Scheduling Tasks and Communications on a Hierarchical System with Message Contention*

Jean-Yves Colin and Moustafa Nakechbandi

LITIS, Université du Havre, IUT
76610 Le Havre, France
{jean-yves.colin,moustafa.nakechbandi}@univ-lehavre.fr

Abstract A Directed Acyclic Graph (DAG) of tasks with small communication delays has to be scheduled on the identical parallel processors of clusters connected by a hierarchical network. The number or processors and of clusters is not limited. Message contention has to be avoided. Task duplication is allowed. In this paper, we present a new polynomial algorithm that computes the earliest start dates of all tasks and spreads these tasks to use few processors per cluster, for a DAG with small communication delays. It also avoids message contention, and always delivers messages on time.

Keywords: Scheduling, DAG, Hierarchical Communications, Message contention, Task Duplication, CPM/PERT

1 Introduction

The efficient use of distributed memory multiprocessors and grids is a very difficult problem. An application is made of different parts, with specific processing times and communication delays, that need to be scheduled carefully. Examples of applications include numerical analysis applications, logistics systems based on heterogeneous distributed computing systems, high performance Data Mining systems, and Automated Document Factories in banking environments.

In the classical scheduling problem with communication delays, a positive processing time is associated to each task of a Directed Acyclic Graph (DAG) and a positive communication delay is associated to each precedence constraint between the tasks of this DAG. The tasks then have to be scheduled on the processors of the distributed memory multiprocessor or grid. This problem is known to be NP-hard in the general case even if the number of available processors is not limited [8]. Many studies are currently available on several aspects of this classical scheduling problem [3] [4] [5] [9] [13] [17] [18].

Task duplication, for example, is used in several studies to lower the communication overheads by executing identical copies of some of the tasks on different processors [1] [4] [5] [12].

* This work is partially funded by the GRR "Transport Logistique et Technologie de l'Information" of the Université du Havre, France.

Hierarchical communications are taken into account in some studies too. The processors are typically grouped into *clusters*, with communication between processors of the same cluster being faster than communications between processors of different clusters [1] [7].

These problems are increasingly recognized to be unrealistic however, because they do not consider message contention [2] [11] [14] [21]. In [15] for example, the authors show the NP-Completeness of the two processor scheduling problem with tasks of execution time 1 or 2 units, unit interprocessor communication latency and message contention. In [6], a CPM/PERT-like polynomial scheduling algorithm for DAG with small communication delays and task duplication is proposed. It is optimal and always avoids message contention, if resources are not limited. It does not consider hierarchical communications, however. More recent studies use heuristics to avoid message contention, and present extensive experimental evaluations to evaluate performance improvements [19] [20].

In this paper, we present a new polynomial algorithm for DAG with small communication delays. The distributed architecture is made of clusters, has a two level communication network and has communication channels that can transmit at most one message at any time. The algorithm computes, if resources are not limited, the earliest start dates of all tasks and spreads these tasks to use few processors per cluster. It also schedules the communications so that message contention is avoided, and always delivers messages on time.

2 The 2LVDS Problem

2.1 The 2LVDS Model

A *2-levels Virtual Distributed System architecture* (2LVDS) is a distributed memory multi-processor architecture (or grid) with a not limited number of homogeneous processors. The processors are grouped into *clusters*. Both the number of clusters and the number of processors in each cluster are not limited. Each processor belongs to one and only one cluster (Fig. 1).

There is a complete communication network between all the processors. Each direct connection between any two processors is made of two unidirectional *channels*, one in each direction. All communications channels inside all clusters are identical and all communications channels between processors of different clusters are identical too, but slower than the intra-cluster channels. Each unidirectional channel may carry at most one message at any time.

An application is represented by a DAG $G = (V, E)$ (or *precedence graph*) where V designates the set of tasks, and E the set of precedence constraints. Formally, a 2LVDS scheduling problem may then be specified by the four parameters V, E, p, c, in which $V = \{1, 2, ...n\}$ is the set of n tasks, E is the set of arcs (i, j) with $(i, j) \in E$ representing a precedence constraint from task $i \in V$ to task $j \in V$, p is the set of processing times with $p_i \in p$ being the processing time of task $i \in V$ on any processor π of the 2LVDS architecture, and c is the set of communications delays. To each arc $(i, j) \in E$ are associated two values $c_{i,j}(1) \in c$ and $c_{i,j}(2) \in c$. $c_{i,j}(1)$ is the positive communication delay of a

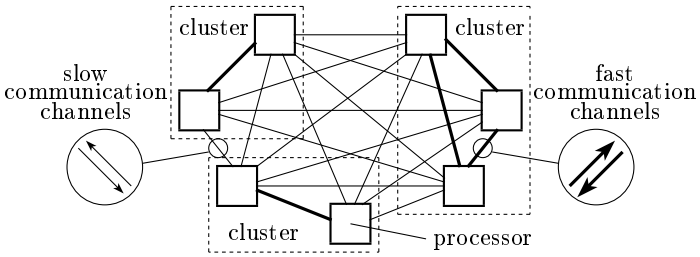

Figure 1. A 2LVDS architecture

message from task i to task j, if i and j are executed on different processors inside the same cluster (*intra-cluster* communication delay). $c_{i,j}(2)$ is the positive communication delay of a message from task i to task j, if i and j are executed in different clusters (*inter-cluster* communication delay), with $c_{i,j}(1) \leq c_{i,j}(2)$. If two communicating tasks i and j are executed on the same processor, there is no need for any communication or its duration is considered negligible, so the communication delay is then 0.

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

All the immediate successors of a task use the same result from this task. This assumption implies that a task needs to send one message only to a given processor, even if several of its successors are to be processed on it, because one message is enough for all. If it does not hold, the task may usually be divided into sub-tasks such that the assumption is satisfied.

Fig. 2 presents an example of such a DAG. The value above each node is its processing time, and the two values above each arc are its two communication delays.

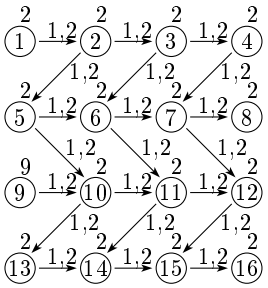

Figure 2. Example of a DAG with two communication delays

Task duplication is allowed. That is, several instances (or *copies*) of the same task may be executed on different processors. We will denote i^k the k^{th} copy of

task i. Because we must take into account the messages in a schedule, we will denote $m(i_k, j_l)$ a message sent from a copy i_k of task i to a copy j_l of task j.

A schedule S of a 2LVDS scheduling problem is then a 5-tuple (F, t^c, π, M, t^m), where

$F(i)$ is the positive number of copies of task $i \in V$,
$t^c(i_k)$ is the starting time of copy i_k of task i, $0 < k \leq F(i)$,
$\pi(i_k)$ is the processor assigned to copy i_k of task i, $0 < k \leq F(i)$,
$M(i, j)$ is the set of all messages sent by copies of task i to copies of task j,
$t^m(m(i_k, j_l))$ is the starting time of message $m(i_k, j_l) \in M(i, j)$.

First, to be feasible, a schedule S must satisfy the following conditions:

- at least one copy of each task is processed, i.e. $\forall i \in V$, $F(i) > 0$,
- at any time, a processor executes at most one copy,
- for each $(i, j) \in E$, for any copy j_l of j, there is one copy i_k of i that is on the same processor or that sends its message on time to j_l, i.e.

  if $\pi(j_l) = \pi(i_k)$ **then**
  $t^c(j_l) \geq t^c(i_k) + p_i$
  else if $\pi(j_l)$ and $\pi(i_k)$ are in the same cluster **then**
  $t^c(j_l) \geq t^c(i_k) + p_i + c_{i,j}(1)$
  else
  $t^c(j_l) \geq t^c(i_k) + p_i + c_{i,j}(2)$
  end if

If, in a schedule S, i_k and j_l satisfy the above condition, we will say that the *Generalized Precedence Constraint* is true for the two copies (in short, that $GPC(i_k, j_l)$ is true).

Second, a feasible schedule S must additionally satisfy the condition that there is no message contention, i.e. in all channels used to transmit at least two messages $m(i_k, j_l)$ and $m(r_t, s_q)$ from a processor π_{i_k} to a processor π_{j_l}, with message $m(i_k, j_l)$ finishing before message $m(r_t, s_q)$, we have

if if $\pi(j_l)$ and $\pi(i_k)$ are in the same cluster **then**
$t^m(m(r_t, s_q)) \geq t^m(m(i_k, j_l)) + c_{i,j}(1)$
else
$t^m(m(r_t, s_q)) \geq t^m(m(i_k, j_l)) + c_{i,j}(2)$
end if

Now, let $C(i_k)$ be the *completion time* of a copy i_k of a task i, i.e. $C(i_k) = t^c(i_k) + p_i$. The *maximum completion time*, or *makespan*, $C_{\max}$ of a solution S is the largest completion time of all copies of all tasks in this solution:

$$C_{\max} = \max_{i \in V, k \leq F(i)} \{t^c(i_k) + p_i\} \ . \tag{1}$$

As usual for this kind of problem, we want to minimize $C_{\max}$, that is, find a feasible solution S^* with the smallest makespan $C^*_{\max}$.

One can note that, if $c_{i,j}(1) = c_{i,j}(2)$, this scheduling problem is actually equivalent to the classical DAG scheduling problem with communication delays

which, in the general case, is a NP-hard problem, even if the number of processors is not limited [16]. For this reason, we will only consider a DAG satisfying the conditions in the following two equations. They guarantee that the DAG has small communication delays. We will denote $PRED(i)$ (respectively $SUCC(i)$) the set of immediate predecessors (resp. successors) of task i in G.

$$\forall i \in V, \min_{g \in PRED(i)} p_g \geq \max_{h \in PRED(i)-\{g\}} c_{h,i}(1) \ . \tag{2}$$

Equation (2) means that processing times are locally superior or equal to the communication delays inside the clusters. It ensures that the earliest start date of any copy of each task may be computed in polynomial time.

$$\forall i \in V, \min_{k \in SUCC(i)} p_k \geq \max_{j \in SUCC(i)-\{k\}} c_{i,j}(2) \ . \tag{3}$$

Equation (3) is very similar to (2). It means too that the processing times are locally superior or equal to the communication delays between the clusters. However, (2) deals with the predecessors of a task and with the intra-clusters communication delays, while (3) deals with the successors, and with the inter-clusters communication delays. Also (2) is true in most cases if (3) is true.

One can note that there is already a trivial solution to the 2LVDS problem: use one cluster only, and schedule all tasks on the processors of this cluster using the algorithm in [6]. This trivial solution, however, is not helpful at all, because real architectures have a limited number of processors in each cluster. For this reason, we propose the following new algorithm 2LVDSOPT. It schedules the tasks and communications in a 2LVDS problem in polynomial time and spreads the tasks on as many clusters as possible to use less processors per cluster.

2.2 The 2LVDSOPT Algorithm

This algorithm has four steps. The first step 2LVDSLWB() computes the earliest start date of all copies of each task of the DAG. The second step 2LVDSCS() computes the critical sequences of the DAG according to the earliest start dates calculated during the first step. The third step 2LVDSCC() computes the graph of the critical sequences of the DAG, and its connected components according to the communication delays $c_{i,j}(1)$. The last step 2LVDSBUILD() computes the solution, scheduling the tasks and communication on the 2LVDS architecture.

Computing the Earliest Start Dates. The first step of 2LVDSOPT computes the earliest start date b_i of all copies of each task i of the DAG. This is done in procedure 2LVDSLWB() (cf. Algorithm 1). Table 1 presents the earliest start dates of each task of the DAG of Fig. 2 computed by procedure 2LVDSLWB().

Computing the Critical Sequences. The second step of 2LVDSOPT computes the critical sequences resulting from the earliest start dates calculated during step 1.

Algorithm 1 procedure 2LVDSLWB(V, E, p, c)

for all tasks $i \in V$ such that $PRED(i) = \varnothing$ **do**
  let $b_i = 0$ {assign 0 to i as its earliest start date b_i}
end for
while there is a task i which has not been assigned an earliest starting date b_i and whose predecessors $h \in PRED(i)$ all have an earliest starting date b_h assigned to them **do**
  let $c = \max_{h \in PRED(i)} b_h + p_h + c_{h,i}(1)$
  find $g \in PRED(i)$ such that $b_h + p_h + c_{h,i}(1) = c$
  let $b_i = \max\left(b_g + p_g, \max_{h \in PRED(i)-\{g\}} b_h + p_h + c_{h,i}(1)\right)$
end while

Table 1. Earliest start dates b_i of the tasks i of the DAG of Fig. 2 computed by procedure 2LVDSLWB() (cf. Algorithm 1)

task i:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
b_i:	0	2	4	6	4	7	9	11	0	9	11	13	11	14	16	18

Let B be the set of the earliest start dates b_i of all tasks of V.

Let GC be the *critical subgraph* of G according to the earliest start dates in B. (i, j) is an arc of GC if $(i, j) \in E$ and $b_j < b_i + p_i + c_{i,j}(1)$. That is, an arc (i, j) in GC means that these two tasks must have copies on the same processor, because there is not enough delay to transmit the result of any copy i_k to a copy j_l from one processor to another processor of the same cluster. GC is always a forest [5]. A *critical sequence* sc of the DAG is a proper path of GC.

The computation is done in procedure 2LVDSCS() (cf. Algorithm 2).

Algorithm 2 procedure 2LVDSCS(V, E, p, c, B)

$GC = \varnothing$
for all arcs $(i, j) \in E$ **do**
  if $b_j < b_i + p_i + c_{i,j}(1)$ **then**
    $GC = GC \cup \{(i, j)\}$
  end if
end for
$s = 0$
for all tasks $i \in V$ **do**
  if task i is a leaf of the critical subgraph GC **then**
    let critical sequence sc_s be the path from the root of the tree in GC that includes task i, to task i
    $s = s + 1$
  end if
end for

Computing the Graph of the Critical Sequences. The third step of 2LVDSOPT builds the undirected graph GSC of the critical sequences sc_s and computes its connected components [10].

Let CC be the set of all computed critical sequences sc_s.

GSC has one node s_s for each critical sequence sc_s of CC computed during the previous step. Also, there is one edge (s_s, s_t) or (s_t, s_s) in GSC if $\exists (i,j) \in E$, with $i \in sc_s$, and $i \notin sc_t$, and $j \in sc_t$, such that $b_j < b_i + p_i + c_{i,j}(2)$. This edge means that there is not enough time to transmit one message between at least one task i of sc_s to another task j of sc_t between two clusters. So sc_s and sc_t must be processed in the same cluster.

The computation is done in procedure 2LVDSCC() (cf. Algorithm 3).

Algorithm 3 procedure 2LVDSCC(V, E, p, c, B, CC)

```
GSC = ∅
for all critical sequences sc_s ∈ CC do
   let s_s be the new node related to sc_s
end for
for all nodes s_s do
   GSC = GSC ∪ {s_s}
   for all nodes s_t ∈ GSC − {s_s} do
      if there is no edge between s_s and s_t in GSC and there is at least one arc (i, j)
      of E with i ∈ sc_s and i ∉ sc_t and j ∈ sc_t, such that b_i < b_i + p_i + c_{i,j}(2) then
         add one edge between s_s and s_t to GSC
      end if
   end for
end for
compute the connected components g_s of GSC
```

Fig. 3 shows the six critical sequences sc_1 to sc_6 found for the DAG of Fig. 2 using the computed earliest start dates in Table 1. It also shows the graph of the critical sequences and its two connected components.

Computing the Solution. The last step of 2LVDSOPT builds a solution with minimal makespan using all the data computed in the preceding phases.

One cluster is allocated to each connected component, and one processor of this cluster is allocated to each critical sequence of this connected component. One copy of each task of each critical sequence is executed at its earliest start date. All messages are sent as soon as the sending copy of the task finishes its execution.

The computation is done in procedure 2LVDSBUILD() (cf. Algorithm 4).

Fig. 4 shows the Gantt chart of the final schedule found for the DAG of Fig. 2. Two clusters, each with three processors, are used. Tasks 1, 2, 9 and 10 have two copies each in this schedule.

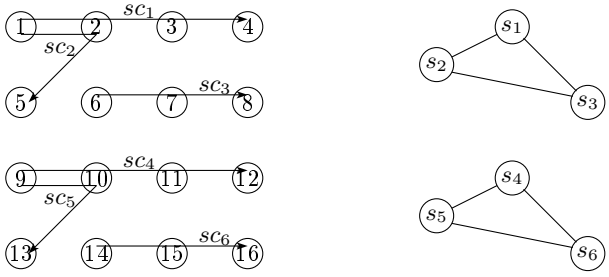

Figure 3. The six critical sequences sc_1 to sc_6 in the critical graph GC of the DAG in Fig. 2 (left), and the graph GSC of these critical sequences with the two resulting connected components (right)

Algorithm 4 procedure 2LVDSBUILD(V, E, p, c, B, CC, GSC)

---

for all connected components $g_c \in GSC$ **do**
  allocate a new cluster Π_c to g_c
  for all node $s_s \in g_c$ **do**
    let sc_s be the critical sequence related to node s_s
    allocate a new processor π_s in cluster Π_c to this critical sequence sc_s
    for all task $i \in sc_s$ **do**
      $F(i) = F(i) + 1$, $t^c(i_{F(i)}) = b_i$, $\pi(i_{F(i)}) = \pi_s$
    end for
  end for
end for
for all copy j_l of task j **do**
  let $\pi(j_l)$ be the processor that executes j_l
  for all task $i \in PRED(j)$ **do**
    if there is no copy of task i on $\pi(j_l)$ and $\pi(j_l)$ does not already receive one message from any copy of i on time for copy j_l **then**
      remove any message m' from any copy of i to processor $\pi(j_l)$
      find one copy i_k that can send its message on time to j_l
      send one message $m(i_k, j_l)$ from copy i_k at date $b_i + p_i$ to processor $\pi(j_l)$
    end if
  end for
end for

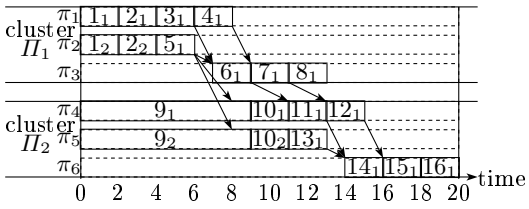

Figure 4. Gantt chart of the solution of the DAG of Fig. 2, using two clusters Π_1 and Π_2, with three processors per cluster (π_1, π_2 and π_3 in Π_1, and π_4, π_5 and π_6 in Π_2)

2.3 Analysis of the Algorithm

Let n be the number of tasks and m be the number of arcs.

The complexity of procedure 2LVDSLWB() is $O(\max(m,n))$, and the complexity of procedure 2LVDSCS() is $O(m)$. The complexity of building the graph of the critical sequences in 2LVDSCC() is $O(n)$ [5], and of computing its connected components is $O(n)$. Thus the complexity of 2LVDSCC() is $O(n)$ too.

Using a graph-level approach, one can show that the complexity of the first part of 2LVDSBUILD() is $O(n^2)$. Because the second part of 2LVDSBUILD() tries, in the worst case, to find one suitable copy of each predecessor for each copy of each task, it is possible to establish that the complexity of this second part is $O(m^2n^2)$. The complexity of procedure 2LVDSBUILD() is then $O(m^2n^2)$.

So the complexity of the overall algorithm is $O(m^2n^2)$.

Also, we have the following theorems.

Theorem 1. *The solution built by* 2LVDSOPT *has minimal makespan.*

Theorem 2. *At least one copy of each task is executed.*

Theorem 3. *The GPC are true for all copies of all tasks.*

Theorem 4. *In the solution computed, each copy of each task receives at least one message on time from at least one copy of each of its predecessor, if a message is needed.*

Theorem 5. *There is no message contention on any unidirectional channel.*

3 Conclusion

A Directed Acyclic Graph of tasks with small communication delays had to be scheduled on the identical parallel processors of several clusters connected by a hierarchical network. The number of processors and of clusters was not limited. Message contention had to be avoided. Task duplication was allowed. We presented a new polynomial algorithm that computes the earliest start dates of tasks and spreads these tasks to use few processors per cluster. It also schedules the communications so that there is no message contention and messages are always delivered on time.

References

1. Bampis, E., Giroudeau, R., König, J.-C.: Using Duplication for Multiprocessor Scheduling Problem with Hierarchical Communications. Parallel Processing Letters 10(1), 133-140 (2000)
2. Beaumont, O., Boudet, V., Robert, Y.: A Realistic Model and an Efficient Heuristic for Scheduling with Heterogeneous Processors. 11th Heterogeneous Computing Workshop (HCW'2002), IEEE Computer Society Press (2002)

3. Bittencourt, L.F., Sakellariou, R., Madeira, E.R.M.: DAG Scheduling Using a Lookahead Variant of the Heterogeneous Earliest Finish Time Algorithm. 18th Euromicro International Conference on Parallel, Distributed and Network-Based Computing (PDP 2010). Pisa, Italy (2010)
4. Bozdag, D., Ozguner, F., and Catalyurek, U.V.: Compaction of Schedules and a Two Stage Approach for Duplication-Based DAG Scheduling. IEEE Transactions on Parallel and Distributed Systems 20(6), 857-871 (2009)
5. Colin, J.-Y., Chrétienne, P.: Scheduling with Small Communication Delays and Task Duplication. Operations Research 39(4), 680-684 (1991)
6. Colin, J.-Y., Colin, P.: Scheduling Tasks and Communications on a Virtual Distributed System. European Journal of Operational Research 94(2) (1996)
7. Colin, J.-Y., Nakechbandi, M: Scheduling Tasks with Communication Delays on 2Levels Virtual Distributed Systems. Proceedings of the 7th Euromicro Workshop on Parallel and Distributed Processing (PDP'99). Funchal, Portugal (1999)
8. Garey, M., Johnson, D.: Computers and Intractability, a Guide to the Theory of NP-Completeness. Freeman (1979)
9. Giroudeau, R., König, J.-C.: Scheduling with Communication Delay,. In: Multiprocessor Scheduling: Theory and Applications, ARS Publishing, 1-26 (2007)
10. Hopcroft, J., Tarjan, R.: Efficient Algorithms for Graph Manipulation. Communications of the ACM 16, 372-378 (1973)
11. Kalinowski, T., Kort, I., Trystram, D.: List Scheduling of General Task Graphs under LogP. Parallel Computing 26, 1109-1128 (2000)
12. Kruatrachue, B., Lewis, T.G.: Grain Size Determination for Parallel Processing. IEEE Software 5(1), 23-32 (1988)
13. Kwok, Y.-K., Ahmad, I.: Static Scheduling Algorithms for Allocating Directed Task Graphs to Multi-Processors. ACM Computing Surveys (CSUR) 31(4), 406-471 (1999)
14. Marchal, L., Rehn, V. , Robert, Y., Vivien, F.: Scheduling Algorithms for Data Redistribution and Load-Balancing on Master-Slave Platforms. Parallel Processing Letters 17(1), 61-77 (2007)
15. Norman, M.G., Pelagatti, S., Thanisch, P.: On the Complexity of Scheduling with Communication Delay and Contention. Parallel Processing Letters 5(3), 331-341 (1995)
16. Papadimitriou, C.B., 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.)
17. Rayward-Smith, V.J.: Scheduling with Unit Interprocessor Communication Delays. Discrete Math. 18, 55-71 (1987)
18. Sarkar, V.: Partitioning and Scheduling Parallel Programs for Execution on Multiprocessors. MIT Press (1989)
19. Sinnen, O., Sousa, L.: Communication Contention in Task Scheduling. IEEE Transactions on Parallel and Distributed Systems 16(6), 503-515, (2005)
20. Sinnen, O., To, A., Kaur, M.: Contention-Aware Scheduling with Task Duplication. Journal of Parallel and Distributed Computing 71(1), 77-86 (2011)
21. Tam, A., Wang, C.L.: Contention-Aware Communication Schedule for High Speed Communication. Cluster Computing 6(4), 339-353 (2003)