

Scheduling with communication delays and data routing in Message Passing Architectures

Aziz MOUKRIM and Alain QUILLIOT

Université de Clermont II, LIMOS
Complexe scientifique des cézeaux
63177 Aubière Cedex
moukrim@ucfma.univ-bpclermont.fr

Abstract. This work deals with the scheduling problem of a directed acyclic graph with interprocessor communication delays. The objective is to minimize the makespan, taking into account the contention in the network induced by the message routing. We propose two heuristics for solving the scheduling and routing problems onto arbitrary networks, taking into consideration the access conflicts to links during the task scheduling. Both heuristics significantly improve the performance of the algorithms which do not consider the contention in the network. The comparison of these heuristics is done on problems with different granularity levels in regard to execution times and number of needed processors.

1 Introduction

With the development of new architectures based on the message passing principle, the interest of scheduling problems with interprocessor communication delays is rapidly increasing. Given a Directed Acyclic weighted Graph (DAG) whose nodes are tasks, the scheduling problem consists in allocating the tasks to processors in such a way that some precedence constraints among the tasks are verified and the makespan is minimized. A precedence constraint from task T_i to task T_j means that T_j needs data from T_i before being started. Moreover, if these two tasks are not assigned to the same processor, a delay must be considered between the completion of T_i and the beginning of T_j to transfer the data. Several cases of this general scheduling problem have been proved to be NP-complete [1][11][2][14] even on an unlimited number of completely connected processors and without taking into account the contention in the network.

However, approximation algorithms have been studied for this problem. They can be separated in two classes: multi-step and one-step methods. The multi-step methods proceed first to a clustering step under the assumption that there is an unlimited number of completely connected processors, and then in the following steps, the clusters are scheduled on the available processors without considering the network contention [7][13][4][5]. The one-step methods schedule tasks directly on the available processors [12][6][3][15]. The ETF heuristic [6] takes the processor distance into account. The MH heuristic considers the communication

volume on the links at the assignment time of tasks and allows the task duplication. Hypertool is developed for Hypercubes and uses the MCP heuristic [15]. It considers neither processor distance nor access conflicts to physical links.

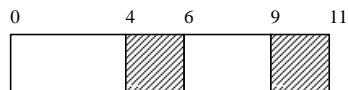
In this paper, we first describe our model and introduce the transfer task notion. Then, we show the limits of approximation algorithms which do not take into account the contention in the network. We compare two heuristics for solving the scheduling problem on a given physical architecture, whose principle is the management of the contention induced by the message routing during the assignment process. They are based on a topologic list. The first one uses an adaptation of Dijkstra algorithm to the shortest path problem in a graph of which arc lengths depend on time. The second one is based on a path classical search by the A^* algorithm. We give an evaluation of these heuristics compared with an extension of the MCP heuristic used in Hypertool system followed by a routing procedure. The performance of these heuristics on grid and tree networks is studied. The considered test problems are obtained on DAGs generated randomly with different granularity levels.

2 Model and assumptions

We consider the *task system* $(\mathcal{T}, \mathcal{E}, p, c)$ where $\mathcal{T} = \{T_1, T_2, \dots, T_n\}$ is the set of non-preemptive *tasks* which constitutes the program, \mathcal{E} represents the set of communication arcs such that $(\mathcal{T}, \mathcal{E})$ is a directed acyclic graph and p_i or $p(T_i)$ is the *execution time* of task T_i . A precedence constraint between T_i and T_j is expressed by an arc $[T_i, T_j]$ with c_{ij} or $c(T_i, T_j)$ its length in \mathcal{E} . It corresponds to the communication cost between T_i and T_j if they are executed on different processors directly connected. We consider the processor network $(\mathcal{P}, \mathcal{F})$ where $\mathcal{P} = \{\pi_1, \pi_2, \dots, \pi_m\}$. An arc $[\pi_i, \pi_j]$ belongs to \mathcal{F} if and only if a direct physical link from processor π_i to processor π_j exists. The makespan of a schedule is denoted by w or C_{max} and the set of predecessor tasks of a task T is denoted by $\Gamma^-(T)$.

If two communicating tasks are assigned to different neighbor processors, we introduce a fictitious task with a duration equal to the necessary transfer time between these two tasks: it is defined as the *transfer task*. We suppose that transfer tasks are preemptive. The global availability of a link $l = [\pi_1, \pi_2]$ between two processors π_1 and π_2 is given by the list of its availability time intervals.

Let us consider a link $l = [\pi_1, \pi_2] : ((0, 4), (6, 9), (11, \infty))$



ensuring the transfers from a processor π_1 to another processor π_2 and a task T_i which has to send data to another task T_j over link l . Let us suppose that $c_{ij} = 4$ and the data transmitted by T_i are available on processor π_1 from time $t = 1$ onwards. The programming of the transfer task on link l is made with preemption and the link l becomes $((0, 1), (7, 9), (11, \infty))$

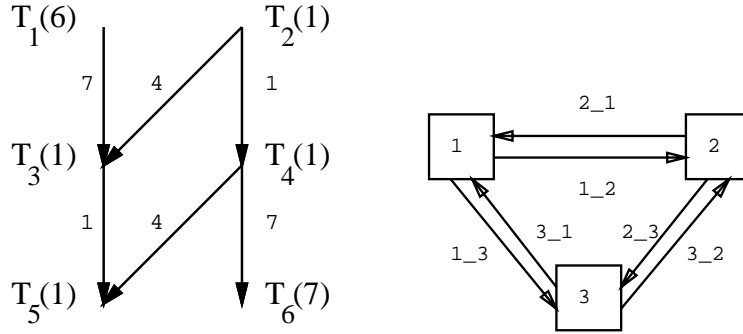
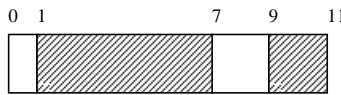


Fig. 1. A task system and a network example.



and the data transmitted by T_i are available on processor π_2 from time $t = 7$ onwards.

3 Approximation algorithms without routing during the task assignment

In this section, we present two approximation algorithms: a multi-step method (DSC) and a one-step method (MCP). We show their limits through the example in Figure 1. That is due to the fact that they do not take into account the actual physical configuration of the network during the task assignment.

3.1 The Modified Critical Path heuristic

The MCP heuristic is proposed by Wu [15] in order to avoid the systematic assignment of the available task of top priority to the first free processor in the list scheduling algorithms. In the MCP heuristic, this task is scheduled to the processor that allows its earliest execution. MCP heuristic can be outlined as follows:

- create a priority list where tasks are classified according to decreasing level: the level of a task is the length of the longest path between this task and a task without successors. The path length is the sum of execution times and communication costs appearing in the path.
- while it remains an unscheduled task, proceed to the assignment of the available task with the highest priority to the processor that allows its earliest execution taking into account the communication costs with its predecessors without considering the network contention.

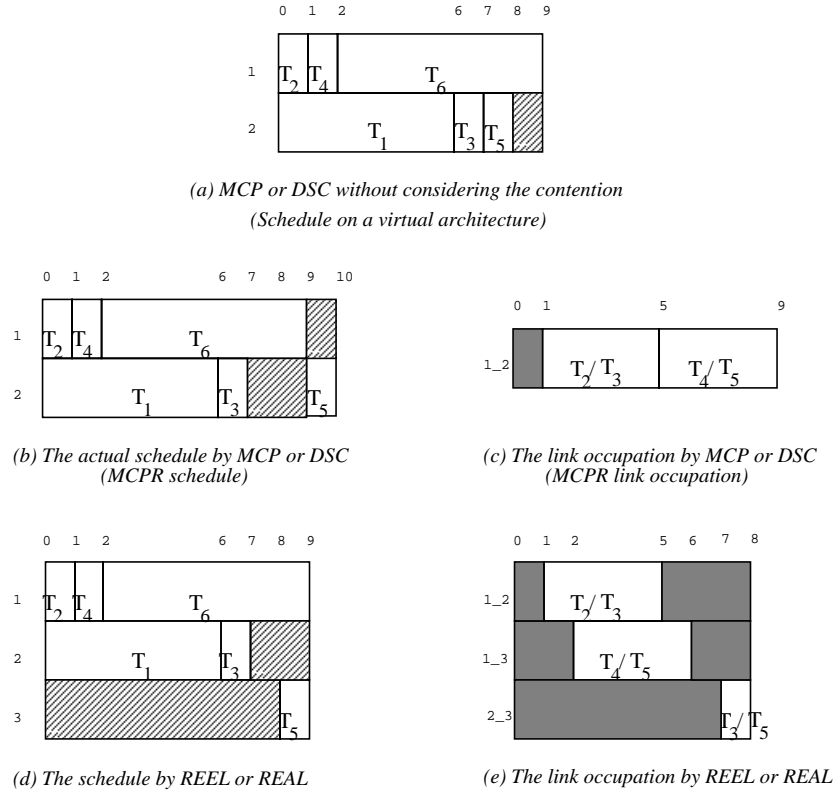


Fig. 2. Different schedules depending on whether contention is taking into account or not.

We consider the MCP heuristic coupled with an assignment strategy using processor distance in the evaluation of the effective communication costs between tasks. The interest of this procedure is shown in [9].

Let us consider the task system and the network in Figure 1 where the number in parentheses after a task identifier corresponds to the execution time of this task. The level of T_5 is 1 and the level of T_6 is 7. The level of T_4 is 15. It is equal to the longest path between T_4 and a task without successors ($\max(p_4 + c_{45} + p_5, p_4 + c_{46} + p_6)$). Continuing in a similar manner, we get the priority list $L = (T_2, T_1, T_4, T_6, T_3, T_5)$.

The MCP heuristic, with or without processor distance strategy, performs as follows. At time 0, T_2 is scheduled in π_1 first, and in the next step T_1 is scheduled in π_2 . At time 1, task T_4 is available and it is scheduled in π_1 because it can start executing at time 1. If it was scheduled in π_2 , it could not start before time 6, and in π_3 , it would be executed at time 2 because it needs to wait 1 unit time to receive the data from π_1 . At time 2, the only available task is T_6 . It is scheduled in π_1 to start executing immediately after T_4 at time 2, and so

on. The obtained schedule by the MCP heuristic for this example is shown in Figure 2.a and makespan $w = 9$.

3.2 The Dominant Sequence Clustering

It has been shown in [4] that DSC is a good clustering algorithm among the multi-step scheduling methods. It is used to schedule task graphs in Pyrrhos software tool which works in two steps.

- Determine a clustering of the task graph on an unbounded number of processors of completely connected architecture. Let u be the number of the obtained clusters.
- Merging and scheduling the u clusters into the m available processors of the physical architecture if $u > m$.

Clustering is deciding which tasks are allocated to the same processor. If both ends of an arc $[T_i, T_j]$ belong to the same cluster, communication cost of $[T_i, T_j]$ is zeroed. By determining a task ordering in each cluster, we obtain a graph which is called a scheduled DAG. The parallel time (PT) of a clustering is given by the longest path in the scheduled graph. This path is called the Dominant Sequence (DS). The DSC algorithm is outlined below.

- Initially each task is a cluster.
- Compute the initial DS . Mark all arcs unexamined.
- While there is an arc unexamined DO
 - Zero an arc in DS if PT does not increase.
 - Mark this arc examined.
 - Find a new DS .
- Endwhile.

The initial clustering in the example of Figure 1 is $C_1 = \{T_1\}, C_2 = \{T_2\}, C_3 = \{T_3\}, C_4 = \{T_4\}, C_5 = \{T_5\}, C_6 = \{T_6\}$. The corresponding parallel time is 17 and $DS = [T_2, T_4, T_6]$. The arc $[T_4, T_6]$ is zeroed and the parallel time is reduced to 16 with the new $DS=[T_1, T_3, T_5]$. Next, the arc $[T_1, T_3]$ is zeroed and the new $DS=[T_2, T_4, T_6]$ with $PT = 10$. Then, $[T_2, T_4]$ is zeroed. Continuing in a similar manner, we obtain two clusters $C'_1 = \{T_1, T_3, T_5\}$ and $C'_2 = \{T_2, T_4, T_6\}$. Finally, $[T_2, T_3]$ and $[T_4, T_5]$ cannot be zeroed. Otherwise, the parallel time will increase. Thus, the resulting clustering is $C'_1 = \{T_1, T_3, T_5\}$ and $C'_2 = \{T_2, T_4, T_6\}$.

Both MCP and DSC heuristics give a schedule whose makespan is $w = 9$. They use only two processors in the network. Unfortunately, during the effective execution, the link $[\pi_1, \pi_2]$ has to ensure some data transfer from task T_2 to T_3 and from T_4 to T_5 . Since a communication channel cannot execute more than one transfer task at a time, the data which are sent by T_4 are available on π_2 only at time 9 (cf. Figure 2.c). The makespan of the schedule given by MCP or DSC heuristic is in fact $w = 10$ (cf. Figure 2.b). However, there is a feasible schedule with makespan $w = 9$ as it is shown in Figure 2.d. and Figure 2.e.

3.3 The Modified Critical Path heuristic followed by Routing

We have seen that MCP and DSC do not give a feasible schedule on the actual network. In this section, we propose a routing procedure to perform a feasible schedule from the one proposed by MCP. This procedure is used to determine the start execution time of tasks on the actual architecture, taking into consideration the data transmission and the access conflicts they generate.

The starting time of some task T is denoted as $s(T)$; the finishing time is $f(T)$; the processor to which T is assigned is $\pi(T)$. Let us consider a task T and a predecessor task of T (T'). The data transferred between T and T' if $\pi(T) \neq \pi(T')$ needs the choice of a routing between these two processors and the execution of a transfer task on each link of this routing.

For a given processor π , we compute the availability time, denoted by $at(T', \pi)$, of data transmitted between T' and T when using the best path to transmit data from $\pi(T')$ to π , taking into account the occupation state of the links. We use an adaptation of Dijkstra algorithm [8] to the shortest path problem in a graph whose arc lengths depend on time.

Keeping from the schedule produced by the MCP heuristic the execution order of tasks on each processor, we use Dijkstra algorithm to determine the start execution time of tasks, taking into consideration the data transmission and the access conflicts they generate. This adaptation of MCP is called MCPR like Modified Critical Path followed by Routing. We consider the tasks by decreasing level and we denote by

- σ , the current planning. That is the occupation state of the communication channel in the network by the transfer tasks between tasks whose starting time has been determined.
- $MC(\pi)$, the current moment of the processor π . That is the availability time of processor π .

Then, the start execution time of T in MCPR is given by

$$s(T) := \max\{MC(\pi(T)), \max_{T' \in \Gamma^{-1}(T)} at(T', \pi(T))\}.$$

After the computation of each availability time $at(T', \pi(T))$, we update current planning σ by the programming of transfer tasks on the links of the best path.

The MCPR schedule of the task system described in Figure 1 is given in Figure 2.b. and Figure 2.c. Its makespan is $w = 10$.

In the next section, we propose a heuristic for solving simultaneously the scheduling and routing problems onto arbitrary networks, taking into account the access conflicts to links during the task scheduling.

4 The PRS heuristic

We propose a heuristic where the assignment of any task T is coupled with the resolution of the routing problem it generates in the case of data transmission.

Algorithm PRS;

First, determine list L of tasks classified according to decreasing level. The current planning σ is empty (no task assigned);

while L is not empty **do**

begin

 Choose the available task T of the top priority; $L := L - \{T\}$;

for each $\pi \in \mathcal{P}$ **do**

 {Here, for each processor π , the planning σ is in the same state before the beginning of the assignment of task T }.

begin

for each $T' \in \Gamma^{-1}(T)$ **do**

begin

 Determine the availability time $at(T', \pi)$ when using the best path $[\pi(T').. \pi]$ to transmit data from $\pi(T')$ to π ;

 Temporary programming of transfer tasks between T' and T on the path $[\pi(T').. \pi]$;

end

$st(T, \pi) := \max\{MC(\pi), \max_{T' \in \Gamma^{-1}(T)} at(T', \pi)\}$;

end

$\pi(T) := \operatorname{argmin}\{st(T, \pi), \pi \in \mathcal{P}\}$;

$s(T) := st(T, \pi(T))$; $f(T) := s(T) + p(T)$; $MC(\pi(T)) := s(T) + p(T)$;

 Plan T on $\pi(T)$ and update σ ;

end-while

Algorithm 1: Simultaneous scheduling and routing algorithm.

This is the PRS heuristic (Algorithm 1) under the french acronym PRS like Placement-Routage-Simultanés. We proceed as follows: for each processor π , we compute the start time of T on π , denoted $st(T, \pi)$, taking into consideration the data transmitted by the predecessor tasks of T . Let $\Gamma^{-1}(T) = \{D_1, \dots, D_k\}$ be the set of predecessor tasks of T . They are classified according to decreasing level which is defined as in MCP. We use a routing algorithm to compute the availability time, denoted by $at(D_1, \pi)$, of data transmitted between D_1 and T on π . After the computation of $at(D_j, \pi)$ ($1 \leq j \leq k-1$), a temporary programming of transfer tasks between T and its predecessors D_l ($1 \leq l \leq j$) is made. Then $at(D_{j+1}, \pi)$ is computed as the availability time of data transmitted between D_{j+1} and T on π , taking into account the current planning σ and the above temporary programming. Before performing the same processing for another processor π' , the temporary programming of transfer tasks between T and its predecessors is canceled. The processor ensuring the best possible start time of T is kept. We have proposed two versions of The PRS heuristic [9] by adopting two strategies for routing algorithm:

- The REEL heuristic under the french acronym REEL like Routage-Etendu-Exact-Localément where the routing algorithm is the adaptation of Dijkstra algorithm to the shortest path problem in a graph of which arc lengths depend on time.

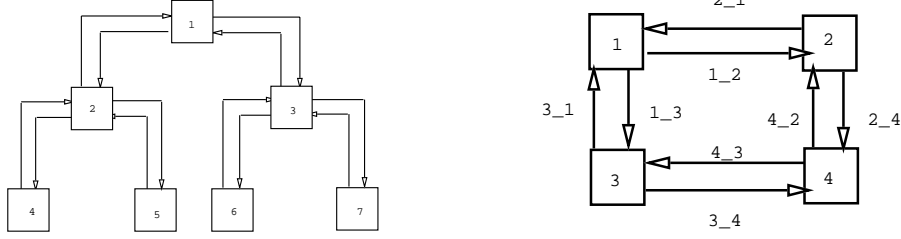


Fig. 3. tree and grid networks

- The REAL heuristic under the french acronym REAL like Routage Etendu Approché Localement where the routing is based on A^* algorithm [10]. Let us remind that we have to determine the best routing for data with length $c(T', T)$ from task T' scheduled on $\pi(T')$ to a processor π . Let π_c be the current processor and $U(\pi_c)$ the finish execution time of the transfer task to π_c of the data. It is calculated as follows: If $\pi_c = \pi(T')$, then $U(\pi_c) = f(T')$. Otherwise, $U(\pi_c)$ is computed step by step, following the generated path $[\pi(T'), \dots, \pi_c]$ given that time $U(\pi(T'))$ is already determined and the duration of a transfer task between two adjacent processors is computed as explained in section 2. Then, the evaluation of processor π_c , denoted by $Val(\pi_c)$, is given by

$$Val(\pi_c) := U(\pi_c) + c(T', T) \times distance(\pi_c, \pi)$$

where $distance(\pi_i, \pi_j)$ designates the length of the shortest path between π_i and π_j in $(\mathcal{P}, \mathcal{F})$ if the length of any arc in \mathcal{F} is 1.

One availability time of data corresponds to each involved path between $\pi(T')$ and π . We denote the best current solution Val^* . It is bounded as follows: $Val_{inf} \leq Val^* \leq Val_{sup}$ where $Val_{inf} = f(T') + c(T', T) \times distance(\pi(T'), \pi)$ (no contention on the links) and Val_{sup} is the availability time if a shortest path between $\pi(T')$ and π in $(\mathcal{P}, \mathcal{F})$ is used, taking into account contention on links.

Let us consider the task system and the network in Figure 1. The priority list is the same as in MCP: $L = (T_2, T_1, T_4, T_6, T_3, T_5)$. Since at the beginning there is no access conflicts to links, the REEL and REAL heuristics perform as MCP: Tasks T_2, T_4, T_6 are scheduled on π_1 , and T_1, T_3 on π_2 . Then task T_5 is considered:

If $\pi(T_5) = \pi_1$, then $s(T_5) = 9$: T_5 is scheduled after T_6 and data of length $c_{35} = 1$ are transmitted on $[\pi_2, \pi_1]$ from $t = 7$ to $t = 8$.

If $\pi(T_5) = \pi_2$, then $s(T_5) = 9$: Since the link $[\pi_1, \pi_2]$ is available only at time 5, data transmitted between T_4 and T_5 are available on π_2 from time $t = 9$ ($5 + 4$).

If data are transmitted from π_1 to π_2 via processor π_3 , they will be available on π_2 from time $t = 10$ ($2 + 4 + 4$).

If $\pi(T_5) = \pi_3$, then $s(T_5) = 8$: Data transmitted from T_4 are transmitted on link $[\pi_1, \pi_3]$ and are available on π_3 from time $t = 6$ ($2 + 4$), and data transmitted from T_3 are sent on link $[\pi_2, \pi_3]$ and are available from time $t = 8$ ($7 + 1$).

Consequently, in the REEL or REAL schedule task T_5 is scheduled on π_3 and the makespan is 9. They out perform DSC and MCP which give a schedule of length 10 after the computation of a feasible schedule by MCPR.

Network	n	Coarse grain				Fine grain			
		D		G		D		G	
		REEL	REAL	REEL	REAL	REEL	REAL	REEL	REAL
Grid 4	10	0.00	0.00	0.00	0.00	0.33	0.33	0.48	0.48
	30	0.78	0.78	1.11	1.13	20.67	23.00	8.72	9.67
	50	0.44	0.44	0.36	0.36	25.44	29.22	7.38	8.19
	70	0.22	0.22	0.15	0.15	45.44	45.56	13.45	14.18
	90	0.89	1.00	0.43	0.47	54.56	52.00	11.38	9.96
	100	4.89	6.00	1.78	2.18	44.33	49.56	7.70	8.30
	150	5.11	6.33	1.30	1.61	97.33	113.11	11.08	12.20
	200	1.00	1.00	0.21	0.21	60.56	62.11	7.64	7.83
	average		1.67	1.97	0.67	0.76	43.58	46.86	8.48
Grid 9	10	0.00	0.00	0.00	0.00	0.44	0.44	0.54	0.54
	30	3.22	3.22	5.23	5.37	20.67	21.56	11.13	11.54
	50	1.22	1.22	1.04	1.04	22.67	23.22	5.53	5.65
	70	2.56	2.67	2.36	2.49	32.67	30.33	10.60	10.00
	90	2.44	3.00	1.25	1.51	39.44	33.33	7.69	6.76
	100	2.11	2.11	1.11	1.11	41.89	45.00	6.48	6.92
	150	2.56	4.22	0.41	0.89	58.33	60.22	7.50	7.82
	200	2.33	2.33	0.92	0.94	63.44	59.22	9.88	9.22
	average		2.06	2.35	1.54	1.67	34.94	34.17	7.42
Grid 16	10	0.00	0.00	0.00	0.00	1.11	1.11	1.84	1.84
	30	1.33	1.78	2.91	3.73	13.22	12.44	6.70	6.35
	50	0.89	0.89	0.76	0.76	24.22	25.00	7.53	7.78
	70	3.89	3.11	4.15	3.50	31.00	33.33	8.81	9.29
	90	6.67	6.78	3.53	3.62	29.44	31.33	6.49	6.81
	100	4.11	4.22	2.04	2.07	39.22	40.44	7.80	8.03
	150	10.22	9.33	3.99	3.53	70.00	74.89	7.27	7.67
	200	4.56	4.33	1.54	1.47	54.89	56.11	6.98	7.42
	average		3.96	3.81	2.37	2.33	32.89	34.33	6.68
general average		2.56	2.71	1.52	1.59	37.14	38.45	7.53	7.68

Table 1. Comparison between REEL and REAL on DAGs with different granularity levels.

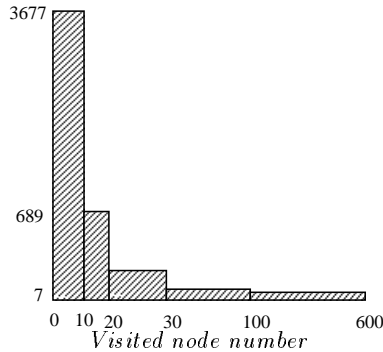


Fig. 4. The distribution of the visited node number for a DAG during the A^* algorithm calls.

5 Comparison between MCPR, REEL and REAL

We consider some benchmarks on grid and tree networks (cf. Figure 3) in order to measure the performances of REEL and REAL heuristics. As for DAGs, we use a random generator described in [9], we generate two kinds of graphs (\mathcal{T}_1 et \mathcal{T}_2) according to the graph granularity; $\mathcal{T}_1: p_i \in [5, 10]$ and $c_{ij} \in [0, 5]$ and $\mathcal{T}_2: p_i \in [5, 10]$ and $c_{ij} \in [10, 15]$. The number of tasks n belongs to $\{10, 30, 50, 70, 90, 100, 150, 200\}$. Numerical experimentations of REEL and REAL on grid networks are presented in Table 1. The results are given in terms of the task number of the DAG and the processor number. For each value of n and m , nine examples are tested.

We make a comparison, of each heuristic h (REEL and REAL), with regard to MCPR by calculating:

- the absolute gain $D: C_{max}(MCPR) - C_{max}(h)$,
- the relative gain $G: 100 * (C_{max}(MCPR) - C_{max}(h)) / C_{max}(MCPR)$.

Independently of the task and processor numbers, we note an improvement due to the REEL (resp. REAL) heuristic compared with the MCPR heuristic of 1,52% (resp. 1.59%) (general average of the relative gain) for coarse grain and 7.53% (resp. 7.68%) for fine grain. These good results were foreseeable especially for fine grain applications. Indeed, the more important the communication flow is, the more important the access conflicts to links are; and taking these conflicts into account during the assignment process improves the global schedule. Moreover, the improvement given by REAL is more or less the same than that one performed by REEL.

5.1 Complexity and execution times

It has been shown in [9] that the complexity of REEL is $O(m^2 n^2 \log n)$. It is comparable to the complexities of Sarkar system [13] and TaskGrapher system [3] which are respectively $O(mn^3)$ and $O(m^3 n^2)$.

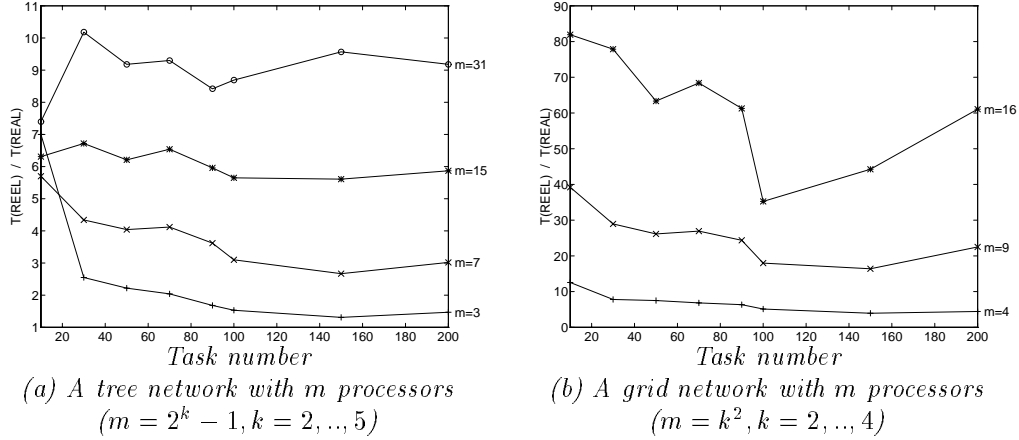


Fig. 5. Comparison of execution times of the REEL and REAL heuristics in terms of task number.

After several tests, we have noticed that the obtained improvements by the REAL heuristic are not significant beyond a certain number of visited nodes in A^* algorithm. This is due to the depth-first paradigm which permits to obtain feasible solutions rapidly. Therefore, we stop the routing procedure after a certain number of visited nodes. This number is chosen equal to m^2 . We have noticed that the number of visited nodes depends closely on the processor configuration. For a tree network, this number does not exceed ten. The histogram (cf. Figure 4) illustrates the distribution of the visited nodes number during the A^* algorithm calls for a DAG with 150 tasks on a grid network of 25 processors. This example is chosen among those with a maximum number of generated nodes. For a task system scheduled on a square grid network with m processors, the number of A^* algorithm calls where the nodes number exceeds m is insignificant (cf. Figure 4).

Figure 5.a (resp. Figure 5.b) represents the ratio of the execution time of REEL, denoted by $T(REEL)$, over the execution time of REAL, denoted by $T(REAL)$, in terms of task number for a tree network (resp. grid network) with m processors such that

$$m = 2^k - 1, k = 2, \dots, 5, \text{ for a tree network.}$$

$$m = k^2, k = 2, \dots, 4, \text{ for a grid network.}$$

When the number of processors increases, the REAL heuristic is still faster than REEL. We also remark that the more the network topology is complicated with cycles, the more important the ratio, $T(REEL)/T(REAL)$, is. This ratio reaches 10.18 for a tree network and 81.93 for a grid network. Therefore, the REAL heuristic is more practical than the REEL heuristic with regard to the execution times.

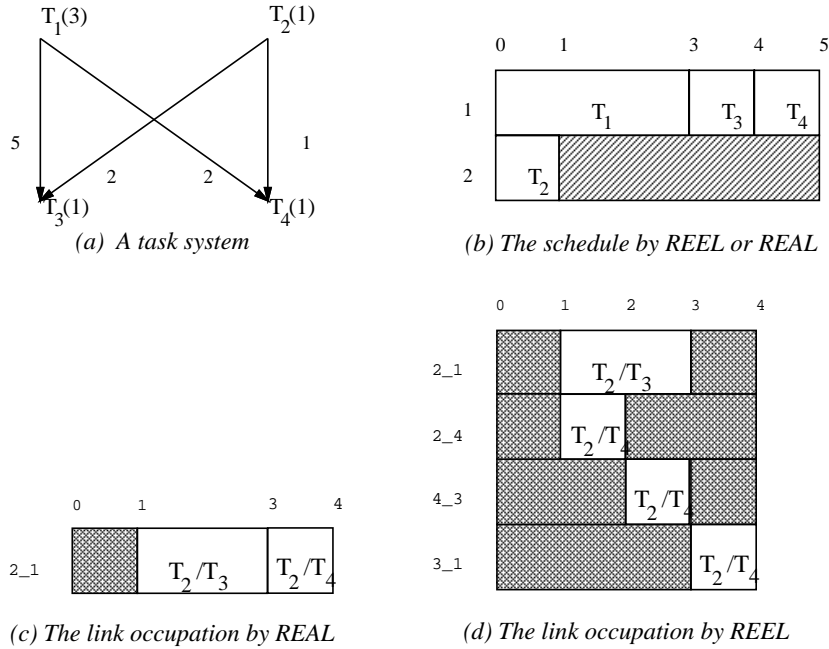


Fig. 6. An example of the number of processors required by REEL and REAL

5.2 The number of processors required

The REEL and REAL heuristics give as an output the number of processors required to execute the different tasks and perform the necessary routing of all data. The REEL heuristic is based on an adaptation of Dijkstra algorithm to the shortest path problem for determining a data routing. In the case there would be several possible paths which would allow the same availability time for data transmitted between two processors, Dijkstra algorithm determines an arbitrary one. However, the REAL heuristic permits to make up a path around one of the shortest paths between the two processors in $(\mathcal{P}, \mathcal{F})$ if the length of any arc in \mathcal{F} is 1, thanks to the order of processor visit: if π is the target processor and π_c is the current processor, the next visited processor π'_c is given by $\pi'_c = \text{Argmin}\{\text{distance}(\lambda, \pi); [\pi_c, \lambda] \in \mathcal{F} \text{ and } \lambda \text{ not visited yet}\}$. Thus, the number of processors required to ensure the routing is minimized. We denote by $P(\text{REEL})$ (resp. $P(\text{REAL})$) the number of processors required by the REEL (resp. REAL) heuristic.

Figure 6.b presents the schedule given by the REEL heuristic as the REAL heuristic for the task system described in Figure 6.a on a square grid network with 4 processors. Figure 6.c (resp. Figure 6.d) presents the occupation times of the network links for REAL (resp. REEL). So, for this example, $P(\text{REEL}) = 4$ whereas $P(\text{REAL}) = 2$.

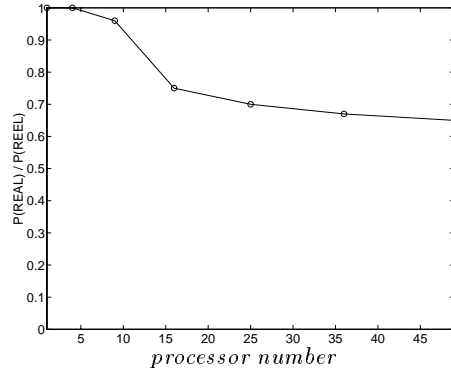


Fig. 7. The average ratio of the number of processors required by REAL over the number of processors required by REEL on grid networks with m processors ($m = k^2, k = 1, \dots, 7$)

Figure 7 shows the average ratio ρ of $P(REAL)$ over $P(REEL)$. It is less or equal to one. When the number of processors is small, the two heuristics use almost all the processors of the network and the number of processors they need is nearly the same. When the number of available processors increases, the number of processors required by REAL becomes less than m whereas the number of processors required by REEL remains often equal to m ; but then $P(REEL)$ and $P(REAL)$ stop increasing, since m approaches the width of the precedence graph of the task system and ρ goes to a limit value. Therefore, the REAL heuristic is better than the REEL heuristic with regard to the number of processors required.

6 Conclusion

In this paper, we have been interested in solving simultaneously the scheduling and routing problems. Significant improvements are obtained compared to the case when the scheduling problem is performed by a heuristic which does not take into account the access conflicts to the communication channels. We have compared two heuristics based on the management of the conflicts induced by message routing during the task assignment by using an adaptation of Dijkstra algorithm to the shortest path problem in a graph of which arc lengths depend on time, REEL, and A^* algorithm, REAL. Both heuristics improve the performance of previously proposed solution. Moreover, REAL out performs REEL with regard to execution times and number of needed processors.

References

1. CHRÉTIENNE Ph., *Task scheduling with interprocessor communication delays*, Eur. J. Op. Res., 57, pp. 348-354, 1992.

2. CHRÉTIENNE Ph., Picouleau C., *Scheduling with communication delays: a survey, Scheduling Theory and its Applications*, P. Chrétienne, E.G. Coffman, J.K. Lenstra, Z. Liu (Eds), John Wiley Ltd 1995.
3. EL-REWINI H., Lewis T.G., *Scheduling Parallel Program Tasks onto Arbitrary Target Machines*. J. of Parallel and Distributed Computing, 9, 1990, pp. 138-153.
4. GERASOULIS A., Yang T., *A Comparison of Clustering Heuristics for Scheduling DAGs on Multiprocessors*, Journal of Parallel and Distributed Computing, Vol. 16, N^o 4, 1992, pp. 276-291.
5. GERASOULIS A. and Yang T., *Efficient Algorithms and a Software Tool for Scheduling Parallel Computation*, Scheduling Theory and its Applications, P. Chrétienne, E.G. Coffman, J.K. Lenstra, Z. Liu (Eds), John Wiley Ltd 1995.
6. HWANG J.J., Chow Y.C., Anger F.D., Lee C.Y., *Scheduling precedence graphs in systems with interprocessor communication times*, SIAM J. Comput., 18(2), pp. 244-257, 1989.
7. KIM S.J. and Browne J.C., *A General Approach to Mapping of Parallel Computation upon Multiprocessor Architectures*. Proc. of Inter. Conf. on Parallel Processing, Vol. 3, 1988, pp. 1-8.
8. MINOUX M., *Structures Algébriques généralisées des problèmes de cheminement dans les graphes. Théorèmes, algorithmes et applications*, R.A.I.R.O. Rech. Op., vol. 10, N^o 6, 1976, pp. 33-62.
9. MOUKRIM A., *Génération automatique de codes parallèles et nouvelles heuristiques d'Ordonnancement pour les machines à passage de messages*, Thèse d'université, Université Blaise Pascal, Clermont-Ferrand, 1995.
10. NILSSON N.J., *Principles of artificial intelligence*, Palo Alto, CA: Tioga, 1980.
11. PAPADIMITRIOU C. and Yannakakis M., *Towards an Architecture-Independent Analysis of Parallel Algorithms*, SIAM J. Comput., 19, (1990), pp. 322-328.
12. RAYWARD-SMITH V.J., *UET scheduling with interprocessor communication delays*, Internal Report SYS-C86-06, School of Information Systems, University of East Anglia, Norwich, United Kingdom, 1986.
13. SARKAR V., *Partitioning and Scheduling Parallel Programs for Execution on Multiprocessors*, The MIT Press, 1989.
14. VELTMAN B., Lageweg B.J., Lenstra J.K., *Multiprocessor scheduling with communication delays*, Parallel Computing 16, 1990, pp. 173-182.
15. WU M.Y., Gajski D., *A programming aid for Hypercube architectures*, The journal of Supercomputing. Vol 2, 1988, pp. 349-372.