Scheduling Tasks with Communication Delays on Multi-Levels Clusters

Jean-Yves Colin, Patrice Colin, Mustafa Nakechbandi
and Frédéric Guinand
Laboratoire d'Informatique du Havre, Université du Havre
Place Robert-Schuman, 76610 Le Havre, France
email:Jean-Yves.Colin@iut.univ-lehavre.fr

March 14, 1999

Abstract

A set of tasks has to be scheduled on the parallel identical processors of the clusters of a multi-levels distributed memory multiprocessor, subject to precedence constraints and small communication delays inside the lowest-level clusters. The architecture model includes networks of shared memory multiprocessors. In this paper, we present a new critical-path like algorithm that finds an optimal solution to this new problem in polynomial time, if task duplication is allowed and the number of available processors is not limited. The solution found is an earliest schedule that spreads the tasks between the multi-levels clusters and the processors.

1 Introduction

For compilers and schedulers to take advantage of distributed memory multiprocessor architectures, new scheduling problems have to be solved efficiently. The main reason why these problems differ from classical scheduling problems is that interprocessor communication delays must be taken into account. The execution of a task cannot start before all results from its predecessors have been transmitted to the processor that has to execute it. These communication delays themselves highly depend on the way tasks are mapped onto processors.

Several studies are currently available on many aspects of this problem. A very good survey can be found in [2]. More recently, in [1] are presented and analysed models for designing parallel programs. And in [6] is presented a list algorithm to schedule tasks and a bound on the performances of the result.

In the first part of this paper, we extend the now classical scheduling problem with communication delays and duplication to consider the problem of scheduling a set of tasks on the parallel identical processors of the clusters of a multi-levels distributed memory multiprocessor, subject to precedence constraints. In the second part, we propose a new critical-path like algorithm that finds an optimal solution to this problem, if task duplication is allowed and the lowest-level communication delays are small. The solution found is an earliest schedule that spreads the tasks between the multi-levels clusters and the processors.

We end with some concluding remarks in the third part.

2 The mL_VDS Scheduling Problem

2.1 Multi-Levels Distributed Memory Architectures

Many scheduling problems with communication delays deal with one-level communication networks. For example, a Virtual Distributed System (VDS) is a one-level distributed memory multiprocessor architecture with a non limited number of identical processors and a complete communication network between these processors, so that each processor is directly connected to each other [3]. In a VDS scheduling problem, a set of tasks has to be scheduled on the processors of a VDS, subject to precedence constraints. In [3], a polynomial algorithm named VDSOPT is presented that builds an optimal solution to this VDS scheduling problem, if the communications delays are small. The overall complexity of the whole algorithm is $O(n^2)$. Its proof can be found too in [4]. Furthermore, if communication delays are large, the VDS scheduling problem is NP-hard in most cases.

Although many distributed memory multiprocessors are one-level architectures, others however have a more complex structure that groups their processors into clusters to reduce costs. In these two-levels architectures, the processors located inside the same cluster use a very fast but small and expensive local communication network (or an even faster shared memory) to communicate, while those located on different clusters must use the cheaper and slower global communication network linking the clusters to exchange datas. The problem of scheduling

tasks on these systems is studied in [5], but the solution proposed there is limited to 2-levels architectures.

The cluster idea is often extended, however, adding levels to use larger clusters of clusters, where processors that are not located inside the same cluster of clusters must use an even cheaper and slower communication network to communicate, etc... An example of a 4-levels architecture may be a network of workstations, in which each workstation includes several general purpose processors linked by a shared memory, the workstations are linked into small clusters of workstations by very fast specialized networks (such as crossbar or omega networks), the small clusters are locally linked into local clusters by fast local area networks, and the local clusters are grouped into large clusters by a slower local or wide area network.

Thus, in the following problem, we consider mL_VDS, a m-levels VDS architecture where identical processors are grouped into 1^{st}-level clusters (either made of a shared memory, inside which communication delays are considered negligible, or made of a very fast network), the 1^{st}-level clusters are grouped into 2^{nd}-level clusters, the 2^{nd}-level clusters are grouped into 3^{rd}-level clusters, etc... up to the $(m\text{-}2)^{th}$-level clusters that are grouped into $(m\text{-}1)^{th}$-level clusters. Each level u is composed of a non limited number of $(u-1)^{th}$-level clusters, and each 1^{st}-level cluster is composed of a non limited number of processors. A complete communication network (or maybe a shared memory in the 1^{st}-level clusters) links the processors of its components, and only these processors.

2.2 The New Problem

A mL_VDS scheduling problem may then be specified by the four parameters (I, U, p, C).

1. $I = \{1, 2, \ldots, n\}$ is a set of n tasks. The processing time of task i is p_i.

2. $G = (I, U)$ is a directed acyclic graph that models the precedence constraints,

3. To each arc (i, j) of U is associated a set $C(i, j)$ of C of positive communication times $\{c_{i,j}(1), c_{i,j}(2), \ldots, c_{i,j}(m)\}$, with $c_{i,j}(u-1) \leq c_{i,j}(u)$, where $c_{i,j}(u)$ represents the communication delay if i and j are executed inside the same u^{th}-level cluster but in different $(u-1)^{th}$-level clusters (if i and j are executed on the same processor, there is no need for a communication, so there is no communication delay). This value may be equal to zero.

The difference with the classical VDS model is then the presence of several different communication delays $c_{i,j}(u)$, in place of a lone communication delay value per arc. Figure 1 is an example of such a graph if m=2, that is in a 2L_VDS scheduling problem. The first value on or below an arc is $c_{i,j}(1)$ and the second value is $c_{i,j}(2)$.

A task is indivisible, starts when all the pieces of informations it needs from its predecessors are available to it, and gives all the informations needed by its successors at the end of its execution.

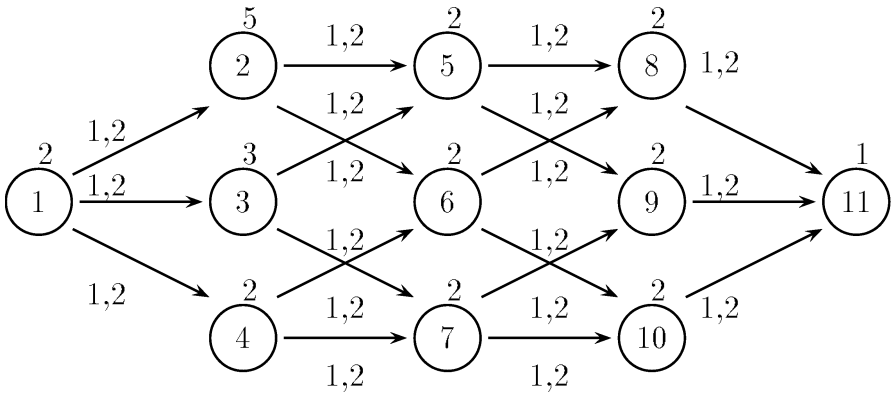

Figure 1: Example of graph with communication delays in a 2L_VDS scheduling problem: the value above each node i is the processing time p_i, the first (resp. second) value above each arc (i, j) is the communication delay $c_{i,j}(1)$ (resp. $c_{i,j}(2)$).

In the following, we will denote $PRED(i)$ (respectively $SUCC(i)$) the set of immediate predecessors (resp. successors) of task i in G.

Task duplication is allowed, that is several instances of the same task may be executed on different processors. This may be useful in some cases. We denotes i_k the k^{th} instance of task i.

A schedule S of a mL_VDS scheduling problem is then a triple (F, t, π), where

1. $F(i)$, $i \in I$ is the positive number of copies of task i.

2. t_{i_k} is the starting time of copy i_k of task i.

3. π_{i_k} is the processor assigned to copy i_k of task i.

A schedule S must also satisfy the following conditions:

1. At least one copy of each task is processed.

2. At any time, a processor executes at most one copy.

3. If (i, j) is an arc of G, then for any copy j_l of task j, there must exist at least one copy i_k of task i such that

$$\begin{array}{ll} t_{j_l} \geq t_{i_k} + p_i & \text{if } \pi_{j_l} = \pi_{i_k} \\ t_{j_l} \geq t_{i_k} + p_i + c_{i,j}(u) & \text{if } \pi_{j_l} \text{ and } \pi_{i_k} \text{ are not in the same } (u-1)^{th}\text{-level cluster} \end{array}$$

If, in a schedule S, i_k and j_l satisfy the two above inequalities, we will say that the **Generalized Precedence Constraint** is true for the two copies (in short, that GPC(i_k,j_l) is true). This Generalized Precedence Constraint means that a task needs its informations from one copy only of each one of its predecessors.

Let C_{i_k} denote the completion time of copy i_k of task i. We then wish to minimize the makespan of the schedule, that is, the maximum task completion time $C_{max} = \max_{i \leq n, k \leq F(i)} C_{i_k}$.

One can first note that the mL_VDS problem is NP-hard in the general case. The proof is based on the fact that any optimal solution to the mL_VDS problem is an optimal solution too to the VDS problem (placing a solution to the mL_VDS problem in one 1^{st}-level cluster results in a solution to the VDS problem). Thus, any polynomial algorithm solving the mL_VDS problem could be used to solve the VDS problem. However, the VDS problem itself is NP-hard in the general case ([7], [8]).

One can note too that there already is an easy solution to this mL_VDS problem, if the 1^{st}-level communication delays are small. Again, assume that

one 1^{st}-level cluster only will be used. This changes the problem to a simple VDS problem, where the communications times $c_{i,j}$ are the 1^{st}-level $c_{i,j}(1)$ values. Using the VDSOPT algorithm presented in [3] then gives in this case an optimal solution, and this solution is an optimal solution too to the mL_VDS problem.

The solution found this way is not really an useful solution, however. So we propose below mL_VDSOPT, a new algorithm that computes an earliest schedule while spreading the tasks on several clusters if possible.

3 The mL_VDSOPT Algorithm

We assume that the following condition holds for the processing times p and the 1^{st}-level communication delays $c(1)$:

$$\min_{g \in PRED(i)} p_g \geq \max_{h \in PRED(i)-\{g\}} c_{h,i}(1) \ .$$

Note that this is in fact the 'small communication delays' hypothesis of [2], [3] and [4], limited here to the 1^{st}-level communication delays.

The mL_VDSOPT algorithm we now propose is made of two important steps.

First is the procedure VDSLWB that computes lower bounds of the starting times of any copy of each task.

The second step is the recursive procedure mL_VDSSOL that at each level u, schedules the copies of the tasks in these clusters if $u = 1$, else builds a graph of sequences, computes the connected components of this graph, then allocates one $(u-1)^{th}$-level cluster to each connected component.

Procedure VDSLWB(G, p, C) can be stated as follows:

For any task i such that $PRED(i) = \emptyset$, let its lower bound b_i be zero;

While there is a task i which has not been assigned a lower bound b_i and whose predecessors in $PRED(i)$ all have lower bounds assigned do

Define

$$C = \max_{k \in PRED(i)} (b_k + p_k + c_{k,i}(1))$$

Let s be such that

$$b_s + p_s + c_{s,i}(1) = C$$

Define the lower bound

$$b_i = \max(b_s + p_s, \max_{k \in PRED(i) - \{s\}} (b_k + p_k + c_{k,i}(1)))$$

Endwhile

Basically, VDSLWB uses only the p processing times and the $c(1)$ set of communication times to computes an earliest execution date b_i for each task i of G.

Table 1 presents the computed earliest execution dates of the tasks of Fig. 1. Figure 2 is the resulting critical graph.

Following C.P.M. terminology, we define an arc (i, j) of U to be critical if $b_i + p_i + c_{i,j}(1) > b_j$. From this definition, it is clear that if (i, j) is a critical arc,

Table 1: Earliest execution dates of tasks from Fig. 1 using the $c(1)$ communications times.

task i	1	2	3	4	5	6	7	8	9	10	11
lower bound b_i	0	2	2	2	7	7	5	10	9	9	12

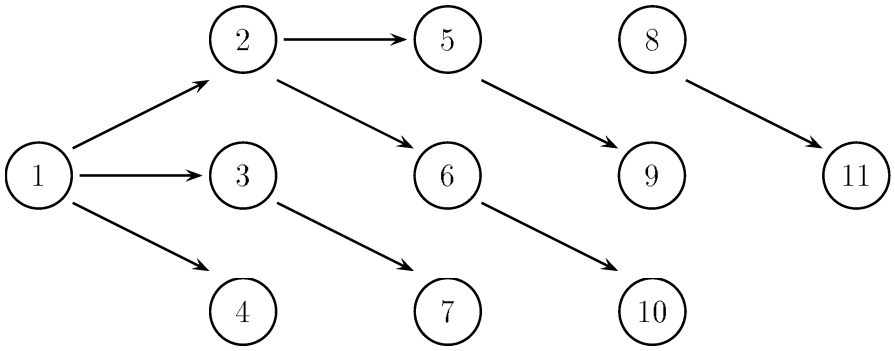

Figure 2: Critical graph extracted from Fig. 1 using the $c(1)$ communications times: the critical sequences are (1,4), (1,3,7), (1,2,6,10), (1,2,5,9) and (8,11).

then in any earliest schedule every copy of task j must be preceded by a copy of task i executed on the same processor.

A critical sequence of the precedence graph is a path such that all of its arcs are critical and it is not a proper subpath of a critical sequence. The critical graph of the VDS scheduling problem is the subgraph of G induced by the critical arcs. The critical graph is always a forest, thus making the search of all critical sequences easy.

The critical graph resulting from the computed lower bounds may be reduced into a u^{th}-level ($u > 1$) sequence graph. This u^{th}-level sequence graph is a non directed graph where:

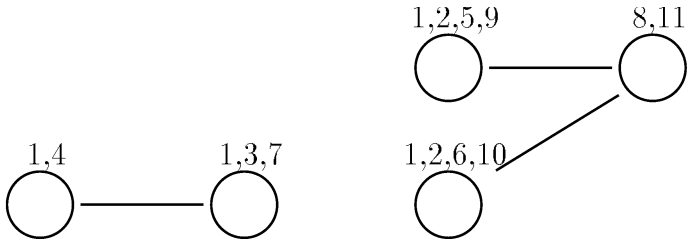

Figure 3: 2^{nd}-level sequence graph extracted from Fig. 2: there are two connected components, one made of two nodes (sequences (1,4) and (1,3,7)) and one made of three nodes (sequences (1,2,5,9), (1,2,6,10) and (8,11)).

1. Each node s represents one sequence of the critical graph.

2. There is an edge between two nodes s_1 and s_2 if there is at least one arc (i, j) of G such that $i \in s_1$, $j \in s_2$ and $b_i + p_i + c_{i,j}(u) > b_j$.

The above definition states that there is an edge between two nodes of the u^{th}-level sequence graph if the two sequences they represent must be in the same $(u-1)^{th}$-level cluster.

Figure 3 presents the 2^{nd}-level sequence graph resulting from the critical graph of Fig. 2. Labels on the nodes are the paths of the corresponding critical sequences.

The sequence graph of Fig. 3 for example includes two connected components, one made of two nodes (sequences (1,4) and (1,3,7)) and one made of three nodes (sequences (1,2,5,9), (1,2,6,10) and (8,11)).

The second important step of our algorithm, procedure mL_VDSSOL(cc, u) (where cc is a connected component of a $(u+1)^{th}$-level sequence graph), builds

the u^{th}-level sequence graph of cc, and allocates one $(u-1)^{th}$-level cluster to each its connected components, or one processor to each of its critical sequences if $u = 1$. It can be stated as follows:

if u=1 then

For each critical sequence sq of cc do

Assign sq to a distinct processor of the current cluster:

one copy of each task of sq must be executed on the

processor assigned to sq;

Execute each copy i_k of each task i at its lower bound

b_i;

EndFor

else

Build the u^{th}-level sequence graph that can be extracted from

cc;

Compute the connected components of this sequence graph;

For each connected component cc_2 of this sequence graph do

assign cc_2 to a distinct $(u-1)^{th}$-level cluster of the

current u^{th}-level cluster;

call mL_VDSSOL($cc_2, u-1$);

EndFor

EndFor

The mL_VDSOPT algorithm is then:

call VDSLWB(G, p, C);

Build a sequence graph sg in which there is one arc between two nodes of sg if there is an arc from a task of the first node toward a task of the second node;

call mL_VDSSOL(sg, m);

Thus, the two connected components of Fig. 3 are first scheduled on two 1^{st}-level clusters. In each 1^{st}-level cluster, a processor is then allocated to each critical sequence of the corresponding connected component. The resulting schedule is represented on Fig. 4.

The overall complexity of the whole mL_VDSOPT algorithm is $O(n^2 \times m^2)$.

4 Some Remarks

One can first note that the solution found has the same makespan than the solution found by VDSOPT when assuming that an as large as needed 1^{st}-level cluster of processors will be used. This implies that mL_VDSOPT builds an optimal solution too (if the 1^{st}-level communications times are inferior or equal to the processing times). The execution dates of the copies of the tasks are the same too, implying that the solution computed is an earliest schedule too (again if the 1^{st}-level communications times are inferior or equal to the processing times).

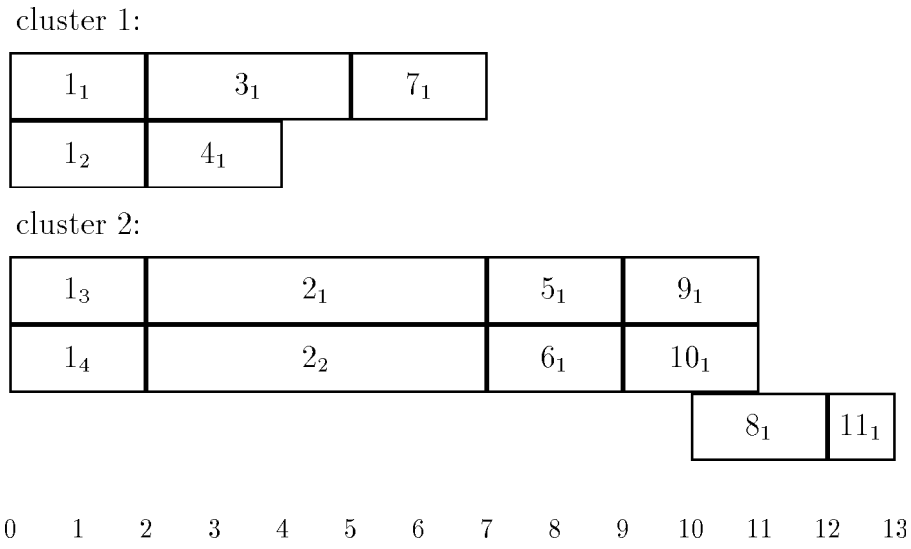

Figure 4: The earliest schedule of copies of the tasks of Fig. 1: one 1^{st}-level cluster is assigned to each connected component of the 2^{nd}-level sequence graph, and, inside each cluster, one processor is assigned to each critical sequence.

So, the new algorithm presented here is an efficient polynomial algorithm that builds an optimal solution to the stated problem. The solution is an earliest schedule on the copies of the tasks. These task copies are spread between the processors clusters.

It is interesting to note too that the optimality of the algorithm does not depend on the speed of the slower communication networks between the clusters of any level strictly above the first one. Actually, the speed of these networks only affects the levels of used clusters, their number at each level, and the number of processors per 1^{st}-level cluster used. Thus the slower these networks, the lower the levels of the clusters used, the less clusters used at each level, and the more processors per 1^{st}-level cluster needed. The worst case being networks between

clusters so slow that the optimal solution found is the degenerate one (everything in one 1^{st}-level cluster) computed by VDSOPT.

Also interesting is the fact that the connected components of the sequence graph are to the 1^{st}-level clusters what the critical sequences are to the processors: one copy of each task of a connected component of the sequence graph of level 1 must be executed in a 1^{st}-level cluster, while one copy of each task of a critical sequence of the critical graph must be executed on a processor. And so on at each higher level. However, the same way VDSOPT does not necessarily use the smallest number of processors, mL_VDSOPT does not necessarily use the smallest number of $(u-1)^{th}$-level clusters per u^{th}-level cluster.

Finally and most importantly, the same way critical-path algorithms are used as the first step of many scheduling heuristics for real shared memory processors with limited numbers of processors (providing priorities for list scheduling algorithms or lower bounds for branch-and-bound methods, for example), this theoretical result may lead to improved heuristics for tasks clustering or tasks scheduling algorithms when targeting real multi-levels distributed multiprocessors, where the number of processors per cluster, and the number of clusters available at each level, are limited. Again, by providing priorities for new list scheduling algorithms, or lower bounds for branch-and-bound methods, for example.

5 Conclusions

A set of tasks had to be scheduled on the parallel identical processors of the clusters of a multi-levels distributed memory multiprocessor, subject to precedence constraints and small communication delays at the lowest level. In this paper, we proposed a new critical-path like algorithm that finds an optimal solution to this problem in polynomial time. The solution found is an earliest schedule that spreads the tasks between the clusters and the processors.

References

[1] Bampis, E., Guinand, F., and Trystram D.: Some Models for Scheduling Parallel Programs with Communication Delays, Discrete Applied Mathematics vol. 172, n^o 1-2 (1997) 5-24.

[2] Chrétienne, P., and Picouleau, C.: Scheduling with Communication Delays: A Survey, in: P. Chrétienne, E. G. Coffman, Jr., J. K. Lenstra and Z. Liu (eds.), Scheduling Theory and its Applications, John Wiley & Sons Ltd, Chichester, England (1995) 65-110.

[3] Colin, J.-Y.: Problèmes d'Ordonnancement avec Délais de Communication: Complexité et Algorithmes, Ph.D. dissertation, Université Pierre et Marie Curie, Paris, France (1989)

[4] Colin, J.-Y., and Chrétienne, P.: C.P.M. Scheduling with Small Communication Delays and Task Duplication, Operations Research vol. 39, n^o 4 (1991) 680-684

[5] Colin, J.-Y., and Nakechbandi, M.: Scheduling Tasks with Communication Delays on 2-Levels Virtual Distributed Systems, Proceedings of the PDP'99 - 7th Euromicro Workshop on Parallel and Distributed Processing, Funchal, Portugal (1999)

[6] Munier, A., and Hanen, C.: Using Duplication for Scheduling Unitary Tasks on m Processors with Unit Communication Delays, Theoretical Computer Science vol. 178, n^o 1-2 (1997) 119-127

[7] Papadimitriou, C. B., and Yannakakis, M.: Toward an Architecture-independent Analysis of Parallel Algorithms, Proceedings of the 20th Annual ACM Symposium Theory of Computing, Santa Clara, California, USA (1988)

[8] Rayward-Smith, V. J.: U.E.T. Scheduling with Unit Interprocessor Communication Delays, Discrete Math. vol. 18 (1987) 55-71