A Shared-Memory Multiprocessor Scheduling Algorithm

Irene Zuccarí, Mauricio Solar, Fernanda Kri, Victor Parada
1 Departamento de Ingeniería Informática, Universidad de Santiago de Chile. http://www.diinf.usach.cl

Abstract. This paper presents an extension of the Latency Time (LT) scheduling algorithm for assigning tasks with arbitrary execution times on a multiprocessor with shared memory. The Extended Latency Time (ELT) algorithm adds to the priority function the synchronization associated with access to the shared memory. The assignment is carried out associating with each task a time window of the same size as its duration, which decreases for every time unit that goes by. The proposed algorithm is compared with the Insertion Scheduling Heuristic (ISH). Analysis of the results established that ELT has better performance with fine granularity tasks (computing time comparable to synchronization time), and also, when the number of processors available to carry out the assignment increases.

1 Introduction

Real-world applications of scheduling is a complex problem in which a large quantity of research have been done in the fields of computer science, artificial intelligence and operational research. Some recently examples are a real application in a printing company shown in [1] and a solution to the Single-Track Railway Scheduling Problem presented in [2]. On the other hand, a comparison of parallel models of genetic algorithm and tabu search to schedule concurrent processes into a parallel machine with homogeneous processors is shown in [3].

In parallel computing there is the problem of finding the best use of the available computer resources of the objective machine with which one is working. After a literature review we could not find any work related that presents a solution that takes into consideration the synchronization time, which is a characteristic of systems with shared memory, in the execution of applications. These applications are usually divided into tasks of different duration that are executed in the available processors. The Latency Time (LT) algorithm [4] approaches this reality, but it has the restriction that the tasks must be of unit duration. This paper presents the
extension of the LT algorithm removing this restriction, allowing tasks with arbitrary
duration to be assigned efficiently.

The paper is organized as follows: section 2 introduces the theoretical
background of the scheduling problem. Section 3 details the LT algorithm and the
extension proposed. Section 4 shows the analysis of the results. Finally, section 5
presents the conclusions.

2 Theoretical Framework

It is always possible to represent a parallel computer application as a task graph. A
task graph is a DAG (Directed Acyclic Graph) defined by a tuple \((V, E, C, T)\), where
\(V = \{n, i \in [1, n]\} (n = |V|)\) is the set of nodes that represent the tasks that must be
iterated; \(E = \{e, i, j \in [1, n]\} (e = |E|)\) is the set of directed edges that indicated
the precedence between tasks \(i\) and \(j\); \(C = \{c, i, j \in [1, n]\}\) is the cost associated with
each edge of the DAG, and \(T = \{t, i \in [1, n]\}\) is the computing time for each task. The
granularity of a graph is the ratio of the average duration of the tasks and the average
of the communications within the DAG (Equation 1).

\[
Gr = \frac{\sum_{i \in V} t_i}{n} / \frac{\sum_{i, j \in V} c_{ij}}{e}
\]  

(1)

Communication is considered when it takes at least 20% of the time occupied in
computing. If it is less than that value, it is considered negligible. Therefore, if \(0 <
Gr < 5\), the granularity is said to be fine, and if \(Gr \geq 5\), the granularity is said to be
coarse [4].

2.1 Scheduling Algorithms

Normally, the inputs to the scheduling algorithms are a task graphs (representing an
applications), and the number of processors \(p\) of the system. The output is a Gantt
chart of the assignment found. The algorithms can consider or not consider the cost
of communication between the tasks, and in that way it is determined whether they
work on machines with shared or distributed memory. Those algorithms that
consider arbitrary communication between the tasks are oriented to the exchange of
messages. Those that do not consider communication have in general been proposed
for machines with shared memory. This is because it is considered that the time
necessary for synchronization between the tasks, \(\Delta t\), is negligible compared to the
duration, \(t_i\), of the tasks (therefore, \(c_{ij}\) is ignored). It can even be applied to machines
with distributed memory, but with the concept of coarse granularity. There are
different approaches to solve the scheduling problem: Meta-Heuristic algorithms [5],
Clustering algorithms [6], Lists algorithms [7].

Meta-heuristic algorithms [5] are defined in a general way and must be modeled
according to the nature of the problem to be solved. Their main advantage is that,
even though they do not deliver an optimum solution, they provide solutions close to
the optimum.

List algorithms have a special case of assignment heuristic which consists in
assigning a priority to each task, and creating a priority list. Then, every enabled task
from the list (its predecessors have already been assigned) it is assigned to an
available processor. Among the algorithms that consider communication the
following can be mentioned: ISH (Insertion Scheduling Heuristic) [8], MCP
(Modified Critical Path) [9] and LT [4]. The ISH algorithm operates as a classical list
algorithm, but when choosing the processor for the next assignment it looks if the
processor has empty time spaces. If so it will consider the rest of the enabled tasks on
the list and use, if possible, those spaces.

A clustering algorithm maps the nodes of the DAG into disjoint groups of tasks
called clusters, which correspond to the sets that group tasks. All the tasks of a given
cluster are executed on the same processor. The main difference between these
algorithms is the heuristic that they have to generate the clusters. Examples are DSC
(Dominant Sequence Clustering) [10] and RC (Reverse Clustering) [6].

3 Extended Latency Time Scheduling Algorithm (ELT)

3.1 Latency Time Scheduling Algorithm (LT)

LT is a list algorithm that operates on DAGs with tasks of unit duration. It assumes a
multiprocessor system with identical processing elements where each processor
carries out only one task at a time until it is completed. This algorithm considers
communication in parallel machines with shared memory, using as input parameter
the synchronization time, \( \Delta t \), when accessing the shared memory, which belongs to
the system. That time is considered comparable to the duration of the tasks, \( t_i \), by
means of the proportionality constant \( k \) (Equation 2).

\[
\Delta t = k t_i
\]  

(2)

LT is designed to operate with DAGs of fine granularity, where \( t_i \leq \Delta t \) or \( t_i = \Delta t \). It
is a two-step algorithm: first, it calculates the priority of each task, which aims at
maximizing the number of enabled tasks and decreasing the latency time used for
synchronization, and second, it generates the Gantt chart corresponding.

3.2 Priority Function of Extended Latency Time Extended (ELT)

The priority function defined for ELT is based on the one used by LT, but more
information is added. To choose the most adequate priority function, nine
combinations of possible ways of prioritizing the information of each node were
defined. Tests were carried out on the sets of DAGs defined in this paper, and the
best priority function found, for each node, was based on the following calculations:
\( l(n_i) \): Size of the longest path starting from the input nodes of the DAG, up to node \( n_i \). To this size must be added the duration of the tasks and the system's time \( \Delta t \).
\( L(n_i) \): Size of the longest path from node \( n_i \) to the output nodes. To this size must be added the duration of the tasks and the system's time \( \Delta t \).
\( L_i \): Size of the longest path that goes through node \( n_i \) (Equation 3).
\[
L_i = l(n_i) + L(n_i) - 1 - t_i
\] (3)

\( CP \): Set of nodes that make up the critical path.
\( LCP \): Size of the DAG's longest path, called Critical Path.
\( Out_i \): Number of immediate successors of node \( n_i \).
\( Hang_i \): Number of tasks achievable by node \( n_i \).
\( T_i \): Duration of each node \( n_i \).
\( SP_i \): priority of the node \( n_i \) (Static Priority).

Calculation of the priority is done as follows:

1. Determine \( L_q, \forall \ i \in [1, n] \) (according to equation 3). Group the tasks that have the same \( L_q \) and arrange them in decreasing order: every new ordered group is labeled as \( L_q \) with \( q: 0, 1, 2, \ldots \)
2. Determine \( Out_i, \forall \ i \in [1, n] \). Group the tasks that have the same \( Out_i \) and arrange them in decreasing order: every new ordered group is labeled as \( OUT \), with \( r: 0, 1, 2, \ldots \)
3. Determine \( Hang_i, \forall \ i \in [1, n] \). Group the tasks that have the same \( Hang_i \) and arrange them in decreasing order: every new ordered group is labeled as \( HANG \), with \( s: 0, 1, 2, \ldots \)
4. Determine \( T_q, \forall \ i \in [1, n] \). Group the tasks that have the same \( T_q \) and arrange them in decreasing order: every new ordered group is labeled as \( T_q \) with \( u: 0, 1, 2, \ldots \)
5. Determine \( priority, \forall \ i \in [1, n] \). To do this, add the subscripts of the subsets \( (L_q, Out, HANG, T_q) \) to which each task belongs. Group the tasks that have the same \( priority \), and arrange them in decreasing order: every new ordered group is labeled as \( PG \), with \( v: 0, 1, 2, \ldots \)
6. Determine \( SP_i = \{ v / n_i \in PG \} \), \( \forall \ i \in [1, n] \).

### 3.3 Assignment’s Extended Latency Time (ELT)

ELT [11] is capable of operating with tasks of arbitrary duration, using the same assignment policy as LT. It should be noted that the relation shown in Equation 2 does not represent this new instance. However, \( \Delta t \) remains constant and is an input parameter to the algorithm. Every task has a time window of a size corresponding to its duration. For every time unit that goes by, the window of the task at the moment processed, are reduced by one unit until they reach zero, at which time the tasks that are enabled can be updated, and the corresponding assignments can be made on the processors that are no longer busy.

The ELT algorithm, whose computing complexity is \( O(n^2) \), is shown below.
ELT Algorithm

\( t = 0 \)
Calculate priority for each node \( n_i \).
For each node, define a Time Window equal to the size of the task.
Put into Unsched all DAG nodes.
Put into Enable all nodes with entry level \( t \text{level} = 1^* \).
Put into group \( n_i \) all Enable nodes.

\( \text{EnabledTasks} := \text{TRUE} \)

**WHILE** Unsched is not empty **DO**

**IF** EnabledTasks **THEN** Schedule(t)
  
  Update Enable with recently assigned nodes.

  Displace the Time Window for each task in Sched(t)

  **IF** any task assigned completed its Time Window **THEN**
  
  EnabledTasks := \text{TRUE}

  ELSE EnabledTasks := FALSE

  UpdateOldTasks(), UpdateGroupTasks()

  **IF** EnabledTasks **THEN**

  Update Unsched removing recently assigned nodes.

  Update-Enable(t)

  \( t := t + 1 \)

**ENDWHILE**

Return Sched

4 Tests and Results

A comparison was made with seven scheduling algorithms obtained from the literature. To save space, the only comparison shown is that with the ISH algorithm, which is the algorithm that gave the best results with respect to ELT. The performance of the ELT algorithm was compared with the ISH algorithm by means of the average of the percentage difference between the results obtained by both algorithms for sets of test DAGs. If the value is positive, it means that the ELT algorithm delivers a best solution in its total parallel time (\( PT \)), and therefore it is better.

The set of tests used consists of 226 DAGs which are divided into three categories: 180 “random structure” DAGs, 60 of which have tasks with random duration (\( CR \)), 60 have tasks with duration between 1 and 2 time units (\( CR_{1.2} \)), and 60 have tasks with duration between 1 and 5 time units (\( CR_{1.5} \)); and 46 DAGs of “known structure” (obtained from the literature for comparison purposes), of which 23 have tasks with arbitrary duration (\( CC \)) and 23 have the same structure of the previous ones, but the tasks have durations between 1 and 2 time units (\( CC_{1.2} \)). All the results obtained are found in [11].

Figures 1, 2 and 3 show the comparison between ELT and ISH for different values of \( \Delta t \) and of \( p \). Figure 1 shows the result of the average of the percentages of

\* Nodes without predecessors.
the $PT$ when both algorithms, were applied to the $\text{CR}_{1.5}$ set. The graph of Figure 2 shows the result of the average of the percentages in the $PT$ when ELT and ISH were applied on the $\text{CR}_{4.2}$ set. And finally, that of Figure 3, the same result obtained on the $\text{CC}$ set.

The best results of the ELT algorithm were obtained, in general, for granularity less than 3 (Figures 1-3). On the other hand, increasing the number of processors available to carry out the assignment of the DAG improves the results obtained by the ELT algorithm in relation to the ISH algorithm. This is explained because the ELT algorithm has a priority function whose purpose is, among other things, to maximize the number of enabled tasks, so having more processors means executing these tasks and decreasing the total parallel time. However, as the number of processors increases, the performance of the ELT improves, but more slowly because ELT is in charge of executing the enabled tasks, and not of balancing the load among the processors. Therefore, as more processing elements appear, there is a point at which some of them are not being occupied, making it possible to minimize the number of processors required for the solution.

![Graph](image.png)

**Fig. 1.** Comparison of the average percentage in the $PT$ between ELT and ISH for $\text{CR}_{1.5}$ ($\Delta t = \Delta t$).
Fig. 2. Comparison of the average percentage in the PT between ELT and ISH for CR_{i,j} (Dt = Δt).

Fig. 3. Comparison of the average difference in TP between ELT and ISH for CC (Dt = Δt).
4.1 Optimums Achieved by ELT

In the results obtained by ELT, optimum assignments were found in agreement with the theoretical optimum parallel time \( PT_{opt} \) value indicated in equation (4), which represents the ideal optimum considering that maximum use is made of all the processors.

\[
PT_{opt} = \left[ \sum_{i=1}^{n} t_i / p \right]
\]  

(4)

Table 1 indicates the number of optimum assignments found for the different sets and test cases performed. The shadowed cells correspond to tests not carried out and cells with “-” indicate that an optimum assignments was not found. The number in parentheses next to each set name is the number of DAGs that it has.

<table>
<thead>
<tr>
<th>Table 1. Optimums obtained by ELT for the test sets used.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CR (60)</td>
</tr>
<tr>
<td>( \Delta t=1 )</td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>p=2</td>
</tr>
<tr>
<td>p=4</td>
</tr>
<tr>
<td>p=6</td>
</tr>
<tr>
<td>p=8</td>
</tr>
<tr>
<td>p=16</td>
</tr>
</tbody>
</table>

The largest number of optimum assignments was obtained with DAGs from tasks with average duration close to the value of \( \Delta t \), showing that for fine granularity the ELT algorithm is capable of generating optimum assignments. It should be noted that in all the sets of tests performed optimum solutions were obtained, but they are not reflected in the graphs because these consider only the average percentage in units of time from the ISH result. Of the 2334 tests performed, 455 correspond to optimum assignments. The \( \text{CR}_{1,2} \) and \( \text{CR}_{1,5} \) sets had the largest number of optimum found, 152 and 261, respectively. This result is attributed to the fact that the durations of the tasks are comparable to synchronization time, to their random structures, and to their having more tasks than CC. It is likely that there are more optimum assignments, but another method would need to be defined to show it, since the optimums are not known.

5 Conclusions

The LT algorithm is based on the idea of working using DAGs with fine granularity. The ELT algorithm keeps this characteristic: its results are improved under two conditions, when the average computing time of the DAG decreases, and when the time required for synchronization increases. The best results are achieved when the
synchronization time takes the value of the greatest duration of the tasks (granularity less than 3).

Like ELT aims at maximizing the number of enabled tasks, when number of processors for carrying out the assignment increases, the performance of the ELT algorithm improves. However, if exist more processors than enabled tasks, the rest of the processors are ignored.

Acknowledgements

This research was partially supported by CONICYT Grant FONDECYT 1030775, Chile.

Bibliographical References