Mineure "Informatique Parallèle et Distribuée"

Contrôle (2 heures)

Eric Goubault & Matthieu Martel

1er avril 2003

- l'exercice et le problème sont indépendants,
- les sections 2.1, 2.2 et 2.3 du problème sont indépendantes, mais demandent une lecture séquentielle du sujet.
- on s'attachera à être clair dans les justifications, et dans le codage.

1 Exercice (7 points): 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 $i \geq 0$):

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

Comme applications visées, on a par exemple:

- l'évaluation d'un polynôme $p(x) = a_0 + a_1 x + \dots + a_m x^m$ par la méthode de Horner: on fait $y_0 = x$, puis $y_{i+1} = x y_i + a_i$ (récurrence linéaire d'ordre 1); alors $p(x) = y_m$.
- la résolution de systèmes linéaires par bandes: soit par exemple

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

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

$$x_i = \sum_{j=i-m+1}^{i-1} -\frac{a_{i,j}}{a_{i,i}} + \frac{b_i}{a_{i,i}}$$

pour $2 \leq i \leq n$.

1.1 Récurrences d'ordre 1

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

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

et on veut calculer y_i pour les indices $0 \leq i \leq n-1$ (pour un n donné).

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+2} en fonction de y_i.

Question 3: Avec l'aide de la question 2, 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? 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.

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 $M(m) = O(m^{2.376})$).

Question 4: 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? Justifier brièvement.

2 Problème (13 points): communications de groupe ou multicast

Une primitive de *communication de groupe* ou *multicast* permet de diffuser un message à un groupe préalablement défini de destinataires. Contrairement à la diffusion (broadcast) que nous avons vue en cours et qui consiste à envoyer un même message d'un noeud $v \in V$ du réseau à l'ensemble $V \setminus \{v\}$ de tous les autres noeuds, un multicast envoie un message identique d'un noeud v à un ensemble $V' \subseteq V \setminus \{v\}$ préalablement défini de noeuds. Le but de cet exercice est d'implémenter une telle primitive de communication et de l'utiliser dans un algorithme distribué de détection de cycles dans un graphe. Les programmes seront écrits en Java avec invocation de méthodes à distance (RMI), et on supposera que les serveurs sont démarrés par l'utilisateur sur les machines souhaitées (par d'activation d'objet à distance).

2.1 Gestion des groupes de diffusion

Nous souhaitons définir une primitive `multicast(int i,String msg)` permettant d'envoyer le message `msg` à l'ensemble des noeuds appartenant au `i`^{ieme} groupe de diffusion. Pour cela nous allons tout d'abord implémenter un serveur permettant de gérer les différents groupes en Java avec invocation de méthode à distance (RMI). Le code correspondant à l'interface `GroupServerInterface` est le suivant [1].

```
public interface GroupServerInterface extends Remote {
    public void addNodeToGroup(int group,String node) throws RemoteException;
    public List getGroup(int i) throws RemoteException;
}
```

`addNodeToGroup(i,node)` ajoute le noeud du réseau décrit par la chaîne `node` au `i`^{ieme} groupe de diffusion. `getGroup(i)` retourne la liste des noeuds appartenant au groupe d'indice `i`. La classe `GroupServer` est partiellement donnée ci-dessous.

```
public class GroupServer extends UnicastRemoteObject implements GroupServerInterface {
```

1. Pour simplifier, on omet de préciser dans tout cet exercice les importations des packages `java.rmi.*`, `java.rmi.server.*` et `java.util.*`.

```
    List[] groups;
    public GroupServer(int maxGroups) throws RemoteException {
        groups = new List[maxGroups];
    }
}
```

Question 1: Ecrire les méthodes `addNodeToGroup` et `getGroup` de la classe `GroupServer` implémentant les prototypes de l'interface [2].

Question 2: Ecrire la méthode `main` pour la classe `GroupServer`. `main` doit enregistrer sur une machine nommée $M1$ un nouveau serveur capable de gérer $NMAX$ groupes. Quelles sont les commandes permettant de démarrer le serveur?

2.2 Implémentation de `multicast`

Nous allons définir une classe `Node` définissant les noeuds d'une applications répartie. Les principales méthodes d'un `Node` sont :

- `register(i,String nodeName)` enregistre auprès du `GroupServer` le noeud courant dans le `i`^{ieme} groupe de diffusion, c.à.d. que le noeud recevra les messages envoyés à ce groupe. `nodeName` est le nom sous lequel l'objet courant a été enregistré auprès du service de nommage du RMI.
- `multiCast(i,msg)` envoie un message à tous les noeuds enregistrés dans le `i`^{ieme} groupe du serveur.
- `send(msg)` est décrite plus tard dans l'énoncé.

L'interface de la classe `Node` est données ci-dessous.

```
public interface NodeInterface extends Remote {
    public void register(int i,String nodeName) throws RemoteException;
    public void multiCast(int group,String msg) throws RemoteException;
    public void send(String msg);
}
```

Question 3: Ecrire la méthode `register(int i,String nodeName)` de la classe `Node`. On supposera que la classe `Node` a été déclarée par

`public class Node extends UnicastRemoteObject implements NodeInterface`.

On suppose que la classe `Node` contient des méthodes `void send(String msg)` et `String recv()` qui gèrent les accés à un buffer synchronisé. Il est possible d'envoyer un message à un objet `n` de type `Node` par la commande `n.send(msg)`. La commande `s = recv()` retourne le premier message du buffer et se bloque si ce dernier est vide, en attendant l'arrivée d'un message.

Question 4: Ecrire la méthode `multicast(int i,String msg)` de la classe `Node` qui envoie le message `msg` à tous les noeuds du groupe `i`.

2.3 Détection de cycles dans un graphe

Nous allons maintenant définir un algorithme distribué utilisant des communications de type multicast pour détecter la présence de cycles dans un graphe orienté. Soit $G = (V, E)$ un graphe orienté. On associe un processus P_{ab} à chaque arc $(a, b) \in E$ et l'on suppose que P_{ab} possède une primitive de communication de type multicast notée $multicast_{ab}(m)$. Par définition, $multicast_{ab}(m)$ diffuse un message m à tous les processus P_{bc}, c.à.d. que le processus correspondant à l'arc (a, b) diffuse ses messages à tous les processus correspondant à des arcs (e_1, e_2) tels que $e_1 = b$.

Question 5: Ecrire un algorithme correspondant à un processus P_{ab} pour détecter la présence d'un ou plusieurs cycles dans le graphe G. Les seules primitives de communication autorisées pour le processus P_{ab} sont $multicast_{ab}(m)$ et `recv()`.

On souhaite étendre la classe `Node` précédente pour implémenter l'algorithme de la question 5. Tout d'abord, on enrichit la classe `Node` par deux champs `int a,b` afin de pouvoir associer un objet `Node` à

2. On pourra utiliser sans les programmer toutes les méthodes classiques sur les listes dont on aura besoin.

chaque arc (a, b) ainsi que par un champ `String nodeHost` contenant le nom de la machine sur laquelle se trouve l'objet `Node` courant. On suppose donné un constructeur `Node(int e1,int e2,String host)` qui initialise correctement `a`, `b` et `nodeHost`.

Question 6: On suppose que les champs `int a,b` et `String nodeHost` sont correctement initialisés. Comment enregistrer le noeud courant auprès du `GroupServer` à l'aide de la méthode `register(int i,String nodeName)` de la question 3 (on donnera les valeurs des deux paramètres)?

Question 7: En supposant que le noeud courant a été correctement enregistré auprès du `GroupServer`, écrire la méthode `void detectCycle()` de la classe `Node` implémentant l'algorithme de la question 5.

On ajoute à la classe `Node` une méthode `main` permettant d'enregistrer un nouveau `Node` correspondant à un processus P_{ab}. `main` commence par :

```
public static void main(String[] args) {
    Node localNode = new Node(Integer.parseInt(args[0]),Integer.parseInt(args[1]),args[2]);
    ...
}
```

Question 8: Compléter la méthode `main` de façon à enregistrer l'objet courant auprès du serveur de nom, de l'ajouter dans dans le bon groupe du `GroupServer`, et à initier la détection de cycle. `main` doit permettre d'ajouter un nouveau processus P_{ab} sur la machine $M2$ par la commande `java Node` a b $M2$.