P-RAM – 2

1 Simulation d'un nouveau type de PRAM

Une PRAM CRTW (comme Concurrent Read Tolerant Write) est une PRAM avec un modèle légèrement différent du modèle CRCW :
- plusieurs lectures simultanées sont autorisées,
- si plusieurs processeurs veulent écrire une même valeur à une même adresse, cette écriture est autorisée
- si plusieurs processeurs veulent écrire des valeurs différentes à une même adresse, le conflit n'est pas résolu et aucune écriture n'a lieu.

▷ **Question 1** *Montrez qu'une exécution sur une PRAM CRTW à n processeurs qui s'effectue en temps t peut être simulée par une PRAM CRCW (en mode arbitraire) en temps $O(t)$.*

2 Compression de tableaux

Soit A un tableau de n entiers et B un tableau de n booléens. On veut calculer la compression du tableau A par le tableau B, c'est-à-dire un tableau C tel que

$$C[i] = \begin{cases} A[j] & \text{avec } j \text{ est le } i^{\text{ème}} \text{ bit non nul de } B \text{ s'il existe} \\ 0 & \text{sinon} \end{cases}$$

Par exemple :

$A=$	[	1	2	3	4	5	6	7	8	9	10]
$B=$	[	1	0	0	1	0	0	0	1	0	0]
$C=$	[	1	4	8	0	0	0	0	0	0	0]

▷ **Question 2** *Proposez un algorithme qui calcule une telle compression sur une PRAM EREW à n processeurs.*

3 Additionneur PRAM

On veut calculer la somme de deux nombres a et b de n bits, que l'on note $a = a_{n-1}a_{n-2}\dots a_1a_0$ et $b = b_{n-1}b_{n-2}\dots b_1b_0$. Pour ceci, on commence par calculer une fonction de propagation de retenue pour chaque bit :

$$\text{prop}_i = \begin{cases} s & \text{si } a_i = b_i = 0 & \text{signifie que la retenue est stoppée} \\ p & \text{si } (a_i = 1 \text{ et } b_i = 0) \text{ ou } (a_i = 0 \text{ et } b_i = 1) & \text{signifie que la retenue est propagée} \\ g & \text{si } a_i = b_i = 1 & \text{signifie qu'une retenue est générée} \end{cases}$$

On a par exemple :

$a=$	0	1	1	0	0	1	1	
$b=$	1	1	0	1	0	1	0	
prop=	p	g	p	p	s	g	p	s (retenue d'entrée)

Plus précisément, si on considère l'addition des bits a_i et b_i, la fonction de propagation prop_i signifie :
- si $\text{prop}_i = s$, la retenue venant des bits $i-1$ est arrêté, la retenue des bits i sera toujours 0,
- si $\text{prop}_i = g$, une retenue est générée, la retenue des bits i sera 1 quel que soit celle des bits $i-1$,
- si $\text{prop}_i = p$, la retenue des bits $i-1$ est transmise sans modification.

On peut aggréger l'action des fonctions de retenue en les composant, et par exemple, considérer l'effet de la retenue des bits $i-1$ sur celles des bits $i+1$: de la même façon, la retenue des bits $i-1$ peut être propagée (p), arrêtée (s), ou générée (g). Pour ceci on compose l'action des fonctions de propagation de retenue à l'aide d'un nouvel opérateur, noté $\otimes$. Formellement, $\text{prop}_i \otimes \text{prop}_{i-1}$ est la fonction de propagation de retenue correspondant à l'action de la retenue des bits (a_{i-1}, b_{i-1}) et (a_i, b_i) sur l'addition des bits (a_{i+1}, b_{i+1}).

▷ **Question 3** *Donnez la table de* $\otimes$.

▷ **Question 4** *Montrez que* $\otimes$ *est associatif.*

▷ **Question 5** *Proposez un algorithme qui calcule efficacement l'addition de deux nombres à* n *bits sur une PRAM EREW à* n *processeurs. Donnez sa complexité.*

4 Composantes connexes sur une P-RAM

On souhaite concevoir un algorithme CREW qui permette de calculer les composantes connexes d'un graphe $G = (V, E)$ dont les sommets sont numérotés de 1 à n. Plus précisément, on cherche un algorithme qui renvoie un tableau C de taille n tel que $C(i) = C(j) = k$ si et seulement si i et j sont dans la même composante connexe et k est le plus petit indice des sommets de cette composante.

Définition 1. À toute étape de l'algorithme, on appellera pseudo-sommet étiqueté par i l'ensemble de sommets $j, k, l, \cdots \in V$ tels que $C(j) = C(k) = C(l) = \cdots = i$. On assimilera le pseudo-sommet i étiqueté par i au sommet étiqueté par i.

Un des invariants de l'algorithme est que le plus petit indice des sommets constituant un pseudo-sommet étiqueté par i est i et que les sommets appartenant à un pseudo-sommet sont dans la même composante connexe. Cette assertion est donc vraie si on initialise C par : pour tout $i \in V = [\![1, n]\!]$: $C(i) = i$. Ceci signifie que chaque processeur se considère au départ comme sommet de référence de sa composante connexe. L'objectif de l'algorithme est de modifier ce point de vue égocentrique.

Définition 2. Une arborescence k-cyclique ($k \geqslant 0$) est un graphe orienté faiblement connexe (c'est-à-dire tel que le graphe non orienté sous-jacent est connexe) tel que :
- tout sommet a un degré sortant égal à 1 et
- il existe exactement un circuit de longueur $k + 1$.

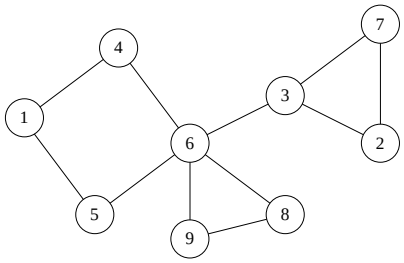
On appelle étoile une arborescence 0-cyclique dans laquelle toutes les arêtes sont incidentes à la racine et l'indice de la racine est le plus petit indice dans l'étoile.

L'invariant précédent est donc que le graphe orienté $(V, \{(i, C(i)) \mid i \in V\})$ est constitué d'étoiles. On peut donc identifier pseudo-sommets et étoiles, le centre de l'étoile étant l'indice du pseudo-sommet. Le calcul des composantes connexes s'effectue en enchaînant plusieurs fois de suite les deux fonctions suivantes :

```
GATHER()
  Pour tout i ∈ S en parallèle
      T(i) ← min{C(j) | {i,j} ∈ E, C(j) ≠ C(i)}
1:    {si l'ensemble est vide, on associe C(i)}
  Pour tout i ∈ S en parallèle
    T(i) ← min{T(j) | C(j) = i, T(j) ≠ i}
2:  {si l'ensemble est vide, on associe C(i)}

JUMP()
  Pour tout i ∈ S en parallèle
B(i) ← T(i)
  Pour j = 1 to log n
  Pour tout i ∈ S en parallèle
T(i) ← T(T(i))
  Pour tout i ∈ S en parallèle
C(i) ← min{B(T(i)), T(i)}
```

▷ **Question 6** *On considère le graphe suivant.*



Appliquer la fonction Gather *sur ce graphe, puis la fonction* Jump*, puis la fonction* Gather*, et ainsi de suite. Il sera instructif d'observer l'effet des opérations sur les graphes orientés* $(V, \{(i, T(i)) \mid i \in V\})$ *et* $(V, \{(i, C(i)) \mid i \in V\})$.

▷ **Question 7** *Montrer qu'après l'application de la fonction* Gather*, les composantes connexes contenant plusieurs pseudo-sommets induisent des arborescences* 1*-cycliques dans le graphe orienté* $(V, \{(i, T(i)) \mid i \in V\})$. *On notera également que le plus petit pseudo-sommet d'une arborescence* 1*-cyclique appartient au cycle.*

▷ **Question 8** *Montrer que la fonction* Jump *transforme une arborescence* 1*-cyclique en étoile (ou pseudo-sommet).*

▷ **Question 9** *Montrer qu'après* $\lceil \log n \rceil$ *enchaînements des fonctions* Gather *et* Jump*, les composantes connexes du graphe sont représentées par les pseudo-sommets induits par* C.

▷ **Question 10** *Quelle est la complexité de l'algorithme ? Combien de processeurs sont utilisés ?*