A Duplication Based List Scheduling Genetic Algorithm for Scheduling Task on Parallel Processors

Dr. Gurvinder Singh  
Department of Computer Science & Engineering,  
Guru Nanak Dev University,  
Amritsar- 143001, Punjab, India  
gsbawa71@yahoo.com

Talvinder Kaur  
Department of Computer Science & Engineering,  
Guru Nanak Dev University,  
Amritsar- 143001, Punjab, India  
Er.talvinder@gmail.com

1. ABSTRACT
Task Scheduling is one of the most challenging NP-complete problems in parallel and distributed computing systems. In general scheduling algorithms with task duplication are of better performance than those without duplication, for fine grain tasks graphs and for networks with high communication latencies [1, 2]. This paper presents an efficient and effective way to allocate tasks of an application in the Computing environment. Generally in list based static scheduling where computation time and communication time are known a-priori. First tasks are prioritized and then the processors that minimize the cost function are assigned to the appropriate tasks. Duplication based scheduling is another category of static scheduling. In this category communication costs among the processors are avoided by duplicating the tasks on same processor. This paper presents a duplication based list scheduling that overwhelms the existing scheduling algorithms in both the categories.

Keywords: DAG, Multiprocessor scheduling, Genetic algorithm, Static task scheduling, heuristics, Critical path, Task duplication

2. INTRODUCTION
The problem of scheduling parallel tasks onto multiprocessors is to simply apportion a set of tasks to processors such that the optimal usage of processors and accepted computation time for scheduling algorithm are obtained [1,2]. The assumption of this paper is based on the deterministic model, that is, the number of processors, the execution time of tasks, the relationship among tasks and precedence constraints are known in advance. The precedence constraints between tasks are represented by a Directed Acyclic Graph (DAG). In addition, the communication cost between two tasks is considered to be non-negligible and the multiprocessor system is not diverse and non-preemptive, that is, the processors are homogeneous, and each processor completes the current task before the new one starts its execution.

The complexity of the scheduling problem is very depended to the DAG, the number of processors, the execution time of tasks and also the performance criteria which would to be optimized.

Many heuristics have been proposed in the literature for tasks scheduling problem as there is no exact solution to NP-complete problem. Task scheduling is characterized into two categories: static scheduling and dynamic scheduling. In static scheduling, which is done at compile time, all the information associated with a parallel program such as task processing time, communication time and data dependencies are known a-priori. In dynamic scheduling, many scheduling decisions of a parallel application are taken at run time. Thus the objective of dynamic scheduling is not only to schedule tasks but also consider the scheduling overhead, fault tolerance issues etc. This paper has consideration to static
scheduling. Various static scheduling heuristics have been proposed in the literature. On basis of the approaches these heuristics use, they have been classified into four groups: list scheduling algorithms, cluster based algorithms, duplication based algorithms and random search algorithms. In list based heuristics, tasks are put in a priority list with each task having unique priority value. Priority of a task depends upon priorities of its ancestors. Now tasks from the priority list are taken one by one following three phases: task selection phase, processor selection phase and status update phase. In task selection phase highest priority task is taken for scheduling. In processor selection phase, extracted task is assigned to a processor that optimizes some predefined cost function. The status update phase updates the status of the system. HEFT, CPOP, LDCP etc. are list based heuristics.

In duplication heuristics the highly communicating tasks are redundantly allocated on the same processors. This is to effectively reduce the start time of the waiting tasks and thus improve overall running time of the applications. Duplication based heuristics are useful in case of Computing System having high communication latencies and low bandwidths.

This paper combines list based scheduling and duplication based scheduling approaches and gives a hybrid static scheduling algorithm. Task duplication approach can effectively be used in any list based heuristic. Idea is to effectively use the time slot on processors during which no task has been scheduled by a list based scheduling algorithm.

Parallel Multiprocessor system scheduling can be classified into many different classes based on the characteristics of the tasks to be scheduled, the multiprocessor system and the availability of information. This paper focus on a deterministic scheduling problem.

A deterministic scheduling problem is one in which all information about the tasks and the relation to each other such as execution time and precedence relation are known to the scheduling algorithm in advance.

The tasks should be non-preemptive i.e. task execution must be completely done before another task takes control of the processor, and the processor environment is homogeneous. Homogeneous of processor means that the processors have same speeds or processing capabilities.

The main objective is to minimize the total task completion time (execution time + waiting time or idle time).

The multiprocessor computing consists of a set of m homogeneous processor

\[ P = \{ p_i : i = 1, 2, 3 \ldots m \} \]

They are fully connected with each other via identical links.

Consider a directed acyclic task graph \( G = (V,E) \) of \( n \) nodes. Each node \( V = \{ T_1, T_2, \ldots, T_n \} \) in the graph represents a task. Aim is to map every task to a set \( P = \{ P_1, P_2, \ldots, P_m \} \) of \( m \) processors. Each task \( T_i \) has a weight \( W_i \) associated with it, which is the amount of time the task takes to execute on any one of the \( m \) homogeneous processors. Each directed edge \( e_{ij} \) indicates dependence between the two tasks \( T_i \) and \( T_j \) that it connects. If there is a path from node \( T_i \) to node \( T_j \) in the graph \( G \), then \( T_i \) is the predecessor of \( T_j \) and \( T_j \) is the successor of \( T_i \). The successor task cannot be executed before all its predecessors have been executed and their results are available at the
processor at which the successor is scheduled to execute. A task is “ready” to execute on a processor if all of its predecessors have completed execution and their results are available at the processor on which the task is scheduled to execute. If the next task to be executed on a processor is not yet ready, the processor remains idle until the task is ready.

The elements set C are the weights of the edges as $C = \{c_k: k = 1, 2, 3…r\}$. It represents the data communication between the two tasks, if they are scheduled to different processors. But if both tasks are scheduled to the same processor, then the weight associated to the edge becomes null [4].

$\text{level}(T_i)$ is defined to be the length of the longest path in the task graph from an entry task to $T_i$, excluding the execution cost of $T_i$. Symmetrically, $\text{blevel}(T_i)$ is the length of the longest path from $T_i$ to an exit task, including the execution cost of $T_i$.

**Genetic Algorithms**

A genetic algorithm starts with an initial population that evolves through generations and to reproduce depends on its fitness [5,6]. In this case, the fitness of an individual is defined as the difference between its makespan and the one of the individuals having the largest makespan in the population. The best individual corresponds to the one having the smallest makespan and the largest fitness.

Next, the operators that compose a genetic algorithm are reviewed. The selection operator allows the algorithm to take biased decisions favor good individuals when changing generations. For this, some of the good individuals are replicated, while some of the bad individuals are removed. As a consequence, after the selection, the population is likely to be dominated by good individuals. Starting from a population $P_1$, this transformation is implemented iteratively by generating a new population $P_2$ of the same size as $P_1$.

Genetic algorithms are based on the principles that crossing two individuals can result an offspring that are better than both parents and slight mutation of an individual can also generate a better individual. The crossover takes two individuals of a population as input and generates two new individuals, by crossing the parents' characteristics. The offspring keep some of the characteristics of the parents.

The mutation randomly transforms an individual that was also randomly chosen. It is important to notice that the size of the different populations is same.

**3. LITERATURE SURVEY**

**3.1. An Efficient Parallel Scheduling Algorithm (1996)**

algorithm, the HPMCP algorithm, is proposed. It produces high-quality scheduling and is much faster than existing algorithms. Although the scheduling algorithms apply to parallel programs, the algorithms themselves are sequential, and are executed on a single processor system. A sequential algorithm is slow. Scalability of static scheduling is restricted since a large memory space is required to store the task graph. A natural solution to this problem is using multiprocessors to schedule tasks to multiprocessors. In fact, without parallelizing the scheduling algorithm and running it on a parallel computer, a scalable scheduler is not feasible.

Samantha Ranaweera and Dharma P. Agrawal [7] introduced a scalable scheduling scheme called STDS for heterogeneous systems. This implies that tasks could potentially have different runtimes on different processors. Schedule length is primarily reduced by selected task duplication. Current task duplication based scheduling schemes are mostly done for homogeneous systems. Comparing the performance of STDS with BIL, another scheduling scheme for heterogeneous systems, it is observed that STDS obtained speed-ups of 6 to 40 generating shorter schedules when sufficient duplication can be carried out. The basic idea behind cluster based scheduling is to cluster tasks that communicate with each other into a single processor. However if the available number of processors is less than the number of clusters generated, these methods fail to deliver acceptable results. In the case of heterogeneous systems, this issue is exacerbated as if only communication is considered, tasks may be allocated to processors on which they have extremely high runtimes. This negates the effects of minimizing communication.

3.3. A Task Duplication Based Scalable Scheduling Algorithm for Symmetric Multiprocessors (2000)
Oh-Han Kang and Dharma P. Agrawal [8] presented a task duplication based scalable scheduling algorithm for Symmetric Multiprocessors (SMP), called S3MP (Scalable Scheduling for SMP), to address the problem of task scheduling. The algorithm pre-allows network communication resources so as to avoid potential communication conflicts, and generates a schedule for the number of processors available in a SMP. The suggested algorithm employs heuristics to select duplication of tasks so that schedule length is reduced/minimized. The performance of the S3MP algorithm has been observed by comparing the schedule length under various number of processors and the ratio of communication to computation cost.

3.4. A Task Duplication Based Scheduling Algorithm with Optimality Condition in Heterogeneous Systems (2002)
Tae-Young Choe and Chan-Ik Park [9] proposed a task scheduling algorithm based on task duplication with an optimality condition to determine whether or not the resulting schedule has the shortest schedule length. The optimality condition is that, which is defined as follows: If a given DAG satisfies optimality conditions of a task scheduling algorithm, the task scheduling algorithm schedules the DAG with the shortest schedule length. Such algorithms have the merit that schedule with the shortest length is guaranteed and no further refinements are required as long as input DAGs satisfy the conditions. They considered scheduling algorithms in heterogeneous systems given two properties. First, the execution time of a task in a processor is the multiplication of the weight of the task and the execution time of the unit data in the processor; Second, communication overhead does not depend on the source processor or the target processor.
Amir Masoud Rahmani and Mojtaba Rezvani [10] proposed a new genetic algorithm, named TDGASA, in which its running time depends on the number of tasks in the scheduling problem. Then, the computation time of TDGASA to find a sub-optimal schedule is improved by Simulated Annealing (SA). The results show that the computation time of the proposed algorithm decreases compared to an existing GA-based algorithm, although, the completion time of the final scheduled task in the system decreases a little.

4. GAPS IN LITERATURE SURVEY
- In TDS, there is a problem that the number of processors must be unrestricted. This condition can hardly matched in practice, in which the number of the processors must be restricted because of equipment space and other reasons in a embedded real-time distributed system.
- Most of the scheduling algorithms assume the presence of a fully-connected network where in all processors can communicate with each other directly. Task scheduling on bus-based SMP is different from that on a fully-connected network.
- Current task duplication based scheduling schemes are mostly done for homogeneous systems. Little work is done on heterogeneous systems.
- One main problem among genetic algorithms is that there is not enough considering of task's precedence, communication cost, computation cost, algorithm complexity and other factors.

5. PERFORMANCE METRICS
Our scheduling objective is to optimize certain metrics. These metrics are makespan, schedule length ratio and processor utilization. These metrics are generally used for evaluation of the scheduling algorithms. These metrics are described below.

**Makespan**
It is the interval between the start of the first task and completion time of the last task. It is the maximum completion time of an algorithm and is calculated by measuring the finishing time FST (n exit) of the scheduled exit task by the algorithm. The algorithm is efficient if it takes less time to execute means low makespan.

Makespan = 30.

**Schedule Length Ratio (SLR)**
SLR is one of the important performance measures of scheduling algorithm. SLR is defined by

$$\text{SLR} = \frac{\text{makespan}}{\sum_{i,j} \text{CP} \min_{i,j}}$$

where makespan is the overall schedule length. The denominator is the sum of minimum computation costs of tasks on the CPmin. CPmin is the critical path which is obtained by minimum computation cost assignment to the nodes of DAGs. The SLR can never be less than one, since the denominator is the lower bound. Algorithm that gives smallest SLR of a graph, is the best algorithm with respect to performance.

Schedule length = 18

**Processor Utilization**
It means for what percentage of the processor is utilized. It is calculated by dividing the execution time of tasks scheduled on the processor by the makespan of the algorithm. Processor utilization (PU) is given as:
PU = \frac{\text{(Total execution time of tasks on processor/makespan)}}{100}

**Speedup**

Another performance measure of scheduling algorithm is speed up. Speed up is defined by

\[
\text{Speed up} = \frac{\min_{P_j \in Q} \sum_{N_i \in V} W_{i,j}}{\text{makespan}}
\]

The numerator is sequential execution time. Sequential execution time is computed by assigning all tasks to single processor that minimizes the overall computation time. The denominator is parallel execution time. Higher the speed up of an algorithm decides the goodness of the algorithm with respect to speed p.

\[
\text{Speedup} = \frac{61}{30} = 2.1
\]

### 6. PROPOSED ALGORITHM

**6.1 Outline of the algorithm**

GAs operates through a simple cycle of stages: creation of population strings, evaluation of each string, selection of the best strings and reproduction to create a new population. The number of genes and their values in each chromosome depend on the problem specification.

In this paper, the number of genes of each chromosome is equal to the number of the nodes (tasks) in the DAG and the gene values demonstrate the scheduling priority of the related task to the node (each chromosome shows a scheduling), where the higher priority means that task must be executed early. In the basic genetic algorithm the initial population is generated randomly, which can cause to generate more bad results. Selection of nodes for duplication is different than duplication based algorithm in proposed algorithm. To reduce the start time of nodes, some algorithms duplicate only the parent nodes as well as some algorithms try to duplicate ancestors at higher level. We have implemented the concept of Task Duplication in MCP Algorithm[4].

Our Algorithm consist of mainly two phases:

1. MCP scheduling phase: Modified Critical Path algorithm initially calculate the value of ALAPs of all nodes and after that erects the list of nodes in increasing order of nodes ALAP. When the ALAP values of two nodes become same, the ALAPs of the children are taken into concern. It should be noted that the MCP algorithm schedules the nodes on the list taking one by one such that a node is scheduled to the processor or work-station that allows the earliest execution start time.

2. Task duplication Phase: The task which have highest effectiveness is duplicated to the idle processor. We made a function to check whether nk can be duplicated on pj or not, it checked the availability of pj at that time as well as checked that is it really worthy to duplicate nk.
6.2 Chromosome representation
According to this genetic algorithm the chromosome is divided into two sections; mapping and scheduling sections. The mapping section contains the processors indices where tasks are to be run on it. The schedule section determines the sequence for the processing of the tasks. Fig. 6.1 shows an example of such a representation of the chromosome. Where, tasks t4, t7, t8 will be scheduled on processor P1, tasks t3, t5 will be scheduled on processor P2, and tasks t1, t2, t6 and t9 will be scheduled on Processor P3. The length of the chromosome is linearly proportional to the number of tasks.

6.3 Initial Population
The initial population of the second part (schedule) of the chromosome can be constructed using ALAP. The algorithm initially calculate the value of ALAPs of all nodes and after that erects the list of nodes in increasing order of nodes ALAP

6.4 Fitness Function
The algorithm uses makespan as objective function for evaluating the schedules of each chromosomes. The proposed GA assigns zero to an invalid chromosome as its fitness value. Our fitness function is defined as

\[ f(i) = \frac{\text{Makespan} - \text{maximum execution time}}{\text{makespan}} \]

where f(i) is the fitness of chromosome i.

6.5 Crossover
Two point crossover operator is used. According to the two point crossover, two points are assigned to bind the crossover region. Since each chromosome consists of a vector of task processor pair, the crossover exchanges substrings of pairs between two chromosomes. Two points are randomly chosen and the partitions between the points are exchanged between two chromosomes to form two offspring. The crossover probability \( \mu_c \) gives the probability that a pair of chromosome will undergo a crossover. An example of a two point crossover is shown in Fig. 6.5.

6.6 Mutation
The mutation probability \( \mu_m \) indicates the probability that an order pair will be changed. If a pair is selected to be mutated, the processor number of that pair will be randomly changed. An example of mutation operator is shown in Fig. 6.6.
Schedule for Phase 1: List Scheduling

P1: T1 → T2 → T5 → T6
P2: T3 → T7
P3: T4 → T9 → T8

Schedule for Phase 2: Task Based Duplication

PI: T1 → T5 → T4 → T8 → T9
P2: T1 → T2 → T6
P3: T1 → T3 → T7

7. CONCLUSION AND FUTURE WORK

In this paper we present a duplication based list scheduling algorithm for scheduling task of an application onto a computing system. Random task graphs are generated by deciding important input parameters like number of nodes, communication to computation cost ratio. This selection makes a wide range of task graphs with various characteristics. Processor utilization, schedule length ratio and makespan for various generations of genetic algorithm are the three metrics that decide goodness of a scheduling algorithm. Experimental results show that duplication based list scheduling heuristic often outperforms many other scheduling algorithms in duplication based algorithm category as well as in list based scheduling algorithm category. Makespan is low and speedup is high. In future we will extend the work to partially connected resources. Availability based scheduling is another future direction to work, where some processors may have internal job queues.
Fig 7.2(a): No. of nodes vs. SLR

Fig 7.2(b): No. of nodes vs. SLR

Fig 7.2(c): No. of nodes vs. SLR

Fig 7.3(a): No. of nodes vs. PU

Fig 7.3(b): No. of nodes vs. PU

Fig 7.3(c): No. of nodes vs. PU
Fig 7.4 No. of Node vs. average Speedup

8. REFERENCES


