Scheduling Fork Graphs under LogP with an Unbounded Number of Processors

Iskander Kort and Denis Trystram

LMC-IMAG
BP53 Domaine Universitaire
38041 Grenoble Cedex 9, France
{kort,trystram}@imag.fr

Abstract This paper deals with the problem of scheduling a specific precedence task graph , namely the *Fork* graph, under the LogP model. LogP is a computational model more sophisticated than the usual ones which was introduced to be closer to actual machines.
We present a scheduling algorithm for this kind of graphs. Our algorithm is optimal under some assumptions especially when the messages have the same size and when the gap is equal to the overhead.

1 Motivation

The last decade was characterized by the huge development of many kinds of parallel computing systems. It is well-known today that a universal computational model can not unify all these varieties. PRAM is probably the most important theoretical computational model. It was introduced for shared-memory parallel computers. The main drawback of PRAM is that it does not allow to take into account the communications through an interconnection network in a distributed-memory machine. Practical PRAM implementations have often bad performances. Many attempts to define standard computational models have been proposed. More realistic models such as BSP and LogP [3] appeared recently. They incorporate some critical parameters related to communications.

The LogP model is getting more and more popular. A LogP machine is described by four parameters L, o, g and P. Parameter L is the interconnection network latency. Parameter o is the overhead on processors due to local management of communications. Parameter g represents the minimum duration between two consecutive communication events of the same type. Parameter P corresponds to the number of processors. Moreover, the model assumes that at most $\lceil \frac{L}{g} \rceil$ messages from (resp. to) a processor may be in transit at any time.

We present in this paper an optimal scheduling algorithm under the LogP model for a specific task graph, namely the *Fork* graph. We assume that the number of processors is unbounded.

In the remainder of this section we give some definitions and notations concerning *Fork* graphs, then we briefly describe some related work. In Sect. 2, the scheduling algorithm is presented.

1.1 About the Fork Graph

A *Fork* graph is a tree of height one. It consists of a root task denoted by T_0 preceeding n leaf tasks $T_1, \ldots, T_n$. The root sends, after its completion, a message to each leaf task. In the remainder of this paper, symbol F will refer to a *Fork* graph with n leaves. In addition, we denote by w_i the execution time of task T_i, $i \in \{0, \ldots, n\}$. The processors are denoted by $p_i, i = 0, 1, \ldots$. We assume, without loss of generality, that task T_0 is always scheduled on processor p_0. Let S be a schedule of F, then $df_i(S)$ denotes the completion time of processor p_i in S where $i \in \{0, 1, \ldots\}$. Furthermore, $df(S)$ denotes the *makespan* (length of S) and df^* the length of an optimal schedule of F.

The contents of the messages sent by T_0 must be considered when dealing with scheduling under LogP. Indeed, assume that T_0 sends the same data to some tasks T_i and T_j. Then assigning these tasks to the same processor (say $p_i, i \neq 0$) saves a communication. Two extreme situations are generally considered. In the first one, it is assumed that T_0 sends the same data to all the leaves. This is called *a common data semantics*. In the second situation, the messages sent by T_0 are assumed to be pairwise different. This is called *an independent data semantics* [4].

1.2 Related Work

The problem of scheduling tree structures with communication delays and an unbounded number of processors has received much interest recently. The major part of the available results focused on the extended Rayward-Smith model. Chrétienne has proposed a polynomial-time algorithm for scheduling Fork graphs with arbitrary communication and computation delays [1]. He has also showed that finding an optimal schedule for a tree with a height of at least two is NP-Hard [2]. More recently, some results about scheduling trees under LogP have been presented. In [6], Verriet has proved that finding optimal *Fork* schedules is NP-Hard when a common data semantics is considered. In [5], the authors have presented a polynomial-time algorithm that determines optimal linear schedules for inverse trees under some assumptions.

2 An Optimal Scheduling Algorithm

We present in this section an optimal scheduling algorithm of F when the number of processors is unbounded ($P \geq n$). Furthermore , we assume that:

- The messages sent by the root have the same size.
- The gap is equal to the overhead: $g = o$. This is the case for systems where the communication software is a bottleneck.
- An independent data semantics is considered.
- $w_i \geq w_{i+1}$, $\forall i \in \{1, \ldots, n-1\}$.

We start by presenting some dominant properties related to *Fork* schedules.

Lemma 1. *Each of the following properties is dominant:*

π_1 *Each leaf task T_i such that $w_i \leq o$ is assigned to p_0.*

π_2 *Processor p_0 executes T_0 at first, then it sends the messages to tasks T_i which are assigned to other processors. These messages are sent according to decreasing values of w_i. Finally, p_0 executes the tasks that were assigned to it in an arbitrary order.*

π_3 *Each processor $p_i (i \neq 0)$ executes at most one task, as early as possible (just after receiving its message).*

Every schedule that satisfies properties $\pi_1 - \pi_3$ will be called a dominant schedule. Such schedules are completely determined when the subset A^* of the leaves that should be assigned to p_0 is known. We propose in the sequel an algorithm (called CLUSTERFORK) that computes this subset. Initially, each task is assigned to a distinct processor. The algorithm manages two variables b and B which are a lower bound and an upper bound on df^* respectively. More specifically, at any time we have $b \leq df^* < B$. These bounds are refined as the algorithm proceeds. CLUSTERFORK explores the tasks from T_1 to T_n. For each task $T_i, i \in \{1, \ldots, n\}$, one of the following situations may be encountered. If the completion time of T_i in the current schedule is not less than B then T_i is assigned to p_0. If df_i is not greater than b, then the algorithm does not assign this task to p_0. Finally, if df_i is in the range $]b, B[$, then the algorithm checks whether there is a schedule S of F such that $df(S) < df_i$. If such a schedule exists, then T_i is assigned to p_0 and B is set to df_i, otherwise T_i is not assigned to p_0 and b is set to df_i.

Theorem 1. *Let A^* be a subset produced by CLUSTERFORK, then any dominant schedule S^* associated with A^* is optimal.*

Finally, it is easy to see that CLUSTERFORK has a computational complexity of $O(n^2)$.

3 Concluding Remarks

In this paper we presented an optimal polynomial time scheduling algorithm for *Fork* graphs under the LogP model and using an unbounded number of processors. This problem was solved under some assumptions which hold for a current parallel machine namely the IBM-SP. We remark that a slight modification in the problem parameters (for instance when the messages are the same) leads to an NP-hardness result. Indeed, in this latter case, scheduling at most one task on each processor p_i, $i \neq 0$ is no more a dominant property and gathering some tasks on the same processor may lead to better schedules.

References

1. P. Chrétienne. Task Scheduling over Distributed Memory Machines. In North Holland, editor, *International Workshop on Parallel and Distributed Algorithms*, 1989.

Algorithm 1 CLUSTERFORK

Begin
{Initially: $df_i = w_0 + (i+1) * o + L + w_i$, $\forall 1 \leq i \leq n$}
$b := 0$;{lower bound on df^*}
$B := MAXVAL$; {upper bound on df^*}
for ($i := 1$ **to** n) **do**
  if ($df_i \geq B$) **then**
    $x_i := 1$; {T_i is assigned to p_0}
    $UPDATE\_DF(i)$;
  else if ($df_i > b$) **then**
    {Look for a schedule such that $df < df_i$}
    $j := i$; $k := 0$; $v := df_0$;
    while ($j \leq n$) **do**
      if ($df_j - k * o \geq df_i$) **then**
        {The assignment of T_j to p_0 is necessary}
        if ($v + w_j - o \geq df_i$) **then**
          $j := n + 1$;
        else
          $v := v + w_j - o$; $k := k + 1$;
      $j := j + 1$;
    if ($j = n + 2$) **then**
      {The looked for schedule does not exist}
      $x_i := 0$; $b := df_i$;
    else {$df^* < df_i$}
      $x_i := 1$; $B := df_i$;
      $UPDATE\_DF(i)$; {Update processors completion times}
  else {$df_i \leq b$}
    $x_i := 0$;
End CLUSTERFORK

2. P. Chrétienne. Complexity of Tree-scheduling with Interprocessor Communication Delays. Technical Report 90.5, MASI , Pierre and Marie Curie University, Paris, 1990.
3. D. E. Culler et al. LogP: A practical model of parallel computation. *Communications of the ACM*, 39(11):78–85, November 1996.
4. L. Finta and Z. Liu. Complexity of Task Graph Scheduling with Fixed Communication Capacity. *International Journal of Foundations of Computer Science*, 8(1):43–66, 1997.
5. W. Löwe, M. Middendorf, and W. Zimmermann. Scheduling Inverse Trees under the Communication Model of the LogP-Machine. *Theoretical Computer Science*, 1997. to appear.
6. J. Verriet. Scheduling Tree-structured Programs in the Logp Model. Technical Report UU-CS-1997-18, Department of Computer Science, Utrecht University, 1997.