MISE EN ŒUVRE PARALLÈLE DE L'ALGORITHME DE STRASSEN

DIDIER AUROUX ET ARNAUD LEGRAND

*Ce rapport a été réalisé dans le cadre du cours d'*Algorithmique Parallèle *de Frédéric Desprez du D.E.A d'Informatique Fondamentale de l'E.N.S. Lyon*

Résumé. Ce rapport est basé sur l'article *A High Performance Parallel Strassen Implementation* de B. Grayson, A.P. Shah et R.A. van de Geijn [GSG95]. Nous y présentons d'abord un bref historique des algorithmes effectuant un produit de matrices puis une distribution récursive des matrices sur une grille torique permettant une mise en œuvre efficace de l'algorithme de Strassen. Enfin, nous étudions de façon théorique les performances et présentons quelques optimisations possibles.

1. Introduction

L'étude de la complexité du produit de matrice dense a commencé dans les années 60 et est toujours d'actualité. S'il est désormais rare d'avoir à multiplier des matrices denses énormes, la régularité des algorithmes résolvant ce problème permet de développer des techniques de parallélisation généralement applicables à d'autres noyeaux d'algèbre linéaire.
Nous rappelons d'abord un certain nombre de résultats théoriques sur la complexité du produit de deux matrices denses. Les seuls algorithmes utilisables en pratique datent du milieu des années 70 et sont basés sur des approches de type *diviser pour régner* (algorithme de Strassen, variante de Winograd). Nous présentons donc la distribution de [GSG95] permettant une mise en œuvre parallèle efficace de l'algorithme de Strassen sur une grille torique. Nous donnons ensuite une étude théorique d'un certain nombre de paramètres de l'algorithme tels que le niveau de récursion optimal, la réutilisation de la mémoire, le facteur d'accélération, ... Enfin, en nous basant sur des mesures de performances de machine classiques, nous observons son comportement avec les meilleurs choix possibles.

2. Produits rapides de matrices denses

2.1. **Historique.** Les formules classiques pour multiplier deux matrices carrées de taille n recquièrent n^3 multiplications mais d'autres algorithmes plus efficaces ont été développés durant les dernières décennies. Pan en 1965 et Winograd en 1967 ont été les premiers à réussir à réduire ce nombre à environ $n^3/2$ multiplications et, pendant quelques années, on a eu du mal à imaginer que cette borne puisse encore être améliorée. C'est Strassen qui, en 1968, donne des formules permettant de calculer le produit de matrices 2×2 avec seulement 7 multiplications et 18 additions. Cette technique appliquée récursivement permet d'atteindre un coût pour le produit de deux matrices carrées de taille n en $\Theta(n^{\log_2 7})$. Winograd en 1971 propose une variante de cet algorithme avec seulement 15 additions, ce qui est le mieux que l'on puisse espérer obtenir avec un produit de matrice récursif basé sur une découpe en quadrants.

Définissons ω comme la borne inférieure des τ tels qu'il existe un algorithme de produits de matrice de complexité $\mathcal{O}(n^\tau)$. Au vu des progrès inattendus concernant ce problème, bien des scientifiques ont commencé à se demander si on avait $\omega > 2$. D'autres algorithmes sophistiqués basés sur des produits tensoriels ont permis d'améliorer encore la borne, notamment Pan en 1984 qui a montré que $\omega \leqslant 2.496$ et Strassen en 1986 qui a encore réduit la borne en montrant que $\omega \leqslant 2.479$. Ce sont Coppersmith et Winograd qui détiennent actuellement le record avec $\omega \leqslant 2.376$. L'algorithme est basé sur un théorème de combinatoire dû à Salem et Spencer (1942) qui donne des ensembles denses d'entiers ne contenant jamais trois termes appartenant à une progression arithmétique. Si ces algorithmes ont un intérêt théorique indéniable, ils sont cependant inutilisables en pratique tant la constante cachée dans le $\mathcal{O}$ est énorme.

Date: 28 janvier 2000.

La question "*A-t-on* $\omega > 2$ *?* " a été baptisée par Bürgisser "*Problème fondamental de la théorie de la complexité algébrique*". Voici un aperçu d'un problème de combinatoire extrêmement lié. Étant donné un groupe abélien G d'ordre n, quel est le plus petit entier $f(n, G)$ tel que : si S est une partie de G de cardinal supérieur ou égal à $f(n, G)$, alors il existe $A, B, C \subseteq S$ non vides et deux à deux distincts tels que $\sum_{a\in A} a = \sum_{b\in B} b = \sum_{c\in C} c$. Si on définit F par $F(n) = \max_G f(n, G)$ et que l'on s'intéresse à $\rho = \lim_{n\to\infty} \frac{\ln n}{F(n)}$, on peut montrer que ρ et ω sont liés. Coppersmith et Winograd ont par exemple montré que si $\rho = 0$ alors $\omega = 2$. On ne sait bien sûr toujours pas si $\rho = 0$ ...

2.2. **Produit de matrices 2 × 2.** Les produits de matrices que nous allons présenter sont basés sur une utilisation récursive de formules valables pour les matrices carrées de taille 2. Nous allons donc présenter trois ensembles de formules permettant de calculer le produit C de deux matrices carrées A et B de taille 2. Ainsi, si on note

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \cdot \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix},$$

on peut utiliser les formules suivantes pour calculer C :

	Post-additions	Produits	Pré-additions
Algorithme Standard	$C_{11} = P_1 + P_2$ $C_{21} = P_3 + P_4$ $C_{12} = P_5 + P_6$ $C_{22} = P_7 + P_8$	$P_1 = A_{11} \cdot B_{11}$ $P_2 = A_{12} \cdot B_{21}$ $P_3 = A_{21} \cdot B_{11}$ $P_4 = A_{22} \cdot B_{21}$ $P_5 = A_{11} \cdot B_{12}$ $P_6 = A_{12} \cdot B_{22}$ $P_7 = A_{21} \cdot B_{12}$ $P_8 = A_{22} \cdot B_{22}$	
Algorithme de Strassen	$C_{11} = P_1 + P_4 - P_5 + P_7$ $C_{21} = P_2 + P_4$ $C_{12} = P_3 + P_5$ $C_{22} = P_1 + P_3 - P_2 + P_6$	$P_1 = S_1 \cdot T_1$ $P_2 = S_2 \cdot B_{11}$ $P_3 = A_{11} \cdot T_2$ $P_4 = A_{22} \cdot T_3$ $P_5 = S_3 \cdot B_{22}$ $P_6 = S_4 \cdot T_4$ $P_7 = S_5 \cdot T_5$	$S_1 = A_{11} + A_{22}$ $S_2 = A_{21} + A_{22}$ $S_3 = A_{11} - A_{12}$ $S_4 = A_{21} - A_{11}$ $S_5 = A_{12} - A_{22}$ $T_1 = B_{11} + B_{22}$ $T_2 = B_{12} - B_{22}$ $T_3 = B_{21} - B_{11}$ $T_4 = B_{11} + B_{12}$ $T_5 = B_{21} + B_{22}$
Variante dûe à Winograd de l'Algorithme de Strassen	$C_{11} = U_1 = P_1 + P_2$ $U_2 = P_1 + P_4$ $U_3 = U_2 + P_5$ $C_{21} = U_4 = U_3 + P_7$ $C_{12} = U_5 = U_3 + P_3$ $U_6 = U_2 + P_3$ $C_{22} = U_7 = U_6 + P_6$	$P_1 = A_{11} \cdot B_{11}$ $P_2 = A_{12} \cdot B_{21}$ $P_3 = S_1 \cdot T_1$ $P_4 = S_2 \cdot T_2$ $P_5 = S_3 \cdot T_3$ $P_6 = S_4 \cdot B_{22}$ $P_7 = A_{22} \cdot T_4$	$S_1 = A_{21} + A_{22}$ $S_2 = S_1 - A_{11}$ $S_3 = A_{11} - A_{21}$ $S_4 = A_{12} - S_2$ $T_1 = B_{12} - B_{11}$ $T_2 = B_{22} - T_1$ $T_3 = B_{22} - B_{12}$ $T_4 = B_{21} - T_2$

Strassen a montré en 1969 que tout produit de matrices 2 × 2 nécessitait au moins 7 multiplications [Str69] et donc que tout algorithme basé sur une découpe récursive en quadrants avait une complexité au mieux en $\Theta(n^{\log_2(7)})$. Deux ans plus tard, Winograd [Win71] propose une variante des formules de Strassen avec seulement 15 additions et R.Probert [Pro73] montre en 1973 que ces formules sont additivement optimales.

2.3. **À la recherche de l'algorithme optimal.** Les formules de Winograd étant optimales pour les matrices carrées de taille 2, elles vont permettre d'atteindre asymptotiquement les meilleures performances possibles pour un algorithme basé sur une découpe récursive en quadrants.

Il ne reste donc plus qu'à trouver le bon niveau de récursion à utiliser pour obtenir la plus petite constante possible du $\mathcal{O}(n^{\log_2(7)})$. En effet, étant donné que pour de petites valeurs de n, les formules standards sont meilleures que les autres, il convient de ne pas appliquer les formules améliorées trop profondément. Dans le cas où la taille de la matrice n'est pas une

puissance de 2, la façon dont on augmente la taille de la matrice pour pouvoir appliquer les formules joue également un rôle important pour atteindre les meilleurs résultats [FP74].
Si on donne le même prix aux additions et aux multiplications, le coût de l'algorithme classique est de $2n^3 - n^2$. Celui de Strassen avec le niveau de récursion optimal est de $3,897n^{\log_2(7)} + \mathcal{O}(n^2)$ si n est une puissance de 2 et est majoré par $4,623n^{\log_2(7)} + \mathcal{O}(n^2)$ sinon. Enfin le coût de la variante de Winograd avec le niveau de récursion optimal est de $3,373n^{\log_2(7)} + \mathcal{O}(n^2)$ si n est une puissance de 2 et est majoré par $4,537n^{\log_2(7)} + \mathcal{O}(n^2)$ sinon. Ce dernier algorithme est donc plus intéressant que l'algorithme classique pour n pair plus grand que 14 et pour tout n supérieur à 35[FP74].

3. Parallélisation

3.1. Distribution des données et Algorithme. L'efficacité de tout algorithme parallèle dépend énormément du placement des données. Aucune des distributions classiques (par blocs, cyclique, cyclique par blocs, …) ne s'avère adaptée à une approche *diviser pour régner* pour calculer un produit de matrice. En fait, pour obtenir la plus grande efficacité possible, il est assez naturel d'essayer de distribuer les données de façon à ce que chaque partie de l'algorithme séquentiel puisse se faire avec la meilleure régulation de charge possible sur tous les processeurs. Ainsi, si on répartit uniformément chacune des plus petites matrices sur lesquelles on va travailler (des matrices de taille $n/2^l$ si on applique une découpe récursive l fois), on obtient une distribution pour les matrices de départ du type de la figure 1.

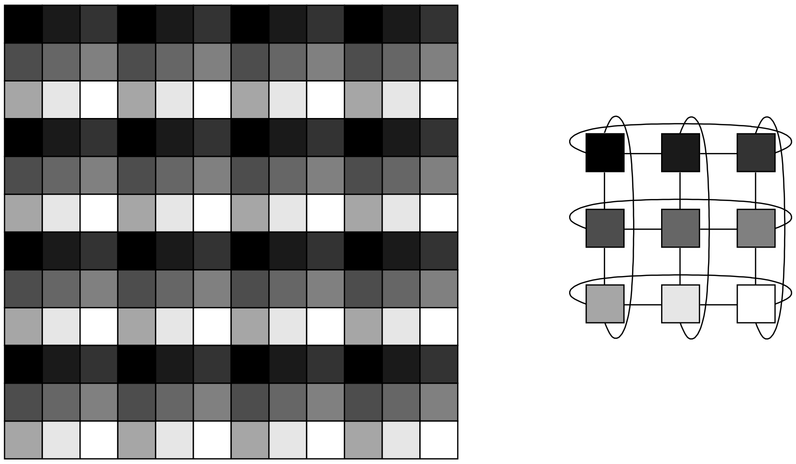


Fig. 1. Une distribution pour la grille 3×3 adaptée aux découpes récursives avec une profondeur 2

Notons, $M_{[i_1,i_2]\times[j_1,j_2]}$ la sous-matrice de M dont les indices sont inclus dans $[\![i_1, i_2]\!] \times [\![j_1, j_2]\!]$. Pour $q \in [\![1, p]\!]$ et $k \in [\![1, 2^l]\!]$, notons $\sigma(k, q) = (k-1)\frac{n}{2^l} + (q-1)\frac{n}{p.2^l} + 1$ et pour p_1, p_2 dans $[\![1, p]\!]$ et (i, j) dans $[\![1, 2^l]\!]$, notons $M_{(i,j),(p_1,p_2)}$ la sous-matrice $M_{[\sigma(i,p_1),\sigma(i+1,p_1)-1]\times[\sigma(j,p_2),\sigma(j+1,p_2)-1]}$. Formellement, le processeur (p_1, p_2) reçoit donc les sous-matrices $A_{(i,j),(p_1,p_2)}$, $B_{(i,j),(p_1,p_2)}$ et $C_{(i,j),(p_1,p_2)}$ pour $(i, j) \in [\![1, 2^l]\!]$. Cette distribution permet de faire toutes les sommes en parallèle à un niveau quelconque de la récursion. Les seuls produits se font au niveau l en utilisant un produit standard optimisé pour la machine sur laquelle on travaille (à l'aide des BLAS 3) et la distribution choisie ne demande pas de redistribution des données. On obtient donc une distribution cyclique par bloc paramétrée par la profondeur de récursion de l'algorithme.

3.2. Complexité de l'algorithme en fonction de la profondeur. Si on itère le procédé jusqu'à une profondeur l, on va effectuer des produits standards sur des matrices de taille 2^l fois plus petites. Notons $T(j)$ le temps nécessaire au calcul d'une matrice intervenant au niveau de récursion j. On aura donc $T(l) = F(\frac{n}{2^l})$ où F est le temps nécessaire pour multiplier deux matrices sur place en parallèle avec le résultat distribué de la même façon que les matrices initiales (en utilisant l'algorithme de Fox par exemple). Par récurrence, en appliquant

la variante dûe à Winograd, on a $T(j-1) = 7\,T(j) + 15\,t_{add}\left(\frac{n}{2^j\,p}\right)^2$ pour $j \in [\![l, 1]\!]$, où t_{add} est le temps nécessaire pour effectuer une addition.
En résolvant la relation de récurrence précédente, on obtient

$$C = T(0) = 7^l T(l) + 15 t_{add}\left(\frac{n}{p}\right)^2 \sum_{k=1}^{l} 7^{k-1}\left(\frac{1}{2^k}\right)^2 \text{, soit}$$

$$\boxed{C = 7^l F(\tfrac{n}{2^l}) + 5 t_{add}\tfrac{n^2}{p^2}\left(\left(\tfrac{7}{4}\right)^l - 1\right)} \qquad (1)$$

Nous continuerons l'étude en supposant que l'algorithme de base pour effectuer un produit sur la grille est l'algorithme de Fox. On a donc : $F_{\textit{théorique}}(N) = 2\frac{N^3}{p^2}\tau + 2\frac{N^2}{p}t_{transf} + (p-1)\,t_{start}$, τ désignant le temps nécessaire pour effectuer une opération élémentaire, t_{transf} le temps de transfert des données entre deux processeurs, et t_{start} la latence au début de chaque communication.

Ainsi, notre algorithme a une complexité théorique égale à :

$$\boxed{\begin{aligned} C_{\textit{\textbf{théorique}}} \;=\;& 2\tfrac{n^3}{p^2}\tau\left(\tfrac{7}{8}\right)^l + \left(2\tfrac{n^2}{p}t_{transf} + 5t_{add}\tfrac{n^2}{p^2}\right)\left(\tfrac{7}{4}\right)^l \\ & + (p-1)\,t_{start}7^l - 5t_{add}\tfrac{n^2}{p^2} \end{aligned}} \qquad (2)$$

3.3. **Calcul de la profondeur optimale.** Pour trouver la profondeur optimale, il suffit donc de dériver la formule (2) par rapport à l. On se ramène donc à résoudre l'équation du troisième degré en $X = 2^l$: $[(p-1)\,t_{start}\ln(7)]\;X^3 + \left[\left(2\,\frac{n^2}{p}t_{transf} + 5\,t_{add}\,\frac{n^2}{p^2}\right)\ln(\frac{7}{4})\right]\;X + \left[2\,\frac{n^3}{p^2}\tau\;\ln(\frac{7}{8})\right] = 0$
On peut résoudre algèbriquement cette équation à l'aide des formules de Cardan et en déduire X puis la valeur optimale de l à n et p fixé. La forme close est cependant assez peu parlante. Si on simplifie le problème en supposant que la latence est nulle, on arrive à

$$\boxed{l_{\textit{\textbf{opt théorique}}} = \log_2\left(\frac{2\,\frac{n^3}{p^2}\tau\;\ln(\frac{8}{7})}{\left(2\,\frac{n^2}{p}t_{transf}+5\,t_{add}\,\frac{n^2}{p^2}\right)\ln(\frac{7}{4})}\right) \approx \log_2(n) - \log_2(p) + c^{te}} \qquad (3)$$

Cette formule donne une idée du comportement de l'algorithme mais nous montrerons dans la section 4 que le comportement sans cette simplification est très semblable. De plus, nous pouvons remarquer que si, a priori, l est majoré par $\log_2(n/p)$ (les matrices que l'on considère au dernier niveau de la récursion sont de taille $\frac{n}{p2^l}$), sa valeur optimale n'en diffère que par une constante dépendant des la machine cible.

3.4. **Utilisation de la mémoire et mesures d'efficacité.**

3.4.1. *Espace Mémoire.* Observons les dépendances entre les différentes variables introduites par l'algorithme (Cf Figure 2). Lorsque l'on fait une addition, on peut écraser l'une des opérandes mais ce n'est par contre pas possible lorsque l'on fait une multiplication. Étant donné qu'à chaque appel récursif, on empile des matrices relativement grosses, on peut alors se demander quel est le nombre minimal de variables temporaires nécessaires compte tenu de ces règles. Il suffit en fait de 3 variables temporaires supplémentaires si on organise bien les calculs (Cf. Figure 3).

En effet, la figure 3 nous montre qu'il faut 7 variables correspondant à 7 espaces mémoire de taille $\left(\frac{n}{p2^k}\right)^2$ pour faire tous les calculs intervenant à la profondeur k de l'algorithme de Strassen-Winograd. Or nous disposons de l'espace alloué pour le résultat de ces opérations (à savoir une matrice de taille $\frac{n}{p2^{k-1}}$), dans lequel nous pouvons stocker les variables Var_4 à Var_7 de la figure 3. Ainsi, en tenant compte de la place nécessaire pour stocker les données A et B et le résultat C, l'espace nécessaire sur chaque machine est de $3\frac{n^2}{p^2} + 3\sum_{k=1}^{l}\left(\frac{n}{p2^k}\right)^2 \leq 4\frac{n^2}{p^2}$, soit seulement la taille d'une matrice supplémentaire (en plus de A, B et C) par rapport

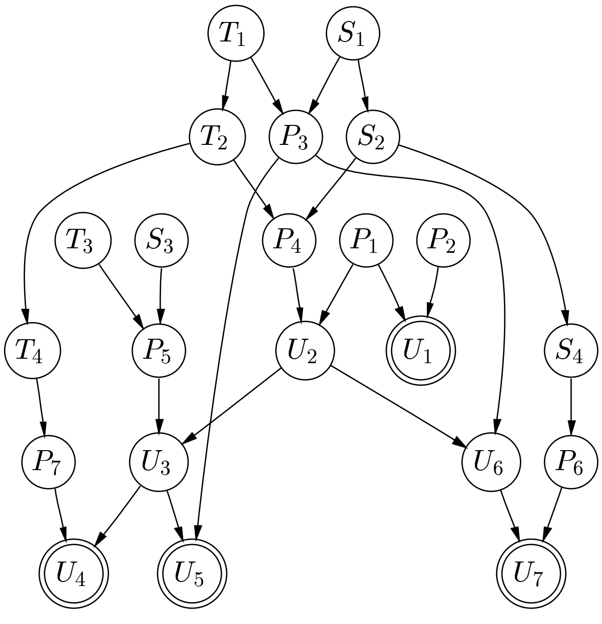


FIG. 2. Dépendances entre les différentes variables temporaires de la variante de Winograd

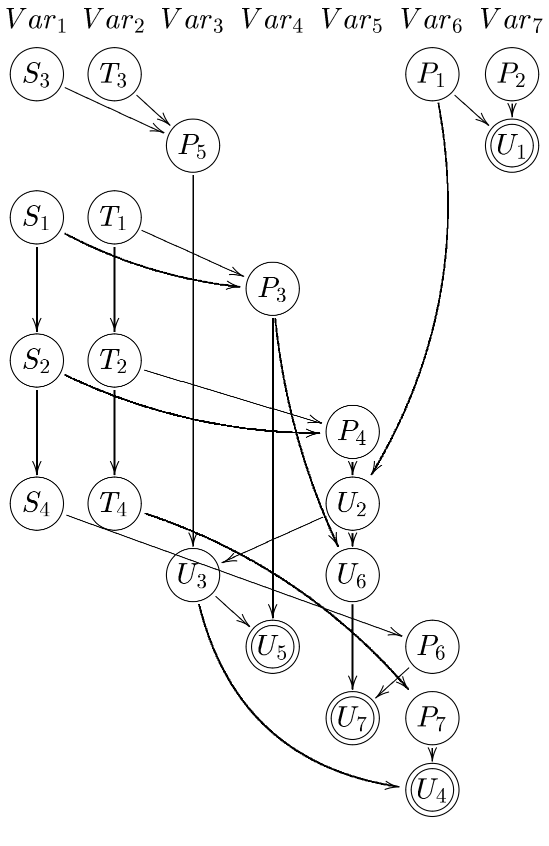


FIG. 3. Organisation économique des calculs

à l'algorithme standard. Chaque machine aura donc moins de mémoire à utiliser que si elle faisait le calcul toute seule à partir du moment où il y a au moins deux machines[1].

3.4.2. *Temps asymptotique.* En remplaçant l par la valeur optimale dans la formule (2) on trouve une complexité équivalente à

[1]Donc à partir du moment où on utilise une machine parallèle.. ☺

$$\boxed{T_{//}(p) \approx c^{te} \frac{n^{\log_2(7)}}{p^{\log_2(7)-1}}} \tag{4}$$

, la constante étant naturellement une fonction croissante de τ, t_{transf}, t_{add}, t_{start}. On retrouve donc un temps asymptotique en $\mathcal{O}(n^{\log_2(7)})$ comme pour l'algorithme de Strassen séquentiel. Des exemples numériques seront présentés plus tard (en section 4) pour valider ce résultat dans le cas où l'on ne suppose pas que la latence est nulle.

3.4.3. *Efficacité.* L'efficacité e vaut donc

$$\boxed{e(p^2) = \frac{T_{seq}/p^2}{T_{//}(p)} \approx c^{te}/p^{0.193}} \tag{5}$$

L'efficacité décroît lorsque $p \to \infty$ mais de façon relativement lente puisque le temps asymptotique est presque en $\frac{1}{p^2}$.

4. Mesures théoriques

Nous allons d'abord présenter les complexités de l'algorithme de Fox sur l'architecture Store & Forward puis Worm-Hole. Nous utiliserons ces formules comme bases pour notre algorithme. Les valeurs numériques utilisées sont issues de mesures réelles (tenant compte des effets de cache et autres problèmes de swap).

4.1. **Mise en œuvre pratique de l'algorithme de Fox.**

4.1.1. *Version Store & Forward.* Si les communications entre les différents processeurs ont lieu suivant le modèle *Store & Forward* (chaque processeur attend d'avoir reçu et stocké un message avant de le transmettre à son voisin), le temps nécessaire à la réalisation d'un produit de deux matrices de taille $N \times N$ est alors le suivant : $F_{SF}(N) = \left(\alpha \frac{N^3}{p^2} + \beta \frac{N^2}{p} + \gamma N + \delta p\right) + 2p^2 t_{start} + 2N^2 t_{transf} + \beta \frac{N^2}{p}$, avec $\left(\alpha \frac{N^3}{p^2} + \beta \frac{N^2}{p} + \gamma N + \delta p\right) = p\left(\alpha\left(\frac{N}{p}\right)^2 + \beta\left(\frac{N}{p}\right)^2 + \gamma \frac{N}{p} + \delta\right)$ représentant le temps nécessaire pour faire p produits de deux matrices de taille $\frac{N}{p}$, $2p^2 t_{start}$ les latences correspondant aux $2p^2$ communications entre les processeurs suivant les deux directions de la grille, $2N^2 t_{transf} = 2p^2 \left(\frac{N}{p}\right)^2 t_{transf}$ les temps de transfert des données (matrices de taille $\frac{N}{p}$) entre les processeurs, et $\beta \frac{N^2}{p} = p\beta\left(\frac{N}{p}\right)^2$ le temps nécessaire pour additionner les p matrices intermédiaires construites lors de l'algorithme de Fox.

La complexité de l'algorithme devient alors :

$$\boxed{\begin{aligned} C_{SF} \;=\; & \alpha \frac{n^3}{p^2}\left(\frac{7}{8}\right)^l + \left(5\, t_{add}\frac{n^2}{p^2} + 2\beta\frac{n^2}{p} + 2\, t_{transf} n^2\right)\left(\frac{7}{4}\right)^l \\ & + \gamma n \left(\frac{7}{2}\right)^l + \left(\delta p + 2\, t_{start} p^2\right) 7^l - 5\, t_{add}\frac{n^2}{p^2} \end{aligned}} \tag{6}$$

On trouve alors la valeur optimale de l de manière analogue au cas thèorique de l'algorithme de Fox, en dèrivant l'expression par rapport àa l, en posant $X = 2^l$ et en résolvant une équation du troisième degré. On remplace alors l par sa valeur optimale dans la formule (2), et on obtient la complexité de l'algorithme uniquement en fonction de n, p et des différents paramètres internes de la machine sur laquelle on se place.

4.1.2. *Version Worm-Hole.* Un autre type de communications entre les différents processeurs est basé sur le modèle *Worm-Hole*, où chaque processeur transmet à son voisin un paquet tout en recevant le paquet suivant. Dans ce cas, le temps nécessaire pour effectuer la multiplication des deux matrices est : $F_{WH}(N) = \left(\alpha \frac{N^3}{p^2} + \beta \frac{N^2}{p} + \gamma N + \delta p\right) + 2pt_{start} + 2\frac{N^2}{p} t_{transf} + \beta \frac{N^2}{p}$. La différence avec la complexité du même calcul dans le cas *Store & Forward* est le facteur p devant les termes en t_{start} et t_{transf} puisqu'il n'y a plus qu'un seul terme de latence et de transfert par rangée de processeurs (on n'effectue qu'une seule communication par rangée de processeurs). La nouvelle complexité de l'algorithme de Strassen (avec la variante de Winograd) est alors la suivante :

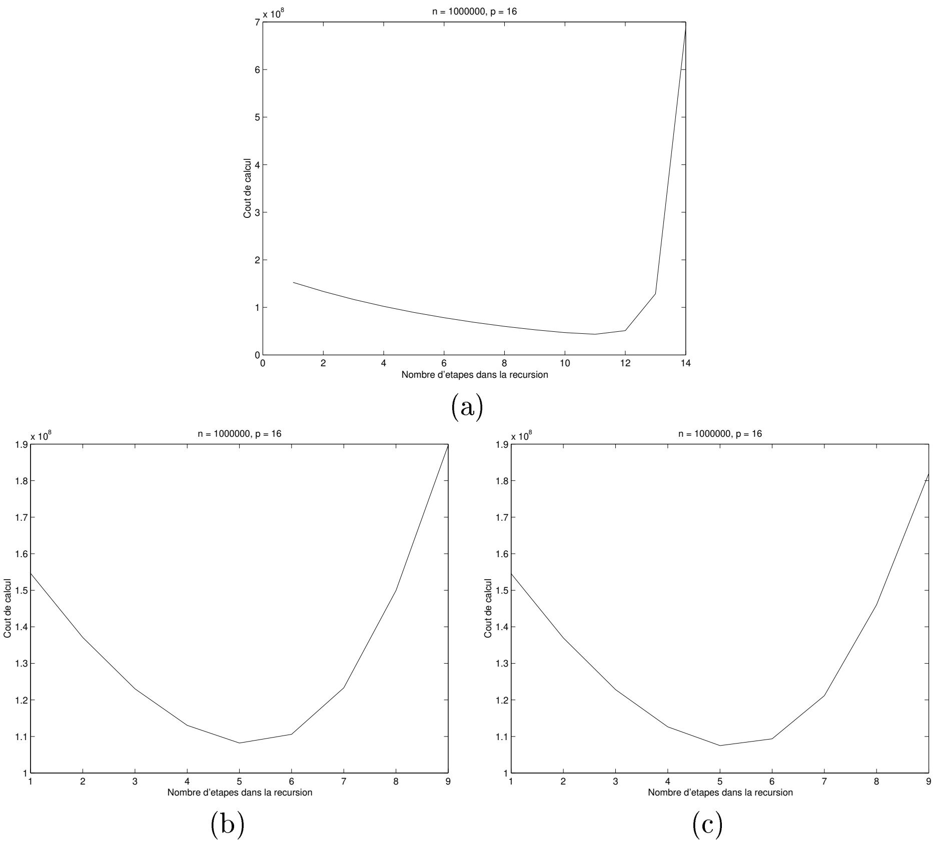


FIG. 4. Paragon : Évolution du temps de calcul en fonction de l, avec $n = 10^6$ et $p = 2^4$, dans le cas théorique (a), Store & Forward (b) et Worm-Hole (c)

$$\boxed{\begin{aligned} C_{WH} \;=\;& \alpha \frac{n^3}{p^2}\left(\frac{7}{8}\right)^l + \left(5t_{add}\frac{n^2}{p^2} + 2\beta\frac{n^2}{p} + 2t_{transf}\frac{n^2}{p}\right)\left(\frac{7}{4}\right)^l \\ & + \gamma n\left(\frac{7}{2}\right)^l + (\delta p + 2t_{start}p)\,7^l - 5t_{add}\frac{n^2}{p^2} \end{aligned}} \qquad (7)$$

On trouve la valeur optimale de l de la même manière que précédemment. Nous allons maintenant appliquer ces résultats sur 3 machines différentes.

4.2. **Paragon.** Cette machine fabriquée par Intel est composé de processeurs i860XP à 50 MHz effectuant 75 MFLOPS chacun. Ils ont chacun 16 ou 32 Mo de mémoire vive et sont interconnectés en grille (Worm-Hole en pratique). Le nombre de processeurs est variable d'une machine à l'autre mais peut dépasser 200. Avec 228 processeurs de calcul, la performance de crête est de 21.6 GFLOPS. Sur cette machine, les valeurs (données en secondes) utilisées ont été les suivantes[2] :

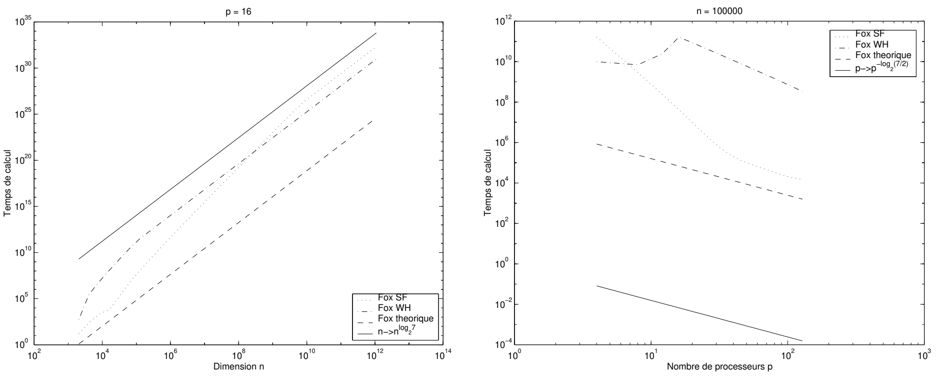
$\alpha = 4.465\,10^{-8}$	$\gamma = -5.39\,10^{-4}$	$t_{add} = 2.233\,10^{-8}$	$t_{start} = 6.42\,10^{-5}$
$\beta = 8.908\,10^{-6}$	$\delta = 0.0$	$\tau = 2.233\,10^{-8}$	$t_{transf} = 2.25\,10^{-8}$

Le rapport α/t_{transf} représente grosièrement le rapport entre les capacités de calcul et de communication de la machine et vaut ici approximativement 2.

4.2.1. *Dépendance en l.* Les valeurs de n et p ont tout d'abord été respectivement fixées à 10^6 et 2^4, et l'évolution du temps de calcul en fonction de l est représentée sur la figure 4.

On constate qu'il y a relativement peu d'écart entre les courbes correspondant aux modèles Store & Forward, et Worm-Hole. Dans tous les cas, on voit apparaître clairement un minimum, et donc une valeur optimale de l. Si on dépasse cette valeur, la complexité peut augmenter de façon désastreuse. Ceci prouve qu'il est important de ne pas pousser trop loin la récursion, ce qui entraînerait beaucoup trop de communications entre les processeurs. A l'inverse, il ne faut pas non plus se contenter de trop peu de niveaux de récursion, ce qui reviendrait quasiment

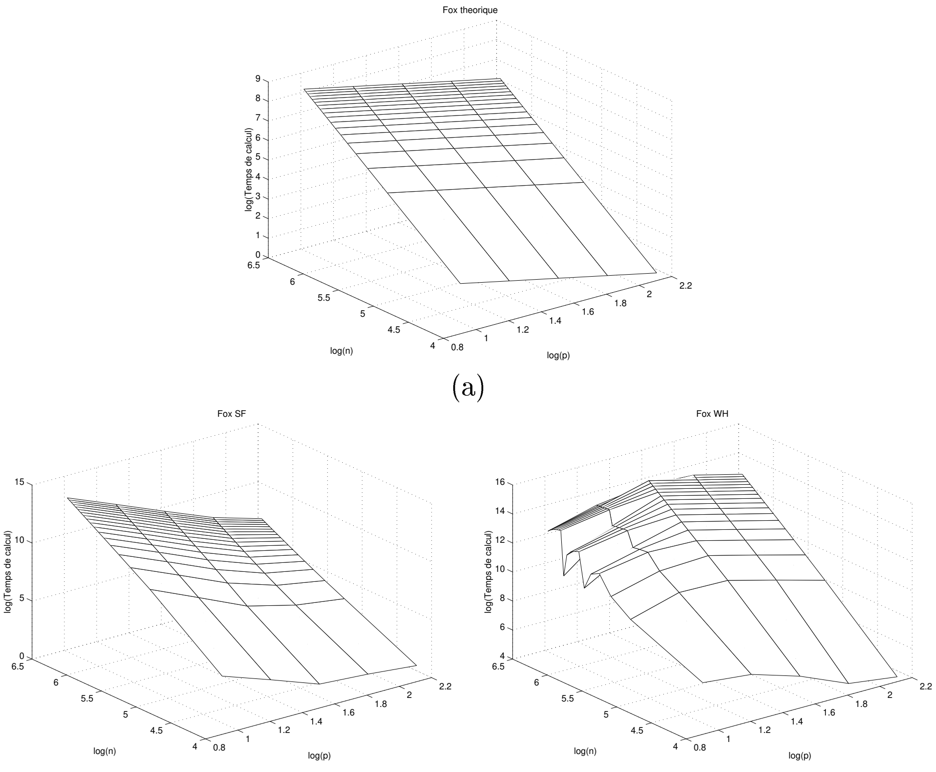
[2] Merci à Stéphane Domas pour nous avoir fourni ces données ainsi qu'un exemplaire de sa thèse.



(a) (b)

FIG. 5. Paragon : Évolution du temps de calcul en fonction de n avec $p = 2^4$ (a) et en fonction de p avec $n = 10^5$ (b), dans le cas où l est optimal

à utiliser l'algorithme classique de multiplication de matrices en $\mathcal{O}(n^3)$. Le calcul de l_{opt} est donc une étape primordiale dans cet algorithme.



(a) (b) (c)

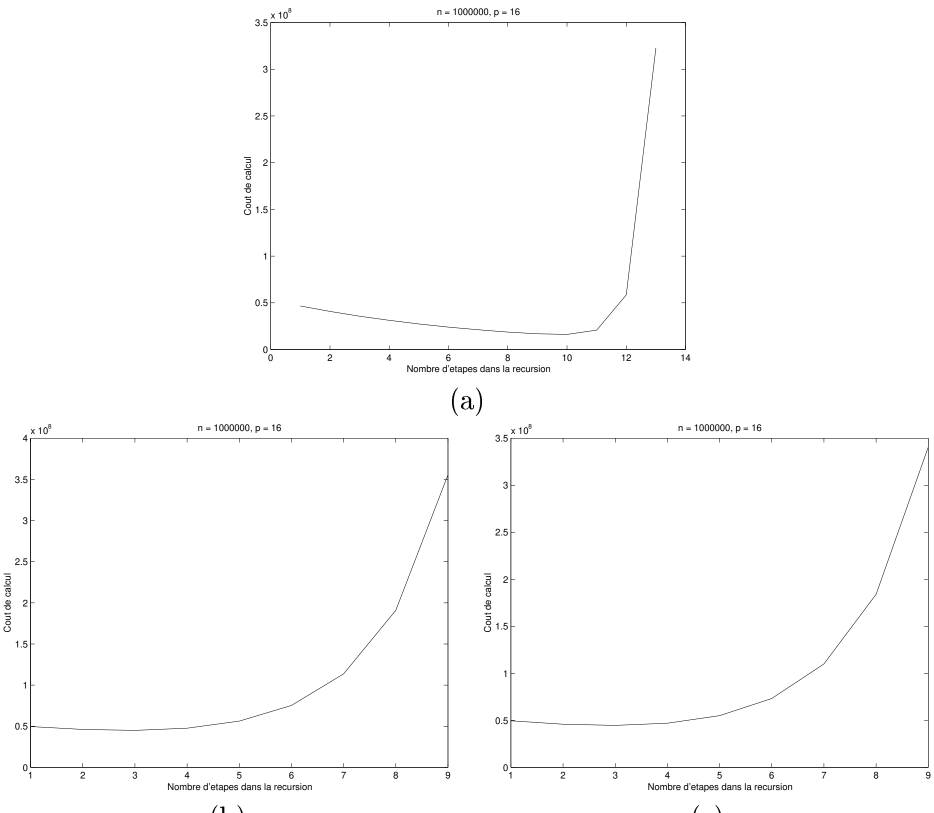
FIG. 6. Paragon : Évolution du temps de calcul en fonction de n et p, l étant toujours choisi de façon optimale, dans le cas théorique (a), Store & Forward (b) et Worm-Hole (c)

4.2.2. *Comportement asymptotique.* Dans la suite, la valeur de l sera considérée comme optimale. L'évolution du temps de calcul en fonction de n est représentée en échelle logarithmique sur la figure 5 (a) pour p fixé à 2^4. Inversement, la figure 5 (b) représente la variation du temps de calcul en fonction de p lorsque n est fixé à 10^5.

Nous utilisons deux droites de référence sur les figures 5 (a) et (b), correspondant aux deux asymptotes en n et p de la complexité de l'algorithme de Strassen en utilisant l'algorithme théorique de Fox. Nous constatons que l'algorithme théorique de Fox a bien un comportement en $C^{te}\frac{n^{\log_2(7)}}{p^{\log_2(7)-1}}$, et que les versions Store & Forward et Worm-Hole tendent approximativement vers le même comportement asymptotique.

4.2.3. *Dépendance en n et p.* L'évolution couplée en fonction de n et p de la complexité du problème est représentée (en double échelle logarithmique) sur la figure 6.

Nous avons, pour des valeurs plus générales de n et p, globalement le même comportement des algorithmes étudiés que sur la figure 5. Même si le comportement asymptotique semble être celui prédit en section 3, il faut noter quelques changements de pentes et quelques irrégularités, notamment pour la version Worm-Hole (figure 6 (c)). Une explication très probable de ce phénomène est le suivant : la valeur optimale de l est calculée de façon algébrique en résolvant une équation du troisième degré (à l'aide des formules de Cardan) mais il est évident que l doit être compris entre 0 et $\lfloor \log_2(n) \rfloor$. Or il arrive que les racines de cette équation ne se situent pas dans cet intervalle. La valeur l_{opt} optimale de l n'est alors pas solution de cette équation, mais est tout de même optimale parmi les valeurs possibles de l. De plus, dans tous les cas, il faut arrondir la valeur trouvée à un entier (partie entière inférieure ou supérieure), ce qui crée une discontinuité supplémentaire dans le calcul de l_{opt} par rapport à n et p.



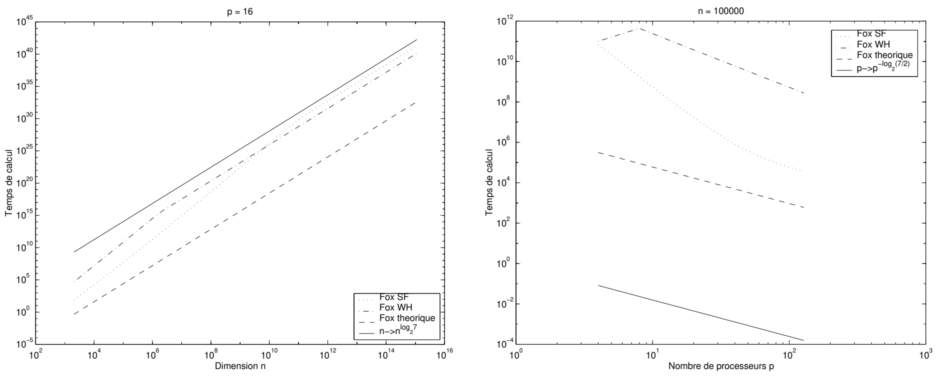
(b) (c)

FIG. 7. SP2 : Évolution du temps de calcul en fonction de l, avec $n = 10^6$ et $p = 2^4$, dans le cas théorique (a), Store & Forward (b) et Worm-Hole (c)

4.3. **SP-2.** L'IBM SP2 est une machine parallèle à base de processeurs RS/6000 interconnectés par un switch. De même que pour la Paragon, le nombre de processeurs est très variable d'une version à l'autre. Sur cette machine, les valeurs utilisées ont été les suivantes :

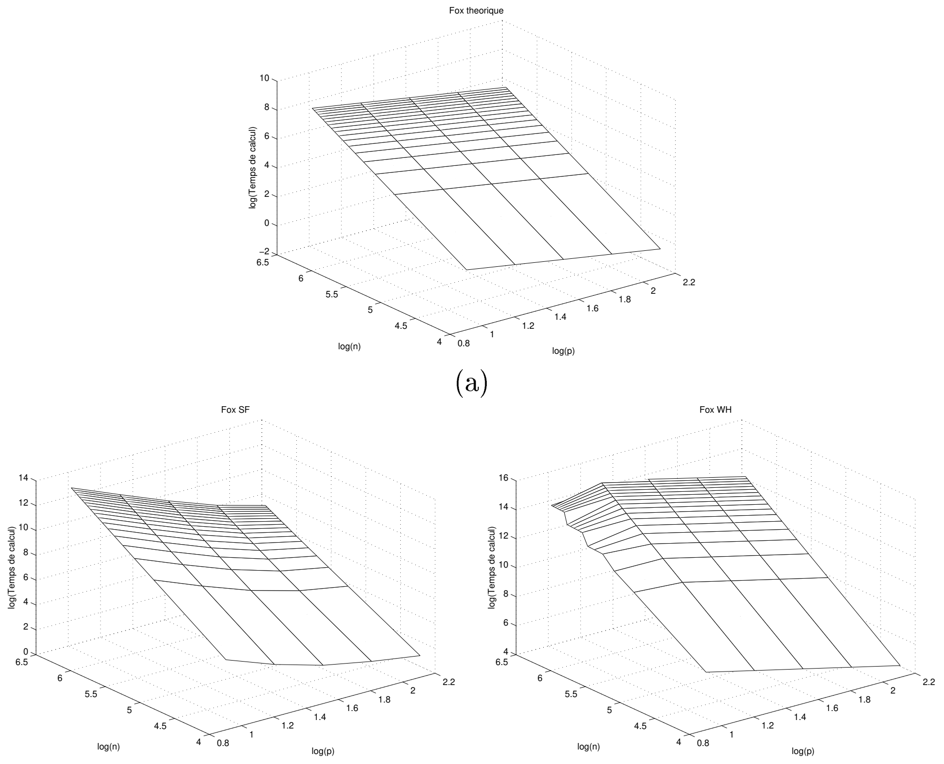
$\alpha = 1.36394\,10^{-8}$	$\gamma = 8.7240\,10^{-4}$	$t_{add} = 6.8197\,10^{-9}$	$t_{start} = 2.1077\,10^{-4}$
$\beta = 1.3244\,10^{-5}$	$\delta = 0.0$	$\tau = 6.8197\,10^{-9}$	$t_{transf} = 3.7592\,10^{-8}$

Le rapport α/t_{transf} représentant le rapport entre les capacités de calcul et de communication de la machine vaut ici approximativement 0.36.



(a) (b)

FIG. 8. SP2 : Évolution du temps de calcul en fonction de n avec $p = 2^4$ (a) et en fonction de p avec $n = 10^5$ (b), dans le cas où l est optimal



(b) (c)

FIG. 9. SP2 : Évolution du temps de calcul en fonction de n et p, l étant toujours choisi de façon optimale, dans le cas théorique (a), Store & Forward (b) et Worm-Hole (c)

4.3.1. *Dépendance en l.* Comme pour la machine Paragon, les valeurs de n et p ont tout d'abord été respectivement fixées à 10^6 et 2^4, et l'évolution du temps de calcul en fonction de l est représentée sur la figure 7. Un minimum apparaît assez clairement sur chacune des trois courbes. Contrairement à la machine précédente, ici les meilleurs résultats sont obtenus pour de petites valeurs de l, et nous avons relativement peu d'écart entre le coût de calcul pour $l = 1$ et celui pour $l = 7$. Ceci est probablement dû au fait que le rapport calcul/communication de cette machine est bien inférieur à ceux de la machine précédente.

4.3.2. *Comportement asymptotique.* l est désormais choisi pour optimiser l'algorithme. On fixe de nouveau p à 2^4 et on fait varier n. L'évolution est représentée sur la figure 8 (a). On fixe alors $n = 10^5$ et on fait varier p. On obtient ainsi la figure 8 (b).

Ces courbes sont représentées en double échelle logarithmique, afin de pouvoir les tracer sur d'assez grands intervalles. Nous avons des résultats similaires à ceux obtenus sur la machine précédente, à ceci près que la convergence des courbes vers leurs asymptotes semble légèrement plus rapide.

4.3.3. *Dépendance en n et p.* L'évolution couplée en fonction de n et p de la complexité du problème est représentée (en double échelle logarithmique) sur la figure 9.

Nous obtenons des résultats similaires à la fois à ceux obtenus avec la machine précédente, et à ceux obtenus dans le cas particulier de la figure 8. Nous avons également la confirmation d'un meilleur comportement du temps de calcul pour des petites valeurs de p, ce que laissait supposer la courbe 8 (b). Toutefois, il y a encore quelques petites irrégularités lorsque p est petit, que l'on peut expliquer par le calcul de l_{opt}, qui ne varie pas continûment avec n et p pour les raisons invoquées au paragraphe 4.2.3.

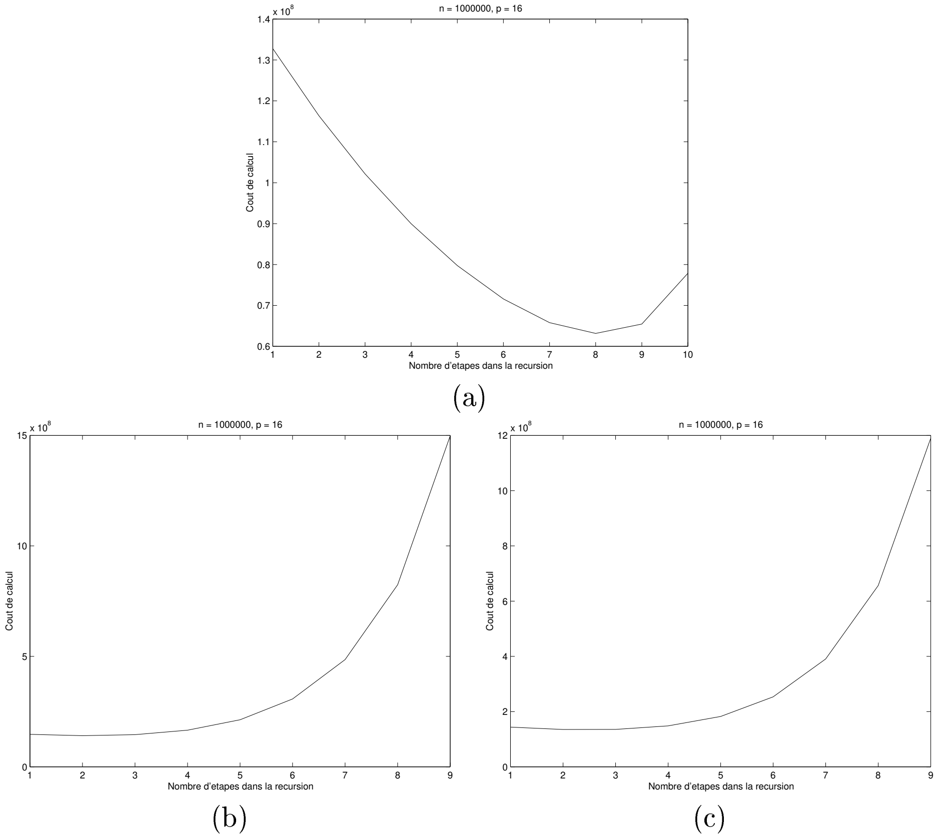


FIG. 10. PoPC : Évolution du temps de calcul en fonction de l, avec $n = 10^6$ et $p = 2^4$, dans le cas théorique (a), Store & Forward (b) et Worm-Hole (c)

4.4. **PoPC.** La PoPC (Pile of PC) est une machine constituée de processeurs Pentium Pro 200 interconnectés par un switch myrinet. Cependant, les valeurs de bande passante et de latence correspondent à un réseau de type Ethernet[3], ce qui nous donne des valeurs correspondant à un réseau de PCs classique :

$\alpha = 3.878119\,10^{-9}$	$\gamma = 2.1819\,10^{-3}$	$t_{add} = 1.9391\,10^{-9}$	$t_{start} = 10^{-3}$
$\beta = 4.9467\,10^{-5}$	$\delta = 0.0$	$\tau = 1.9391\,10^{-9}$	$t_{transf} = 10^{-6}$

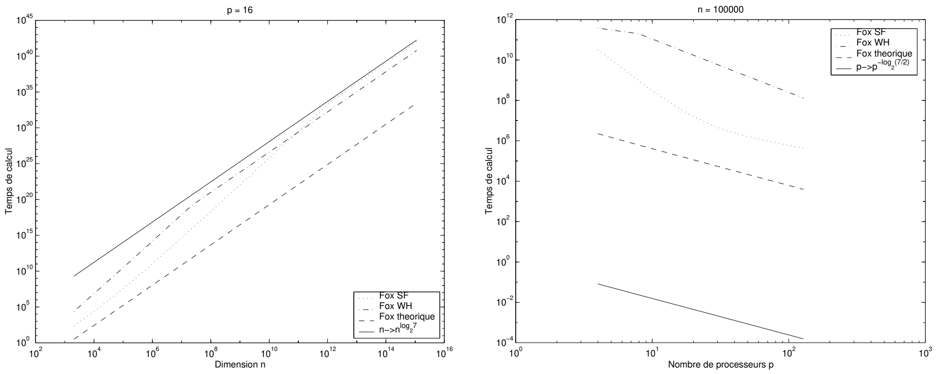
[3]À l'époque des mesures, les librairies tirant parti de myrinet étaient en cours de développement. On notera que sur ce type de réseau, la diffusion en Worm-Hole se fait pour le même prix que le Store & Forward puisqu'aucune des diffusion ne peut se faire en parallèle. Nous n'en tenons pas compte et effectuons les calculs suivants en supposant que l'architecture d'interconnexion est une grille …

Le rapport α/t_{transf} représentant le rapport entre les capacités de calcul et de communication de la machine vaut ici approximativement 0.0038.

4.4.1. *Dépendance en l.* On fixe de nouveau les valeurs de n et p à 10^6 et 2^4 respectivement, et l'évolution du temps de calcul en fonction de l est représentée sur la figure 10.

Nous observons encore une certaine différence entre la courbe obtenue pour l'algorithme de Fox théorique et celles obtenues pour les versions pratiques (Store & Forward, et Worm-Hole), ces deux dernières étant une fois de plus très comparables. Mais dans les trois cas, nous pouvons voir qu'il y a une valeur de l meilleure que les autres (même si pour les versions pratiques de l'algorithme de Fox, la complexité change peu pour les petites valeurs de l).

4.4.2. *Comportement asymptotique.* l étant fixé (optimal), la complexité ne dépend plus que de n et p. On fixe alors p à 2^4 et on fait varier n. L'évolution est représentée sur la figure 11 (a).



(a) (b)

FIG. 11. PoPC : Évolution du temps de calcul en fonction de n avec $p = 2^4$ (a) et en fonction de p avec $n = 10^5$ (b), dans le cas où l est optimal

On suppose maintenant $n = 10^5$ et on fait varier p. On obtient ainsi la figure 11 (b).

Une fois de plus, les résultats obtenus sont similaires à ceux obtenus pour les machines précédentes, à savoir une convergence des courbes vers les asymptotes de pentes respectives $\log_2(7)$ et $\log_2(7) - 1$.

4.4.3. *Dépendance en n et p.* L'évolution couplée en fonction de n et p de la complexité du problème est représentée (en échelle logarithmique) sur la figure 12.

Ces surfaces nous montrent que le comportement du temps de calcul est globalement le même pour des valeurs petites ou grandes de n et p. De plus, les irrégularités constatées dans le cas Worm-Hole pour les petites valeurs de p (pour les machines précédentes) ont quasiment disparues.

5. Conclusion

Nous avons obtenu de façon théorique plusieurs résultats sur la complexité de l'algorithme de Strassen-Winograd utilisé en parallèle, notamment sur son comportement asymptotique. L'étude pratique qui a suivi, sur 3 machines relativement différentes du point de vue de leurs rapport capacité de calcul/capacité de communication, et les résultats obtenus confirment globalement l'étude théorique préliminaire. De plus, il y a relativement peu de différences de comportement de la complexité de l'algorithme sur ces 3 machines.

Nous avons montré que l'algorithme de Strassen-Winograd a une complexité asymptotique en $C^{te}\frac{n^{\log_2(7)}}{p^{\log_2(7)-1}}$, et des mesures pratiques nous fournissent des approximations de la constante :

Constante	Fox théorique	Fox Store & Forward	Fox Worm-Hole
Paragon	$9.37\ 10^{-8}$	$2.87\ 10^{-5}$	$\mathbf{1.05\ 10^{-5}}$
SP2	$3.53\ 10^{-8}$	$1.02\ 10^{-5}$	$\mathbf{3.89\ 10^{-6}}$
PoPC	$2.33\ 10^{-7}$	$\mathbf{4.83\ 10^{-4}}$	$1.11\ 10^{-4}$

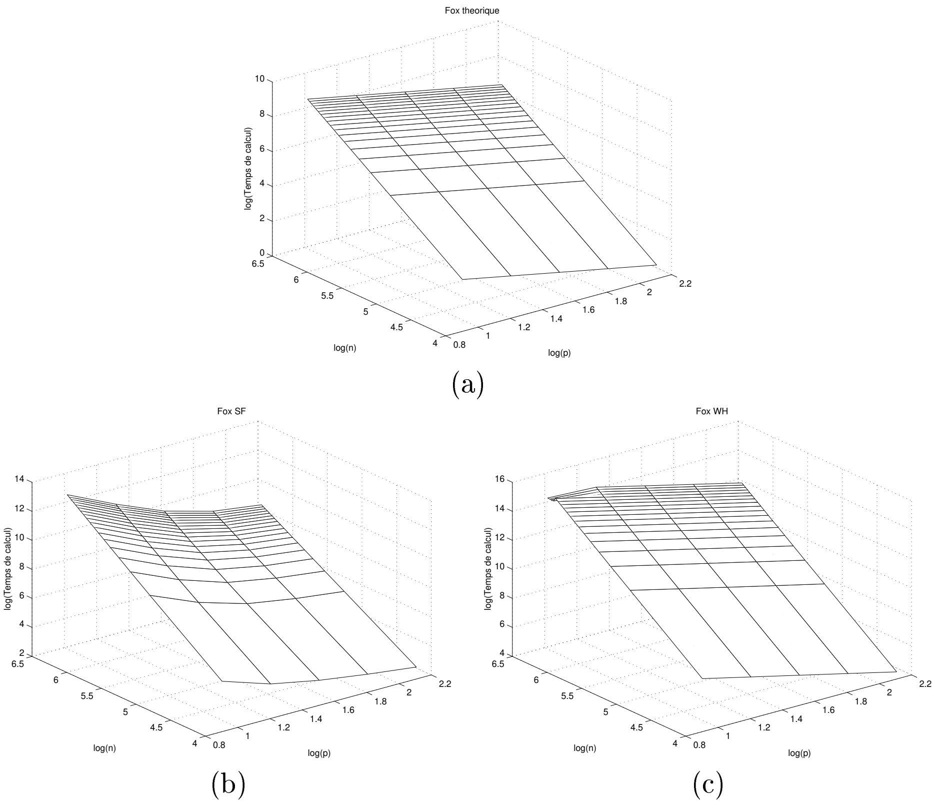


FIG. 12. PoPC : Évolution du temps de calcul en fonction de n et p, l étant toujours choisi de façon optimale, dans le cas théorique (a), Store & Forward (b) et Worm-Hole (c)

La meilleure des 3 machines étudiées est donc l'IBM SP2 (qui est une machine aux processeurs plus rapides que la Paragon et interconnectés par un réseau aussi rapide), suivie par Paragon puis PoPC (certainement limité par son réseau d'interconnexion Ethernet), même si encore une fois ces données ne sont à prendre en compte que pour de grandes valeurs de n et p.

Il peut enfin être intéressant de comparer les temps nécessaires pour effectuer le produit de deux matrices de taille n avec plusieurs algorithmes. Par exemple avec $n = 10^6$ (soit près de 4 To) et $p = 128$ (ce qui est aussi énorme), on obtient les temps de calcul suivants :

Algorithme classique (séquentiel)	Fox Seul	Strassen parallèle
1400 années	32 jours	11 jours

On a donc une différence sensible entre le temps d'exécution de l'algorithme de Fox et la version parallélisée de Strassen. Plus la taille de la matrice est importante, plus il devient intéressant d'utiliser la version parallélisée de Strassen par rapport à l'algorithme de Fox seul et cette tendance s'inverse quand le nombre de processeur augmente.

Une étude théorique plus poussée pourrait donc permettre de déterminer à partir de quelle taille de matrices, il est intéressant de passer de l'algorithme de Fox à celui de Strassen, puis Strassen-Winograd en fonction du nombre de processeurs disponibles puisque l'étude réalisée ici n'a été quasiment qu'asymptotique.

Nous avons également montré quelles étaient les étapes importantes de la mise en œuvre de cet algorithme de façon parallèle : tout d'abord le calcul de la profondeur optimale de récursion, dont nous avons vu qu'elle influait beaucoup sur le temps de calcul, la gestion de la mémoire et l'optimisation du stockage des données et des calculs intermédiaires, qui permettent de réduire considérablement la quantité de mémoire nécessaire sur chaque processeur, et enfin nous avons montré que l'efficacité, même si elle tend vers 0 lorsque p croît, est tout à fait intéressante.

Références

[FP74] P.C. Fischer et R.L. Probert, *Efficient procedures for using matrix algorithms*, Automata, Languages and Programming (Springer Verlag, ed.), Lecture Notes in Computer Science, no. 14, 1974, pp. 413–427.

[GSG95] Brian Grayson, Ajay Pankaj Shah et Robert A. van de Geijn, *A high performance parallel strassen implementation*, Tech. report, University of Texas, June 1995.

[Pro73] R. Probert, *On the complexity of matrix multiplication*, Tech. Report CS-73-27, Dept of Applied Analysis and Computer Science, University of Waterloo, 1973.

[Str69] V. Strassen, *Gaussian elimination is not optimal*, Numer. Math., no. 13, 1969, pp. 354–356.

[Win71] S. Winograd, *On multiplication of* 2×2 *matrices*, Linear Algebra and its Applications, vol. 4, 1971, pp. 381–388.

E-mail address: `[Didier.Auroux,Arnaud.Legrand]@ens-lyon.Fr`