An Augmentation-Based Algorithm for Task Scheduling in Parallel Systems

Raj Vemulapalli
Cap Gemini America
Des Moines, IA 50266

Abstract

The problem of scheduling tasks on parallel systems has been shown to be computationally intractable in its general form as well as many restricted cases. In this paper, we introduce a two-step Augmentation based algorithm for scheduling general task graphs in parallel systems. Several experimental studies have been conducted to compare the performance of the proposed technique with several known heuristics. The obtained results show that the augmentation based algorithm out-performed other heuristics on most of the randomly-generated task graphs.

1. Introduction

Task Scheduling is one of the most challenging problems in parallel and distributed computing. It is known to be NP-complete in its general form as well as several restricted cases [1]. In this paper, we propose an algorithm that employs a two-step augmentation technique for solving the general scheduling problem with and without considering the communication cost. In the first step, the algorithm augments the input task graph, by adding as few relations (precedence edges) as possible, in order to obtain an interval order. In the second step, the algorithm uses the scheduling algorithms obtained in [1,2] to find an optimal schedule of the augmented graph. We use the same notation and assumptions used in [1]. A task graph is an interval order when its elements can be mapped into intervals on the real line and two elements are related if and only if the corresponding intervals do not overlap [1,2]. Interval orders have a useful property that relates the successors of a pair of nodes in any interval order.

Lemma 1 [2] For any interval order \( P = (V,E) \) and \( u,v \in V \), either \( N(v) \subseteq N(u) \) or \( N(u) \subseteq N(v) \) where \( N(v) = \{ u \mid (v,u) \in E \} \).

A simple greedy algorithm to find an optimal schedule in the case when the execution time of all interval ordered tasks is same was presented by Papadimitriou and Yannakakis in [2]. This basic idea is reflected in Algorithm 1 below.

Algorithm 1:
1. The number of successors of each node is used as its priority.
2. Whenever a processor becomes available, assign it the ready task with the highest priority.

2. Interval order augmentation

Given a task graph \( G = (V,A) \), we can find an augmented task graph \( I = (V,A \cup AI) \) that is interval ordered, which is constructed by adding some additional arcs \( AI \) to \( G \). The following theorem shows that this augmentation process is always possible.

Theorem 1: Given a task graph, \( G = (V,A) \), there always exists another graph \( I = (V,A \cup AI) \) such that \( I \) is an interval order, where \( AI \) is a set of directed edges augmented to \( G \).

Figure 1 shows a task graph and a possible augmented interval order that has only one more edge. Although it is preferable to find the minimum number of directed edges that need to be augmented to a task graph to form an interval order, this problem is rather difficult. Given the history of similar augmenting problems, this problem is not likely to be tractable. Currently, we are working on the construction of a proof that this augmentation problem is NP-hard. However, the following algorithm provides an efficient heuristic for obtaining an augmented interval order from a given task graph. The algorithm is in the form of a function that returns the augmented interval order.

Algorithm 2:
1. The number of successors of each node is used as its priority.
2. Until all tasks are assigned for execution, pick the ready task with the highest priority and assign it to the processor that could start its execution the earliest. Ties are broken in a way such that the task is assigned after the task with the lowest priority.

Figure 1: (a) A task graph and (b) Its augmented interval order
Algorithm 3:
Input: A Task Graph \( G = (V, A) \)
Output: An augmented interval Order \( I = (V, A \cup A_1) \)

Function Augment(G)
Begin
\( A_1 \leftarrow \emptyset \)
While \((V, A \cup A_1)\) is not an interval order do
begin
let \( u \) be the node with the highest number of successors and let \( v \) be the node with the second highest number of successors
\( A_1 \leftarrow A_1 \cup (u, w), \) where \( w \in N(v) \) and \( w \notin N(u) \)
end
forevery edge \((x, y) \in A_1\) do
if \((V, A \cup A_1 - (x, y))\) is an interval order
then \( A_1 \leftarrow A_1 - (x, y) \)
Augment(G) \( \leftarrow (V, A \cup A_1) \)
End.

Algorithm 3 augments a given graph \( G = (V, A) \) with directed edges and produces an interval order. The while loop adds one edge at a time, \((u, w)\), where \( u, w \in V \) and \((u, w) \notin A_1\), until the input graph \( G \) augmented by the new edges becomes an interval order. As there may be some unnecessary edges in the newly constructed interval order \( I \), the for every loop tries to remove such extra edges. In each iteration, it removes one edge at a time from the edges that were added to the augmenting graph \( G \), selected randomly, as long as the graph maintains the property of being an interval order. It is easy to see that Algorithm 3 can be implemented in \( O(n^2) \). Figure 1 shows a possible example of a task graph and its augmented interval order constructed using Algorithm 3.

3. The scheduling algorithm

In this section, we present a two-step scheduling algorithm for an arbitrary task graph \( G = (V, A) \). In the first step, Algorithm 3 is used to construct an augmented interval order \( I = (V, A \cup A_1) \) for the input task graph. In the second step, based on whether the communication cost among the processors is considered, one of the two optimal algorithms introduced in Section 2 is used to provide an optimal schedule of the augmented graph \( I \). Clearly, a schedule for \( I \) will be a feasible schedule for \( G \) since it honors every precedence relation of \( G \). How close the optimal schedule of \( I \) to the optimal schedule of \( G \)? The answer depends highly on how many extra relations were added to \( G \). If \( I \) is constructed from \( G \) by adding few relations, relative to the number of the original relations in \( G \), then the two schedules are more likely to be very similar. In the following algorithm, the Boolean variable \( \text{COM} \) is assumed to be true when communication cost needs to be considered and false otherwise.

Algorithm 4:
Input: A Task Graph \( G = (V, A) \), \( N \) processors and a Boolean variable \( \text{COM} \)
Output: A feasible schedule of \( G \).

Begin
\( I = \text{Augment}(G) \)
If \( \text{COM} \) then Algorithm 2 \( (I, N) \)
else Algorithm 1 \( (I, N) \)
End.

Figure 2: The Proposed Two Step Scheduling Technique

Figure 2 shows the results of applying Algorithm 4 on the graph of Figure 1(a) without and with communication.

Table 1 and Table 2 below present a summary of the experimental results that demonstrates that Interval Order Augmentation performs better than other scheduling algorithms in most cases. When communication is not considered, results of comparing our Interval Order Augmentation (IOA) algorithm with the Highest Level First with No Estimated Time (HLFNET) scheduling algorithm are tabulated. When the communication cost is considered, we compared the performance of our Interval Order Augmentation with the performance of the Insertion-Scheduling Heuristic (ISH). It is clear from the results that the proposed algorithm performs extremely well in the case when communication cost is not considered. However, it performed only fairly when the communication effect is encountered.

Table 1: Results of Comparing IOA with HLFNET

<table>
<thead>
<tr>
<th></th>
<th>IOA performed better than HLFNET</th>
<th>IOA performed the same as HLFNET</th>
<th>HLFNET performed better than IOA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>64% of the times</td>
<td>56% of the times</td>
<td>76% of the times</td>
</tr>
</tbody>
</table>

Table 2: Results of comparing IOA with ISH

<table>
<thead>
<tr>
<th></th>
<th>IOA performed better than ISH</th>
<th>IOA performed the same as ISH</th>
<th>ISH performed better than IOA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>6% of the times</td>
<td>96% of the times</td>
<td>18% of the times</td>
</tr>
</tbody>
</table>

References


Proceedings of the 1996 Hawaii International Conference on System Sciences (HICSS-29)
1060-3425/96 $10.00 © 1996 IEEE