Critical Node Dynamic Scheduling Algorithm for a Multiprocessor System

Hao Fan¹², Haojiang Deng¹, Xiaozhou Ye¹*¹
¹National Network New Media Engineering Research Center, Institute of Acoustics, Chinese Academy of Sciences, Beijing 100190, China
²University of Chinese Academy of Sciences, Beijing 100190, China
*Corresponding author(E-mail: yezx@dsp.ac.cn)

Abstract
An efficient task scheduling which task can be scheduled to proper processor aiming to minimize the schedule length, is a classic problem in multiprocessor system. In this paper, we analyze two typical list scheduling algorithms, the MCP algorithm and the BDCP algorithm, and show that they have one or more significant drawbacks which can lead to poor performance. The proposed list scheduling algorithm named the critical node dynamic scheduling (CNDS) algorithm is different in some ways. First, it guarantees the nodes on critical path have the high priority and can be scheduled to the same processor. Second, it constructs a critical path list and an exit node list to be scheduled. Third, it dynamically selects the parent nodes of critical node and exit node when they are ready to be scheduled but their parent nodes are not scheduled. Fourth, it selects a suitable processor for a parent node of critical node or exit node by considering the potential start time of critical node or exit node. Experiments and comparisons are carried out for all three algorithms under various scheduling conditions, and the result shows that the CNDS algorithm outperforms the previous algorithms. Besides, the CNDS algorithm is economical in terms of processors used and improves the utilization of the heaviest load processor. Moreover, the CNDS algorithm has admissible time complexity and is suitable for a wide range of task scheduling graph.

Key words: Scheduling algorithm, list scheduling, critical node, schedule length, multiprocessor system

1. INTRODUCTION
The objective of task scheduling on multiprocessor system is to schedule tasks to proper processor in order to minimize the schedule length(Jin, 2008). It is well known, however, that multiprocessor scheduling for most precedence-constrained task graphs is an NP-complete problem in its general form (Geng, 2013). To tackle the problem, simplifying assumptions have been made regarding the task graph structure representing the program and the model for the parallel processor systems(Srikanth,2015). Three categories of heuristic scheduling algorithm, i.e. list scheduling(Arabnejad,2014;Fard,2014), task duplication scheduling (Gupta,2015;Singh,2016) and clustering scheduling (Gil, 2014;Dehsangi,2015), have been proposed in the literature. In order to be in practical use, a scheduling algorithm should have low complexity and should be economical in terms of the number of processors used (Mouhamed,1990). Because of these, list scheduling algorithm outperforms others by a considerable margin.

In this paper, we firstly analyze two typical list scheduling, the modified critical-path (MCP) algorithm (Wu,1990) and the balanced dynamic critical path (BDCP) algorithm(Shi,2001), and find some drawbacks on them. To tackle the drawbacks, we propose a new list scheduling algorithm based on critical path, which is called the critical node dynamic scheduling (CNDS) algorithm. The algorithm has admissible time complexity and economical used processors. It is suitable for different types of graph.

The remainder of this paper is organized as follows. In the next section, we describe the background of the scheduling problem and analyze the MCP algorithm and the BDCP algorithm. In section 3, we describe our CNDS algorithm. In section 4, we use an example to illustrate these algorithms. In section 5, we provide and compare the experimental results. In section 6, we make the concluding remarks.

2. BACKGROUND
A parallel program can be represented by a directed acyclic graph (DAG), \( G = (V,E) \), where \( V \) is the set of nodes and \( E \) is the set of edges. A node in the DAG represents a task in the parallel program. The value of node, the computation cost, represents the task execution time. An edge in the DAG represents the communication messages and precedence constraints among the nodes. The value of edge, the communication cost, represents the time required for communicating the data from one task to another. In an edge, the source node is called the parent node and the destination node is called the child node. In the DAG, a node which has no parent node is called an entry node while a node which has no child node is called an exit node. A node cannot start to execute
before it gathers all of the messages from its parent nodes. The communication-to-computation ratio (CCR) of a parallel program is defined as its average communication cost divided by its average computation cost. We assume that the communication time is zero if the parent node and the child node are assigned to the same processor. A critical path (CP) of the DAG is a set of nodes and edges, forming a path from an entry node to an exit node, of which the sum of computation costs and communication costs is the maximum. The CP of the DAG potentially determines the schedule length because the cumulative computation costs of the nodes on the CP is the lower bound on the schedule length.

It is well known, the list scheduling algorithm include three steps to assign tasks of the graph. Firstly, it must determine the priority of the tasks in the graph according to the task dependencies. Then it can build the tasks scheduling list based on the priority of the tasks. Secondly, it selects a high priority task in the tasks scheduling list. Thirdly, it assigns the chosen task to a proper processor on where it can get a better performance than other processors. Finally, it repeats the step 2 and step 3 to schedule tasks until there is no task in the tasks scheduling list. In the following, we present the MCP algorithm and the BDCP algorithm and analyze them.

The MCP algorithm is designed based on an attribute called the latest possible start time (LPST) of a node, also named the as-late-as-possible (ALAP) time. Firstly, it computes the ALAP time for all nodes. Then, it creates a list for each node which consists of its ALAP time and all its descendants in decreasing order. It sorts these lists in decreasing order and creates a node list. Finally, it selects the first node in the list to a proper processor which allows its earliest execution. It repeats selecting nodes until the list is empty. The complexity of the MCP algorithm is shown to be $O(n^2 \log n)$. Although the MCP algorithm adopts the strategy of the critical path of the graph, the node selection is static which means that the priority of node may not always order the nodes for scheduling according to their relative importance. What’s more, the nodes to be scheduled to proper processor only consider the start time of them not the potential start time of their children.

The BDCP algorithm is designed based on an attribute called the earliest possible start time (EPST) of a node. Firstly, it computes the EPST time and the static schedule length for all nodes. Secondly, it creates a list in decreasing order according to the EPST time. Then, it selects the first task in the list to a proper processor which gains the smallest current schedule length. After that, it updates the EPST time of its child node. It repeats the above steps until there is no task in the list. The complexity of the BDCP algorithm is shown to be $O(n^3)$. Although the BDCP algorithm adopts the strategy of the dynamic critical path of the graph, the nodes to be scheduled to proper processor only consider the start time of them not the potential start time of their children.

What’s more, it could not guarantee the nodes on the CP can be scheduled in the same processor so that it can reduce the communication cost among tasks.

3. THE PROPOSED ALGORITHM

In this section, we present the proposed CNDS algorithm which aims at achieving high performance, low complexity, economical numbers of used processors and high utilization of heaviest load processors. Before presenting our algorithm, it is necessary to define some attributes and formulas.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n_i$</td>
<td>The node number of a task in the DAG</td>
</tr>
<tr>
<td>$w_i$</td>
<td>The computation cost of node $n_i$</td>
</tr>
<tr>
<td>$c_{ij}$</td>
<td>The communication cost of the directed edge from node $n_i$ to $n_j$</td>
</tr>
<tr>
<td>exit($G$)</td>
<td>The set of exit nodes in the DAG</td>
</tr>
<tr>
<td>pred($i$)</td>
<td>The parent nodes of node $n_i$</td>
</tr>
<tr>
<td>succ($i$)</td>
<td>The child nodes of node $n_i$</td>
</tr>
<tr>
<td>EPST($n_i$)</td>
<td>The earliest possible start time of the node $n_i$</td>
</tr>
<tr>
<td>LPST($n_i$)</td>
<td>The latest possible start time of the node $n_i$</td>
</tr>
<tr>
<td>CSL</td>
<td>The current schedule length</td>
</tr>
<tr>
<td>EST($n_i$, $J$)</td>
<td>The earliest start time of the node in $n_i$ processor $J$</td>
</tr>
<tr>
<td>AEST($n_i$, $J$)</td>
<td>The absolute start time of the node in $n_i$ processor $J$</td>
</tr>
<tr>
<td>PE($n_i$)</td>
<td>The processor which contains node $n_i$</td>
</tr>
<tr>
<td>$IT(J)$</td>
<td>The idle time of processor $J$</td>
</tr>
</tbody>
</table>
**DEFINITION 1.** The earliest possible start time of the node \( n_i \), denoted by \( EPST(n_i) \) is defined as follows:

\[
EPST(n_i) = \begin{cases} 
0, & \text{if it is an entry node, } pred(i) = \emptyset \\
\max_{n_{i\in pred(i)}} \left\{ EPST(n_j) + w_j + c_{i,j} \right\}, & \text{otherwise} 
\end{cases} \tag{1}
\]

According to Definition 1, the \( EPST \) value can be computed by traversing the DAG in breadth-first search from the entry nodes. When \( EPST(n_i) \) is to be computed, all the \( EPST \) value of parent nodes of node \( n_i \) should have been computed.

**DEFINITION 2.** The current schedule length of task graph, denoted by CSL is defined as:

\[
CSL = \max_{n_{i\in\text{exit}(G)}} \left\{ EPST(n_i) + w_i \right\} \tag{2}
\]

According to Definition 2, the CSL value is computed by the maximum value among all the earliest possible finish time of exit nodes. It can be used to be computed the latest possible start time which is described in the following definition.

**DEFINITION 3.** The latest possible start time of the node \( n_i \), denoted by \( LPST(n_i) \) is defined as follows:

\[
LPST(n_i) = \begin{cases} 
CSL-w_i, & \text{if it is an exit node, } succ(i) = \emptyset \\
\min_{n_{j\in succ(i)}} \left\{ LPST(n_j) - w_j - c_{j,i} \right\}, & \text{otherwise} 
\end{cases} \tag{3}
\]

According to Definition 3, similar to the computation of \( EPST \), the \( LPST \) value can be computed by traversing the DAG in breadth-first search from the exit nodes. Note that the \( LPST \) value should be computed after the CSL value has been computed. When \( LPST(n_i) \) is to be computed, all the \( LPST \) value of child nodes of node \( n_i \) should have been computed.

**DEFINITION 4.** The earliest start time of the node \( n_i \) on processor \( J \), denoted by \( EST(n_i, J) \) is defined as follows:

\[
EST(n_i, J) = \begin{cases} 
0, & \text{if it is an entry node, } pred(i) = \emptyset \\
\max_{n_{i\in pred(i)}} \left\{ EST(n_j, J) + w_j + r(PE(n_j), J)c_{i,j} \right\}, & \text{otherwise} 
\end{cases} \tag{4}
\]

\[
r(PE(n_j), J) = \begin{cases} 
0, & \text{if } PE(n_j) = J \\
1, & \text{otherwise} 
\end{cases} \tag{5}
\]

According to Definition 4, the \( EST \) value can be computed when all the \( EST \) value of parent nodes of \( n_i \) have been computed. It can be used to be computed the absolute start time which is described in the following definition. Note that the communication time among two nodes can be zero if they are in the same processor.

**DEFINITION 5.** The absolute earliest start time of the node \( n_i \) on processor \( J \), denoted by \( AEST(n_i, J) \) is defined as follows:

\[
AEST(n_i, J) = \begin{cases} 
EST(n_i, J), & \text{if } \left[ EST(n_i, J) + w_i \right] \in IT(J) \\
\Delta, & \text{otherwise} 
\end{cases} \tag{6}
\]

According to Definition 5, the \( AEST \) value is the absolute earliest start time considering the slot time on processor. Note that the \( \Delta \) value is the smallest value which satisfies the condition that the task \( n_i \) can be computed on processor \( J \) without time conflict.

### 3.1 List Creation

As we describe in section 2, the CP of a task graph determines the schedule length. Thus, the proposed algorithm guarantees the CP nodes have a high priority in being scheduled. According to the critical path algorithm in graph theory, it is known that if \( EPST(n_i) \) is equal to \( LPST(n_i) \), node \( n_i \) must be a critical node, namely, node \( n_i \) is on the CP of the DAG.

**RULE 1.** If \( EPST(n_i) = LPST(n_i) \), then \( n_i \) is a CP node.

According to Rule 1, we calculate the \( EPST \) and the \( LPST \) of all nodes and determine the CP nodes. Then we select all the CP nodes to create a CP scheduling list by increasing order of the nodes’ \( EPST \) which means that the earliest ready task can be scheduled in high priority. By constructing the CP scheduling list, the CP nodes and their parent nodes can be scheduled in order, but the exit nodes not on the CP cannot be scheduled. So we need to create an exit scheduling list by increasing order of the nodes’ \( EPST \), similar to the construction of the CP scheduling list.
3.2 Node Selection

While we are able to construct the scheduling list, we still need a method to select node to be scheduled. The CNDS algorithm selects the top node of the scheduling list to be scheduled.

RULE 2. A node $n_i$ can be scheduled, if its parent nodes have been scheduled, otherwise, its unscheduled parent nodes should be scheduled firstly.

According to Rule 2, if the parent nodes of the chosen node have been scheduled, we select it to be scheduled. Otherwise, we traverse the parent unscheduled nodes of it in breadth-first search and construct another scheduling list which consists of all these nodes by increasing order of the nodes' $EPST$, similar to the construction of the CP scheduling list. Then we select the top node of the new list to be scheduled following Rule 2 until this list is empty. The above new list created is a dynamic operation and it is generated dynamically by the selection of CP node or exit node. With all the operations of node selections in the CP scheduling list and the exit node scheduling list above, the nodes of task graph can be all selected in proper priority.

3.3 Processor Selection

While we are able to select a CNDS node, we still need a method to select a proper processor for scheduling.

RULE 3. The CP nodes should be scheduled in the same processor.

As the CP nodes have a great influence on the CSL, we guarantee the CP nodes can be scheduled in the same processor so that we can shorten the communication cost between the nodes on the CP and we can improve the utilization of the heaviest load processor. High utilization means that the heaviest load processor has small number of slices and other processors have much idle time so that they can run tasks from another task graph and it is benefit for large-scale parallelism.

RULE 4. The exit nodes not on the CP are scheduled to the processor which can achieve the smallest value of $AEST$.

The exit nodes are not significant on the task scheduling and have little influence on the other nodes, so we just guarantee they can be scheduled as early as possible.

RULE 5. The nodes in the dynamic list is scheduled to the processor which the bottom node of the list can achieve the smallest value of $AEST$.

For those nodes in the dynamic list, we must guarantee they can be scheduled to proper processor so that the bottom node of the list can achieve the smallest value of $AEST$ as the schedule strategy of bottom node has significant influence on the task graph scheduling length and the bottom node’s finish time is the schedule length of dynamic list.

3.4 The CNDS Algorithm

The CNDS algorithm is formalized in this section. It uses two steps: schedule the CP scheduling list and schedule the exit node scheduling list. Both two steps are in the similar operation and the scheduling algorithm is shown in Algorithm 1. The difference between two steps is that the scheduling strategy of the CP scheduling list schedules the CP nodes in the same processor while the scheduling strategy of the exit node scheduling list schedules the exit nodes according to the $AEST$ value. Based on Rule 2, the proposed algorithm handles the ready node and unready node in two different ways. For the ready node in the scheduling list, the proposed algorithm selects a proper processor to schedule the task according to Rule 3 and Rule 4. For the unready node in the scheduling list, the proposed algorithm constructs a new list consists of its unscheduled parent nodes and schedules them to a proper processor following the Rule 5.

The CNDS algorithm is shown in Algorithm 2. It computes the $EPST$ value and the $LPST$ value for all nodes, and constructs a CP scheduling list according to Rule 1 and an exit node scheduling list. The proposed algorithm schedules the nodes in the two lists respectively. The complexity of the algorithm is $O(n^2)$ where $n$ is the number of tasks.

ALGORITHM 1. Schedule the List

1) While there are unscheduled nodes in the scheduling list do
2) Select the first node from the list to schedule
3) If it is ready then
4) For the CP node, place it in the same processor as other CP nodes. For the exit node, compute the $AEST$ value, and select the processor that minimize the $AEST$ value
5) else
6) Construct a list consists of the parent nodes of the chosen node which are not be scheduled, and then sort the list by the increasing order of the $EPST$
7) While there are unscheduled nodes on the list do
8) Select the first node from the list to schedule
ALGORITHM 2. the CNDS Algorithm

1) Compute EPST and LPST for all nodes
2) Construct a CP scheduling list and an exit node scheduling list
3) Schedule the CP scheduling list
4) Schedule the exit node scheduling list

4. AN APPLICATION EXAMPLE

In this section, an example task is used to illustrate the effectiveness of the proposed algorithm. The task graph used is a parallel Gaussian elimination task graph and is shown in Fig. 1. Note that the edges which are the CPs in this task graph are shown with thick arrows.

The schedule of the Gaussian elimination task graph generated by the MCP algorithm is shown in Fig. 2. The MCP algorithm schedules the task graph in the order: n1, n3, n7, n4, n9, n5, n12, n10, n6, n14, n11, n16, n15, n2, n8, n13, n17, n18. The MCP algorithm schedule length is 520, the number of processors used is 4 and the utilization of PE 0 is 77 percent. It is shown form this example that the CP nodes are not scheduled in high priority. When a CP node and a node not on the CP are ready for being scheduled, a CP node may not be scheduled first. It is also shown that the MCP algorithm does not take care of the scheduling effect of the child nodes of a node.
The schedule of the Gaussian elimination task graph generated by the BDCP algorithm is shown in Fig. 3. The BDCP algorithm schedules the task graph in the order: n1, n3, n7, n4, n6, n2, n8, n9, n10, n11, n12, n14, n13, n15, n16, n17, n18. The BDCP algorithm schedule length is 490, the number of processors used is 4 and the utilization of PE 0 is 86 percent. Although the BDCP algorithm adopts the dynamic CP, it is shown form this example that the CP nodes are not scheduled in high priority. When a CP node and a node not on the CP are ready for being scheduled, a CP node may not be scheduled first. It is also shown that the MCP algorithm does not take care of the scheduling effect of the child nodes of a node.

The schedule of the Gaussian elimination task graph generated by the CNDS algorithm is shown in Fig. 4. The CNDS algorithm schedules the task graph in the order: n1, n3, n7, n4, n9, n12, n5, n10, n14, n16, n17, n6, n11, n15, n18, n2, n8, n13. The CNDS algorithm schedule length is 440, the number of processors used is 3 and
the utilization of PE 0 is 100 percent. In accordance with the calculation of the EPST value and LPST value for all nodes, a CP scheduling list and an exit node scheduling list are constructed. The order of the CP scheduling list is: n1, n3, n7, n9, n12, n14, n16, n17, n18. All nodes in the CP scheduling list must be scheduled to PE 0. The order of the exit node scheduling list is: n2, n8, n13. At the first step, the highest CP node n1 is selected for scheduling. After scheduling the first three CP nodes { n1, n3, n7 }, n9 is selected for scheduling, but its parent node n4 is not ready. So a new dynamic scheduling list { n4, n9 } is constructed, and the scheduling strategy of n4 must consider the AEST value of n9. By computation, n4 is scheduled to PE 0 so that n9 can get the smallest AEST value. The same situation takes place in scheduling the CP nodes { n14, n18 }, and new dynamic scheduling lists { n5, n10, n14 } and { n6, n11, n15, n18 } are constructed. At last, the exit nodes { n2, n8, n13 } of the exit node scheduling list are selected for scheduling. Through computation of their AEST value on each processor, n2 and n8 is scheduled to PE 2 and n13 is scheduled to PE 1 so that their AEST value can be minimized. The CNDS algorithm guarantees that the CP nodes of the task graph have the high priority and can be scheduled first. The CNDS algorithm also assures that the CP nodes are scheduled in the same processor. The CNDS algorithm dynamically selects the parent nodes of CP node or exit node and considers of their scheduling influence on their child CP node or exit node.

Figure 4 The schedule of the Gaussian elimination task graph generated by the CNDS algorithm

In this example, the CNDS algorithm achieves the smallest scheduling length, the least number of processors used and the highest utilization of the heaviest load processor.

5. PERFORMANCE AND COMPARISON

In this section, we present a performance comparison of all three algorithms. For this purpose, we generate a large set of random task graph to test the algorithms. The performance comparison is carried out in three ways. First, we compare the normalized schedule length (NSL) (Hall,2011) scheduled by the algorithms in different CCR and the result is shown in Fig. 5. The NSL of the DAG is defined as:

\[ NSL = \sum_{a \in CP} w_a + \sum_{a \in CP} \frac{c_{ij}}{\text{schedule length}} \]  

(7)

Second, we compare the number of processors used scheduled by the algorithms in the same case and the result is presented in Fig. 6. Third, we compare the utilization of the heaviest load processor also in the same case and the result is displayed in Fig. 7.
As can be observed from Fig. 5, the three algorithms have a good performance when the CCR of the task graph is more than 1. The larger the value of CCR is, the better performance the three algorithms can achieve. It is because there are much communication cost can be optimized when the related nodes are scheduled to the same processor. It is also known that the CNDS algorithm outperforms other algorithms especially when the CCR of the task graph is more than 0.5 and the optimization performance is almost 10 percent.

As can be shown from Fig. 6, the larger the value of CCR is, the less the number of processors uses. It is because the three algorithms schedule many related nodes to the same processor so that the communication cost can be minimized and the schedule length can be optimized. The CNDS algorithm has an outstanding economical number of processors used when the CCR of task graph is less than 1 while the BDCP algorithm has an excellent performance when the CCR of task graph is more than 1. The CNDS algorithm can save almost one processor used when the CCR is 0.1 and 0.5. In a word, the CNDS algorithm has an economical number of processors.
As can be presented from Fig. 7, the CNDS algorithm has an excellent performance on the utilization of the heaviest load processor and the utilization of the heaviest load processor is more than 90 percent in each case. So the CNDS algorithm can take full advantage of one core to calculate, have a small number of slice time to waste and benefit for large-scale parallelism.

6. CONCLUSIONS

In this paper, we have proposed a new scheduling algorithm based on list scheduling which has admissible time complexity. We construct two scheduling lists, the CP scheduling list and the exit node scheduling list, and schedule the nodes in the two lists separately. We guarantee the CP nodes can be scheduled to the same processor. We dynamically construct the list consists of the selected node and its unscheduled parent nodes and schedule the unscheduled parent nodes considering the influence on the selected node. By experiments with three algorithms, our proposed algorithm shows that it has a better scheduling performance in the schedule length and the utilization of the heaviest load processor than other algorithms, and is economical in terms of processors used. The performance of our proposed algorithm makes it a viable choice for compile-time scheduling of various task graphs onto multiprocessors.

REFERENCES


