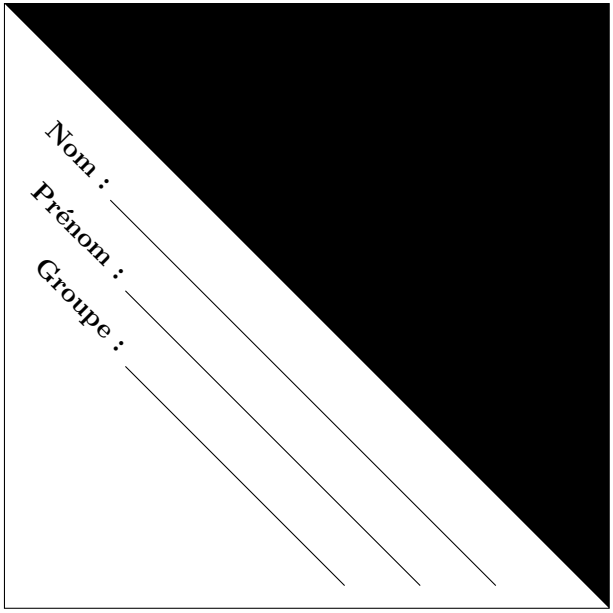
Université Côte d'Azur
Polytech Nice Sophia

Département : SI (4ème année)
Cours : Algorithmique Parallèle
Date : 21 février 2024

Durée : 1h
Documents : Aucun hormis une page A4



L'utilisation des documents et de n'importe quel dispositif électronique sont interdits. Vos réponses doivent être rédigées de manière claire. Le poids de chaque exercice est donné à titre indicatif et peut-être modifié lors de la correction finale des copies.

1 Simulation d'une PRAM CW dite "associative" avec une PRAM EW (5pts)

Cet exercice traite du comportement d'une instruction d'écriture d'une PRAM ayant le mode CW « associative / combinaison ».

1. On le rappelle, dans ce mode, avant d'écrire une valeur, si plusieurs processeurs veulent écrire à la même adresse en mémoire globale, alors ces valeurs sont d'abord automatiquement combinées par la machine, en un temps jugé égal à O(1) (par exemple, la combinaison est faite par un circuit cablé et bien sûr parallèle, et on considère qu'il est donc très rapide pour effectuer la combinaison !) en utilisant une opération binaire associative, puis le résultat est écrit à l'adresse mémoire concernée.

   Imaginez un problème à résoudre en parallèle, où une telle opération d'écriture très puissante serait utile. Insistez bien sur le choix que vous faites pour l'opération binaire.

   Pourquoi, à votre avis, cette opération binaire doit-elle être associative ?
2. La question porte maintenant sur la possibilité de reproduire un tel comportement, alors qu'on ne disposerait que d'une PRAM n'autorisant pas les écritures concurrentes, donc une EW PRAM.

   Que pensez vous qu'il faut faire pour que toute opération d'écriture soit finalement effectuée par la EW PRAM : plus précisément, donnez un algorithme parallèle et générique (en notant $\bullet$ l'opération binaire associative) que la EW PRAM exécuterait afin de reproduire une telle opération d'écriture de la PRAM CW associative. Donner son temps d'exécution parallèle, en posant n égal au nombre de valeurs que n processeurs désirent écrire, et qui doivent donc être combinées avant écriture.
3. En conclusion : quel est le surcoût en temps parallèle d'un algorithme PRAM CW associatif de complexité en temps parallèle $T(n)$ lorsque on doit l'exécuter sur une EW PRAM ?

   Si on pose que l'algo sur la PRAM CW a une complexité en temps qui fait qu'il appartient à la classe de complexité *NC*, est-ce que sur la EW PRAM la nouvelle complexité en temps fait qu'il appartient toujours à *NC* ? (Pour rappel, la classe de complexité *NC* regroupe les problèmes pour lesquels il existe un algorithme PRAM de temps parallèle poly- logarithmique en fonction de la taille des données d'entrée, en utilisant un nombre polynomial de processeurs.)

2 OpenMP (6 pts)

Quelques petites questions sur les concepts, de manière "graduelle".

1. OpenMP propose un modèle de programmation dit *fork - join*. Expliquer en 2 ou 3 phrases maximum ce que cela signifie
2. Dans une région parallèle, supposons qu'on commence par une première instruction (par exemple un print, ou comme ci dessous, une initialisation d'une variable) et juste après, le code déclare un pragma de "work-sharing" pour une boucle for.

   ```
   #pragma omp parallel
   {
   int j=omp_get_thread_num();
   #pragma omp for
   for(i = 0; i < size; i++) //do something
   }
   ```

   Supposons que le nombre de threads configuré pour l'exécution soit de 4. Notez bien la présence de la paire de {} délimitant la région parallèle.

   Que vaut j ? A-t-on une seule ou plusieurs instances de j ?

   Expliquer pourquoi les itérations de la boucle for seront distribuées entre les threads de la région parallèle.

   A présent, imaginez que la paire de {} délimitant la région parallèle ait disparu (ait été omise, cf message sur le canal slack...). Expliquer le fait que la boucle for devienne non parallèle.
3. Vous supposerez une répartition statique des itérations de la boucle for, fonction du nombre de threads et du nombre d'itérations à réaliser. Dans l'ordre thread 0 reçoit les x premières itérations, thread 1 les x suivantes, etc.

   Qu'affiche comme lignes le code suivant. Sont-elles toujours les mêmes, et est-ce que l'ordre des lignes est toujours identique.

   ```
   int main(int argc, char* argv[]) {
   omp_set_num_threads(4);
   #pragma omp parallel
   {
   int j=omp_get_thread_num();
   int mod=0;
   #pragma omp for
     for (int i=0; i<8; i++) {
      mod=j*i;
      printf("In FOR: Thread %d on iteration %i, mod = %d\n", j,i, mod);
     }
   printf("Thread %d mod = %d\n", j, mod);
   }
   ```

3 Tri (4 pts)

Rappel en image de cet exercice du TD2. L'opération effectuée sur la matrice montrée sur la figure 1 est un test ">" entre la valeur en ligne et la valeur en colonne. Ensuite, on voit que chaque ligne de la matrice est sommée, pour obtenir un vecteur. Puis, les déplacements à réaliser pour obtenir une nouvelle séquence, mais cette fois triée, sont obtenus grâce au dessin à droite de la figure (flèche descendante). Notez que dans cette version, on peut faire le tri en place. Mais ce ne sera plus le cas par la suite.

Le but de l'exercice est d'étendre cet algorithme, afin de permettre la présence de potentiellement plusieurs copies de plusieurs des variables du tableau à trier. Exemple (3,8,3,1,8,7,1,3). Le nouveau tableau résultat et trié, devra être (1,1,3,3,3,7,8,8).

1. Appliquez la première phase, identique

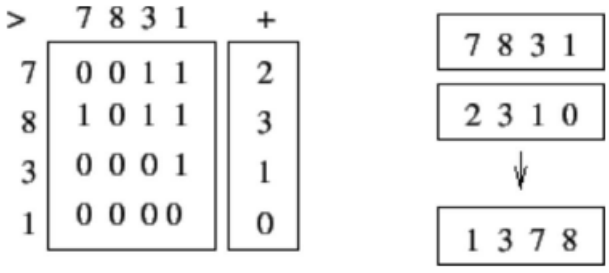


Figure: Exemple de tri sur la séquence de valeurs 7;8;3;1

FIGURE 1 – Tri

2. Déplacez les éléments dans un nouveau tableau, dont les cases seront initialisées d'une manière spécifique. De quelle PRAM avez vous besoin. Que constatez vous après ces déplacements ?
3. Terminez le travail en vous aidant d'un tableau de flags (sans omettre d'expliquer comment ce tableau de flags est construit).
4. Quel est le temps de calcul PRAM total
5. Calculer le travail effectué par cet algorithme.
6. Par rapport à un tri en séquentiel dont la complexité en temps serait de O(NxlogN), est-ce que cet algorithme PRAM est optimal ?

4 Recherche dans un tableau (5 points)

Nous allons étudier le problème de reconnaissance de forme sur des tableaux unidimensionnels. Nous supposerons donnés un entier n et un tableau *tab* de taille n dont les cases contiennent 0 ou 1. Nous supposerons que les indices du tableau sont compris entre 0 et $n-1$. Le but est de trouver le nombre maximal de 0 contigüs (c'est à dire figurant dans des cases consécutives) dans le tableau.

Nous utiliserons le tableau suivant comme exemple :

i	0	1	2	3	4	5	6	7
tab[i]	0	0	1	1	0	0	0	1

Supposons qu'on veuille programmer (en séquentiel, ou en parallèle, peu importe) une fonction *nombreZerosGauche(i, tab, n)* prenant en paramètres le tableau *tab* de taille n ainsi qu'un indice i compris entre 0 et $(n-1)$. Cette fonction renvoie le nombre de cases contigües du tableau contenant 0 à gauche de la case d'indice i, cette case étant comprise. D'un point de vue formel il s'agit de renvoyer 0 si la case d'indice i contient un 1 et sinon de renvoyer

$$max\{j - i + 1 \ tel \ que \ i \leq j \leq (n-1) \ et \ \forall k \in [i,j], tab[k] == 0\}$$

Sur le tableau exemple précédent, en prenant comme indice $i = 3$ nous obtenons 0 et pour l'indice $i = 5$, *nombreZerosGauche* doit retourner 2. Notez que si la case i contient 1, alors *nombreZerosGauche* doit retourner 0.

Dit autrement, la fonction *nombreZerosGauche* permet d'obtenir ce tableau :

i	0	1	2	3	4	5	6	7
tab[i]	0	0	1	1	0	0	0	1
nbZerosGauche[i]	1	2	0	0	1	2	3	0

Et le but ultime, ce serait de connaitre quel est le nombre maximal de 0 contigüs (ici 3) que *tab* possède.

Question Donner les grandes lignes d'une solution parallèle PRAM permettant d'obtenir le tableau $nbZerosGauche$ ci-dessus. Bien décomposer les étapes en indiquant les tableaux intermédiaires et supplémenatires nécessaires.

Quelle sorte de PRAM avez-vous besoin d'utiliser.

Quelle est la complexité du temps parallèle de cette solution.