

**Cours : “Parallélisme”
Travaux dirigés
E. Goubault, S. Putot**

TD 2

15 décembre 2014

1 Récurrences linéaires

Le problème qui nous préoccupe dans cette section est de calculer efficacement, en parallèle, des suites récurrentes linéaires d'ordre $m \geq 1$, de la forme (pour $i \geq 0$):

$$\begin{aligned} y_0 &= a_0^0 \\ \dots &\dots \dots \\ y_{m-1} &= a_0^{m-1} \\ y_{m+i} &= a_m^{m+i} y_{m+i-1} + \dots + a_1^{m+i} y_i + a_0^{m+i} \end{aligned}$$

Comme applications visées, on a par exemple:

- l'évaluation d'un polynôme $p(x) = a_0 + a_1x + \dots + a_kx^k$ par la méthode de Horner: on fait $y_0 = a_k$, puis $y_{i+1} = xy_i + a_{k-i-1}$ (récurrence linéaire d'ordre 1); alors $p(x) = y_k$.
- la résolution de systèmes linéaires par bandes: soit par exemple

$$A = \begin{pmatrix} a_{1,1} & 0 & 0 & 0 & \dots & \dots & 0 \\ a_{2,1} & a_{2,2} & 0 & 0 & \dots & \dots & 0 \\ a_{3,1} & a_{3,2} & a_{3,3} & 0 & & & \vdots \\ 0 & a_{4,2} & a_{4,3} & a_{4,4} & & & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ & & & 0 & a_{n,n-2} & a_{n,n-1} & a_{n,n} \end{pmatrix}$$

La solution de l'équation $Ax = b$ peut être trouvée par la récurrence suivante: $x_1 = \frac{b_1}{a_{1,1}}$, $x_2 = \frac{1}{a_{2,2}}(b_2 - a_{2,1}x_1)$,

$$x_i = \left(\sum_{j=i-m}^{i-1} -\frac{a_{i,j}}{a_{i,i}} x_j \right) + \frac{b_i}{a_{i,i}}$$

pour $3 \leq i \leq n$.

1.1 Récurrences d'ordre 1

On se place dans le cas de récurrences linéaires d'ordre 1 ($m = 1$): c'est à dire,

$$\begin{aligned} y_0 &= a_0^0 \\ y_{i+1} &= a_1^{i+1} y_i + a_0^{i+1} \end{aligned}$$

et on veut calculer y_i pour les indices $0 \leq i \leq n - 1$ (pour un n donné, que l'on pourra supposer être une puissance de 2).

Question On suppose que a_1^i est toujours égal à un. Comment calculer efficacement la suite des y_i , $0 \leq i \leq n - 1$ sur une machine PRAM EREW, CREW, CRCW? Quelle sera l'efficacité de l'algorithme?

Corrigé: Quand les a_1^i sont tous égaux à un, il s'agit de calculer toutes les sommes partielles des a_0^j , pour $j = 0, \dots, i$, sur le processeur P_i . Il s'agit donc d'utiliser la technique de saut de pointeur: sur une PRAM EREW, CREW et CRCW on a un calcul en temps $O(\log n)$ utilisant un total de $O(n)$ opérations sur n processeurs. L'efficacité est donc de l'ordre de $O(\frac{1}{\log n})$. Par le théorème de simulation de Brent, on aurait pu simuler cet algorithme avec $O(\frac{n}{\log(n)})$ processeurs, et ainsi atteindre une efficacité de 1.

Question On se place maintenant dans le cas d'une récurrence linéaire d'ordre 1 générale, c'est à dire que les a_1^i sont maintenant quelconques. Calculer y_i en fonction de y_{i-2} .

Corrigé:

$$y_i = a_1^i a_1^{i-1} y_{i-2} + a_1^i a_0^{i-1} + a_0^i$$

Question On suppose maintenant $n = 2^k$. A l'aide de la question précédente, comment modifier l'algorithme de la question 1 pour calculer efficacement la suite des y_i , $0 \leq i \leq n - 1$ sur une machine PRAM CREW, en utilisant la technique de saut de pointeur? On donnera un pseudo-algorithme sous la même forme que les algorithmes donnés dans le polycopié (chapitre PRAM). Quelle est l'efficacité de cet algorithme? Justifier brièvement.

Corrigé: Moralement, on organise n processeurs P_i , $0 \leq i \leq n - 1$ en liste chaînée, P_{i+1} pointant vers P_i . Au début, on associe la valeur $p[i]$ à chaque P_i , avec $p[0] = a_0^0$ et $p[i] = 0$ si $i > 0$, ainsi que les coefficients $a[i] = a_1^i$ si $i > 0$ (0 si $i = 0$) et $b[i] = a_0^i$ si $i \geq 0$. Ensuite on itère le pseudo-code suivant:

```
forall proc i s. t. next[i] != nil in parallel {
  p[i] = _i a[i]*p[next[i]] + b[i];
  a[i] = _i a[i]*a[next[i]];
  b[i] = _i a[i]*b[next[i]] + b[i];
  next[i] = next[next[i]];
}
```

Alors au bout de k itérations, $p[n - 1]$ vaut y_{n-1} . On peut le prouver par récurrence, en montrant qu'à l'étape $1 \leq e \leq k$, les y_i , $0 \leq i < 2^e$ sont calculés. Par exemple pour $k = 2$ on a les étapes suivantes:

- Etape 1: $p[0] = a_0^0$, $p[1] = a_1^1 a_0^0 + a_0^1$, $p[2] = a_0^2$, $p[3] = a_0^3$ et $a[1] = 0$, $a[2] = a_1^2 a_1^1$, $a[3] = a_1^3 a_1^1$. Enfin, $b[1] = a_1^1 a_0^0 + a_0^1$, $b[2] = a_1^2 a_0^1 + a_0^2$, $b[3] = a_1^3 a_0^2 + a_0^3$ et $next[1] = nil$, $next[2] = 0$, $next[3] = 1$.
- Etape 2: $p[2] = a_1^2 a_1^1 a_0^0 + a_1^2 a_0^1 + a_0^2 = y_2$,
 $p[3] = a_1^3 a_1^2 (a_1^1 a_0^0 + a_0^1) + a_1^3 a_0^2 + a_0^3$
 $= a_1^3 a_1^2 a_1^1 a_0^0 + a_1^3 a_1^2 a_0^1 + a_1^3 a_0^2 + a_0^3$
 $= y_3$

On a aussi: $a[2] = 0$, $a[3] = 0$, $b[2] = a_1^2 a_1^1 a_0^0 + a_1^2 a_0^1 + a_0^2$, $b[3] = a_1^3 a_1^2 (a_1^1 a_0^0 + a_0^1) + a_1^3 a_0^2 + a_0^3$, et tous les $next[i]$ sont à nil .

Cet algorithme prend un temps $O(\log(n))$ et utilise $O(n)$ opérations arithmétiques, et n processeurs. L'efficacité est donc de l'ordre de $O(\frac{1}{\log(n)})$. Par le théorème de simulation de Brent, on aurait pu transformer cet algorithme afin qu'il utilise seulement $O(\frac{n}{\log(n)})$ processeurs, et on aurait pu ainsi atteindre une efficacité de 1.

1.2 Récurrence linéaire d'ordre supérieur

On se place maintenant dans le cas général $m \geq 1$, et on suppose disposer d'un algorithme PRAM permettant de faire la multiplication de deux matrices carrées $m \times m$ en temps $O(\log(m))$, et utilisant $O(M(m))$ opérations (on pourra supposer ici $M(m) = O(m^{2.376})$) sur $O(m^3)$ processeurs.

Question: Comment améliorer l'algorithme de la question 3 afin de calculer y_i , $0 \leq i \leq n-1$, sur une PRAM, dans le cas d'une récurrence d'ordre m ? Quelle est l'efficacité de l'algorithme sur une PRAM CREW? Justifier brièvement.

Correction:

Il suffit de poser:

$$\bullet b_0 = \begin{pmatrix} a_0^0 \\ a_0^1 \\ \dots \\ a_0^{m-1} \end{pmatrix}, b_i = \begin{pmatrix} 0 \\ 0 \\ \dots \\ a_0^{m+i-1} \end{pmatrix} \text{ (pour } i \geq 1), Y_i = \begin{pmatrix} y_i \\ y_{i+1} \\ \dots \\ y_{i+m-1} \end{pmatrix}$$

$$\bullet A_i = \begin{pmatrix} 0 & 1 & \dots & 0 \\ 0 & 0 & \dots & 0 \\ \dots & & & \\ 0 & 0 & \dots & 1 \\ a_1^{m+i} & a_2^{m+i} & \dots & a_m^{m+i} \end{pmatrix}$$

Ainsi, la récurrence d'ordre m revient à la récurrence linéaire d'ordre un, dans un espace m -dimensionnel:

$$\begin{aligned} Y_0 &= b_0 \\ Y_i &= A_i Y_{i-1} + b_i \quad 1 \leq i < n \end{aligned}$$

On utilise alors exactement le même algorithme qu'à la question précédente. Mais maintenant, les multiplications n'ont plus un coût unitaire mais en $O(\log(m))$. Donc le temps d'exécution parallèle sera de $O(\log(n)\log(m))$. Le nombre d'opérations arithmétiques (donc la complexité séquentielle) est maintenant non plus de l'ordre de n , mais de nm . On a besoin maintenant de l'ordre de nm^3 processeurs. Ainsi, l'efficacité est de l'ordre de $\frac{nm}{\log(n)\log(m)nm^3} = \frac{1}{\log(n)} \frac{1}{m^2 \log m}$.