RAIRO-Oper. Res. 49 (2015) 369–381
DOI: 10.1051/ro/2014048

RAIRO Operations Research
www.rairo-ro.org

SCHEDULING 2-DIMENSIONAL GRIDS WITH LARGE COMMUNICATION DELAYS *

E.M. Daoudi[1], D. Trystram[2,3] and F. Wagner[2]

Abstract. Most parallel and distributed platforms available today show a large unbalance between slow communications and fast local computations which makes the scheduling of tasks much harder to handle efficiently. The so called *scheduling with large communication delays* problem has been proved to be NP-hard even in some restricted cases. Its status regarding approximation is still unknown. In this work, we propose to study this challenging problem for graphs with a specific structure of 2-dimensional grids. More precisely, we provide lower and upper bounds for both sub-problems of unbounded number of processors and two processors. We show in both cases that the proposed scheduling algorithms are close to lower bounds.

Keywords. Parallel processing, scheduling, large communication delays.

Mathematics Subject Classification. 90B35.

1. Introduction

1.1. Context and motivation

The large scale parallel and distributed platforms available today are characterized by several new features that highly influence the way resources should be allocated. In particular, the communications between remote computing units are

Received 2 October, 2012. Accepted 26 September, 2014.

* *Supported by the Mohammed I university in the frame of the PGR program*

[1] University Mohammed I, Faculty of Sciences, LaRI laboratory, Oujda, Morocco. m.daoudi@fso.ump.ma

[2] Grenoble Institute of Technology, France. trystram@imag.fr; Frederic.Wagner@imag.fr

[3] Institut Universitaire de France

Article published by EDP Sciences

very costly compared to the computations of elementary operations. Hence, one crucial point is to deeply understand the impact of such large communications on the performance of parallel algorithms. Let notice that this is a rather old problem which focused regularly researches over the last decades (in the eighties when the machines evolved from shared-memory to distributed memories, in the nineties with the development of local clusters built with desk components linked by local interconnection networks, around the change of the millennium with the apparition of the computational grid paradigm and today this the target of very large scale exa-scale systems). The problem of efficiently handling large communications has still no satisfiable solution today and it becomes more and more important with the increase of the number of nodes.

In the classical scheduling theory, a parallel program is usually represented by a directed acyclic graph. The program is partitioned into elementary tasks whose vertices are the instructions and the edges correspond to data movements of intermediate results [6]. Two models have been proposed for handling large communications [5]. First, the parallel tasks model [10] and all its variants (rigid, moldable, malleable tasks) [24] which corresponds to an artificial increase of the task granularity in order to reduce the unbalance. Here, the communications inside the parallel tasks are implicitly hidden and thus, neglected. In the same category, bottom-up approaches have been developed for clustering the tasks according to some local criterion related to communications [17, 23]. The second model is an extension of the delay model introduced in [21] where the communications between tasks allocated in different processors are explicit and thus, can be optimized. This delay is a function of amount of data to be transferred and network topology. The start-up time of the communication is commonly dominant with regard to the size of the messages.

The purpose of this paper is to study the scheduling problem with large communication within the delay model for the specific structure of 2-dimensional grid. We target the minimization of the maximum completion time (makespan). This graph corresponds to a broad class of applications including linear algebra and matrix computations or image processing [14]. It is an important theoretical subject since it may help in better understanding the frontier between polynomial and hard problems. Moreover, the novel analysis of lower bounds provided here should be useful for a broader class of task graphs for instance to guide exact exploration algorithms [22]. Let us start with a review of the most important related works.

1.2. Related works

The problem of determining efficient scheduling algorithms with communication delays has been initiated in the mid eighties with the emergence of distributed-memory parallel architectures. Since then, a lot of studies have been devoted to the complexity or analysis of scheduling algorithms for graphs with specific structures. Most of these works focused on minimization of the makespan which corresponds to the maximum completion time of an application. When the communication times

are small in regard to computations, the problem has more or less been solved: several variants have been shown to be polynomial and those which are hard have solutions with good approximation ratios. Some theoretical results lead to practical implementations into compilers [23]. The problem with large communication is much harder and has no satisfiable solution today.

The model of scheduling with communication delays has been introduced in [21]. The complexity was also studied in this paper and Rayward-Smith showed that the problem is NP-hard for an arbitrary number of processors and unit processing times and unit communication delays (this problem is denoted by $P|prec, p_j = 1, c_{ij} = 1|C_{\max}$ [6]). Later, this problem restricted to trees has also been proved to be NP-hard by Lenstra *et al.* [15]. Assuming an unbounded number of processors and any constant communication delays, we know from the study of Jakoby and Reischuk on large delays that the problem $P_\infty|prec, p_j = 1, c_{ij} = c > 1|C_{\max}$ is NP-hard [13]. It remains NP-hard for graphs structured in trees. For the restricted problem with two processors, Afrati *et al.* proved that the problem is NP-hard in the strong sense even when restricted to the class of binary trees [1]. In contrast, they provide a polynomial-time algorithm for complete binary trees.

For large communication delays, the question of the existence of a polynomial-time approximation algorithm for both cases of unbounded and bounded number of processors is still open. Giannakos *et al.* proved that there is no hope to find an approximation better than $1 + \frac{1}{c+3}$ for infinite number of processors [4] unless $\mathcal{P} = \mathcal{NP}$. The best known algorithms for any structure of graphs have an approximation ratio which depends on c. The best known approximation ratio is $\frac{2(c+1)}{3}$, established by Giroudeau *et al.* in [11]. Since the lower bound is in $\mathcal{O}(1)$ and the upper bound is linear in c, there is still a huge gap to solve the question. We partially fill the gap here for 2-dimensional grids.

In order to be exhaustive in our presentation, let us mention that there is an alternative to improve the previous results by allowing replication of some suitable tasks. With duplication, the problem with an unbounded number of processors remains NP-complete [19]. For an arbitrary number of processors, the best ratio is in $\mathcal{O}(\sqrt{c})$ [16]. However, for some specific structures of precedence, it is possible to obtain constant approximations. For instance, for trees, there exists a 2-approximation algorithm [18] with unbounded number of processors where duplication is allowed. As before, the question of the existence of an approximation ratio that does not depend on c is still open. Moreover, such approaches based on duplication are very costly from the view point of resource consumption.

Notice that many authors studied other methods without theoretical guarantees. Several papers proposed solutions based on meta-heuristics, for instance the recent genetic algorithm proposed in [20]. Another interesting approach is to use greedy list policies like Earliest Task First [12].

1.3. Contributions

In this work, we propose an analysis of the problem of scheduling a 2-dimensional grid graph and show almost-tight results. We target the minimization of the

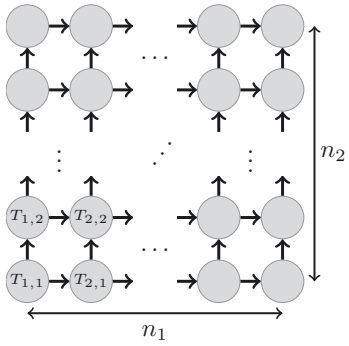

FIGURE 1. Representation of a $2D\text{-}grid(n_1, n_2)$.

makespan (denoted by $C_{\max}$). Our goal within this work is to study the approximation of this problem assuming large communication delays. Using the standard 3-fields notation of scheduling problems [6], we consider the following two instances: the problem with unbounded number of processors $P_\infty|2D\text{-}grid; p_j = 1; c_{ij} = c > 1|C_{\max}$ and the problem restricted to two processors $P_2|2D\text{-}grid; p_j = 1; c_{ij} = c > 1|C_{\max}$.

For the first problem, we provide a lower bound on the optimal time $C^*_{\max}(n) \geq \frac{65}{81}nc$ and then, we propose an algorithm that constructs a scheduling in time of order nc. This analysis is based on a detailed study of the particular case of a square $c \times c$ grid.

For the second problem, we derive a lower bound on the optimal time $C^*_{\max}(n, 2) \geq \frac{n^2}{2} + c + 1$ and then, we show that the algorithm based on the same principle as before is asymptotically optimal. More precisely, the graph is scheduled in time $\frac{n^2}{2} + 3c$.

2. Description of the problem

We consider the scheduling problem with large communication delays of task graphs structured as 2-dimensional grids on a multiprocessor system composed of p identical processors. A 2-dimensional grid of size n_1 by n_2 (denoted in short by *2D-grid*(n_1, n_2)) is defined as the set of tasks $T_{i,j}$, for $1 \leq i \leq n_1$ and $1 \leq j \leq n_2$, which respect the following precedence constrains:

$T_{i,j} \ll T_{i+1,j}$, for all $1 \leq i < n_1 - 1$

$T_{i,j} \ll T_{i,j+1}$ for all $1 \leq j < n_2 - 1$

where the relation $T_{i,j} \ll T_{i',j'}$ means that the execution of task $T_{i,j}$ must be completed before $T_{i',j'}$ starts its execution.

We consider the UET-LCT model (which stands for Unit Execution Time – Large Communication Times) where the tasks have unit execution times and the communication delay (denoted by c) between two distinct processors is a constant greater than the time for executing a task [8].

Furthermore, we assume that the communications between two processors can not be pipelined (classical 1-port half-duplex model) : each processor can at any time receive one (and only one) communication or send one communication but not receive and send simultaneously. It can however overlap communications with computations (start new computations while sending previously computed results for example).

3. Unbounded number of processors

We present in this section one of the main results of this paper: a new lower bound on the optimal completion time for scheduling a 2D-grid(n,n). More precisely, we show that $C^*_{\max} > \left\lfloor \frac{n}{c} \right\rfloor \times \left(\frac{65}{81}c^2 - \frac{8}{27}c - \frac{1}{9}\right)$.

We start with the analysis of an intermediate result by proving a lower bound for the particular case of a square 2D-grid(c,c).

3.1. Scheduling a 2D-grid(c,c)

We consider the restricted case of 2D-grid(c,c) (square grid of size $c \times c$). As c is the communication delay, it seems difficult to obtain a really parallel solution and we are indeed going to show that most computations are processed sequentially by a single processor.

First let us start by bounding the maximal number of communications executed in a sequence by any processor in any optimal solution. For a given solution we denote by $\gamma_{\max}$ the maximal number of communications executed sequentially and $\gamma^*_{\max}$ the maximum of $\gamma_{\max}$ on any optimal solution.

It is clear that the sequential solution runs in time c^2. Since any optimal solution achieves at most this time and each communication takes a time c, no optimal solution contains a sequence of more than c communications and $\gamma^*_{\max} < c$.

In fact, any upper bound on $C^*_{\max}$ can be used to obtain an upper bound on $\gamma^*_{\max}$. In a first step we start by providing a better upper bound on $C^*_{\max}$.

3.1.1. Upper Bound on $C^*_{\max}$ for 2D-grid(c,c)

Theorem 3.1. $\gamma^*_{\max} < \frac{8c}{9} + \frac{2}{3}$.

Proof. The proof is constructive: we build a schedule that partitions the square grid as illustrated on Figure 2 using two processors P_1 and P_2, where x is an integer $\leq c$ to be determined later.

We parallelize the execution of the tasks between both processors P_1 and P_2. In the resulting schedule, the communications do not generate any waiting time for P_1. We proceed as follows: first, P_1 executes the tasks located on the left column during a time equal to $c - x$ (step 1). Then, it immediately starts communicating with P_2 which will be able to process tasks at time $c - x + c$. Then, P_1 continues columns by columns to generate as fast as possible all communications toward P_2 (step 2). After that, it spends some time doing all tasks independent from P_2

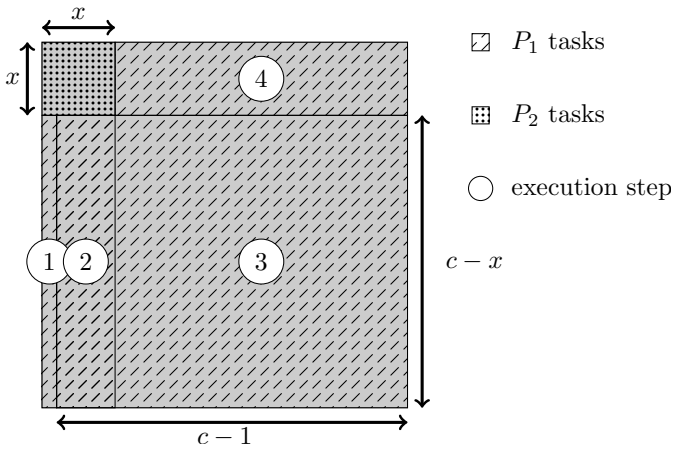

FIGURE 2. Tasks allocation for two processors.

in step 3. During this time, P_2 will process all its tasks and issue all outgoing communications. Finally, P_1 will finish all the remaining tasks in step 4.

P_2 will have completed all incoming communications at time $c - x + c \times x$. It will then process its bottom right task for one unit of time. Finally it executes all outgoing communications taking a time $c \times x$.

In order to avoid waiting times for P_1 all the communications should be overlapped by computations. This means that the time spent for computing tasks in steps 2 and 3 should be at least equal to the time required for communications and thus, $(c-1)(c-x) \geq (1 + 2(c \times x))$.

To minimize $C_{\max}$, x should be maximized. As $x \leq \frac{c^2-c-1}{3c-1}$, we take $x = \left\lfloor \frac{c^2-c-1}{3c-1} \right\rfloor$ and deduce that $x > \frac{c}{3} - 1$.

We are now able to bound the makespan of this schedule: $C_{\max} = c^2 - x^2$ since P_1 never waits. Therefore $C_{\max} < c^2 - (\frac{c}{3} - 1)^2$ and $C^*_{\max} \leq C_{\max} < \frac{8c^2}{9} + \frac{2c}{3}$. We obtain $\gamma^*_{\max} < \frac{8c}{9} + \frac{2}{3}$ as a direct result. □

3.1.2. Lower Bound

First, we start by introducing some additional useful notations. We consider here any schedule with no particular restriction. $\pi(i, j)$ is defined by the function that returns the index of the processor assigned to execute task $T_{i,j}$ within the schedule.

We also introduce a set of *boxes* B_k for the analysis of the schedule. Informally, each box is a rectangular shaped subset of tasks. Let P_t denote the processor executing the top right task of the box. We call P_t the *box processor* (even if all the tasks in the box are not necessarily executed on P_t). We also denote by (i_s, j_s) the coordinates of the bottom left task of the box and (i_e, j_e) the coordinates of the top right task of the box. Figure 3 illustrates the notion of box on a *non-optimal* schedule.

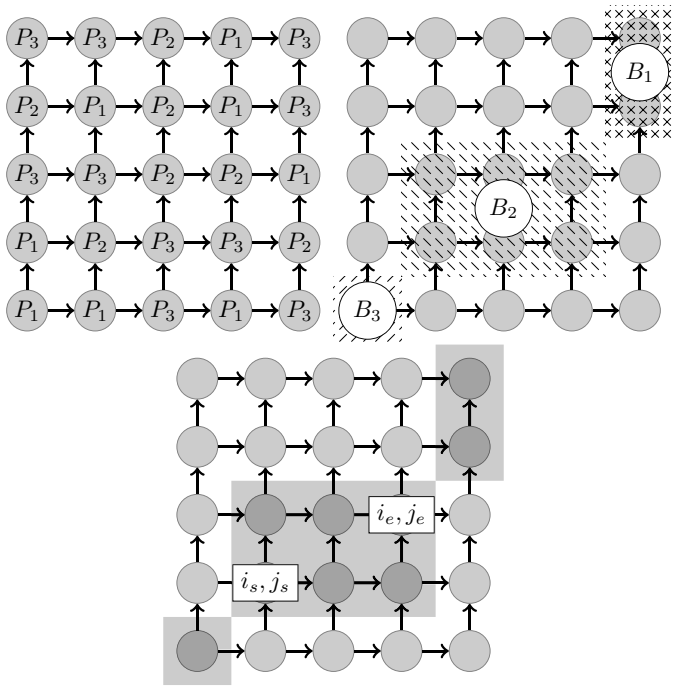

FIGURE 3. Left: **non-optimal** allocation π of tasks to processors. Right: three corresponding boxes. Bottom: top-right and bottom-left tasks in box 2.

Definition 3.2.

A box is a rectangular set of tasks defined by $(i_s, j_s), (i_e, j_e)$ which respects the following constraints :

- $\forall i | i_s \leq i \leq i_e, \exists j | j_s \leq j \leq j_e$ and $\pi(i, j) = t$
- $\forall j | j_s \leq j \leq j_e, \exists i | i_s \leq i \leq i_e$ and $\pi(i, j) = t$
- $\forall i | i_s - 1 \leq i \leq i_e, \pi(i, j_s - 1) \neq t$
- $\forall j | j_s - 1 \leq j \leq j_e, \pi(i_s - 1, j) \neq t$

The set of boxes is built iteratively starting from the first box containing the top right task $T_{n,n}$ and continuing with the box just below and to the left as depicted in Figure 3. The tasks outside B_1 and on its bottom left boundary are all assigned to a different processor than P_1. The top right task in B_2 is then chosen below and to the left of the bottom left task of B_1 and the process iterates building B_2 until all tasks across the bottom left boundary are assigned to a different processor than P_2. This process of building boxes is repeated until reaching a border of the grid.

Finally, we denote by r the number of boxes in the schedule. w_k and h_k denote respectively the width and height of B_k.

Note that since this algorithm is fully deterministic it will yield to a *unique* decomposition into boxes.

Lemma 3.3. *For any box B_k with $k \neq r$, the execution of the top right task in the box will require a sequence of $w_k + h_k$ communications after the end of the predecessor box.*

Proof. The networking model implies that all incoming communications to the box processor are issued sequentially. Therefore, we only need to bound the number of incoming communications for each box. Now, the top right task of the box depends on all other tasks of the box being completed. Moreover, by construction, any task across the bottom left border will generate a communication: for any task under the bottom line, it is not executed in the box processor and there exists a task on the same column which is requiring a communication. For any task on the left, there is for the same reason a task on the same line which requires a communication. Therefore, we need a sequence of $w_k + h_k$ communications. □

Note: We emphasize that the communications are not always taking place with the edges crossing the edge of the box. For example, in Figure 3, task T_{i_e,j_e} of B_2 will require (at least) three communications for the bottom edge of its box. From left to right : T_{i_e-2,j_e-2} to compute T_{i_e-2,j_e-1} (P_1 to P_2); T_{i_e-1,j_e-1} to compute T_{i_e-1,j_e} (P_3 to P_2); T_{i_e,j_e-1} to compute T_{i_e,j_e} (P_3 to P_2). Computing T_{i_e-1,j_e-1} requires no communication from T_{i_e-1,j_e-2} since both tasks are done on P_3.

Lemma 3.4. *The last box contains the first task: $T_{1,1} \in B_r$.*

Proof. The proof is by contradiction. Assume without loss of generality that the box building process reaches $i = 1$ (the left edge) before reaching $j = 1$. Now, we start by executing $T_{1,1}$. Later, we will go on the box B_r and then $B_{r-1}, \ldots,$ and finally B_1. Each box B_k requires a sequence of at least w_k communications. Moreover as the boxes are processed in a sequence (because of the dependencies) we have a total number of communications which is larger or equal to $\sum_{k=1}^{r} w_k = c$. There is a contradiction because this cannot happen in any optimal schedule. □

Lemma 3.5. *In any optimal schedule, the half-perimeter of the bottom left box B_r is strictly larger than c:*

$$w_r + h_r > c$$

Proof. As all the boxes are executed in sequence, we clearly need to issue a sequence of $\sum_{k=1}^{r-1}(w_k + h_k)$ communications by Lemma 3.3. Moreover Lemma 3.4 implies that $\sum_{k=1}^{r} w_k = \sum_{k=1}^{r} h_k = c$. Combining these two equations, we deduce that there exists a sequence of $2c - w_r - h_r$ communications. Since we cannot have c communications in an optimal solution, we conclude that $w_r + h_r > c$. □

Theorem 3.6.

$$C^*_{\max} > \frac{65c^2}{81} - \frac{8c}{27} - \frac{1}{9}$$

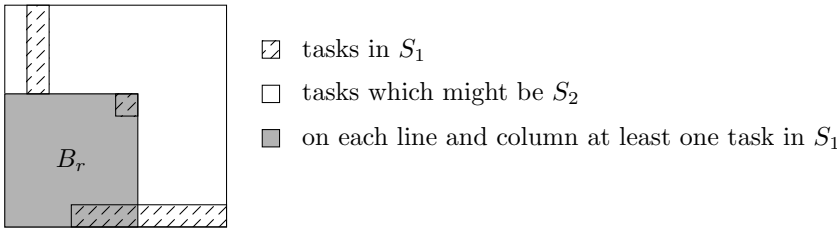

FIGURE 4. Constraints on S_1.

Proof. Without loss of generality let P_1 be the box processor of B_r. The set of tasks is partitioned into two subsets, namely, the set S_1 of tasks executed on P_1 and the set S_2 of tasks executed on any other processor.

We consider only the communications taking place between S_1 and S_2 and intend to maximize the area (*i.e.* the number of tasks) in S_2 while still validating the $\gamma^*_{\max}$ constraint of Theorem 3.1.

We now use Lemma 3.5. Since $w_r + h_r$ is larger than c, the communications constraint implies that in at least $w_r + h_r - c$ lines or columns, there is a continuous set of tasks on P_1 up to the top or right border of the grid (where no communication is needed anymore). Without loss of generality, we assume that there is a line of S_1 tasks reaching the right border. Since there is now at least a S_1 task on each x coordinate, these tasks generate c communications upwards unless at least the end of one column also consists in S_1 tasks. Figure 4 illustrates these constraints.

Now, we know that there is at least one S_1 task on each line and on each column. Any line or column not consisting *entirely* of S_1 tasks generates at least one communication. Since there are c lines, c columns and the number of allowed communications is less than c we deduce that more than c lines or columns are entirely filled with S_1 tasks. This directly implies that $|S_1| > |S_2|$.

Without considering communications nor the processors involved in S_2 we can deduce that $C^*_{\max} \geq |S_1|$ since all tasks of S_1 are executed on the same processor. In order to minimize this bound we therefore need to maximize $|S_2|$ since $|S_1| + |S_2| = c^2$.

We know from Theorem 3.1 that $\gamma^*_{\max} < \frac{8c}{9} + \frac{2}{3}$. Now we maximize the area of S_2 while validating this constraint on the number of communications which is itself depending on the perimeter of S_2. The ideal configuration for S_2 (which cannot be reached) is therefore as a square which is the best way to maximize the area of a shape while keeping its perimeter below a given limit. Let x be the side of this square, we have $2x < \frac{8c}{9} + \frac{2}{3}$ and thus, a maximal surface of $x^2 < \frac{16c^2}{81} + \frac{8c}{27} + \frac{1}{9}$.

From here we conclude that the area of S_1 is greater than $c^2 - (\frac{16c^2}{81} + \frac{8c}{27} + \frac{1}{9}) = \frac{65c^2}{81} - \frac{8c}{27} - \frac{1}{9}$. Since this is more than half of the total area it is also a lower bound of $C^*_{\max}$. □

3.2. Extension to an arbitrary 2D-grid(n_1, n_2)

Theorem 3.7. *For any rectangle grid of size* (n_1, n_2)*:*

$$C^*_{\max} > \min\left(\left\lfloor \frac{n_1}{c} \right\rfloor, \left\lfloor \frac{n_2}{c} \right\rfloor\right) \times \left(\frac{65c^2}{81} - \frac{8c}{27} - \frac{1}{9}\right).$$

Proof. We build a set of $\min\left(\left\lfloor \frac{n_1}{c} \right\rfloor, \left\lfloor \frac{n_2}{c} \right\rfloor\right)$ sub-grids of size $c \times c$ aligned on the diagonal. All these grids are completed in sequence because in each grid the bottom left task depends on the result of the top right task of the previous grid. We conclude the proof by applying Theorem 3.6 on each sub-grid. □

Corollary 3.8. *For any square grid of size* (n, n) *:*

$$C^*_{\max} > \left\lfloor \frac{n}{c} \right\rfloor \times \left(\frac{65c^2}{81} - \frac{8c}{27} - \frac{1}{9}\right).$$

3.3. Approximation algorithm for unbounded number of processors

In this section we propose a parallel algorithm which provides an upper bound of the optimal execution time. We assume a multiprocessor system composed of an unbounded number of processors, denoted each by P_k, with $k \geq 1$.

If $2c$ does not divide n we complete the grid by adding rows of virtual tasks until a new total of $n' = \left\lceil \frac{n}{2c} \right\rceil \times 2c$.

We subdivide the grid into $\frac{n'}{2c}$ slices and we allocate each slice S_k to processor P_k. Each slice S_k, for $1 \leq k \leq \dfrac{n'}{2c}$ is composed of $2c$ consecutive rows of the grid.

The algorithm is then pretty straightforward. Each processor executes all tasks from its slice. All tasks or communications are executed as soon as possible. If at any time the processor has the choice between several tasks, it executes in priority the task $T_{i,j}$ with minimal i. Figure 5 illustrates the execution of the algorithm on a $(6, 6)$ grid with $c = 1$.

The height of $2c$ of each slice enables us to overlap most communications by some computations. Consider any a processor P_k executing the column i of S_k. During the first c tasks of the column, P_k sends all data needed by P_{k+1} for the first task of column $i-1$. During the last c tasks of column i, P_k receives the data needed for the execution of the first task of the next column. Taking for example column 3 in S_2 in Figure 5 we can see that between $t = 7$ and $t = 8$, P_2 sends the data from $T_{2,4}$ to P_3 while between $t = 8$ and $t = 9$ it receives data from $T_{4,2}$.

This implies that once a slice starts it executes all its tasks without interruption.

The last slice is starting at time $t = (\frac{n'}{2c} - 1) \times 3c = (\lceil \frac{n}{2c} \rceil - 1) \times 3c$ and ending therefore at no more than $(\lceil \frac{n}{2c} \rceil - 1) \times 3c + 2cn < n(2c + 3/2)$.

Since the last task of the last slice depends on all others, it is also the last to be executed and we deduce immediately that $C_{\max} < n(2c + 3/2)$.

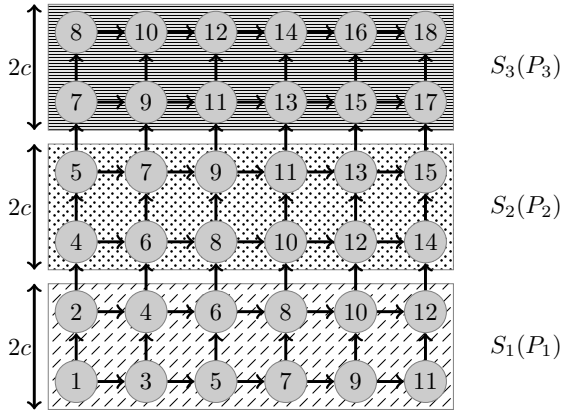

FIGURE 5. Tasks-Partitioning into Slices ($c = 1$) ; the labels denote the execution times.

4. Algorithm for 2 processors

In this section, we study the complexity of the 2D-grid(n,n) scheduling problem in the case where the number of processors is restricted to 2 and propose an asymptotically optimal scheduling.

4.1. A lower bound for $p = 2$

Proposition 4.1. *Let $C^*_{\max}(n, 2)$ be the optimal execution time of the 2D-grid(n,n) using 2 identical processors, then $C^*_{\max}(n, 2)$ satisfies:*

$$\frac{n^2}{2} + c + 1 \leq C^*_{\max}(n, 2).$$

Proof. According to the tasks precedence graph, it is clear that at least two communications are necessary:

- at least, one communication is necessary after the execution of the task $T_{1,1}$ which precedes all tasks. So, during the $1 + c$ first steps, at most only one processor is busy to execute at most $1 + c$ tasks;
- at least, one communication is necessary before the beginning of execution of task $T_{n,n}$ which succeeds all tasks. So, during the $1 + c$ last steps, at most only one processor is busy to execute at most $1 + c$ tasks.

Since there are two processors, we deduce that :

$$\frac{n^2 - 2(1 + c)}{2} + 2(1 + c) = \frac{n^2}{2} + 1 + c \leq C^*_{\max}(n, 2). \qquad \square$$

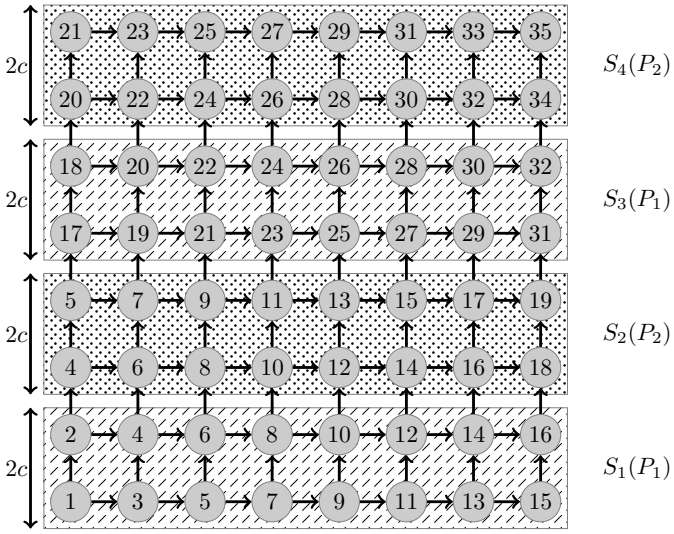

FIGURE 6. 2-processors Algorithm ($c = 1$).

4.2. An asymptotically optimal algorithm for $p = 2$

We start by the same decomposition into slices as in Section 3.3: with n' rows cut into $\frac{n'}{2c}$ slices of height $2c$.

The slices are then distributed to the processors in a cyclic manner.

Execution is kept the same as in the previous algorithm: slice by slice, columns by columns.

The example of Figure 6 describes the execution steps of the algorithm for $n = 8$ and $c = 1$.

If n is larger than $2c$ no processor is ever waiting between start and end of its execution. Processor P_1 executes $\lceil \frac{n'}{4c} \rceil$ slices and starts at $t = 0$. Processor P_2 executes $\lfloor \frac{n'}{4c} \rfloor$ slices and starts at $t = 3c$. We directly derive that $C_{\max} < \max(2cn(\lceil \frac{n'}{4c} \rceil), 3c+2cn(\lfloor \frac{n'}{4c} \rfloor)) < \max(2cn(\frac{n}{4c}+3/2), 3c+2cn(\frac{n}{4c}+1/2)) < \max(\frac{n^2}{2}+3cn, \frac{n^2}{2}+cn+3c) < \frac{n^2}{2}+3cn$

5. Concluding remarks

We have presented in this paper several results for scheduling 2-dimensional grids with large communication times. For an unbounded number of processors (at least greater than $\frac{n}{2c}$), we introduced a new lower bound analysis on the optimal completion time together with an upper bound. The ratio between both bounds is close to 1 and asymptotically optimal for two processors. While the complexity of our problem is still an open question, we proved that our restricted graph structure leads to a better significantly performance guarantees than the general scheduling problem with arbitrary graphs.

Moreover, we believe that the analysis provided in this paper may be used in a broader context. First it may be easily extended to grids of higher dimensions or to more complex networking models like full-duplex. Secondly, it clarifies the key graph properties which restrict the parallel execution. We may thus expect the analysis to extend to more general classes of graphs.

References

[1] F. Afrati, E. Bampis, L. Finta and I. Milis, Scheduling trees with large communication delays on two identical processors. *J. Scheduling* **8** (2005) 179–190.

[2] R.J. Anderson, P. Beame and W. Ruzzo, *Low overhead parallel schedules for task graphs*, in Proc. of SPAA (1990).

[3] E. Bampis, J-C. König and D. Trystram, A low overhead schedule for a 3D-grid graph. *Parallel Proces. Lett.* **2** (1992).

[4] E. Bampis, A. Giannakos and J-C. König, On the complexity of scheduling with large communication delays. *EJOR* 94 (1996).

[5] E. Bampis, F. Guinand and D. Trystram, Some models for scheduling parallel programs with communication delays. *Discrete Appl. Math.* **72** (1997).

[6] J. Blazewicz, K. Ecker, E. Pesh, G. Schimdt and J. Weglarz, *Handbook on Scheduling – from theory to applications.* Springer (2007).

[7] E.G. Coffman and P.J. Denning, *Operating system theory.* Prentice Hall (1972).

[8] P. Chrétienne and C. Picouleau, Scheduling with communication delays: A survey, in *Scheduling Theory and its Applications.* Wiley (1995).

[9] M. Cosnard and D. Trystram, *Algorithmes et architectures parallèles.* Inter Editions (1993).

[10] M. Drozdowski, Scheduling for Parallel Processing. Springer (2009).

[11] R. Giroudeau, JC. Konig, F. Moulai and J. Palaysi, Complexity and approximation for the precedence constrained scheduling problem with large communication delays, in Proc. EuroPar, LNCS, Springer (2005).

[12] J. Hwang, Y. Chow, F. Anger and C. Lee, Scheduling precedence graphs in systems with interprocessor communication times. *SIAM J. Comput.* **18** (1989) 244–257.

[13] A. Jakoby and R. Reischuk, The complexity of scheduling problems with communication delays for trees, in Proc. Scandinavian workshop on Algorithmic Theory (1992).

[14] V. Kumar, A. Grama, A. Gupta and G. Karypis, *Introduction to parallel computing: design and analysis of algorithms.* The Benjamin/Cummings Publishing company (1994).

[15] J. Lenstra, M. Veldhorst and B. Veltman, The complexity of scheduling trees with communication delays. *J. Algorithms* **20** (1996).

[16] R. Lepere and C. Rapine, *An asymptotic $O(\ln n / \ln \ln n)$-approximation algorithm for the scheduling problem with duplication on large communication delay graphs*, in Proc. STACS (2002).

[17] R. Lepère and D. Trystram, *A new clustering algorithm for large communication delays*, in Proc. of IPDPS (2002).

[18] A. Munier, Approximation algorithms for scheduling trees with general communication delays. *Parallel Computing* (1999).

[19] C. Papadimitriou and M. Yannakakis, Towards an Architecture-Independent Analysis of Parallel Algorithms. *SIAM J. Comput.* **19** (1990) 322–328.

[20] J. Pecero, D. Trystram and A. Zomaya, *A new genetic algorithm for scheduling for large communication delays*, in Proc. EuroPar, LNCS, Springer (2009).

[21] V. Rayward-Smith, UET scheduling with unit interprocessor communication delays. *Discrete Appl. Math.* **18** (1987) 55–71.

[22] A. Semar Shahul and O. Sinnen, Scheduling task graphs optimally with A*. *J. Supercomputing* **51** (2010) 310–332.

[23] T. Yang and A. Gerasoulis, List scheduling with and without communication delays. *Parallel Computing* **19** (1993) 1321–1344.

[24] D. Trystram, *Scheduling parallel applications using malleable tasks on clusters*, in Proc. of IPDPS (2001).