SCHEDULING OF PARALLEL CONSTRAINED TASK GRAPHS ON MULTIPROCESSOR SYSTEMS
Shailza Kamal
Department of Computer Science, Baba Hira Singh Bhattal Engg. College, Lehragaga (Punjab), India.

Abstract- Efficient scheduling of computationally intensive program is one of the most essential and most difficult issues to achieve high performance in a parallel computing. Finding an optimal solution for a scheduling problem is NP-complete. Therefore it is necessary to have heuristic to find a reasonably good schedule. This work focuses on task scheduling (multiprocessor scheduling) keeping in view its significance for the most complex scientific and engineering applications. So this work aims at implementing list based scheduling heuristic with different types of task priorities and to test the appropriateness of multiprocessors system over uniprocessor system in regards to prominent performance parameters like schedule length and speed.

Keywords- static scheduling, dynamic scheduling, normalized schedule length, makespan.

I. Introduction
Advances in the hardware and software technologies have led to increase in the use of large scale parallel and distributed system for database, real time, defence and large scale commercial applications. One of the biggest issues in such systems is the development of effective techniques for the distribution of processes of a parallel program on multiple processors. The problem is how to distribute (schedule) the processes among processing elements to achieve some performance goals, such as minimizing execution time, minimizing communication delays and maximizing resource utilization. Following are the key factors for scheduling evaluation:

Two main scheduling approaches are: Static Scheduling: It predicts the program execution behaviour at compile time. Dynamic Scheduling: It predicts the program execution behaviour at run-time. Mainly on parallel and distributed system, static scheduling is commonly used because it predict the program execution behaviour at compile time and it perform a partitioning of smaller tasks into coarser-grain processes in an attempt to reduce the communication costs.

This paper addresses only the static scheduling problem. The task interaction graph model, in which vertices represent parallel processes and edges denote the inter-process interaction is usually used in static scheduling of loosely- coupled communicating processes (since all tasks are considered as simultaneously and independently executable, there is no temporal execution dependency) to a distributed system. Scheduling problems may broadly classify as:

(1) Job scheduling and (2) Task scheduling.

Job scheduling deals with the scheduling of independent jobs; whereas tasks scheduling is related to scheduling of tasks (processes) belonging to a single application program. Generally the objective of job scheduling is to have good load balancing among the processors, whereas for the later minimization of overall execution time is the main concern [8]. The prime objective behind both is to maximize the parallel system’s throughput by completing maximum number of jobs in the given time span. This work focuses on task scheduling (multiprocessor scheduling) keeping in view its significance for the most complex scientific and engineering applications.

II. Problem Definition
Parallel Machine scheduling also known as Parallel Task Scheduling, involves assignment of multiple tasks onto the system architecture’s processing components. So, the precedence-constrained parallel applications in scientific and engineering fields are most typical application model.
The problem of efficient scheduling of the parallel application task graphs (task scheduling) onto a given multiprocessor system with the purpose of reducing the overall make-span shall be the main focus of this work.

### III. Scheduling Heuristics

The objective of scheduling is to minimize the overall finish-time of the parallel program by proper allocation of the tasks to the processors and arrangement of execution sequencing of the tasks. Scheduling is done in such a manner that the precedence constraints among the program tasks are preserved. The overall finish-time of a parallel program is commonly called the schedule length or makespan. Some variations to this goal have been suggested.

#### 3.1 List Scheduling

The objective of scheduling is to map the tasks onto the processors (machines) and order their execution so that task dependencies are satisfied and minimum overall scheduling length (makespan) is achieved [2] mainly on parallel and distributed system. Most scheduling algorithms are based on the so-called list scheduling technique [18]. List based heuristics are the most primitive ones that arrange the nodes of the graphs in the form of a list, based on some priority in general which may be calculated statically and dynamically. The highest priority reads task is then selected and schedule on the most suitable processor. Based on the priority calculation phase (compile time or run-time), the list scheduling can be static or dynamic.

#### 3.2 Clustering Based Scheduling

Clustering based scheduling is the second most widely studied and exploited techniques in the history of multiprocessor scheduling. However, unlike the list based approach that allocates and assigns a task to a processor in a single step, cluster is basically a two phase heuristics, in the first phase, tasks that must go together on the same processor, owing to their higher interprocessor communication costs are decided upon. These tasks are then mapped to a common cluster under the ambience of cluster heuristics. There by generating unbounded number of clusters for the whole task graph. In the second phase, these clusters are mapped onto the physical processors, and the tasks with in a cluster are sequentilized using a priority based approach [4,12].

#### 3.3 Duplication Based Scheduling

Duplication heuristics are more effective, in general, for fine grain tasks graphs and for networks with high communication latencies. However, most of the available duplication algorithms are designed under the assumption of unbounded availability of fully connected processors, and lie in high complexity range. Low complexity optimal duplication algorithms work under restricted cost and/or shape parameters for the task graphs.[4]

### IV. Basic Application/Task Model

A parallel program can be represented by a node- and edge-weighted directed acyclic graph (DAG), in which the node weights represent task processing times and the edge weights represent data dependencies as well as the communication times between tasks.

A DAG - $G = (V, E, w, c)$ - that represents the application to be scheduled

- $V = \{v_i : i = 1, \ldots, N\}$ represents the set of tasks.
- $E = \{e_{ij} : \text{data dependencies between node } n_i \text{ and node } n_j\}$
- $w(n_i) \rightarrow$ represents the node $n_i$'s computation cost
- $(e_{ij})$—represents the communication cost between node $n_i$ and node $n_j$}

![Figure 1: Application directed acyclic graph](image-url)

As the DAG scheduling problem is NP-complete in general, a number of heuristics have been proposed.
V. Static List Based Scheduling

Static scheduling of a program represented by a directed task graph on a multi-processors system to minimize the program completion time is a well-known problem in parallel processing [9]. Since finding an optimal schedule is an NP-complete problem in general, researchers have resorted to devising efficient heuristics.

Table 1: Priority attributes for the given DAG

<table>
<thead>
<tr>
<th>Node</th>
<th>s_level</th>
<th>t_level</th>
<th>b_level</th>
</tr>
</thead>
<tbody>
<tr>
<td>t_1</td>
<td>75</td>
<td>0</td>
<td>99</td>
</tr>
<tr>
<td>t_2</td>
<td>55</td>
<td>28</td>
<td>71</td>
</tr>
<tr>
<td>t_3</td>
<td>45</td>
<td>24</td>
<td>55</td>
</tr>
<tr>
<td>t_4</td>
<td>50</td>
<td>22</td>
<td>60</td>
</tr>
<tr>
<td>t_5</td>
<td>35</td>
<td>24</td>
<td>57</td>
</tr>
<tr>
<td>t_6</td>
<td>25</td>
<td>52</td>
<td>27</td>
</tr>
<tr>
<td>t_7</td>
<td>30</td>
<td>50</td>
<td>34</td>
</tr>
<tr>
<td>t_8</td>
<td>35</td>
<td>50</td>
<td>43</td>
</tr>
<tr>
<td>t_9</td>
<td>20</td>
<td>78</td>
<td>20</td>
</tr>
</tbody>
</table>

The objective of scheduling is to minimize the completion time of parallel application by properly allocating the tasks to the processors. A parallel program, therefore, can be represented by a node- and edge-weighted directed acyclic graph (DAG), in which the node weights represent task processing times and the edge weights represent data dependencies as well as the communication times between tasks [1].

For the DAG shown in Figure 1, the priority list according to different priority schemes i.e. t_level, b_level and s_level are shown on Table 1.

A. Phase 1: Task Priority Assignment

In phase 1 the task priority can be done in different manners according to the different priority levels and the task sequence or list is constructed.

- static_level based task priority list:
  - < T1, T2, T4, T3, T5, T8, T7, T6, T9>
- bottom_level based task priority list assignment:
  - < T1, T2, T4, T5, T3, T8, T7, T6, T9>
- top_level based task priority list assignment:
  - < T9, T8, T7, T6, T5, T2, T3, T4, T1>

B. Phase 2: Processor Selection Phase

In this phase, tasks are picked up one by one from the above generated list and assigned to a processor that can execute it at the Earliest Start Time or the earliest execution (finish time) time so that the overall makespan of the task graph is minimized.

1) List Scheduling, based on Static level:

Gantt chart representation for static level priority.

(2) List Scheduling, based on Bottom Level Priority

Gantt chart representation for bottom level priority.

(3) List scheduling, based on Top-level priority
VI. Experimentation

A performance comparison of three priority heuristics namely static-level priority, bottom-level priority and top-level based priority on a multiprocessor system comprising of different number of processors. For this purpose a randomly task graph is first generated with different design parameters like number of task nodes, varying execution and communication costs and random indegree and outdegree of these nodes. Different priority levels are then calculated for each of these nodes in a depth first search manner and a sorted list is prepared. These nodes are then scheduled on different number of processors by selecting the most suitable processor.

A. Performance Metrics

The performance and comparison of different priority based list schedules are based on the following metrics.

1. Normalized schedule length (NSL)

NSL gives the normalized makespan generated by an algorithm with respect to the absolute lower bound, which is nothing but the maximum static level of a task in DAG. The main performance is the scheduling length of its output schedule. Since a large set of task graph with different properties is used, it is necessary to normalize the schedule length to lower bound. It is the ratio of actual makespan to the critical path length of the DAG (representing lower bound of the makespan). Ideally it should be one.

2. Efficiency

Efficiency of any particular task can be calculated by dividing the speed-up to the total number of processors. More the parallelism in the given DAG, efficiency will be higher.

Efficiency = average speed-up / number of processors

B. Simulation Parameters

In this work, the random task graphs (DAGs) are generated with 25, 50, 75, 100, 150, 250 and 500 number of nodes. The multiprocessor network is fully connected network with number of processors as 4, 8 and 16. The other parameters are as shown in the table.

1. Performance analysis of NSL on different no. of processors

Figure 2: average NSL for three priority levels on different no. of nodes for p=4
2. Performance analysis of average Efficiency of three priority levels on different no. of processors

Figure 3: average NSL for three priority levels on different no. of processors for p=16

Figure 4: average efficiency graph for three priority levels on different no. of nodes for p=4

Figure 5: average efficiency graph for three priority levels on different no. of nodes for p=16
VII. Conclusion

In this work, a comparison of list scheduling heuristic based on different priority schemes is taken up and results presented after exhaustive simulations on a randomly generated set of task graphs. Scheduling is one of the vital design issues for achieving the optimal efficiency of parallel architectures. Parallel programs are assumed to be available in the form of directed acyclic task graphs. A simulator is developed and tested through simulation on multiprocessor system with different number of processors using random task graphs with widely varying design parameters. Performance is analyzed using NSL, Speedup and efficiency metrics.

Based on the results obtained following major conclusion are drawn-

- List scheduling is a simple and effective scheduling heuristic.
- Different priority schemes in the first phase of task sequence generation phase can significantly affect the performance of the scheduling algorithm.
- Bottom level priority scheme is seen to be most effective in comparison to s-level and t-level priority heuris

VIII. Future Scopes

This paper mainly focused on the static task scheduling problem. In static scheduling, knowledge about the characteristics of the application such as task execution time, communication cost, and task dependencies are assumed to be available before execution, and the schedule is generated off-line. One of the directions to extend this work is to develop dynamic scheduling algorithms to overcome the limitations imposed by static scheduling. Dynamic schedulers use information available at run-time to make scheduling decisions.

REFERENCES