REVUE FRANÇAISE D'AUTOMATIQUE, D'INFORMATIQUE ET DE RECHERCHE OPÉRATIONNELLE. RECHERCHE OPÉRATIONNELLE

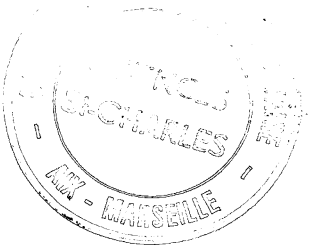
M. MARRAKCHI

Un algorithme parallèle optimal pour la résolution d'un système triangulaire

Revue française d'automatique, d'informatique et de recherche opérationnelle. Recherche opérationnelle, tome 27, n° 3 (1993), p. 273-280.

<http://www.numdam.org/item?id=RO_1993__27_3_273_0>

Recherche opérationnelle/Operations Research
(vol. 27, n° 3, 1993, p. 273 à 280)



UN ALGORITHME PARALLÈLE OPTIMAL POUR LA RÉSOLUTION D'UN SYSTÈME TRIANGULAIRE (*)

par M. MARRAKCHI ([1])

Communiqué par Jacques CARLIER

Résumé. – *On présente un algorithme parallèle optimal pour le graphe de précédence de type 2-pas à tâches de longueur constantes. Ce dernier est obtenu en segmentant l'algorithme séquentiel résolvant un système linéaire triangulaire. Pour un problème de taille n et un nombre de processeurs p inférieur à (n + 2)/4, nous montrons l'optimalité de l'algorithme parallèle construit.*

Mots clés : Algorithmes parallèles, graphe 2-pas, résolution de système triangulaire, complexité.

Abstract. – *We present in this paper an optimal parallel algorithm for 2-steps graph with constant tasks. This graph occures in the parallelisation of triangular linear system resolution. For a problem of size n and p processors lower then (n + 2)/4, we show the optimality of this parallel algorithm.*

Keywords: Parallel algorithms, 2-steps graph, triangular linear system resolution, complexity.

1. INTRODUCTION

La résolution d'un système triangulaire est une étape principale dans les méthodes résolvant un système dense. La parallélisation de cet algorithme a été étudiée par plusieurs auteurs [3, 7, 8]. Dans ce papier, nous présentons la parallélisation de l'algorithme résolvant un système triangulaire. Pour un problème de taille n et un nombre de processeurs égal à p où $0 < p \leq (n+2)/4$, nous construisons un algorithme parallèle optimal. Nous subdivisons l'algorithme séquentiel en tâches élémentaires qui sont ordonnées par une relation de précédence : la relation $T \ll T'$ signifie, l'exécution de la tâche T' ne débute que si celle de la tâche T est terminée [1, 4, 5]. L'architecture sous laquelle nous programmons l'algorithme parallèle est une architecture SIMD/MIMD à mémoire partagée. Pour plus de détails du modèle théorique sur lequel nous programmons l'algorithme parallèle, nous renvoyons le lecteur à [2].

(*) Reçu en février 1991.

([1]) Département Informatique, Faculté des Sciences, Route de Soukra, km 4, 3038 Sfax, Tunisie.

Recherche opérationnelle/Operations Research, 0399-0559/93/03/$ 4.00

2. GRAPHE D'ORDONNANCEMENT

Étant donné le système triangulaire inférieur

$$Ax = b, \qquad où \quad A = (a_{ij})\ 1 \leq i, j \leq n$$

une matrice carrée triangulaire de taille n non singulière et x, b deux vecteurs de taille n. La décomposition en tâches est la suivante en supposant initialement que chaque mémoire x_i contient b_i $1 \leq i \leq n$:

Pour $i := 1$ **à** n **faire exécuter** $T_{i,i} : \left\langle x_i := \dfrac{x_i}{a_{ii}} \right\rangle$

Pour $j := i + 1$ **à** n **faire exécuter** $T_{j,i} : < x_j := x_j - a_{ji}x_i >$.

La tâche $T_{i,i}$ correspond au calcul de x_i, tandis que la tâche $T_{j,i}$ élimine cette dernière de la j-ième équation. Les contraintes d'ordonnancement sont :

(A) $T_{i,i} \ll T_{j,i}$, $i + 1 \leq j \leq n$, $1 \leq i \leq n - 1$

(B) $T_{j,i} \ll T_{j,i+1}$, $2 \leq j \leq n$, $1 \leq i \leq j - 1$.

La figure 1 présente l'allure du graphe d'ordonnancement qui est le graphe 2-pas dans le cas où $n = 5$ [2]. Une unité de temps est définie comme le temps mis pour effectuer soit une soustraction munie d'une multiplication soit une division [9]. Le temps mis pour exécuter une tâche quelconque est une unité de temps. Le temps séquentiel de l'algorithme étant alors $n\ (n + 1)/2$ unités de temps.

Notations. – 1. $L\ (T_{k,j}) = \{T_{k,j}\ ;\ T_{k+1,j}\ ;\ T_{k+2,j}\ ;\ \ldots\ ;\ T_{n,j}\}$.

2. $L\ (T_{k,j}\ ;\ T_{i,j}) = \{T_{k,j}\ ;\ T_{k+1,j}\ ;\ T_{k+2,j}\ ;\ \ldots\ ;\ T_{i,j}\}$ où $i \geq k$. Si $i < k$, $L\ (T_{k,j}\ ;\ T_{i,j})$ est vide.

3. $D(T_{k,j}) = \{T_{k,j}\ ;\ T_{k+1,j-1}\ ;\ T_{k+2,j-2}\ ;\ \ldots\}$, c'est l'ensemble de toutes les tâches $T_{a,b}$ telles que $a + b = k + j$ et $b \leq j$.

4. $D\ (T_{k,j}\ ;\ T_{u,v}) = \{T_{k,j}\ ;\ T_{k+1,j-1}\ ;\ T_{k+2,j-2}\ ;\ \ldots\ ;\ T_{u,v}\}$, c'est l'ensemble de toutes les tâches $T_{a,b}$ telles que $a + b = k + j = u + v$, $v \leq b \leq j$ et $k \leq a \leq u$. Si $k > u$ ou $v > j$, $D\ (T_{k,j}\ ;\ T_{u,v})$ est vide.

Exemple illustratif. – Dans le cas où $n = 5$, soient les ensembles suivants :

$L\ (T_{2,1}) = \{T_{2,1}\ ;\ T_{3,1}\ ;\ T_{4,1}\ ;\ T_{5,1}\}$; $\quad L\ (T_{3,2}) = \{T_{3,2}\ ;\ T_{4,2}\ ;\ T_{5,2}\}$;

$L\ (T_{4,3}) = \{T_{4,3}\ ;\ T_{5,3}\}$; $\quad L\ (T_{5,4}) = \{T_{5,4}\}$;

$D\ (T_{2,1}) = \{T_{2,1}\}$; $\quad D\ (T_{3,3}) = \{T_{3,3}\ ;\ T_{4,2}\ ;\ T_{51}\}$

Remarques :

1. Si la taille de la matrice est un entier pair, la tâche $T_{n,1}$ du graphe est un élément de $D\ (T_{1\ +\ n/2,\ n/2})$, si non, elle appartient à $D\ (T_{(1\ +\ n)/2,\ (1\ +\ n)/2})$.

2. Soient $D\ (T_{i,j})$ et $D\ (T_{u,v})$, si $i + j \leq u + v$, $D\ (T_{i,j})$ se situe (au sens large) au-dessus de $D\ (T_{u,v})$.

3. DESCRIPTION DE L'ALGORITHME

Dans ce qui suit, nous décrivons l'algorithme parallèle construit pour résoudre un système triangulaire inférieur. L'algorithme s'exécute en fait en deux phases séquentielles. Notons m et s respectivement le plus petit et le plus grand des deux entiers $2p$ et $i + p$.

Phase (1) **exécuter** $\{T_{1,1}\}$;

pour $i := 1$ **à** $2p - 1$ **faire**

début exécuter $L(T_{i+1,i} ; T_{m,i}) \cup D(T_{m+1,i-1} ; T_{p+i,p})$; (*a*)

$\{T_{i+1,i+1}\} \cup L(T_{m+1,i} ; T_{2p,i}) \cup D(T_{s+1,2p+i-s-1})$; **fin** ; (*b*)

Dans cette phase, les processeurs exécutent les tâches des ensembles $L(T_{i+1,i} ; T_{2p,i})$; $D(T_{2p+1,i-1})$, $\{T_{1,1}\}$, $\{T_{i+1,i+1}\}$ $(1 \leq i \leq 2p - 1)$. Pour que chaque ligne (*a*) et (*b*) contiennent exactement p tâches, il faut que les ensembles $D(T_{2p+1,i-1})$, $(1 \leq i \leq 2p - 1)$ se situent au-dessus de la diagonale contenant $T_{n,1}$. Autrement dit, il faut que $2p + 1 \leq n + 1$ $(1 \leq i \leq 2p - 1)$. Alors, p doit vérifier la condition $0 < p \leq (n + 2)/4$. L'exécution du graphe commence à l'instant $t = 0$ par

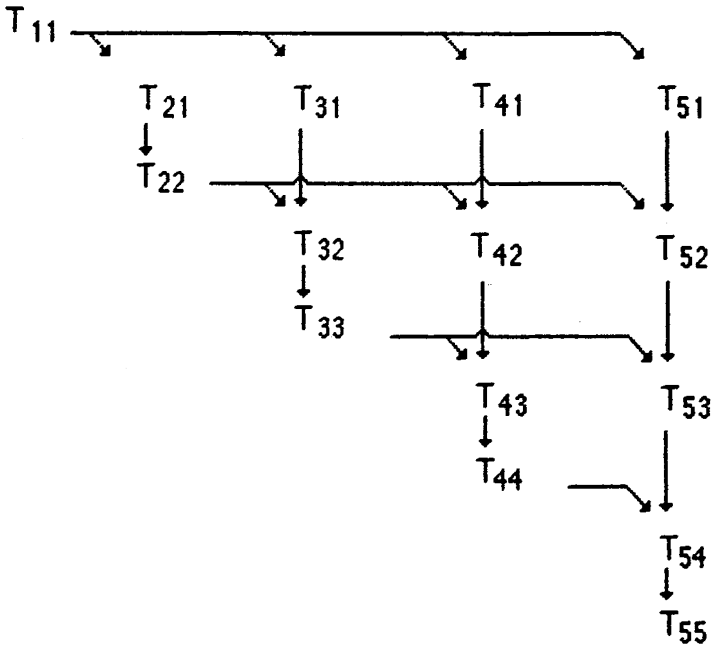


Figure 1. – Graphe d'ordonnancement pour une matrice de taille 5×5.

la tâche $T_{1,1}$. Avec p processeurs, il est évident qu'il est possible d'exécuter toutes les tâches de chaque ligne en parallèle en une seule unité de temps. A l'instant $2i-1$ (resp. $2i$), l'algorithme commence l'exécution de la ligne (a) [resp. (b)] de la i-ième itération.

Phase (2) **exécuter** $D(T_{2p+1,2p-1})$;
pour $i := 2p$ **à** $n-1$ **faire**
début exécuter $D(T_{i+1,i})$; $D(T_{i+1,i+1})$; **fin** ;

Dans cette phase, les processeurs exécutent les tâches des ensembles $D(T_{2p+1,2p-1})$, $D(T_{i+1,i})$ et $D(T_{i+1,i+1})$ avec $2p \leqslant i \leqslant n-1$. L'exécution de cette phase aura lieu juste après la fin de l'exécution de la première phase, c'est-à-dire à l'instant $4p-1$. Pour définir l'ordre d'exécution des tâches des ensembles $D(T_{2p+1,2p-1})$, $D(T_{i+1,i})$ et $D(T_{i+1,i+1})$ avec $2p \leqslant i \leqslant n-p-1$, on introduit la contrainte supplémentaire (C) suivante [2] :

$$T_{k,j} \leq\leq T_{k+1,j-1}$$

où la notation $T \leq\leq T'$ signifie que l'exécution de la tâche T' ne peut débuter avant celle de la tâche T (mais la simultanéité est possible). Plus précisement, les processeurs exécutent les tâches dans l'ordre suivant :

$$T_{2p+1,2p-1} \leq\leq T_{2p+2,2p-2} \leq\leq \cdots \leq\leq T_{2p+1,2p} \leq\leq T_{2p+2,2p-1} \leq\leq \cdots \leq\leq T_{k,j} \leq\leq T_{k+1,j-1} \leq\leq \cdots \leq\leq T_{n-p,n-p} \leq\leq \cdots \leq\leq T_{n-1,n-2p+1} \leq\leq T_{n,n-2p}.$$

Si q processeurs se libèrent, l'algorithme les affecte à l'exécution des q tâches suivantes. Le lemme suivant calcule l'instant $t_{k,j}$ où débute l'exécution d'une tâche $T_{k,j}$ des ensembles $D(T_{2p+1,2p-1})$, $D(T_{i+1,i})$ et $D(T_{i+1,i+1})$ où $2p \leq i \leq n-p-1$, ($\lfloor x \rfloor$) est le plus grand entier inférieur ou égal à x).

LEMME :

1. *Si*

$$T_{k,j} \in \bigcup_{2p \leq i \leq n/2} D(T_{i+1,i}),$$

$$t_{k,j} = 1 + \left\lfloor \frac{(i-2)(i+1)+k}{p} \right\rfloor$$

2. *Si*

$$T_{k,j} \in \bigcup_{2p-1 \leq i \leq n/2} D(T_{i+1,i+1}),$$

$$t_{k,j} = 1 + \left\lfloor \frac{(i-1)(i+1)+k-1}{p} \right\rfloor$$

3. *Si*

$$T_{k,j} \in \bigcup_{n/2 \leq i \leq n-p-1} D(T_{i+1,i}),$$

$$t_{k,j} = 1 + \left\lfloor \frac{(n-1)(n+2) - 2(n-i)^2 - 2n + 2k - 2}{2p} \right\rfloor$$

4. *Si*

$$T_{k,j} \in \bigcup_{n/2 \leq i \leq n-p-1} D(T_{i+1,i+1}),$$

$$t_{k,j} = 1 + \left\lfloor \frac{(n-1)(n+2) - 2(n-i)^2 - 2i - 2k - 2}{2p} \right\rfloor$$

Pour déterminer l'instant $t_{k,j}$ où l'exécution d'une tâche débute, il suffit de calculer $N_{k,j}$: c'est la somme du nombre de tâches exécutées dans la phase (2) avant $T_{k,j}$ (au sens strict) dans l'ordre imposé par (C) et du cardinal des tâches exécutées dans la phase (1) privée de la tâche $T_{1,1}$. On a alors :

$$t_{k,j} = 1 + \left\lfloor \frac{N_{ki}}{p} \right\rfloor$$

Dans le cas où $T_{k,j}$ appartient à l'ensemble

$$\bigcup_{2p \leq i \leq n/2} D(T_{i+1,i}) \quad [\text{resp.} \bigcup_{2p-1 \leq i \leq n/2} D(T_{i+1,i+1})],$$

$N_{k,j}$ est égal à : $k + (i-2)(i+1)$ [resp. $(i-1)(i+1) + k - 1$], ce qui prouve les expressions 1 et 2.

Pour déterminer $N_{k,j}$ d'une tâche $T_{k,j}$ de l'ensemble

$$\bigcup_{n/2 \leq i \leq n-p-1} D(T_{i+1,i}) \quad [\text{resp.} \bigcup_{n/2 \leq i \leq n-p-1} D(T_{i+1,i+1})],$$

on retranche du nombre total de tâches du graphe privé de la tâche $T_{1,1}$, *i. e.* $(n-1)(n+2)/2$, celui des tâches situées après la tâche $T_{k,j}$ y compris cette dernière dans l'ordre imposé par (C), *i. e.*, $(n-1)^2 + n - k + 1$ [resp. $(n-1)^2 + i - k + 1$]. Ce qui achève la démonstration des expressions 3 et 4.

A la fin de l'exécution des tâches de $D(T_{n-p,n-p})$, les processeurs se synchronisent et commencent l'exécution de la partie inférieure du graphe bordée supérieurement (au sens strict) par $D(T_{n-p,n-p})$. On affecte à chaque processeur une colonne des p dernières colonnes : le processeur P_j

exécute la colonne $n - p + j$ $(1 \leq j \leq p)$ y compris la tâche diagonale $T_{n-p+j,\, n-p+j}$. A l'instant,

$$t_{n-p+1,\, n-p} = t_{n,\, n-2p} + 1 = \left\lfloor \frac{(n-1)(n+2) - 2p^2 - 2}{2p} \right\rfloor + 1,$$

l'algorithme commence l'exécution des p tâches de $D(T_{n-p+1,\, n-p})$. Le début de l'exécution d'une tâche $T_{k,j}$ aura lieu à l'instant

$$t_{k,j} = t_{n-p+1,\, n-p} + j - (2n - 2p - k + 1).$$

A l'instant,

$$t_{n,\, n} + 1 = \left\lfloor \frac{(n-1)(n+2) - 2p^2 - 2}{2p} \right\rfloor + 2p + 1,$$

l'exécution du graphe se termine.

Pour mieux comprendre l'algorithme, nous présentons l'exemple suivant avec $n = 10$ et $p = 3$. (i, j dans la colonne s signifie que la tâche $T_{i,j}$ s'exécute par le processeur P_s) :

	1	2	3	Instant où l'exécution débute
Phase (1)	1,1	–	–	0
	2,1	3,1	4,1	1
	2,2	5,1	6,1	2
	3,2	4,2	5,2	3
	3,3	6,2	7,1	4
	4,3	5,3	6,3	5
	4,4	7,2	8,1	6
	5,4	6,4	7,3	7
	5,5	8,2	9,1	8
	6,5	7,4	8,3	9
	9,2	10,1	6,6	10
Phase (2)	7,5	8,4	9,3	11
	10,2	7,6	8,5	12
	9,4	10,3	7,7	13
	8,6	9,5	10,4	14
	8,7	9,6	10,5	15
	10,6	9,7	8,8	16
	–	9,8	10,7	17
	–	9,9	10,8	18
	–	–	10,9	19
	–	–	10,10	20

LEMME : *L'algorithme exécute toutes les tâches du graphe 2-pas en respectant les contraintes d'ordonnancement.*

Il est clair que la phase (1) [resp. phase (2)] exécute les tâches de la partie supérieure (resp. inférieure) du graphe bordée inférieurement (au sens large) (resp. au sens strict) par $\{T_{2p,2p}\}$ et $D(T_{2p,2p-1})$. Il est facile aussi de vérifier que les contraintes d'ordonnancement sont respectées dans les deux phases. Il suffit de comparer l'instant $t_{k,j}$ où l'exécution de la tâche $T_{k,j}$ commence avec $t_{j,j}$ et $t_{k;j+1}$ les instants de début d'exécution des tâches $T_{j,j}$ et $T_{k,j+1}$.

4. OPTIMALITÉ

Dans cette section, on montre que l'algorithme construit est optimal. Supposons qu'il existe un algorithme $\mathcal{A}^*$ qui exécute le graphe 2-pas en un temps T^*_p inférieur au temps T_p de l'algorithme $\mathcal{A}$ construit. A l'instant $t_{n-p+1,n-p}$ les p processeurs se situent sur la diagonale $D(T_{n-p+1,n-p})$ pour l'algorithme $\mathcal{A}$. Au même instant, les p processeurs se situent sur ou au-dessous de $D(T_{n-p+1,n-p})$ pour l'algorithme $\mathcal{A}^*$. En effet, si un processeur est strictement au-dessus de la diagonale $D(T_{n-p+1,n-p})$, T^*_p ne sera pas inférieur à T_p. Effectuons la division euclidienne de $N_{n-p+1,n-p}$ par p, on a $N_{n-p+1,n-p} = qp + r$ avec $0 \leq r < p$. Si $r = 0$, entre la fin de l'exécution de la tâche $T_{1,1}$ et le début de l'exécution de $D(T_{n-p+1,n-p})$, les processeurs sont tous actifs pour l'algorithme $\mathcal{A}$. Ce qui entraîne que pour $r = 0$, l'algorithme $\mathcal{A}$ s'exécute en temps minimal. Pour r vérifiant $0 < r < p$, les processeurs, dans l'algorithme $\mathcal{A}$, balaient le graphe 2-pas, privé de la tâche $T_{1,1}$, sans interuption jusqu'à la synchronisation des processeurs à la fin de l'exécution de $D(T_{n-p,n-p})$. Donc on a $p - r$ processeurs libres après l'exécution des r dernières tâches de $D(T_{n-p,n-p})$. A l'instant $t_{n-p+1,n-p}$, $\mathcal{A}^*$ a exécuté au maximum $p-r$ tâches de plus que $\mathcal{A}$. Ces tâches se situent obligatoirement sur ou en dessous de $D(T_{n-p+1,n-p})$. Il existe au moins une tâche $T_{k,j}$ de ce dernier ensemble dont l'exécution n'est pas encore commencée. Ce qui entraîne que

$$T^*_p \geq t_{n-p+1,n-p} + 2p = T_p$$

(2 p représente le temps minimal mis pour exécuter les tâches du chemin séparant $T_{k,j}$ de $T_{n,n}$, $T_{k,j}$ et $T_{n,n}$ comprises).

THÉORÈME :

1. *Le temps mis pour exécuter l'algorithme construit est* :

$$T_p = \left\lfloor \frac{(n-1)(n+2) - 2p^2 - 2}{2p} \right\rfloor + 2p + 1$$

2. *Il n'existe aucun algorithme parallèle exécutant le graphe 2-pas avec p processeurs en un temps strictement inférieur a* T_p.

5. CONCLUSION

L'algorithme parallèle présenté ci-dessus n'est exécutable que si $0 < p \leq (n+2)/4$. Dans le cas général où n et p vérifient $(0 < p \leq n)$, on montrera, dans un autre papier, qu'il existe un algorithme parallèle optimal pour le graphe 2-pas où la longueur des tâches est constante.

REMERCIEMENTS

Je remercie vivement le P. Zaher Mahjoub dont les remarques et discussions ont permis d'améliorer le contenu et la présentation de cet article.

RÉFÉRENCES

1. M. COSNARD et Y. ROBERT, Algorithme parallèle : une étude de complexité, *Technique et Science Informatiques, 6,* 2, 1987, p. 115-125.
2. M. COSNARD, M. MARRAKCHI, Y. ROBERT et D. TRYSTRAM, Parallel Gaussian elimination on an MIMD computer, *Parallel Computing, 6,* 1988, p. 275-296.
3. D. EVANS et R. C. DUNBAR, The parallel solution of triangular system of equations, *I.E.E.E. Transactions on Computers,* c-32, n°2, February 1983, p. 201-204.
4. S. P. KUMAR, *Parallel algorithms for solving linear equations on MIMD computers,* PhD Thesis, Washington State University, 1982.
5. R. E. LORD, J. S. KOWALIK, et S. P. KUMAR, Solving linear algebraic equations on an MIMD computer, *J. A.C.M. 30,* 1, 1983, p. 103-117.
6. M. MARRAKCHI, *Techniques d'ordonnancement et algorithme parallèle en algèbre linéaire,* Thèse de l'I.N.P.G., Université de Grenoble, juillet 1988.
7. M. MARRAKCHI et Y. ROBERT, *Optimal scheduling algorithms for parallel iterative methods on multiprocessor systems,* rapport 693, janvier 1988, IMAG, Laboratoire TIM3, Grenoble.
8. N. M. MISSIRLIS, Scheduling parallel iterative methods on multiprocessor systems, *Parallel Computing, 5,* 1987, p. 295-302.