

P-RAM

Résumé: Nous revenons sur la séparation des modèles et étudions un algorithme de calcul des composantes connexes sur une CREW.

1 Séparation CREW/CRCW

▷ **Question 1** Donner un algorithme PRAM CRCW permettant de calculer avec $O(n^2)$ processeurs en temps constant le maximum d'un tableau A composé de n éléments. *Réponse* □

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

Une P-RAM CRCW est donc strictement plus puissante qu'une P-RAM CREW. Nous venons de séparer les modèles CRCW et CREW, en exhibant un algorithme pour lequel un facteur $O(\log p)$ sépare les temps d'exécution avec p processeurs d'un même problème. Ce facteur $O(\log p)$ est maximal.

2 Composantes connexes

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 $j, k, l, \dots \in V$ tels que $C(j) = C(k) = C(l) = \dots = i$. 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 $i \in V = \llbracket 1, n \rrbracket$: $C(i) = i$. 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 ($k \geq 0$) 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.

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 :

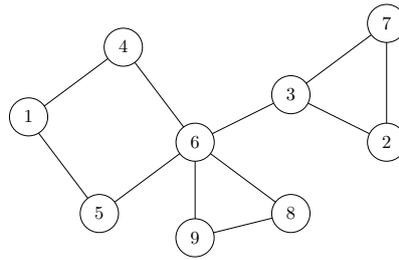
```

GATHER()
1: Pour tout  $i \in S$  en parallèle :
2:    $T(i) \leftarrow \min \{C(j) \mid \{i, j\} \in E, C(j) \neq C(i)\}$ 
   {si l'ensemble est vide, on associe  $C(i)$ }
3: Pour tout  $i \in S$  en parallèle :
4:    $T(i) \leftarrow \min \{T(j) \mid C(j) = i, T(j) \neq i\}$ 
   {si l'ensemble est vide, on associe  $C(i)$ }

JUMP()
5: Pour tout  $i \in S$  en parallèle :
6:    $B(i) \leftarrow T(i)$ 
7: Pour  $j = 1$  à  $\log n$  :
8:   Pour tout  $i \in S$  en parallèle :
9:      $T(i) \leftarrow T(T(i))$ 
10: Pour tout  $i \in S$  en parallèle :
11:    $C(i) \leftarrow \min \{B(T(i)), T(i)\}$ 

```

▷ **Question 3** 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 $(V, \{(i, C(i)) \mid i \in V\})$. Réponse □

▷ **Question 4** 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. Réponse □

▷ **Question 5** Montrer que la fonction **JUMP** transforme une arborescence 1-cyclique en étoile (ou pseudo-sommet). Réponse □

▷ **Question 6** 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**. Réponse □

▷ **Question 7** Quelle est la complexité de l'algorithme ? Combien de processeurs sont utilisés ? Réponse □

3 Réponses aux exercices

▷ **Question 1, page 1**

Il suffit d'utiliser un processeur pour chaque comparaison et de leur faire écrire le résultat au même endroit, ce qui nous conduit donc à l'algorithme 1.

```

CALCUL_MAXIMUM( $A, n$ )
1: Pour tout  $i \in \llbracket 1, n \rrbracket$  en parallèle :
2:    $m[i] \leftarrow \text{VRAI}$ 
3: Pour tout  $i, j \in \llbracket 1, n \rrbracket^2$  en parallèle :
4:   Si  $A[i] < A[j]$  Alors
5:      $m[i] \leftarrow \text{FAUX}$ 
6: Pour tout  $i \in \llbracket 1, n \rrbracket$  en parallèle :
7:   Si  $m[i] = \text{VRAI}$  Alors
8:      $max \leftarrow m[i]$ 
9: Renvoyer  $max$ 
    
```

Algorithme 1: Algorithme CRCW pour calculer le maximum d'un tableau.

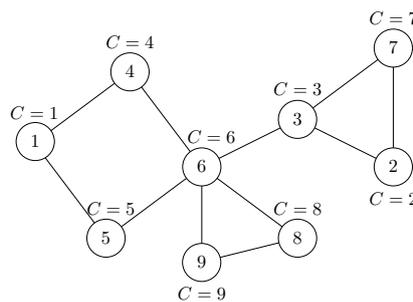
La P-RAM opère en mode CRCW consistant, car tous les processeurs écrivent la même valeur FALSE. À la première et troisième étapes on utilise seulement n processeurs, tandis qu'à la deuxième étape on a $n(n - 1)$ processeurs P_{ij} , $i \neq j$, chacun responsable d'une comparaison $A[i] < A[j]$.

▷ **Question 2, page 1**

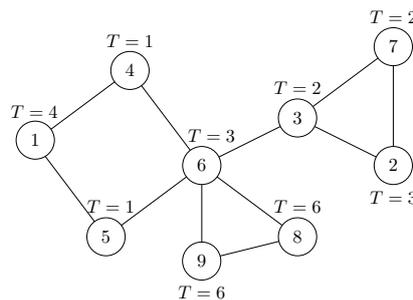
L'algorithme du maximum nous fournit la réponse. Plus généralement, une P-RAM CRCW de n processeurs est capable de calculer en temps constant la fusion $\bigotimes_{1 \leq i \leq n} e_i$ de n éléments (e_1, \dots, e_n) , où \otimes est une loi associative. Cette opération de réduction ne peut pas s'effectuer en moins de $O(\log n)$ étapes sur une P-RAM CREW : avec ce modèle, en temps constant on peut fusionner au plus un nombre constant d'éléments en une seule valeur.

▷ **Question 3, page 2**

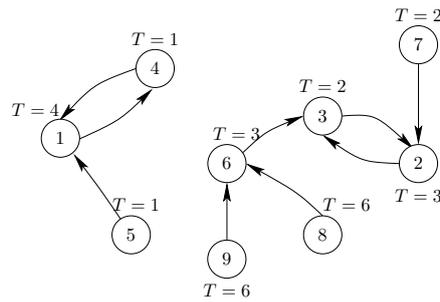
Avant le début de l'algorithme, les valeurs de C sont initialisées comme ci-dessous :



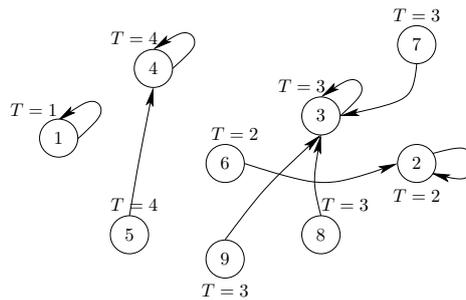
Après l'application de la fonction GATHER, les valeurs de C sont inchangées et celles de T sont les suivantes :



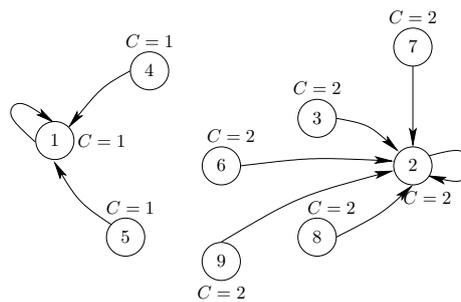
Si on considère le graphe orienté $(V, \{(i, T(i)) \mid i \in V\})$, on peut s'apercevoir qu'il n'est constitué que d'arborescences 1-cycliques :



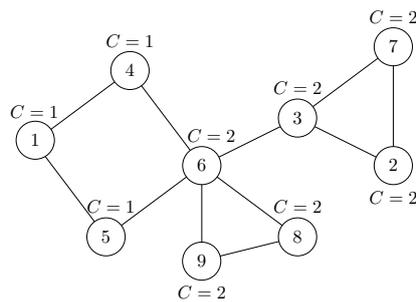
Après le saut de pointeur de la fonction JUMP, il ne reste plus que des étoiles :



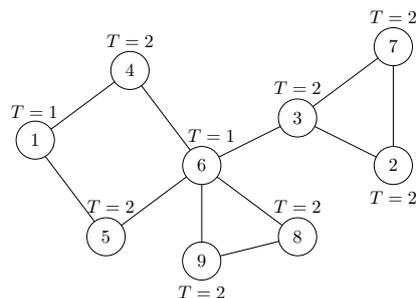
Ces étoiles sont fusionnées dans le graphe $(V, \{(i, C(i)) \mid i \in V\})$ lors de la dernière opération de JUMP :



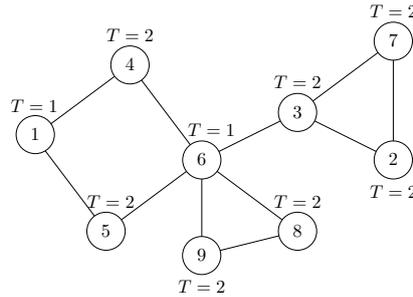
On notera qu'il ne reste plus que deux pseudo-sommets. On se retrouve dans la situation suivante au début de l'appel à la fonction GATHER :



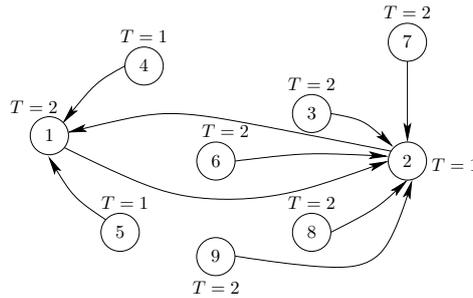
Après la première étape, T est mis à jour comme suit :



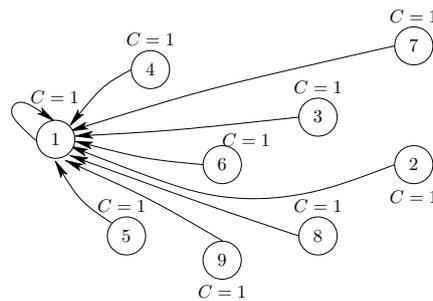
et enfin comme ceci :



On se retrouve donc avec le graphe orienté $(V, \{(i, T(i)) \mid i \in V\})$ constitué d'arborescences 1-cycliques :



Les pseudo-sommets 1 et 2 sont donc fusionnés à la fin de l'appel à JUMP :



On a bien calculé les composantes connexes du graphe.

▷ **Question 4, page 2**

Tout d'abord, il est clair que lorsqu'une composante connexe ne contient qu'un seul pseudo-sommet, l'étoile correspondante est transférée dans T sans modification.

Si une composante connexe contient plusieurs pseudo-sommets, par contre, T décrira un ensemble d'arborescences 1-cycliques contenues dans cette composante. En effet, tout pseudo-sommet de cette composante contient au moins un sommet adjacent à un sommet d'un autre pseudo-sommet. GATHER fait pointer le représentant de chaque pseudo-sommet – *via* T – vers le représentant d'un autre pseudo-sommet, tout en laissant pointer les autres sommets vers leur pseudo-sommet initial. En clair, si deux groupes se touchent, leurs représentants se retrouvent liés dans le graphe orienté induit par T et les autres sommets continuent de pointer vers leur représentant respectif. Chaque composante de ce graphe orienté doit contenir au moins une boucle car le degré sortant de chaque sommet vaut 1. Il y a au plus une boucle car sinon il y aurait un pseudo-sommet avec deux valeurs pour T . Enfin la boucle en question ne peut être que de longueur 2, sinon (si elle était de longueur 1) i et $T(i)$ seraient identiques ou (si elle était de longueur supérieure à 2) il existerait un sommet i sur la boucle tel que $T(i)$ n'est pas le plus petit indice des pseudo-sommets adjacents au pseudo-sommet i .

▷ **Question 5, page 2**

Cette étape fusionne tous les sommets d'une même arborescence 1-cyclique en une étoile indexée par le sommet de plus petit indice en utilisant la technique de saut de pointeur. En effet, étant donné la configuration d'une étoile, à l'issue des sauts de pointeur, chaque sommet a pour valeur de T l'une des anciennes valeurs d'un des sommets de la boucle. La dernière étape permet donc d'assigner à tous les sommets la plus petite valeur des sommets de l'arborescence à laquelle ils appartiennent.

▷ **Question 6, page 2**

Il suffit de montrer que le nombre de pseudo-sommets diminue au moins de moitié à chaque étape. Concentrons-nous sur les représentants des pseudo-sommets et intéressons-nous au graphe induit par T sur ces sommets. Dans un tel graphe, deux pseudo-sommets i et j sont connectés si, et seulement si, il existe deux sommets k et l connectés dans le graphe initial et tels que $C(k) = i$ et $C(l) = j$. Dans l'exemple précédent, cela revient à avoir un graphe composé des pseudo-sommets 1 et 2 reliés par une arête après la première application de GATHER. La fonction JUMP fusionne tous les pseudo-sommets ainsi reliés en un seul pseudo-sommet. Ainsi le nombre de pseudo-sommets dans une même composante connexe diminue au moins de moitié à chaque étape et $\lceil \log n \rceil$ enchaînements de GATHER et JUMP suffisent à calculer les composantes connexes.

▷ **Question 7, page 2**

Premièrement, on peut remarquer que la boucle séquentielle de la fonction JUMP implique que le temps de calcul total est au moins $O(\log^2 n)$, et ce quel que soit le nombre de processeurs. On va d'abord montrer que ce temps peut être atteint avec $O(n^2)$ processeurs.

On peut déjà remarquer qu'avec autant de processeurs, la première et la dernière boucle de JUMP prennent un temps $O(1)$ et le saut de pointeur un temps $O(\log n)$. En fait, $O(n)$ processeurs suffisent pour arriver à un tel temps de calcul de la fonction JUMP. Si on veut arriver à un temps total de l'ordre de $O(\log^2 n)$, on va donc devoir montrer que la fonction GATHER peut s'exécuter en temps $O(\log n)$.

Le calcul du maximum de n valeurs peut se faire en temps $O(1)$ avec n^2 processeurs. Cependant, ce sont les maxima de plusieurs ensembles que l'on veut calculer et il faut raffiner donc l'approche (voir algorithme 2).

```

1: Pour tout  $i \in S$  en parallèle :
2:    $T(i) \leftarrow \min \{C(j) \mid \{i, j\} \in E, C(j) \neq C(i)\}$ 
     {si l'ensemble est vide, on associe  $C(i)$ }
   La boucle précédente peut être transformée en le code suivant

3: Pour tout  $i, j \in S$  en parallèle :
4:   Si  $\{i, j\} \in E$  Et  $C(i) \neq C(j)$  Alors
5:      $Temp(i, j) \leftarrow C(j)$ 
6:   Sinon
7:      $Temp(i, j) \leftarrow \infty$ 
8: Pour tout  $i \in S$  en parallèle :
9:    $Temp(i, 1) \leftarrow \min \{Temp(i, j) \mid j \in S\}$ 
10: Pour tout  $i \in S$  en parallèle :
11:   Si  $Temp(i, 1) = \infty$  Alors
12:      $T(i) \leftarrow C(i)$ 
13:   Sinon
14:      $T(i) \leftarrow Temp(i, 1)$ 

```

Algorithme 2: Précision sur la mise en œuvre des calculs de maxima.

La première boucle se fait clairement en temps $O(1)$ avec $O(n^2)$ processeurs (en réalité avec $O(|E|)$ processeurs) sur une CREW. Les deux boucles suivantes se font en temps $O(\log n)$ avec $O(n^2)$ processeurs en divisant pour régner.

Le calcul des composantes connexes s'effectue donc bien en temps $O(\log^2 |V|)$ avec $O(|V| + |E|)$ processeurs. Cependant, la fonction JUMP gaspille des ressources (le «diviser pour régner» est trop gourmand en ressources). Les calculs de minima peuvent également être optimisés en terme de ressources. Le théorème de Brent nous permet donc de diminuer le nombre de processeurs à $O\left(\frac{n^2}{\log n}\right)$ (en réalité à $O\left(\frac{|E|}{\log |V|} + |V|\right)$ processeurs) sans modifier le temps de calcul.