www.ijecs.in
International Journal Of Engineering And Computer Science ISSN:2319-7242
Volume 3 Issue 6 June, 2014 Page No. 6615-6619

Critical Analysis of Various Parallel Scheduling Algorithms

Shreya Sharma[1]

[1]Department of Computer Science, Guru Nanak Dev University,
Amritsar, India
Er.shreyasharma@gmail.com

Abstract – ***In this paper, we have discussed various parallel scheduling algorithms and there drawbacks. Out of static scheduling algorithms the Dynamic Critical Path (DCP) Algorithm is the best algorithm. It has an admissible time complexity, is economical in terms of the number of processors used and is suitable for a wide range of graph structures. But multiprocessor scheduling problem is NP-complete in nature even with simplifying assumptions, and becomes more complex under relaxed assumptions such as arbitrary precedence constraints, and arbitrary task execution and communication times. Therefore, a Genetic approach based algorithm was proposed with an objective to simultaneously meet the goals of high performance, scalability, and fast running time known as Parallel Genetic Scheduling (PGS) algorithm. And even outperforms DCP algorithm best known in terms of performance and time complexity.***

Keywords: Parallel scheduling algorithms, Genetic Algorithms, Multiprocessor scheduling problem, DCP, PGS.

1. Introduction

Processing of multiple tasks simultaneously on multiple processors is called parallel processing. These parallel programs consist of multiple active simultaneously solving a given problem. This given task is divided into multiple subtasks using divide-and-conquer technique and each one of them is processed on different CPUs and therefore is known as parallel processing.

Perhaps, [6] the most crucial component of efficient parallel processing system is the scheduling and allocation of the modules of a parallel program to the processors. Because its modules must be properly arranged in time and space in order to optimize the performance. Given a parallel program represented by a task graph, where the nodes represent the tasks and the edges represent the communication costs and precedence constraints among the tasks, a scheduling algorithm determines the execution order of tasks and a mapping algorithm determines the allocation of these tasks to processors.
The task scheduling problem can be best represented as a weighted Directed Acyclic Graph (DAG). In this DAG, a node or vertex represents a task, and the directed edge shows the dependency between two tasks. The basic aim of task scheduling is to schedule tasks onto processors and minimize the makespan of the schedule i.e. the finishing time of the last task relative to the beginning time of the first task i.e. to minimize the execution time. This is equivalent to maximization of the speedup which is defined as the time required for sequential execution divided by the time required for parallel execution.

1.1 Classification of Dag Scheduling Algorithms

Static DAG scheduling problems are still working with major changes in the model graph and machines. In general, the parallel programs come in a variety of structures, and the number of such algorithms has recently been designed to deal with the arbitrary graphs.
These algorithms can be further divided into two categories: The algorithms which consider computational cost of all tasks as uniform, while others have arbitrary value. Some of the algorithms have Intertask communication value equal to zero, that is, there is precedence in the graph but without cost. Scheduling can be done with or without task duplication.
The main aim of the *Task duplication based (TDB) scheduling algorithms* is to reduce the communication overhead by allocating redundant nodes to multiple processors. In this different strategies can be used to select parent nodes for duplication.
Non-TDB algorithms assuming arbitrary task graphs with arbitrary costs on the nodes and edges can be divided into two categories; some scheduling algorithms assume the availability of an unlimited number of processors, while other algorithms assume a limited number of processors. And are named as *UNC (unlimited groups) scheduling algorithm and BNP (limited number of processors) scheduling algorithm* respectively, in both classes of algorithms, processors are assumed to be fully connected and no attention is given to linking strategies contention or routing used for communication. The technique used by the UNC algorithms is also called clustering. At the beginning of the programming process, each node is considered as a group. In the following steps, two groups are merged if the merger reduces the completion time. This procedure continues until the merger cluster can be merged. The reasoning behind

UNC algorithms is that they can take advantage of using multiple processors to further reduce the length of the schedule. However , the groups generated by the UNC may need a post-processing step to match the groups of processors as the number of available processors may be less than the number of bunches .

Some algorithms have been designed to reflect the more general model in which the system is supposed to consist of an arbitrary network topology, whose links are not without contention. These algorithms are called *APN (arbitrary processor network)* scheduling algorithms. They also schedule messages on the network communication links.

2. Dynamic Critical Path Scheduling Algorithm

Yu-Kwong Kwok and lshfaq Ahmad (1996) [5] proposed a static scheduling algorithm for allocating task graphs to fully connected multiprocessors under the UNC category. In this a graph with n number of nodes are scheduled where $w(n_i)$ is the computation cost on node n_i and c_{ij} is the communication cost of edge between n_i and n_j. The dynamic critical path length(DCPL) is calculated in the end by keeping the account of the schedule length at each step $i(SL_i)$. In this it is stated that the scheduling process proceeds, the CP can change dynamically. That is, a node on a CP at one step may not be on the CP at the next step. This is because the communication costs among nodes may be changed to zero if the nodes are scheduled to the same processor. The intermediate scheduling step CP is known as *dynamic critical path* (DCP).

Essentially, the DCP algorithm examines a node n_i for scheduling if, among all nodes, n_i has the smallest difference between its *ALST* (Absolute- Latest-Start-Time) and *AEST* (Absolute-Earliest-Start-Time).

The *AEST* values can be computed by traversing the task graph in a breadth-first manner beginning from the entry nodes so that when is to be computed, all the *AEST* values of n_i*'s* parent nodes are available. The *AEST* of n_i is then simply the latest data arrival time among all its parent nodes. But the communication among two nodes is taken to be zero if they are in the same processor. Therefore, absolute earliest start time of a node n, in a processor J, denoted by AEST (n,, J) is recursively defined as follows:

$$\max_{1\le k\le p}\{AEST\left(n_{i_k}, PE\left(n_{i_k}\right)\right) + w\left(n_{i_k}\right) + r\left(PE\left(n_{i_k}\right), j\right)c_{i_k j}\}$$

Similar to the computation of the *AEST* values, the values of the *ALST* can also be computed by traversing the task graph in a breadth-first manner but in the reverse direction i.e. bottom up. ALST value is computed after the *DCPL* has been computed and is as follows:

$$\min_{1\le m\le q}\{AlST\left(n_{i_m}, PE\left(n_{i_m}\right)\right) - w(n_i) - r\left(PE\left(n_{i_m}\right), j\right)c_{ii_k}\}$$

where n_i has q children nodes and n_{i_m} is the m^th child node.

$$ALST(n_i, J) = DCPL - w(n_i)$$

if it is an exit node. The value of difference between the AEST and ALST is equivalent to the value of the node's *mobility*, defined as:

$$\{Cur\_CP\_Length - b - level(n_i) + t - level(n_i))\}$$

The DCP algorithm uses a *look ahead* strategy to find a better cluster for a given node.

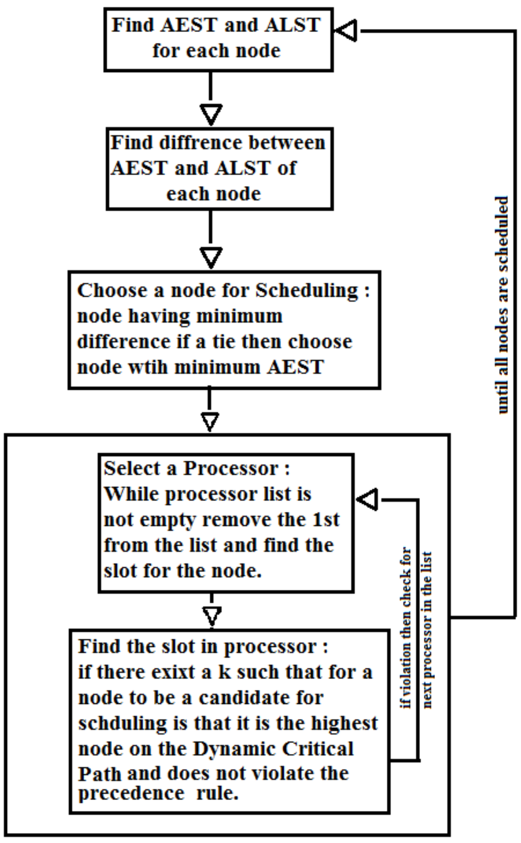
Find AEST and ALST for each node
Find diffrence between AEST and ALST of each node
Choose a node for Scheduling : node having minimum difference if a tie then choose node wtih minimum AEST
Select a Processor : While processor list is not empty remove the 1st from the list and find the slot for the node.
Find the slot in processor : if there exixt a k such that for a node to be a candidate for schduling is that it is the highest node on the Dynamic Critical Path and does not violate the precedence rule.
if violation then check for next processor in the list
until all nodes are scheduled

Figure 1: Working of DCP Algorithm

The time-complexity of the DCP algorithm is O (v^3). Since the DCP algorithm examines the first unscheduled node on the current critical path by using mobility measures, it constructs optimal solutions for fork and joins graph structures.

DCP algorithm is different from the others in the following sense [9]:

- It assigns dynamic priorities to nodes at each step based on the *dynamic critical path* (defined below) so that the schedule length reduces monotonically.
- It changes the schedule on each processor *dynamically* in that the start times of nodes are not fixed until all nodes have been considered.
- It uses an intelligent way [9] to select suitable processor for a node by "looking ahead" the potential start time of the *critical* child node on that processor.
- It schedules relatively unimportant nodes to the processors already in use in order not to overuse processors.

3. Genetic Algorithm

Multiprocessor task scheduling is an NP-hard optimization problem, as the time needed to solve it optimally grows exponentially with the number of tasks and there exist no algorithms for finding an optimal solution in polynomial time. And it becomes more complex under relaxed assumptions such as arbitrary precedence constraints, and arbitrary task execution and communication times. As finding the best way to maximize efficiency in task scheduling process can be extremely complex. Even for a single program there are multiple tasks & constraints and limited number of resources. Therefore, Genetic algorithm is used to which tries to find an optimal solution because unlike heuristic methods genetic algorithm operate on a population of solutions rather than a single solution.

Genetic algorithms (GAs) [10] are adaptive procedures derived from Darwin's principal of survival of the fittest in natural genetics. In searching a large state-space, multi-model state-space, or n-dimensional surface, [3] a genetic algorithm may offer significant benefits over more typical search of optimization techniques.
GA's are general-purpose, stochastic search methods that use the principles inspired by natural selection and genetics (i.e., crossover, mutation). Each point in the well-defined search space of a given problem is called chromosomes. A GA operated by iteratively generating a population of chromosomes that are encoded of the candidate solutions. The quality of the solution represented by a chromosome is evaluated with a function called fitness. The fitness of each chromosome is a measurement of performance of the design variables as defined by the objective function and the constraints of the problem. GAs use global search techniques to explore different regions of the search space simultaneously by keeping track of a set of potential solutions of diverse characteristics. Gas has been widely used for the scheduling problem in number of ways showing the potential of using this class of algorithms for scheduling. Implementation of a genetic algorithm is shown in the following flowchart

3.1 Genetic Operators

- *Selection Operator:* generates a new population of chromosomes by selecting chromosomes from the old population based on their fitness scores. The selection criterion is that chromosomes with higher fitness score should have a higher probability of surviving to the next generation.

- *Crossover Operator:* is a genetic operator used to diverge the encoding of a chromosome or chromosomes from one iteration to the next. Crossover generates new individuals that have some segments of both parent's genetic objects.

- *Mutation Operator:* is a GA operator that modifies one or more gene values in an individual/ chromosome from its preliminary shape. The result of this modification can be entirely new gene values being added to the gene group. The genetic algorithm may be capable to reach at better solution with these new gene values. Mutation is a vital operator of the genetic search as it prevents the population from being idle at any local optima.

4. Parallel Genetic Scheduling (PGS):

Kwok[7] presented a parallel genetic algorithm, called the PGS algorithm, for multiprocessor DAG scheduling. And used this approach so that the re-combinative nature of a genetic algorithm can potentially be determined and an optimal scheduling list leads to an optimal schedule.
Parallelization of the algorithm is based on a approach in which the parallel processors *Np* communicate to exchange the best chromosomes with exponentially decreasing periods. And, the parallel processors perform exploration of the solution-space at the early stages and exploitation at the later stages.

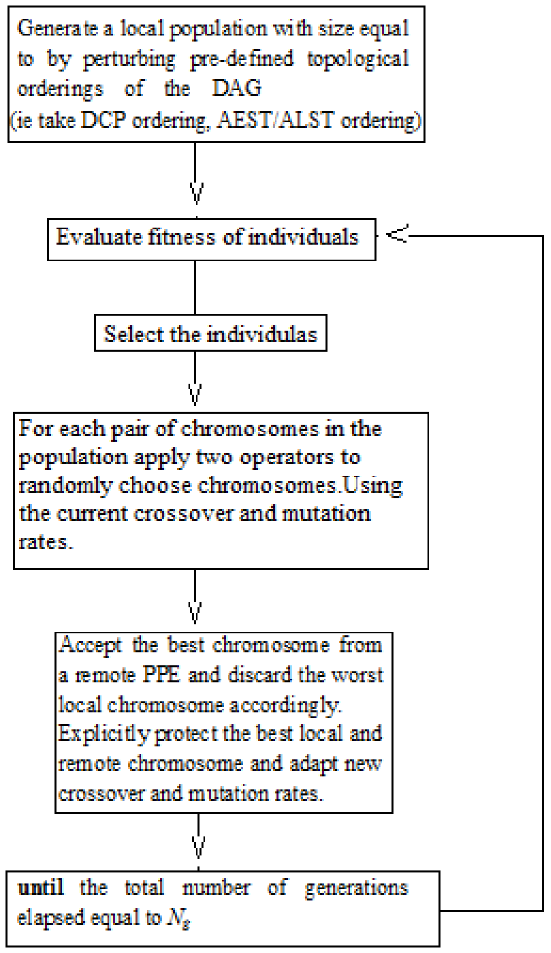
Generate a local population with size equal to by perturbing pre-defined topological orderings of the DAG
(ie take DCP ordering, AEST/ALST ordering)
Evaluate fitness of individuals
Select the individulas
For each pair of chromosomes in the population apply two operators to randomly choose chromosomes.Using the current crossover and mutation rates.
Accept the best chromosome from a remote PPE and discard the worst local chromosome accordingly.
Explicitly protect the best local and remote chromosome and adapt new crossover and mutation rates.
until the total number of generations elapsed equal to N_g

Figure 2: Working of PGS Algorithm

In his experimental studies, he found that the PGS algorithm generates optimal solutions for more than half of all the cases in which random task graphs were used. In addition, the PGS algorithm demonstrates an almost linear speedup and is therefore scalable.
While the DCP algorithm has already been shown to outperform many of the leading algorithms, the PGS algorithm is even better since it generates solutions with comparable quality while using significantly less time due to its effective parallelization. An extra advantage of the PGS algorithm is its scalability, and with the use of more parallel processors, the algorithm can also be used for scheduling large task graphs.
Although the PGS algorithm has shown encouraging performance, further improvements are possible if an optimal set of control parameters, including crossover rate, mutation rate, population size, number of generations, and number of parallel processors used could be determined which is an open research problem.

4.1 Experimental Results

Kwok [7] has compared PGS algorithm with the DCP algorithm. The results in case of Gaussian elimination task graphs are shown in table 1 and revealed that the performance of both algorithms was somewhat similar form more than half of the cases.

Table 1: Avg. Ratios of the schedule lengths generated by the PGS algorithm to that of the DCP algorithm for the Gaussian

elimination task graphs (matrix size ranges from 9 to 18) with three CCRs using 2, 4, 8, and 16 PPEs

CCR	*0.1*	*1.0*	*10.0*
No. of PPEs	*Avg. Ratio*		
2	0.97	1.01	1.01
4	1.00	1.02	1.01
8	0.98	0.99	1.03
16	1.01	0.97	1.05

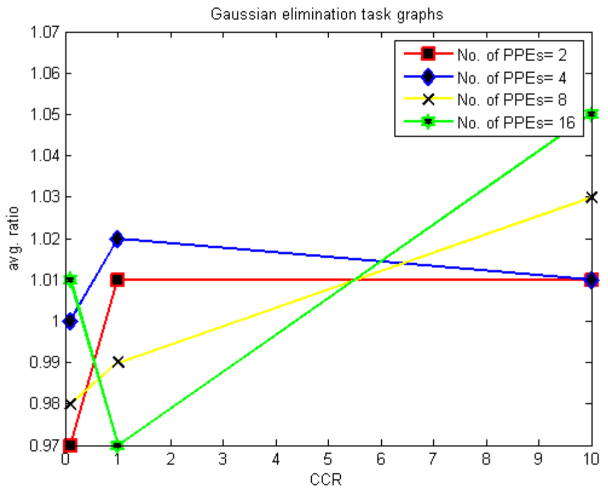

Figure 3: Graphical Illustration of Table 1

The results for the LU- decomposition task graphs are shown in table 2. In this case DCP outperformed the PGS algorithm by producing better schedule lengths. But the reason behind this could be that LU- decomposition task graphs have multiple critical paths, therefore minimization of schedule length is difficult.

Table 2: Avg. Ratios of the schedule lengths generated by the PGS algorithm to that of the DCP algorithm for the LU-decomposition task graphs (matrix size ranges from 9 to 18) with three CCRs using 2, 4, 8, and 16 PPEs

CCR	*0.1*	*1.0*	*10.0*
No. of PPEs	*Avg. Ratio*		
2	1.05	1.05	1.15
4	1.05	1.07	1.10
8	1.03	1.05	1.03
16	1.04	1.03	1.10

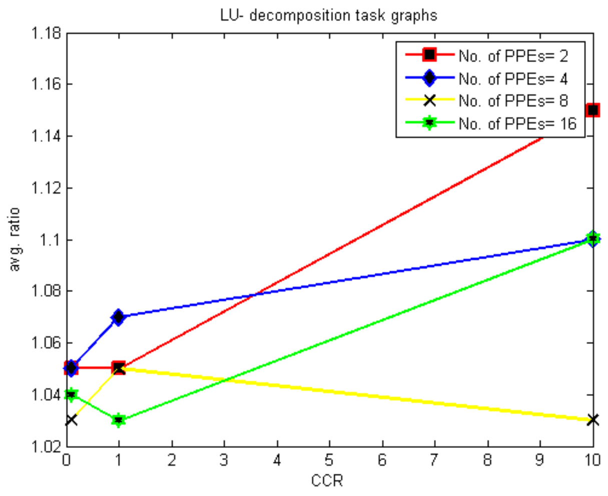

Figure 4: Graphical Illustration of Table 2

The results for the Laplace equation solver task graphs are shown in table 3. In this case also the overall performance of DCP was better than PGS because all the paths in a Laplace equation solver task graph are critical paths. But the average difference in schedule length is not much.

Table 3: Avg. Ratios of the schedule lengths generated by the PGS algorithm to that of the DCP algorithm for the Laplace equation solver task graphs (matrix size ranges from 9 to 18) with three CCRs using 2, 4, 8, and 16 PPEs

CCR	*0.1*	*1.0*	*10.0*
No. of PPEs	*Avg. Ratio*		
2	1.05	1.21	1.18
4	1.14	1.20	1.20
8	1.10	1.21	1.12
16	1.11	1.12	1.15

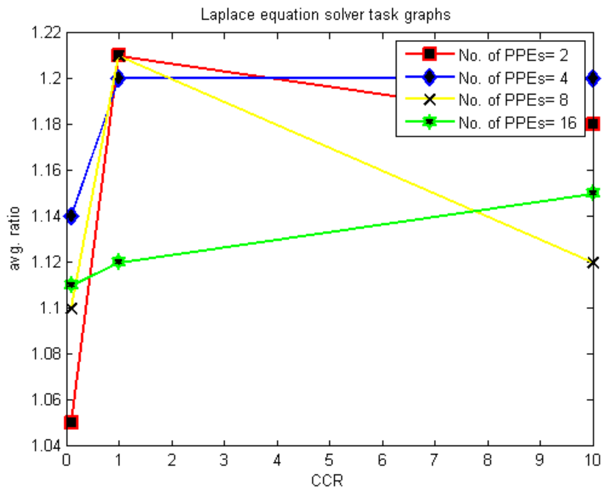

Figure 4: Graphical Illustration of Table 3

From these three types of graphs the results indicate that the performance of the PGS algorithm is comparable to that of DCP algorithm. According to Kwok[7] the reason behind these types of results could be the no. of generations. Therefore, he varied the no. of generations for random type of graph and concluded that PGS algorithm outperformed DCP algorithm for all the smaller values of communication to computation ratio (CCR) (ie. 0.1 and 1) as well as the higher values i.e. CCR= 10. And the PGS algorithm was faster than the DCP algorithm. Following table shows the avg. running time of both the algorithms.

Table 4: Average running times using 1 PPE.

Matrix Size	DCP	PGS
9	5.17	6.17
10	5.55	6.15
11	6.73	8.73
12	7.92	9.92
13	9.63	11.63
14	12.43	13.43
15	16.34	18.34
16	19.32	20.32
17	23.89	25.89
18	28.25	30.25

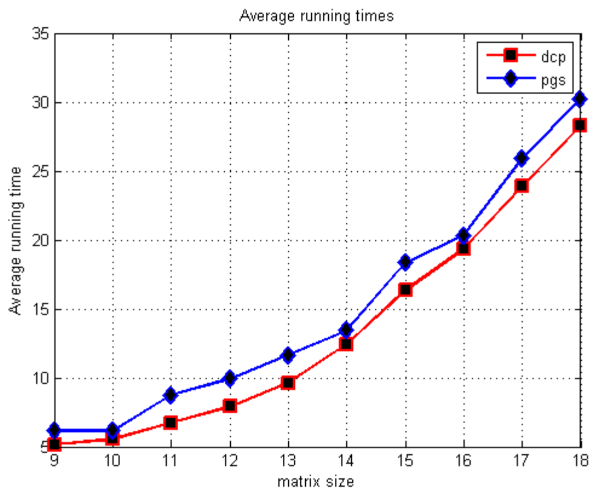

Figure 6: Graphical Illustration of Table 4

5. Conclusion and Future Work

This paper has evaluated various parallel scheduling algorithms and there shortcomings. Among the static scheduling algorithms the Dynamic Critical Path algorithm outperforms than the other algorithms and has an admissible time complexity, is economical in terms of the number of processors used and is suitable for a wide range of graph structures. But it has been found that the multiprocessor scheduling problem is NP-complete in nature even with simplifying assumptions. It is also found to be more complex under relaxed assumptions such as arbitrary precedence constraints, and arbitrary task execution and communication times. Therefore, a Genetic approach based algorithm has been used with an objective to simultaneously meet the goals of high performance, scalability, and fast running time known as Parallel Genetic Scheduling algorithm. It has outperformed than DCP algorithm to evaluate best known in terms of performance and time complexity.

In near future the use of Particle swarm optimization based evolutionary algorithms will be done to enhance the results further. However scalability of the processors and the jobs is also neglected in this research work in near future the scalability issues will also be considered**.**

References

[1] Yi-wen Zhongiz and Jian-gang Yang,“A genetic algorithm for tasks scheduling in parallel Multiprocessor systems”,proceedings of the second international conference on machine learning and cybernetics, 2003

[2] Min-You Wu , Ishfaq Ahmad and Yu-Kwong Kwok, “Analysis, Evaluation, and Comparison of Algorithms for Scheduling Task Graphs on Parallel Processors”, 1996

[3] R Sivaraj “A review of selection methods in genetic algorithm”, IJEST, May 5, 2011

[4] Yu-Kwong Kwok and Ishfaq Ahmad, “Benchmarking and Comparison of the Task Graph Scheduling Algorithms”, 1999

[5] Yu-Kwong Kwok and lshfaq Ahmad “Dynamic Critical Path Scheduling: An effective technique for allocating task graphs to multiprocessors”, , IEEE transaction on parallel and distributed system , Vol. 7, No. 5, May 1996

[6] Yu-Kwong Kwok “Efficient Algorithms for Scheduling and Mapping of Parallel Programs onto Parallel Architectures”, 1994

[7] Yu-Kwong Kwok and Ishfaq Ahmad, “Efficient Scheduling of Arbitrary Task Graphs to Multiprocessors using A Parallel Genetic Algorithm”, Journal of Parallel and Distributed Computing, July 1997

[8] Min-You Wu, “Efficient Local Search for DAG Scheduling”, Senior Member, IEEE, June 2001

[9] YK Kwok, “Static scheduling algorithms for allocating directed task graphs to multiprocessors”, IEEE, 1999.

[10] David E. Goldberg, “Genetic Algorithms in search optimization and machine learning”, published by Pearson Education, 2004