Révisions pour le partiel

1 Exercices du partiel de l'année dernière

1.1 Produit de matrice sur P-RAM

1. Expliquer comment multiplier deux matrices carrées de taille n en temps $O(\log n)$ sur une P-RAM CREW avec $O(n^3)$ processeurs.
2. Améliorez l'algorithme précédent pour obtenir le même temps $O(\log n)$ en utilisant seulement $O(n^3/\log n)$ processeurs.

1.2 Produit matrice-vecteur sur un anneau

On veut effectuer un produit matrice-vecteur $Y = A \cdot X$, où A est une matrice carrée de taille n, et X, Y sont deux vecteurs à n composantes. La difficulté nouvelle est que A est supposée symétrique, et seule sa partie triangulaire inférieure est accessible (stockée en mémoire).

La machine cible est un anneau de p processeur P_i , $0 \leq i \leq p-1$. On supposera n divisible par $2p$.

Concevoir un algorithme parallèle pour cette opération :

1. Déterminez la fonction d'allocation de A, X et Y.
2. Écrivez le programme des processeurs.
3. Quelles sont les spécificités de votre programme (recouvrement calcul-communication, granularité) ?
4. Étude des performances : votre algorithme est-il optimal ? Si oui, démontrez-le, si non essayer de l'améliorer !

N'hésitez pas à proposer plusieurs variantes...

2 Recherche des racines dans une forêt

Soit $\mathcal{F}$ une forêt d'arbres binaires. Chaque nœud i d'un arbre est associé à un processeur $P(i)$ et possède un pointeur vers son père $pere(i)$. On va chercher des algorithmes EREW et CREW pour que chaque nœud connaisse la racine de son arbre (notée $racine(i)$), et ainsi prouver l'intérêt des lectures concurrentes.

▷ **Question 1** *Donner un algorithme PRAM CREW pour que chaque nœud détermine racine(i). Démontrer que l'algorithme propose n'utilise que des lectures concurrentes et déterminer sa complexité.*

Réponse. L'algorithme naturel pour chercher les racines utilise la technique de saut de pointeur, un nœud et ses ascendants ayant la racine. Une transformation directe du code nous conduit donc a l'algorithme 1.

Calcul_Racine
Pour tout i en parallèle **:**
Si $pere(i) = NIL$ **Alors**
$racine(i) = i$ x
Tant que il existe un nœud i tel que $pere(i) \neq NIL$ **:**
Pour tout i en parallèle **:**
Si $pere(i) \neq NIL$ **Alors**
Si $pere(pere(i)) = NIL$ **Alors**
$racine(i) = racine(pere(i))$
$pere(i) = pere(pere(i))$.

Algorithm 1: Algorithme CREW pour le calcul des racines.

Cet algorithme est bien de type CREW puisque les seules écritures effectuées par le processeur i concernent des données qui lui sont propres($racine(i)$ et $pere(i)$). Par contre, cet algorithme n'est pas EREW puisque plusieurs processeurs peuvent avoir le même père (surtout à la fin !) et donc peuvent accéder simultanément à la même donnée en lisant $pere(i)$ par exemple.

En utilisant l'analyse de complexité effectuée en cours pour le calcul de la distance à la fin de la liste s'applique et on peut ainsi vérifier que tout les nœuds des arbres de la forêt connaissent leur racine en temps $O(\log d)$ où d est la profondeur maximale des arbres.

▷ **Question 2** *Sur un modèle EREW, quelle est la complexité qu'on peut espérer ?*

Réponse. Plaçons nous dans le pire des cas pour le modèle EREW, c'est à dire le cas où la forêt n'est constituée que d'un seul arbre, et considérons le nombre de processeurs qui peuvent apprendre qui est la racine a chaque étape. Dans le modèle EREW, le nombre de processeurs ayant accès à l'information ne peut que doubler à chaque étape (1 processeur peut au plus lire une case mémoire où est détenue l'information). Dans le cas d'une forêt à un seul arbre, un processeur au plus connaît la racine au départ, et il faut donc $O(\log n)$ étapes pour propager l'information à tous les processeurs.

On vérifie donc ainsi que le modèle CREW est strictement plus puissant que le modèle EREW. Nous allons maintenant montrer comment simuler une lecture simultanée en $O(\log n)$ sur une EREW. La simulation des écritures concurentes en $O(\log n)$ étapes peut se faire en utilisant le même type de technique.

▷ **Question 3** *Comment simuler une lecture concurrente en $O(\log n)$ sur une EREW.*

Réponse. La simulation de la lecture concurrente utilise le fait qu'on sait trier n nombres en temps $O(\log n)$ sur une EREW. On peut utiliser le même type de technique pour les écritures concurentes.

- Dans une première étape chaque processeur écrit dans une case mémoire le couple $< x_i, i >$, où i est son numéro de processeur et x_i est l'adresse de la case mémoire à laquelle il souhaite accéder.
- Dans la deuxième étape, on effectue un tri des couples $< x_i, i >$ selon la première composante, ce qui «regroupe» les processeurs qui désirent accéder à la même donnée. On obtient ainsi une liste triée y dont les composantes sont des couples de la forme $y_i =< x_k, \sigma(i) >$ où σ est une permutation de $[1, n]$.
- Dans la troisième étape, le processeur P_i regarde les cases $< x_k, \sigma(i) >$ et $< x_{k'}, \sigma(i+1) >$. Si $x_k = x_{k'}$, c'est a dire si $P_{\sigma(i)}$ et $P_{\sigma(i+1)}$ cherchent à accéder à la même donnée, alors $suivant(\sigma(i)) = \sigma(i+1)$. Dans le cas contraire, on pose $suivant(\sigma(i)) = NIL$. Il est facile de vérifier que cette étape n'utilise que des lectures indépendantes (si $< x_k, \sigma(i) >$ et $< x_{k'}, \sigma(i+1) >$ sont lus en deux temps) et qu'on obtient ainsi l listes chaînées contenant chacune les processeurs qui désirent accéder à une même donnée.
  La dernière étape de l'algorithme est une simple adaptation de l'algorithme de saut de pointeur vu en cours pour propager l'information à lire dans toute la liste et qui peut s'exprimer par l'algorithme 2, dont la preuve de la correction est immédiate.

```
Simule
  Pour tout i en parallèle :
    Si suivant(i) = NIL Alors
      val(i) = Lecture
    Sinon
      val(i) = NaN
  Tant que il existe un nœud i tel que suivant(i) ≠ NIL :
    Pour tout i en parallèle :
      Si suivant(i) ≠ NIL Alors
        val(i) = val(suivant(i))
        suivant(i) = suivant(suivant(i))
```

Algorithm 2: Dernière étape de la simulation d'une lecture concurrente sur une EREW.

▷ **Question 4** *Donner un algorithme efficace CRCW pour calculer (en le moins de temps possible) le produit de deux matrices booléennes (dans ce cadre, l'addition est remplacée par le ou et la multiplication par le et).*

Réponse. Ce n'est pas très compliqué. On va utiliser $O(n^3)$ processeurs et réaliser le produit de matrice en temps $O(1)$, ce qui conduit bien à un algorithme efficace optimal en temps. L'astuce est qu'il est facile de faire le *ou* de n nombres en temps $O(1)$ sur une CRCW. En effet, le résultat est 1 dès que l'une des opérandes est 1 et il suffit donc que chaque processeur écrive 1 dans une case prédéfinie s'il possède un 1

et ne fasse rien sinon. Formellement l'algorithme peut être décrit comme suit.
Le processeur $P_{i,j,k}$ lit les valeurs $a_{i,k}$ et $b_{k,j}$, effectue le *et* des deux valeurs et stocke le résultat dans $c_{i,j,k}$.
Le processeur $P_{i,j,k}$ écrit 1 dans la case $c_{i,j}$ si $c_{i,j,k} = 1$ et ne fait rien sinon.