Etude de la complexité de la parallélisation de la grille 2D sur un modèle d'architecture multiprocesseur

- **El Mostafa DAOUDI et Abdelhak LAKHOUAJA**
Université Mohamed 1er, Faculté des Sciences
Département de Mathématiques et d'Informatique, Oujda, MAROC

- **Pierre MANNEBACK**
Laboratoire P.I.P., Faculté Polytechnique de Mons
Rue de Houdain 9, 7000 Mons, BELGIQUE

Résumé: Dans ce travail nous donnons une borne inférieure du temps d'exécution parallèle pour le graphe de précédence des tâches grille 2D sur un modèle d'architecture multiprocesseur MIMD. Les coûts des communications sont pris en compte et sont considérés comme étant une fonction linéaire de la taille des données échangées entre processeurs. Ensuite nous montrons que la borne inférieure peut être atteinte en utilisant une architecture multiprocesseur à mémoire distribuée: l'anneau des processeurs.

Mots clés: Architecture parallèles, Graphe de précédence des tâches, Grille 2D, Complexité des algorithmes parallèles.

1. Introduction

La parallélisation d'algorithmes peut être étudiée et modélisée à l'aide des graphes de précédences des tâches. Les sommets sont des tâches élémentaires et les arcs définissent des éventuelles communications entre les tâches extrémités si ces dernières sont exécutées sur 2 processeurs différents. Notons qu'une tâche élémentaire est un bloc d'instructions, indivisible (dans le sens qu'elle est exécutée par le même processeur sans interruption). Elle est complétement définie par ses entrées, ses sorties et par son temps d'exécution [5,6].

Plusieurs applications numériques peuvent être modélisées par les graphes de précédences des tâches ayant la forme d'une grille comme la multiplication matricielle et la résolution des systèmes triangulaires pour la méthode du gradient conjugué préconditionné [7,8]. Cette famille de graphes a fait l'objet de plusieurs travaux dans le but d'étudier la complexité de leurs parallélisations sur différents modèles d'architecture parallèles [1,2]. Un grand nombre de travaux concernant la recherche d'algorithmes optimaux considèrent le cas idéal où la communication est négligée (tâche élémentaire ayant un temps d'exécution unitaire noté modèle UET: Unit Execution Time) ou la supposent égale à une unité de temps (modèle UET-UCT: Unit Execution Time - Unit Communication Time) [1,3].

Dans ce papier nous étudions la complexité de la parallélisation du graphe de tâches grille 2D, en supposant que le coût de communication est une fonction linéaire de la taille des données échangées entre 2

processeurs différents. Cette extension a l'avantage de fournir des modèles proches des machines parallèles existantes. Les résultats de complexité que nous obtenons pour la grille 2D sont des généralisations de quelques résultats obtenus dans la littérature [3].

2. Définitions et modèle de parallélisation

Le modèle d'architecture parallèle considéré est de type machine MIMD composée de p processeurs identiques, qui communiquent entre eux par l'intermédiaire d'une mémoire partagée ou via un réseau complet. On suppose que le coût de communications de m données, entre 2 processeurs différents, est modélisé par:

$$c(m)=\beta+m\tau$$

où β est le temps d'initialisation et τ est le temps de transfert. De plus nous supposons qu'un processeur peut commencer une communication avant d'avoir terminé une autre.

Le processus de parallélisation d'algorithmes conduit à un *overhead*, surcoût en français, du temps d'exécution parallèle (théoriquement celui-ci est égal au temps d'exécution séquentiel divisé par le nombre de processeurs). L'*overhead* est dû au temps de communication et au temps d'inactivité des processeurs imposée par les contraintes de précédences. Dans le but de diminuer cet *overhead*, on utilise les notions de job et de graphe d'ordonnancement introduites dans [1,4].

Un job est défini comme une suite indivisible de tâches UET (dans le sens qu'elles sont attribuées au même processeur et sont exécutées sans interruption). Il faut noter qu'il n'y a pas de communications entre les tâches appartenant au même job et qu'un job ne peut communiquer avec un autre qu'avant ou après son exécution. Le coût de communication est considéré si deux jobs sont adjacents et sont attribués à 2 processeurs différents. Le regroupement des tâches UET en jobs donne lieu à un nouveau graphe de précédence, où les sommets sont des jobs et les contraintes de précédences entre les tâches se transforment en contraintes de précédences entre les jobs tel que si T appartient au job J et T' appartient au job J' et si T<<T' alors J<<J' (<< est la realtion de précédence). De plus on suppose que le nouveau graphe obtenu est sans circuit.

Dans la suite nous considérons le graphe de précédence des tâches grille 2D de taille (n,n). Il est défini par les contraintes de précédences suivantes où les tâches $T_{i,j}$ sont UET et << est la relation de précédence:

$$T_{i,j} << T_{i',j} \quad \text{pour } 1 \leq i < i' \leq n$$
$$T_{i,j} << T_{i,j'} \quad \text{pour } 1 \leq j < j' \leq n$$

T<<T' veut dire que l'exécution de la tâche T doit être terminée avant que celle de T' ne commence [5,6]. Sur la figure 1, on représente une grille 2D de taille (6,6).

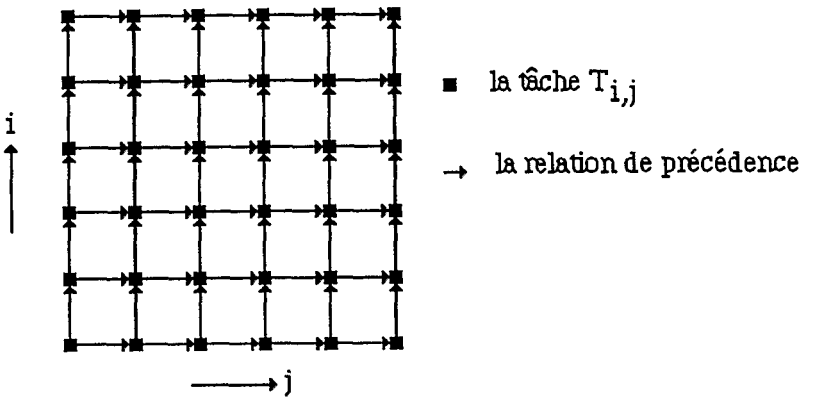


Figure 1: Graphe de précédence des tâches grille 2D

Pour étudier la complexité de la parallélisation de la grille 2D, nous utilisons la notion de jobs. Nous supposons que chaque job contient un nombre fixe égal à $m_x m_y$, pour $1 \leq m_x, m_y \leq n$, de tâches UET, de la façon suivante: le job $J_{u,v}$, pour $1 \leq u \leq n/m_y$ et $1 \leq v \leq n/m_x$, est un ensemble de $m_x m_y$ tâches UET $T_{i,j}$, pour $(u-1)m_y < i \leq um_y$, $(v-1)m_x < j \leq vm_x$. Ce regroupement en jobs conduit à un nouveau graphe de précédence de la forme d'une grille 2D de taille $(n/m_y, n/m_x)$ où les sommets sont les jobs et les contraintes de précédences sont définies par:

$$J_{u,v} << J_{u',v} \quad \text{pour } 1 \leq u < u' \leq n/m_y$$
$$J_{u,v} << J_{u,v'} \quad \text{pour } 1 \leq v < v' \leq n/m_x$$

J<<J' veut dire que l'exécution du job J doit être terminée avant que celle de J' ne commence. Sans perte de généralité, on suppose que $m_x \leq m_y$. Sur la figure 2, on présente le graphe de précédence des jobs où chaque job est de taille (2,2).

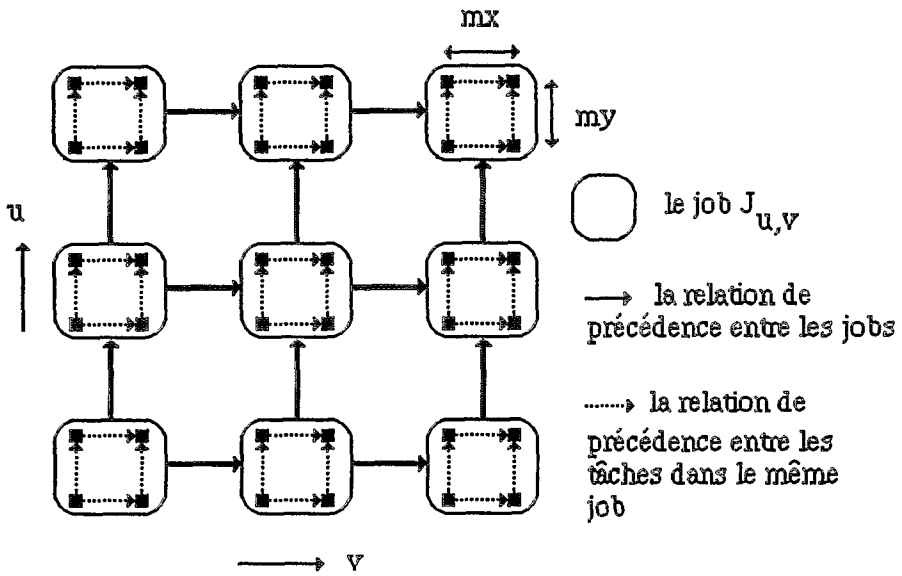


Figure 2: Graphe de précédence des jobs grille 2D

Le temps d'exécution d'un job J est égal à la somme des temps d'exécutions des tâches UET appartenant à ce job qui est égal à $W(J)=m_x m_y$. Suivant les relations de précédences, le nombre de jobs indépendants (qui peuvent être exécutés en parallèle) est au plus égal à $\eta(m_$ Par conséquent le nombre de processeurs qui peuvent être utilisés

Lemme 1: On suppose que chaque job est de taille fixe égale à (m_x,m_y), pour $1 \leq m_x,m_y \leq n$. Une borne inférieure du temps d'exécution parallèle d'une grille 2D de taille (n,n) est égale à:

$$B(m_x,m_y) = nm_y + nm_x + nc(m_x)/m_y - c(m_x) - m_xm_y$$

Preuve:

- Cas $c(m_x) \leq m_xm_y$.

Notons par t_i, pour $1 \leq i \leq n/m_y$, la date du début d'exécution du job diagonal $J_{i,i}$ et par T_i la date de fin de son exécution. On a:

$$T_i = t_i + W(J_{i,i}) = t_i + m_xm_y.$$

Suivant les relations de précédences, le job $J_{i+1,i+1}$ ne commence son exécution qu'après la fin d'exécution des jobs

$$\{J_{i,i} , J_{i,i+1} , J_{i+1,i}\}.$$

Notons que les jobs $J_{i,i+1}$ et $J_{i+1,i}$ sont indépendants et donc peuvent être exécutés en parallèle par 2 processeurs. Nous déduisons que la date t_{i+1} du début d'exécution du job $J_{i+1,i+1}$ vérifie:

i) $t_{i+1} \geq T_i + 2m_xm_y$ si $J_{i,i+1}$ et $J_{i+1,i}$ sont exécutés dans le même processeur

ii) $t_{i+1} \geq T_i + m_xm_y + c(m_x)$ si les jobs $J_{i,i+1}$ et $J_{i+1,i}$ sont exécutés en parallèle sur 2 processeurs différents.

De i) et ii) nous déduisons que

- les jobs diagonaux $\{J_{i,i}$, pour $1 \leq i < n/m_y\}$ et
- les jobs $\{J_{n/my,j}$, pour $n/m_y \leq j \leq n/m_x$ $\}$

ce qui conduit à la borne inférieure

$$B(m_x,m_y) = (2m_xm_y + c(m_x)(\frac{n}{m_y} - 1) + (\frac{n}{m_x} - \frac{n}{m_y} + 1)m_xm_y$$

- **Cas $m_xm_y \leq c(m_x)$**

Soit a le plus petit entier tel que $c(m_x) \leq am_xm_y$ (a est la partie entière supérieure de $c(m_x)/m_xm_y$). Notons par w_i, pour $1 \leq i \leq n/m_y$, le temps du début d'exécution du job $J_{i,1+(i-1)a}$ et par T_i le temps de fin de son exécution. On a:

$$T_i = w_i + W(J_{i,1+(i-1)a}) = w_i + m_xm_y$$

Suivant les relations de précédences, le job $J_{i+1,1+ia}$ ne commence son exécution qu'après l'exécution de l'ensemble des jobs:

$$J = \{J_{i,j}, \text{ pour } 1+(i-1)a \leq j \leq 1+ia \} \cup \{J_{i+1,j}, \text{ pour } 1+(i-1)a \leq j \leq ia \}$$

Notons que les jobs $J_{i,j}$ et $J_{i+1,j'}$ pour $j' < j$ sont indépendants et par conséquent peuvent être exécutés en parallèles par 2 processeurs. Nous déduisons que:

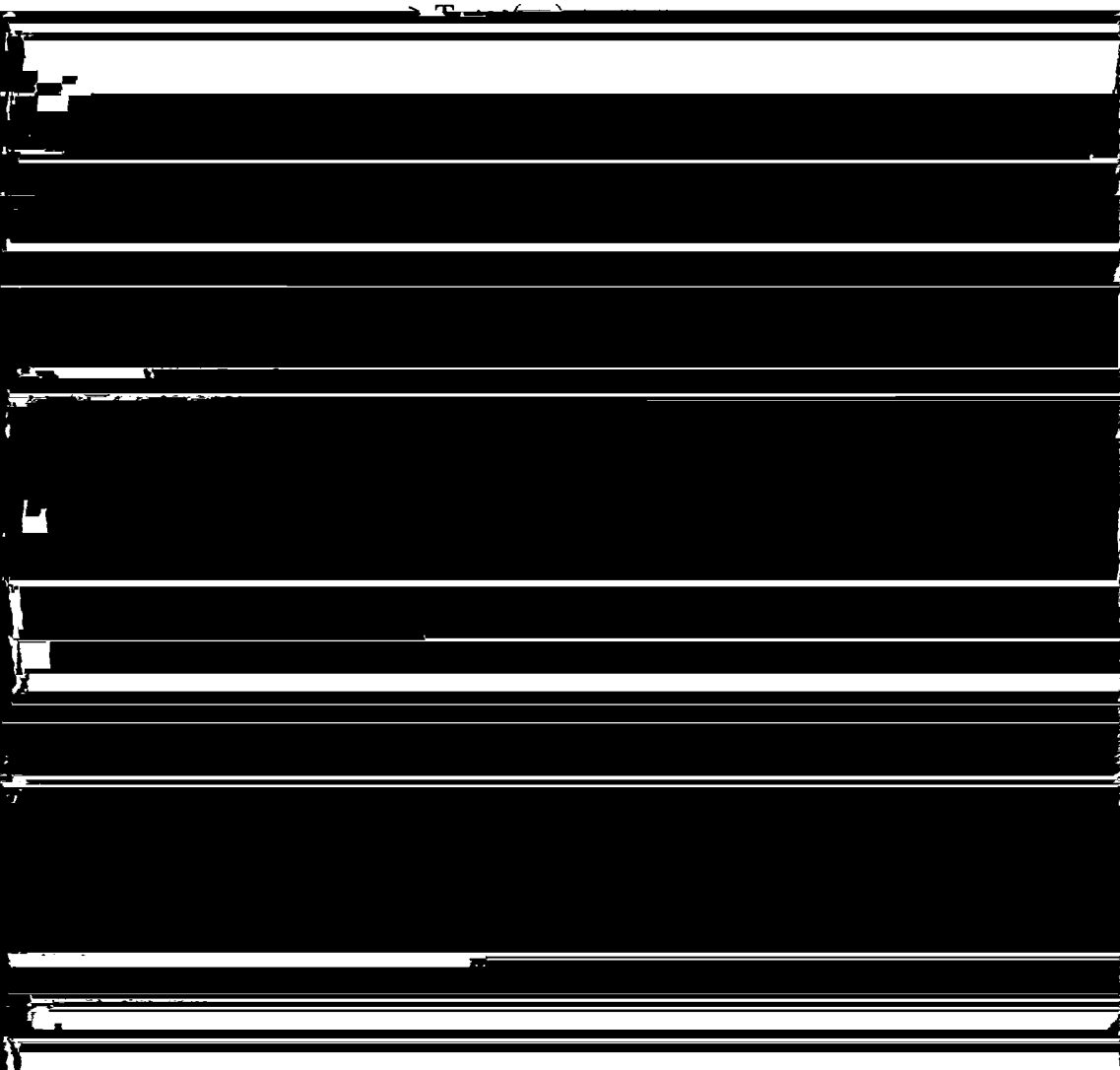
i) Si un seul processeur est utilisé pour exécuter les jobs de l'ensemble J alors

$$w_{i+1} \geq T_i + 2am_xm_y$$

ii) Si 2 processeurs sont utilisés pour exécuter les jobs de l'ensemble J, et puisque $c(m_x) \leq am_xm_y$, alors nous déduisons qu'après avoir terminé l'exécution du job $J_{i,1+(i-1)a}$ par un processeur, celui ci communique m_x données à l'autre processeur pour exécuter le job $J_{i+1,1+(i-1)a}$ et simultanément continue à exécuter les a jobs $J_{i,j}$, pour $(i-1)a \leq j \leq 1+ia$.

Comme nous supposons de plus qu'un processeur peut commencer une communication avant d'avoir terminé une autre, alors à la fin d'exécution de chaque job $J_{i,j}$, pour $(i-1)a \leq j \leq 1+ia$, il envoie à l'autre processeur m_x données pour exécuter les jobs $J_{i+1,j}$, pour $(i-1)a \leq j \leq 1+ia$.

par conséquent w_{i+1} vérifie:



Corollaire 2: Pour $m_x=m_y=1$ une borne inférieure du temps d'exécution parallèle est égal à:

- 2n-1 si c(1)=0 (modèle UET)
- 3n-2 si c(1)=1 (modèle UET-UCT)

Maintenant nous analysons $B(m_x,m_y)$ dans le but d'obtenir une borne inférieure du temps d'exécution pour toute valeur de m_x et m_y. Posons c=c(1), nous déduisons le résultat suivant:

Proposition 3: Une borne inférieure du temps d'exécution est $(2+c)n+O(1)$ si $c\leq 1$ et $(1+2\sqrt{c})n+O(1)$ sinon.

Elle est accomplie par les jobs de taille optimale $(1,m^*_y)$, avec $m^*_y=1$ si $c\leq 1$, et $m^*_y=\sqrt{c}$ sinon.

Preuve:

On choisit les paramètres m_x et m_y dans le but de minimiser $B(m_x,m_y)$. Il est clair que $B(1,m_y)\leq B(m_x,m_y)$. Donc il suffit de minimiser $B(1,m_y)$. L'analyse de $B(1,m_y)$ montre qu'il existe un point stationnaire défini par $m^*_y=\sqrt{\frac{nc}{n-1}}$, ce qui conduit au résultat asymptotique donné dans la proposition 3.

4. Un algorithme asymptotiquement optimal sur une architecture parallèle à mémoire distribuée

Considérons un anneau composé de $p=n/m^*_y$ processeurs. Chaque processeur P_i, pour $1 \leq i \leq p-2$, est connecté aux processeurs P_{i-1} et P_{i+1} et en plus P_0 est connecté au processeur P_{p-1}. On suppose que les calculs et les communications peuvent se chevaucher et que chaque processeur peut communiquer au moins $\sqrt{c}$ données en pipeline.

Proposition 4: La borne inférieure de la proposition 3 peut être atteinte sur un anneau composé de n/m^*_y processeurs.

Preuve:

Nous proposons un algorithme qui atteint cette borne sur un anneau de composé de $p=n/m^*_y$ processeurs. Les jobs sont choisis de tailles optimales égales à $(1,m^*_y)$ et sont attribués aux processeurs ligne par ligne de la manière suivante; au processeur P_i, pour $0 \leq i \leq p-1$, on affecte les jobs $J_{i+1,v}$, pour $1 \leq v \leq n$.

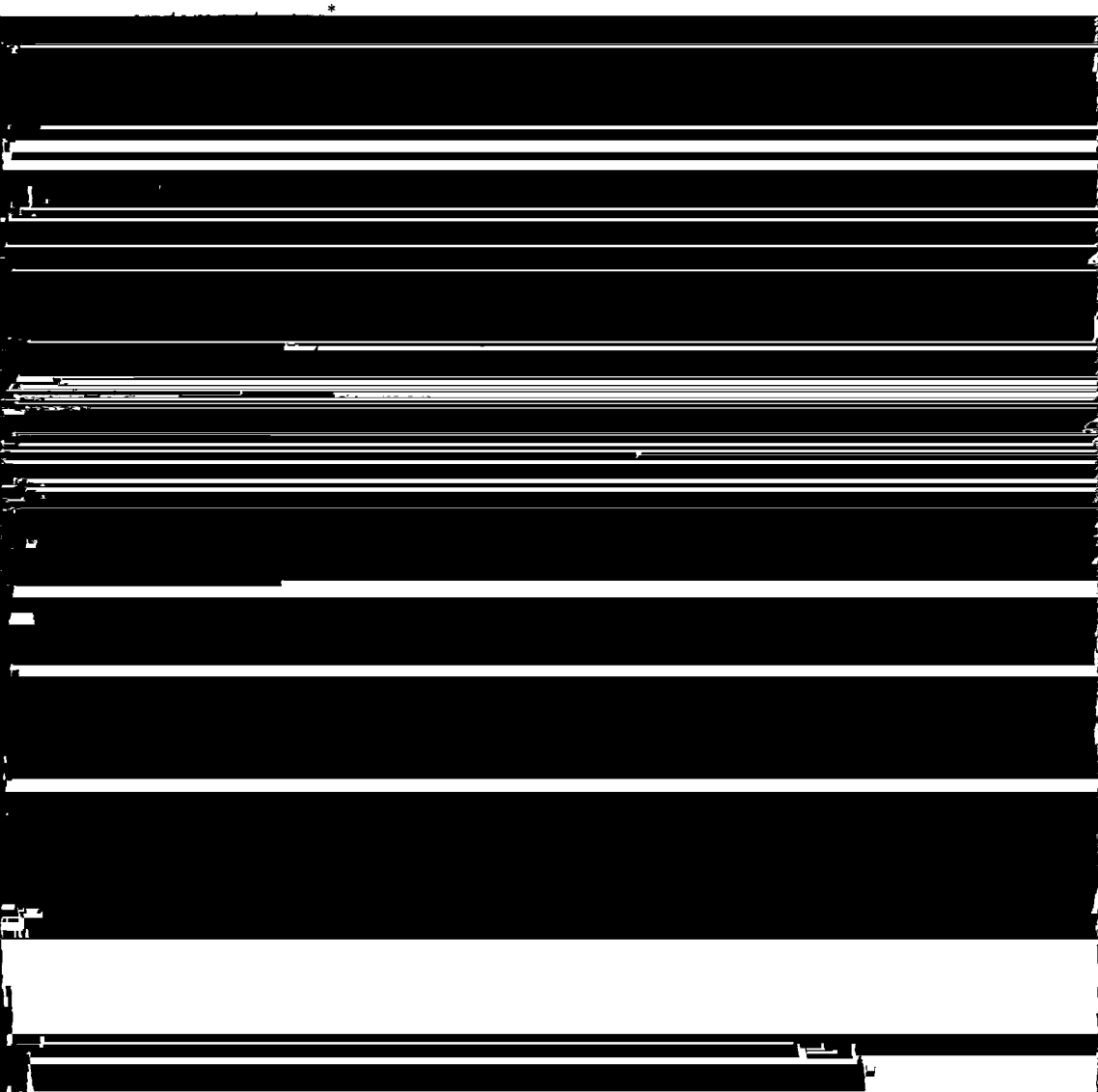
Il est clair que le processeur P_i, commence son exécution au temps $(m^*_y+c)i$. Nous allons montrer qu'il la termine au temps $(m^*_y+c)i+nm^*_y$. Par conséquent le temps d'exécution de l'algorithme et égal à celui du dernier job, $J_{n/m^*y,n}$, qui est exécuté par le processeur P_{n/m^*y-1} donc égal à $(m^*_y+c)(n/m^*_y-1)+nm^*_y$.

Pour cela nous montrons par récurrence sur i, pour $1 \leq i \leq n/m^*_y$ que si l'exécution du job $J_{i,v}$, pour $1 \leq v \leq n-1$, commence au temps t alors celle de $J_{i,v+1}$ commence au temps $t+m^*_y$. Par conséquent le processeur P_i termine son exécution au temps $(m^*_y+c)i+nm^*_y$.

L'hypothèse de récurrence est vrai pour i=0 (Processeur 0). Supposons que la propriété est vraie pour i.

Soit t le temps du début d'exécution du job $J_{i+1,v}$.

- D'après les relations de précédence, l'exécution du job $J_{i,v}$ est terminée au temps t-c.
- D'après l'hypothèse de récurrence, le job $J_{i,v+1}$ termine son exécution au



Si nous supposons que les jobs sont choisis de taille $(1,m^*_y)$ et sont attribués aux processeurs ligne par ligne d'une manière entrelacé tel qu'au processeur P_i, pour $0 \leq i \leq p-1$, on affecte les jobs $J_{i+1+jp,v}$, pour $1 \leq v \leq n$, j est un entier tel que $i+1+jp \leq n/m^*_y$ alors on a le résultat suivant.

Corollaire 5: La borne inférieure de la proposition 3 peut être atteinte sur un anneau composé de $p = \frac{n}{1+c} + O(1)$ processeurs si $c \leq 1$ et $p = \frac{n}{1+\sqrt{c}} + O(1)$ sinon.

Exemple:

n=6 et c=2 alors $m^*_y=2$

Processeur 2:	10 12 14 16 18 20
	9 11 13 15 17 19
Processeur 1:	6 8 10 12 14 16
	5 7 9 11 13 15
Processeur 0:	2 4 6 8 10 12
	1 3 5 7 9 11

5 Conclusion

[4] E. Bampis, J-C. König, D. Trystram, "Impact of communications on the complexity of the parallel gaussian elimination", Parallel Computing 17 (1991)

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

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

[7] E.M. Daoudi, P. Manneback, "Implementation of ICCG algorithm on distributed memory architecture", Iterative Methods in Linear Algebra, North-Holland (1992)

[8] Y. Robert, The impact of vector and parallel architectures on the gaussian elimination algorithms, Halsted Press (1990)