

Cours : “Calcul Parallèle”
Travaux dirigés
E. Goubault & T. Heindel

TD 2

28 janvier 2013

1 Ordonnanceur à tourniquet

On cherche à implémenter un ordonnanceur à tourniquet pour les threads JAVA en JAVA.

L'idée est que l'on doit définir un thread ordonnanceur qui gère une file d'attente de threads “utilisateurs” JAVA et les ordonner un à un pour un quantum de temps élémentaire donné (une constante de votre programme). Cet ordonnanceur ne gèrera pas de notion de priorité.

Chaque thread utilisateur devra s'enregistrer auprès de l'ordonnanceur en appelant la méthode `addThread()` (à écrire). On pourra aussi écrire une méthode `removeThread` qui désinscrit un thread auprès de cet ordonnanceur. On utilisera la classe `CircularList` donnée sur la page web du cours, qui fournit une implémentation de liste circulaire, avec les méthodes:

- `public synchronized void insert(Object o)` d'insertion d'un objet dans la file.
- `public synchronized void delete(Object o)` qui efface l'objet de la liste.
- `public synchronized Object getNext()` qui prend l'élément suivant dans la file.

Avant de commencer, on répondra aux questions suivantes: quelle doit être la priorité du thread ordonnanceur? du thread utilisateur actif? des threads non encore actifs?

2 Tri pair-impair

On suppose que l'on dispose d'un réseau linéaire de p processeurs $P_1, \dots, P_p$, c'est à dire que les processeurs sont organisés sur une ligne, et que chacun peut communiquer avec son voisin de gauche et son voisin de droite s'il en a un. On veut trier n entiers, avec p qui divise n . Chaque processeur a au départ $\frac{n}{p}$ entiers, et on veut trouver un algorithme parallèle qui permette de faire en sorte que la sous-suite sur P_i soit ordonnée et inférieure à la sous-suite ordonnée P_j , pour tout $i < j$. C'est à dire tout simplement que l'on veut trier en parallèle les données réparties sur les processeurs P_i .

En s'inspirant du réseau de tri pair-impair vu en cours, écrire un tel algorithme en JAVA. Pour ce faire, on utilisera la classe `Buffer` suivante, pour coder les échanges de données entre chaque processus, chacun implémentant un comparateur du réseau de tri.

```
class Buffer {
    int val;
    Buffer() { val = -1; }

    synchronized void write(int v) {
```

```

    while (val!=-1) {
        try { wait(); } catch (InterruptedException e) {};
    };
    val = v;
    notifyAll();
}

synchronized int read() {
    while (val==-1) {
        try { wait(); } catch (InterruptedException e) {};
    };
    int n = val;
    val = -1;
    notifyAll();
    return n;
}
}

```

3 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: $x_1 = \frac{b_1}{a_{1,1}}$, $x_2 = \frac{1}{a_{2,2}}(b_2 - a_{2,1}x_1)$,

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

pour $3 \leq i \leq n$.

3.1 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 1** 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?

☞ **Question 2** 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} .

☞ **Question 3** 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.

3.2 Récurrence linéaire d'ordre supérieur

On se place maintenant dans le cas général $m \geq 1$, et on suppose disposer d'un algorithme PRAM CREW 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 4** Comment améliorer l'algorithme de la question précédente 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.