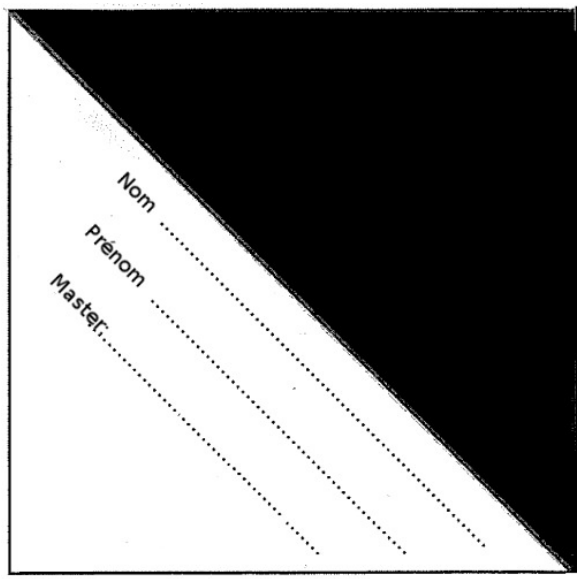
Université Nice Sophia Antipolis
Polytech



Parallelisme et Distribution

Durée : 120 minutes
Document autorisé : une feuille A4 manuscrite
Toute réponse doit être justifiée

Points : | | | |

Exercise 1: Semaphores

On considère les deux fonctions suivantes, utilisant une variable partagée x.

$function\ haut()$	$function\ bas()$

`for i from 1 to 10`
`  x++`
`end for`
	`for i from 1 to 15`
	`  x--`
	`end for`

1. Initialement $x = 15$. Que vaudra x si chaque fonction est exécutée par un thread différent, et pourquoi ?

2. En utilisant un sémaphore, modifiez les fonctions pour que la fonction $haut()$ s'exécute toujours complètement avant la fonction $bas()$.

3. Est-ce que cette solution permet de tirer partie d'une machine multi-coeur ?

.....
.....
.....
.....
.....

4. Proposez une solution qui augmente le parallélisme tout en maintenant une exécution déterministe.

.....
.....
.....
.....
.....
.....
.....
.....
.....

Exercise 2: Filtrage de tableau

Le but de cet exercice est de filtrer un tableau pour ne retenir que les éléments répondant à un critère exprimé par une fonction $f(x)$ qui retourne $true/false$.

Par exemple, si $f(x)$ retourne $true$ si x est pair, alors le tableau $A = [6, 2, 3, 4, 5, 6, 7]$ sera filtré en $B = [6, 2, 4, 6]$. La taille de B doit être calculée au plus juste, i.e. il ne peut y avoir de cases vides et les tableaux **ne sont pas dynamiques**.

Version séquentielle

1. Proposez une version séquentielle (appelée $filtre\_seq$) de cet algorithme.

2. Quelle est la complexité algorithmique ?

Version parallèle

Nous allons maintenant écrire une version parallèle cet algorithme. On supposera que le tableau A a une taille $n = 2^m$.

Le principe général est le suivant :

1. Appliquer f sur chaque élément du tableau et construire un tableau R de résultats où le résultat de f est indiqué par les valeurs 0 ($false$) ou 1 ($true$).
2. Appliquer un *sum-prefix* sur R pour obtenir R_SUM
3. Utiliser R_SUM pour connaitre la taille du tableau final B
4. Placer les éléments filtrer dans B.

Construction du tableau de bits R

On considère $A = [4, 5, 12, 8, 9, 10, 34, 87]$ et la fonction $f(x)$ définie précédemment (pair/impair).

1. Appliquez l'étape 1 de l'algorithme et construisez le tableau R correspondant.

2. Écrivez un algorithme parallèle prenant A et f en paramètre et construisant R.

3. Quelle PRAM nécessite cet algorithme ?

4. Quelle est sa complexité ?

Calcul de R_SUM

1. Rappelez brièvement ce que calcule un *sum-prefix*

2. Appliquez *sum-prefix* sur le tableau R obtenu précédemment pour obtenir R_SUM

3. Quelle PRAM nécessite cet algorithme ?

4. Quelle est sa complexité ?

5. Comment utiliser R_SUM pour connaitre la taille du tableau B ?

.....
.....
.....
.....
.....

Calcul de B

1. En utilisant R et R_SUM, écrivez l'algorithme parallèle permettant de construire B.

.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....

2. Quelle PRAM nécessite cet algorithme ?

.....
.....
.....
.....
.....

3. Quelle est sa complexité ?

Algorithme complet

1. Quelle PRAM nécessite l'algorithme complet, i.e avec les 4 étapes ?

2. Quelle est sa complexité ?

3. Est-il optimal en travail ?

4. Comment traiter le cas $n \neq 2^m$?

Exercise 3: Tri fusion de Batcher sur PRAM

On rappelle que le tri fusion de Batcher utilise des comparateurs pour construire des réseaux $FUSION_m$, tel que $FUSION_1$ fusionne deux listes triées de 2^1 éléments.

1. Dessinez $FUSION_2$

2. Comment construit-on un réseau $FUSION_m$?

3. Combien de comparateurs sont nécessaire ?

4. Combien faut-il de processeurs sur une PRAM pour implémenter un comparateur ?

5. En déduire le nombre de processeurs nécessaires pour implémenter $FUSION_m$

6. Comment construit-on TRI_m qui trie une liste de 2^m éléments ?

7. Combien de processeurs sont-ils nécessaires pour implémenter TRI_m sur une PRAM ?

Exercise 4: Algorithmique distribuée sur anneau

On considère N machines ($P_0 ... P_{N-1}$) organisées en anneau unidirectionnel. Initialement chaque machine i possède une donnée différente X_i et à la fin de l'algorithme, P_0 a toutes les données. Cette opération est appelée *Gather*, par opposition au *Scatter* vu en cours. Le modèle de programmation utilisé ici est *Message Passing* et seules les primitives *Send()* et *Receive()* sont disponibles.

1. Donnez les formules permettant trouver les identifiants des voisins de la machine i

2. Quelles sont les communications possibles pour la machine i sur cette topologie ?

3. Ecrivez l'algorithme **Gather()**.

4. En considérant que la donnée X_i a une taille de 1, quel est le coût de cet algorithme dans le modèle de

communication $\beta + L\tau$?

. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .

Ceci est un brouillon