A Scheduling Algorithm for Fork-Join DAG in Bus-based and Heterogeneous Environment

Lisheng Wang, Liguo Chen, Kete Wang
Department of Computer Science
College of Electronic and Information Engineering
Tongji University
Shanghai, China
lishwang@tongji.edu.cn
chenliguo1989@163.com

Abstract—The Fork-Join task graph is one of the basic modeling structures for parallel processing. However, many previous scheduling algorithms ignore to economize processors and minimize the total completion time. What's more, many algorithms don't consider the competition caused by bus-based clusters and the heterogeneous of processors in real applications. This paper presents a new algorithm for Fork-Join task graph, considering economy of processors and minimization of the total completion time, the non-parallel communication, and heterogeneous environment as well. We propose a task scheduling algorithm based on task duplication which randomly generated a number of Fork-Join task graphs by producing the task execution time and communication time. Simulation results show that the proposed algorithm has less total completion time and less number of processors than other compared algorithms for more practical applications.

Keywords—Task scheduling, Task duplication, Heterogeneous, Bus-based, Fork-join task graph

I. INTRODUCTION

The goal of task scheduling algorithm is to allocate the tasks of a parallel program to processors in order to minimize the completion time of the program and economize processors. Since it has been shown the multiprocessor scheduling problem is NP-complete, many researchers have proposed scheduling algorithms based on heuristics [1-14]. It is well known that the complexity and quality of scheduling algorithm largely depend on the task graph structure [3, 4]. The fork-join structure is one of the basic modeling structures for parallel processing.

Since the communication time between tasks assigned to the same processor is considered to be negligible, task duplication is one way of reducing overhead of the interprocessor communication. By using this approach, some of more critical tasks of a parallel program are duplicated on more than one processor. This can potentially reduce the start time of the waiting tasks and eventually improve the overall completion time of the entire program. It has been shown that the scheduling algorithms based on task duplication always perform better than any other scheduling algorithms without task duplication.

There are several task duplication based scheduling schemes. When certain conditions are met for the given task set with parallel homogeneous processors, TDS[1] algorithm proposed is able to find an optimal schedule. However, it ignores to economize the processors so that the speedup of parallel programs is decreased. TDS algorithm assumes that algorithms work in interconnected network environment and the communication between the processors can be executed in parallel. However, each processor in the sending and receiving multiple messages are serial in a not interconnected network, such as the bus-based clusters. We should not only consider the start time and end time of the task, but also consider the start and the end time of communication between processors due to the serial communication. Task Scheduling Algorithm (TSA_FJ) [7] only tries to utilize the idle time slot of the homogeneous processors, but the workload of other processors are not considered. This algorithm assumes that the communication between processors must be the serial. What’s more, many algorithms assume that each processor has the same execution speed and the same communication time to fixed length messages. Experimental results show that this assumption would not result in good results in the practical application. Task scheduling algorithm should be based on a heterogeneous environment and take into account the serial communication between processors for better adapting to the actual systems.

In this paper, we propose a new algorithm based on fork-join task graphs that can be used in heterogeneous systems according to the algorithms used in homogeneous environment.

The algorithm based on bus-based clusters uses task duplication method to improve the scheduling performance. Section 2 describes the systems model and problem definition. Section 3 presents the analysis of our approach, followed by a description of the proposed algorithm and an example to demonstrate the operation of the algorithm. Section 4 includes the performance results and comparisons with other algorithms. We provide concluding remarks in the last section.

II. SYSTEMS MODEL AND PROBLEM STATEMENT

We assume that a parallel program can be expressed as a directed acyclic graph (DAG), which is also called a task graph. A DAG consists of a tuple (V, E, w, c), where V, E, w, c are the set of task nodes, the set of communication edges, the set of computation costs associated with the task nodes,
and the communication costs associated with the edges respectively. \(w(ni)\) is the computation cost for task \(ni\) and \(c(ni,nj)\) is the communication cost for edge \((ni,nj)\) that connect task \(ni\) and task \(nj\). If task \(ni\) and task \(nj\) are assigned to the same processor, then \(c(ni,nj)\) is zero. The edge \(c(ni,nj)\) represents the precedence constraint between task \(ni\) and task \(nj\). In other words, task \(nj\) can start only after the completion of task \(ni\).

The fork-join structure is one of the basic modeling structures for parallel processing. A fork-join task graph is a hybrid of an in-tree task graph and an out-tree task graph, which is shown in Figure 1. It has a fork node that spawns a number of children nodes. The output edges of the children nodes are connected to a join node. We assume the set of task nodes \(V=\{n1,n2,\ldots,nm\}\), the fork node \(nx\), the join node \(nz\), the set of intermediate nodes \(S=\{n1,n2,\ldots,nm\}\) and assume bus-based and heterogeneous clusters systems. How to compute \(SL\) is shown in below:

\[
SL = \max_{ni \in S} \{cct_{ori}(ni)+w(nz)/p0 \}
\]

If task node \(ni\) is assigned on processor \(p0\), then \(cct_{ori}(ni)=cst_{ori}(ni)=ct_{ori}(ni)\), that is \(c(ni,nz)=0\).

### III. THE PROPOSED ALGORITHM FOR FORK-JOIN TASK GRAPHS

#### A. analysis of the algorithm

Since each task is independent in set \(S\), tasks in set \(S\) can be divided into two disjoint parts, namely \(S= S1 \cup S2\), \(S1 \cap S2 = \emptyset\), the tasks in set \(S1\) are assigned to \(p0\). At first, \(S1\), \(nx\), \(nz\) are assigned to processor \(p0\), so the completion time of set \(S1\) is given, which can be recorded \(Len(S1)\). As \(S2\), \(nx\) are assigned to processors \(p1, p2, \ldots\), the communication time between tasks of set \(S2\) and task node \(nz\) affects the final total completion time. What’s more, different scheduling methods to the tasks of set \(S2\) lead to different completion time for the tasks of set \(S2\). Record the set \(S2\) earliest completion time for the \(Len(S2)=\max_{ni \in S2} \{cct_{ori}(ni)\}\). The total completion time after determining set \(S1\) and set \(S2\) is \(SL = \max \{Len(S1), Len(S2)\} + w(nz)/p0\). It’s easy to calculate \(Len(S1)\), so we first consider how to calculate \(Len(S2)\).

The tasks of set \(S2\) are assigned to processor \(p1, p2, \ldots\). The most direct way to copy task node \(nx\) and assign each task in set \(S2\) to different processors at the same time in order to minimize the maximum communication completion time.

At first, quick sort tasks from set \(S2\) in non-ascending order by the sum of \(w(nik)+c(nik,nz)\). Now we assume the results satisfy

\[
\sum_{ik \in S2} w(nik)+c(nik,nz) \geq \sum_{ij \in S2} w(nij)+c(nij,nz)
\]

Code of \(Len(S2)\) is shown in Figure 2:

\[
\text{int } Len(S2) \\
\{ \\
\quad cct_{ori}(n0)=w(n1)+c(n1,nz); \\
\quad \text{for } k=2 \text{ to } j \\
\quad \quad cct_{ori}(n0)=\max(cct_{ori}(n0),w(nk)+c(nk,nz)/p0); \\
\quad \text{return } cct_{ori}(n0); \\
\} 
\]

Figure 2: Computation of \(Len(S2)\)

\(Len(S1)\) and \(Len(S2)\) in the calculation are only relevant to how to divide set \(S1\) and \(S2\). Set \(S1\) and \(S2\) by the different the total completion time correspond to different \(SL\).

Therefore, the goal of scheduling algorithm is how to determine set \(S1\) and set \(S2\) for corresponding to a minimum of the \(SL\).

#### B. The Proposed Algorithm

The new algorithm includes two steps:

1. **Step 1:** Determine set \(S1\) and set \(S2\) as it is shown in Figure 2.
2. **Step 2:** Schedule reasonable tasks in set \(S2\) in order to save processors.
In step 1, the initial state of the sets is $S_1=\emptyset$ and $S_2=S$. If $S_2$ is not empty, recursively select task node from set $S_2$ which makes $Q=\max\{\text{Len}(S\cup ni), \text{Len}(S_2-ni)\}$ minimal. If $Q=\max\{\text{Len}(S_1),\text{Len}(S_2)\}$, then let $S_1=S_2-ni$, $S_2=\emptyset$; if $Q>\max\{\text{Len}(S_1),\text{Len}(S_2)\}$, then let set $S_1$ and set $S_2$ remain.

### Step 1: Identification of the Algorithm

After determining the set $S_1$, $S_2$ and the total completion time $SL$ under without increasing $SL$ or the start time of the task $nx$ condition, the tasks are assigned to processors used as much as possible in step 2.

After considering step 1 and step 2, the algorithm in bus-based and heterogeneous clusters environment is shown in Figure 4.

### An Example

In this section, we use an example of fork-join task graph in Figure 1 to illustrate the procedure of the proposed algorithm. In this example, we assume that the set of processors $P=\{p_0, p_1, p_2, \ldots, p_4\}$, their speed of execution order is denoted by $1\geq 1/2 \geq 1/3 \geq 1/4$, communication speed of a unit traffic between the main processor $p_0$ and other processor is denoted by $1\geq 1/2 \geq 1/3 \geq 1/4$.

**Step 1:** Identify the set $S_1$, $S_2$ and the total completion time $SL$, the results are in Table I.

**Step 2:** Schedule the tasks of the set $S_2$. Considering orderly the following each task of set $S_2$, the algorithm searches the scheduled processors to select which processor suits for inserting the task.

If the inserting condition is satisfied, the algorithm would modify the tasks in set $S_1$, $S_2$ and assign the tasks to the suited processor, such as task $n_1$. Otherwise, it assigns both entry task $nx$ and $ni$ to a new processor, such as task $n4$.

**Table I. Processor of Determining to the Set $S_1$, $S_2$ and $SL$**

<table>
<thead>
<tr>
<th>S1</th>
<th>S2</th>
<th>Len(S1)</th>
<th>Len(S2)</th>
<th>$\text{cst}_\text{or}(n_3)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\emptyset$</td>
<td>${n_1, n_2, n_3, n_4}$</td>
<td>5</td>
<td>86</td>
<td>86</td>
</tr>
<tr>
<td>${n_3}$</td>
<td>${n_1, n_2, n_4}$</td>
<td>17</td>
<td>51</td>
<td>51</td>
</tr>
<tr>
<td>${n_4}$</td>
<td>${n_1, n_2, n_3}$</td>
<td>27</td>
<td>26</td>
<td>27</td>
</tr>
</tbody>
</table>

| $SL=\text{cst}_\text{or}(n_3)+n_3=27+6=33$ |

**Step 3:** When all tasks in set $S_2$ are assigned to processors, the algorithm schedules the task $nx$ to the main processor $p_0$. In the end, the processor allocation and the schedule time obtained by the algorithm are shown in Figure 6. (shadow denotes the overlap time between the main processor itself computing and communicating with other processors.)

**Table II. Comparison of Schedule Results with Other Algorithms**

<table>
<thead>
<tr>
<th>n</th>
<th>$\text{parameter}$</th>
<th>The Algorithm</th>
<th>TSA_FJ</th>
<th>TDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>SL</td>
<td>23</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>4</td>
<td>SL</td>
<td>33</td>
<td>34</td>
<td>36</td>
</tr>
<tr>
<td>5</td>
<td>SL</td>
<td>36</td>
<td>36</td>
<td>38</td>
</tr>
<tr>
<td>6</td>
<td>SL</td>
<td>41</td>
<td>42</td>
<td>43</td>
</tr>
<tr>
<td>7</td>
<td>SL</td>
<td>51</td>
<td>51</td>
<td>57</td>
</tr>
<tr>
<td>8</td>
<td>SL</td>
<td>52</td>
<td>52</td>
<td>57</td>
</tr>
</tbody>
</table>

From the comparison in Table II, we can see that the algorithm based on fork-join task graphs that can be used in bus-based and heterogeneous systems has less the total completion time and much less processors. The algorithm outperforms the other algorithms in terms of the total completion time and the number of used processors. Thus,
the speedup and efficiency of the algorithm are better than other algorithms.

V. CONCLUSION

This paper presents a better algorithm to schedule tasks of fork-join graph onto processors, which is based on fork-join task graphs that can be used in bus-based and heterogeneous environment. The performance of the algorithm has been compared with other algorithms in terms of the total completion time and the number of used processors. Simulation results showed that the algorithm has better speedup and efficiency than other compared algorithms. Therefore, the algorithm achieves considerable performance for practical applications where can be used in bus-based and heterogeneous systems.

REFERENCES


