Task Scheduling in Multiprocessor System using Fork-Join Method (TSFJ)

Ranjit Rajak¹ & C.P.Katti²
School of Computer and System Sciences
Jawaharlal Nehru University (JNU)
New Delhi-110067, India
ranjit.jnu@gmail.com¹, cpkatti@yahoo.com²

ABSTRACT

The performance of the multiprocessor system depends on how the tasks are scheduled in multiprocessors. If we allocate tasks wisely then we can get good performance of the system. The primary goal of task scheduling in multiprocessor system is to minimize the total execution time, so that we can achieve maximum speed-up and efficiency. The application program is represented by directed acyclic graph (DAG) in scheduling problem. In this paper, we have proposed an algorithm for task scheduling in multiprocessor system based on fork-join method. Tasks are allocated to the processors follows fork-join structure. The performance of the TSFJ algorithm is measured based on scheduling length, speedup and efficiency. Further, we have shown the comparative study between TSFJ algorithm and heuristic algorithms.

KEYWORDS

Task Scheduling, Directed Acyclic Graph, Parallel Processing, Scheduling Length, Speedup

1 INTRODUCTION

The task scheduling in multiprocessor system is called multiprocessor scheduling [1]. Scheduling of multiple tasks on the parallel system or multiprocessor system is known as NP-complete problem. This problem is considered as one of the challenging problem in the parallel computing [2]. Here the application program is represented by Directed Acyclic Graph (DAG). The major objective of task scheduling is to minimize the overall scheduling length. Task scheduling consists of three components [3] are performance of homogenous processors, task mapping on to the processors and order of execution of task on the processors.

This paper presents a task scheduling algorithm using fork-join method. Here, we have find out fork-join structure at each level of a given DAG. Allocation of tasks on the processor is based on fork-join condition. The results show that TSFJ algorithm performs better in comparison to heuristic of scheduling algorithms.

We have assume that proposed scheduling algorithm is static in nature because inter processor communication, task dependency and the definite number of homogeneous processors are known in advance before scheduling.

The rest part of the paper is organized as follows: Section 2 gives a brief description of a task scheduling model and Fork-Join structure. The proposed algorithm described in section 3 and heuristic algorithms in section 4. The comparison matrices, performance analysis are provided in section 5 and 6. Finally, come to conclusion in section 7.

2 TASK SCHEDULING MODEL

The multiprocessor system consists of \( m \) finite numbers of homogeneous processors \( P \) are fully connected to each other.

\[
P = \{P_1, P_2, \ldots, P_m\}
\]

\[\text{Figure 1. Processors are fully connected}\]

The task scheduling is represented by directed acyclic graph (DAG). The DAG model [4] consists of two tuples \( G=(V,E) \) where \( V=\{T_1, T_2, \ldots, T_n\} \) finite set of tasks and \( E=\{e_{ij}\} \) edges that connects two tasks \( T_i \) and \( T_j \).
There are two types of time used in task scheduling: execution and communication time. The execution time is associated with each task $T_i$ and communication time between two tasks and associated with each edge. The execution time is denoted by $E(T_i)$ and communication time is denoted by $C(T_i)$.

The communication time between two tasks is zero if they are scheduling on same processor. One condition always exists here, that is precedence constraints should be maintained. The task $T_j$ executes if and only if all its predecessor are executed. Task scheduling is classified into deterministic and non deterministic scheduling [5]. Deterministic scheduling is known as static scheduling and non-deterministic scheduling is known as dynamic scheduling. In deterministic scheduling, all the information about tasks are well known in advance. That is communication time, execution time and their precedence constraint are known during compile time and in case of non deterministic scheduling, it is known during execution time.

In this paper, we consider only static scheduling problem[2]. There are various types of constraint of static scheduling: execution time, communication time, number of tasks and number of processors[4]. The layout of a DAG model with six nodes is shown in figure[4].

2.1 FORK-JOIN STRUCTURE

Two primitive graphs are used for this proposed algorithm. These structures are Fork and Join of the task graph. Fork-Join structure graphs are very useful to understand the optimality of scheduling algorithms[5]. This is because DAG can be decomposed into a number of fork and join structures. A fork is a structure that consists of only one parent and multiple children. Similarly in join, it is a structure that consists of multiple parents and only one child.

2.1.1 FORK STRUCTURE

The fork structure consists of one parent task $T_p$ and multiples children task $\{T_1, T_2, ..., T_c\}$. The execution time for parent tasks are represented by $E(T_p)$ and $E(T_1), E(T_2), ..., E(T_c)$ for the children tasks. The communication time between parent task and child task are represented by $C($Parent task, Child Task$)$. The fork condition is given by[5]:

$$C(T_p,T_1)+E(T_1) \geq C(T_p,T_2)+E(T_2) \geq ... \geq C(T_p,T_c)+E(T_c)$$

2.1.2 JOIN STRUCTURE

The fork structure consists of one parent task $T_p$ and multiples children task $\{T_1, T_2, ..., T_c\}$. The execution time for parent tasks are represented by $E(T_p)$ and $E(T_1), E(T_2), ..., E(T_c)$ for the children tasks. The communication time between parent task and child task are represented by $C($Parent task, Child Task$)$. The fork condition is given by[5]:

$$C(T_p,T_1)+E(T_1) \geq C(T_p,T_2)+E(T_2) \geq ... \geq C(T_p,T_c)+E(T_c)$$
The join structure consists of multiple parent \{T_1,T_2,T_3...T_P\} and only one child \{T_c\}. Here execution time and communication time is same as fork structure. The join condition is given by[5]
\[E(T_1)+C(T_1,T_c)\geq E(T_2)+C(T_2,T_c)\geq \ldots \geq E(T_P)+C(T_P,T_c)\]

3 PROPOSED ALGORITHM

We propose a new task scheduling algorithm based on the method of fork-join(TSFJ). Here, we consider that DAG model consists of only one entry and one exit task. This scheduling algorithm works on bounded number of processors and all the processors are identical. The TSFJ algorithm works based on fork-join structure at each level. If there is a fork structure at any level then parent task is allocated on any given processor. The child task is allocated to parent task ‘s processor if the child has fork maximum and satisfied precedence constraint and other children task are allocated on different processors. The fork maximum is found based on execution and communication between parent and child task. Similarly, if it is a join structure then a child task is allocated on a processor where parent task is allocated and rest children tasks are allocated on different processors. The allocation of child task depends on join maximum and precedence constraint. Join maximum is found also based on execution and communication time between parent and child task. The detail of proposed TSFJ algorithm is given below.

3.1 TSFJ Algorithm

The procedure for fork join concept in task scheduling of multiprocessor system

**STEP1:** Take a DAG as input that consists of one entry task and one exit task.

**STEP2:** Identify the fork and join structure at each level of a given DAG.

**STEP3:** If there is a fork structure at any level then apply fork condition to find Fmax( fork Maximum) of the children. i.e.

Find distance between a parent(T_p) and children(T_i) using following

\[T_{-\text{dis}}(T_i)=E(T_i)+C(T_i,T_p) \quad \text{where } i=1 \text{ to } c \text{ children tasks.} \]

\[\text{Fmax}=\text{Max}(T_{\text{-dis}}(T_1),T_{\text{-dis}}(T_2),\ldots T_{\text{-dis}}(T_c))\]

**STEP4:** Allocated parent task on a processor(P_i). Check Fmax task if it is maintain the precedence constraint or not.

If Fmax is not maintaining the precedence constraint then choose next Fmax task.

Allocate Fmax Task on the processor where parent processor is allocated and other task is allocated to the processors(P_{i+1},P_{i+2},..P_{i+n-1}).

**STEP5:** If there is a join structure at any level then apply join condition for find Jmax( Join Maximum) of the parents. i.e.

Find distance between parents(T_i) and a child(T_c) using following

\[T_{\text{-dis}}(T_i)=E(T_i)+C(T_i,T_c) \quad \text{where } i=1 \text{ to } p \text{ parent tasks.} \]

\[\text{Jmax}=\text{Max}(T_{\text{-dis}}(T_1),T_{\text{-dis}}(T_2),\ldots T_{\text{-dis}}(T_p))\]

**STEP6:** Allocate child to the processor where Jmax of parent task is allocated and also check their precedence constraint before allocation.

3.2 An Illustrative Example of TSFJ Algorithm

We have taken a DAG model[5] with nine tasks T_1, T_2, T_3, T_4, T_5, T_6, T_7, T_8 and T_9. Now we have to find fork and join structure at each level of given DAG model.
At level 0: Only one fork structure which consists of one parent task (T₁) and five children tasks (T₂, T₇, T₃, T₄, T₅).
Now, we find the distance between a parent task (T₁) and children tasks (T₂, T₇, T₃, T₄, T₅). Then we find Fmax.

\[ T_{\text{dis}}(T₂) = C(T₁, T₂) + E(T₂) = 4 + 3 = 7 \]
\[ T_{\text{dis}}(T₇) = C(T₁, T₇) + E(T₇) = 10 + 4 = 14 \]
Similarly, \( T_{\text{dis}}(T₃) = 4 \), \( T_{\text{dis}}(T₄) = 5 \) and \( T_{\text{dis}}(T₅) = 6 \).

\[ F_{\text{max}} = \max\{7, 14, 4, 5, 6\} = 14 \]
Fmax is 14 for task T₇ but this does not satisfying precedence constraint. So we take next Fmax.
Fmax = 7 which is Task T₂.
The parent task T₁ and child task T₂ will be allocated on the same processor and other children tasks will allocate on different processors. If satisfies previous Fmax is the precedence constraint then it will also allocate on the parent task’s processor. Here, T₇ satisfies precedence constraint and it will also be allocated on same processor where T₁ is allocated. The schedule of level 0 is given below in the Gantt Chart.

At level 1: Only one fork structure in which parent task (T₂) and their children tasks (T₆, T₇).
\[ T_{\text{dis}}(T₆) = 5 \] and \( T_{\text{dis}}(T₇) = 5 \).
Fmax = 5 which can either T₆ or T₇. Here taking T₆ allocates on processor P₂ where parent task (T₂) is allocated. Here T₇ is already allocated. Below task scheduling after level 1 is given

At level 2: There are join structures at this level. The first join structure has two parents task (T₁, T₂) and one child task (T₇). Second join structure also has two parents task (T₃, T₄) and one child task (T₈).

First Join Structure:
We first find the distance between parents task (T₁, T₂) and a child task (T₇). After this we find Jmax.
\[ T_{\text{dis}}(T₁) = C(T₁, T₇) + E(T₇) = 2 + 10 = 12 \]
\[ T_{\text{dis}}(T₂) = C(T₂, T₇) + E(T₇) = 3 + 1 = 4 \]
Jmax = \( \max\{12, 4\} = 12 \) for the parent task T₁.
In case of join structure, a child task and Jmax parent task are allocated on same processor. i.e T₇ will be allocated on processor P₁ which is already allocated.

Second Join Structure:
Similarly, we also find that the distance between parents task (T₃, T₄) and one child task (T₈).
\[ T_{\text{dis}}(T₃) = 4 \]
\[ T_{\text{dis}}(T₄) = 5 \]
Jmax = 5 for parent task T₄.
Now, child task T₈ and parent task T₄ will be on the same processor P₃.
Task scheduling after level 2 is given below.

At level 3: Also, there is one join structure. In this structure, there are three parents task (T₆, T₇, T₈) and one child task (T₉).
Again, find the distance between the parents task (T₆, T₇, T₈) and one child task (T₉) and Jmax.
\[ T_{\text{dis}}(T₆) = 9 \]
\[ T_{\text{dis}}(T₇) = 10 \]
\[ T_{\text{dis}}(T₈) = 9 \]
Jmax = 9 for parent task T₆.
The child task T₉ and parent task T₆ are on the same processor. The task scheduling after level 3 is given below.
Figure 9. Task Schedule generated after level 2 using Join structure.

After level 3, there is no more task in DAG. So, we can see that scheduling length is 17 units.

4 HEURISTIC ALGORITHMS

Here, we compare our proposed TSFJ algorithm with heuristic algorithms. We take four heuristic algorithms for comparison: Highest Level First with Estimate Time (HLFET)[9], Modified Critical Path (MCP)[8], Earliest Time First (ETF)[10] and Dynamic Level Scheduling (DLS)[11] algorithms.

All the four heuristic algorithms are used priority attributes: top level (t-level), bottom level (b-level)[6], static level (sl), dynamic level (dl) and As Late As Possible (ALAP)[7,8].

The scheduling length and priorities of four heuristic algorithms are given in table. The scheduling length is calculated based on DAG with nine nodes[5].

<table>
<thead>
<tr>
<th>Heuristic Algorithms</th>
<th>Scheduling Length</th>
<th>Priorities Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. HLEFET [5]</td>
<td>19</td>
<td>static level (sl)</td>
</tr>
<tr>
<td>2. MCP [5]</td>
<td>20</td>
<td>As Late As Possible (ALAP)</td>
</tr>
<tr>
<td>3. ETF [5]</td>
<td>19</td>
<td>static level (sl)</td>
</tr>
<tr>
<td>4. DLS [5]</td>
<td>19</td>
<td>dynamic level (dl)</td>
</tr>
</tbody>
</table>

5 COMPARISON MATRICES

The comparative study between TSFJ algorithm and Heuristic algorithms has been done on basis of five matrices: scheduling length, speedup, efficiency, normalized scheduling length and load balancing.

6 PERFORMANCE ANALYSES

In this section, we evaluate the performance of proposed TSFJ algorithm and Heuristic algorithms. All the performance analysis has been done on a given a DAG model with nine tasks.

The TSFJ algorithm has scheduling length 17 as scheduling generated after level 3. Now we find speedup, efficiency, normalized scheduling length and load balancing of DAG model with nine nodes.

Speedup = 30/17 = 1.764
Efficiency = (1.76/4) x 100 = 44
Normalized Scheduling Length = 17/(17+6+11+8)/4 = 1.619

Table 3. Performance Analysis of Heuristic Algorithms [4] and TSFJ Algorithm
The proposed TSFJ algorithm gives minimum scheduling length as compared to HLFET, MCP, ETF, and DLS algorithms. We also compared their performance on the basis of speedup, efficiency, normalized scheduling length, and load balancing. The proposed algorithms have better speedup and efficiency than heuristic algorithms. The normalized scheduling length of TSFJ is almost same as ETF and DLS algorithms. The load balancing is achieved higher in TSFJ algorithm and lower in ETF and DLS algorithms. The algorithms have been tested on DAG with nine tasks. It can also be tested on more than nine nodes.

8. REFERENCES

7. Kwok,Y.K.,Ahmad,,:Dynamic Critical Path Scheduling : An Effective Techniques for Allocating Tasks Graph onto