UNIVERSITÉ DE NICE
SOPHIA ANTIPOLIS

U.F.R Sciences

ANNEE UNIVERSITAIRE 20__ / 20__ Session : ____

U.E. _____

Épreuve de _____

Note de l'épreuve
15

(1) Le candidat doit inscrire ici : ses noms, prénoms, lieu et date de naissance, puis rabattre suivant le pointillé le coin de la copie et le coller.

Il est interdit au candidat de signer sa copie ou d'y inscrire un signe quelconque pouvant en indiquer la provenance.

Wpar $(n, p) = (n-1)\log n$
tilisés lors de la
esseurs pour que
ières récursions

Nombre d'intercalaires 1 Nice, le 10 Décembre 09

6/7

Exercice I

1) int nbInversion (int[] A, int n) {
int nb ← 0;
for (int i ← 1; i < n; i ← i+1)
if (A[i] > A[i+1]
nb ← nb + 1;
return nb;
}

$T_{seq}(n) = n - 1$

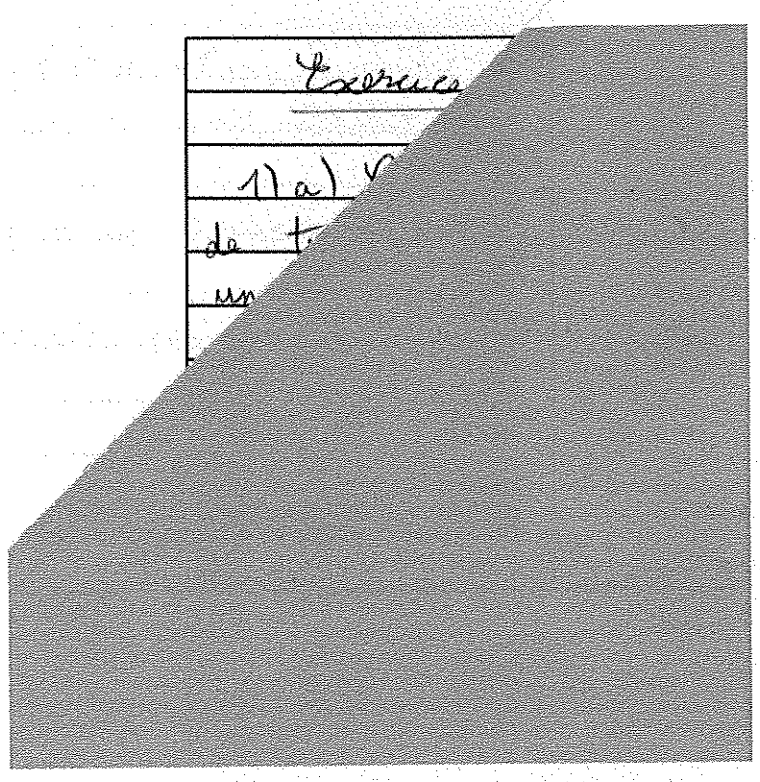
2) a) int nbInversion (int[] A, int n) {
int nb ← 0;
for (int i ← 1; i < n; i ← i+1) en // avec i privé
if (A[i] > A[i+1]
nb ← nb + 1; avec reduce (+ ; nb)
return nb
}

Coût de réduction proportionnel à CW

Une PRAM CREW avec n-1 processeurs est nécessaire, l'accès en lecture d'une case i pouvant être effectuée en même temps lors des tests A[i-1] > A[i] et A[i] > A[i+1]

OK

1/6



Exercice

1) a) V

de t

un

On a $T_{par}(n) = 1$ et $W_{par}(n,p) = n-1$ qui est optimal puisqu'identique à $T_{seq}(n)$

(Une PRAM EREW était également possible en prenant les cases 2 par 2 avec $\frac{n-1}{2}$ processeurs effectuant 2 tours - le 1er tour avec $A[1] > A[2]$, $A[3] > A[4]$... le 2nd avec $A[2] > A[3]$, $A[4] > A[5]$ - où $W_{par}(n,p) = 2 \cdot \frac{n-1}{2} = n-1$)

2/3

Attention que la traduction suivante dans le protocole CREW

2) b) int nbInversions (int[] A, int n) {

if (n == 1) return 0;

if (n%2 == 0) {

if (A[n/2] > A[n/2 + 1])

return 1 + nbInversions (A[1 ... n/2])
+ nbInversions (A[n/2 + 1 ... n]);

else

return nbInversions (A[1 ... n/2])
+ nbInversions (A[n/2 + 1 ... n]);

} else

return nbInversions (A[1 ... ⌈n/2⌉])
+ nbInversions (A[⌈n/2⌉ ... n]);

Une PRAM CREW est nécessaire lors d'une récursion où n est impair, $A[\lceil n/2 \rceil]$ est lu dans les deux sous-ta

2/

On a $T_{par}(n) = \log n$ et $W_{par}(n,p) = (n-1)\log n$ puisque n processeurs sont utilisés lors de la dernière récursion.

Il faudrait alors utiliser $\frac{n-1}{\log n}$ processeurs pour que ce soit optimal en transformant les dernières récursions en séquentiel.

2/3

Exercice 2

1) SMP signifie Shell Memory Processor

0,5

2) L'accélération maximale est bornée par sa partie séquentielle puisque $S(p,n) = \frac{T_{seq}(n)}{T_{par}(p,n)}$

$= \frac{(1-f)+f}{f+\frac{(1-f)}{n}} = \frac{1}{f+\frac{(1-f)}{n}} = \frac{n}{nf+1-f} < \frac{1}{f}$ et non par le nombre de processeurs

2,5/4,5

1,5

3) Il est possible de paralléliser un for avec l'annotation #pragma omp parallel for en veillant toutefois que la variable soit locale à la boucle d'incrémentation

0,5

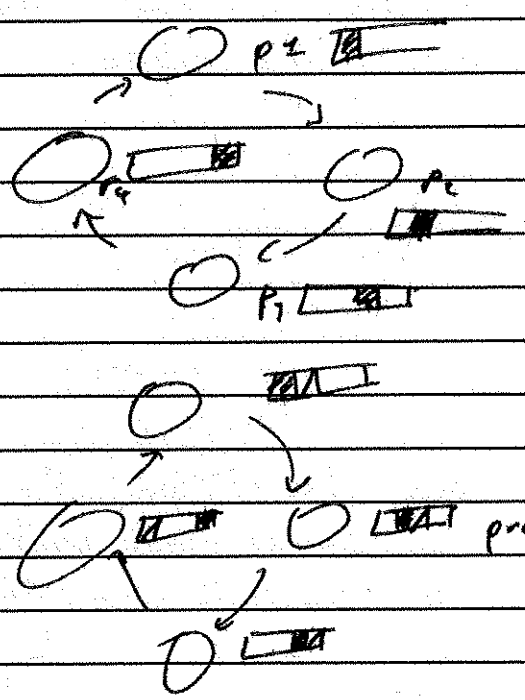
c'est toujours le cas même sans OpenMP

3/6

Exercice III

I.

cet algorithme propage l'adresse de chaque dans la case du tableau lui correspondant c'est à dire que chaque cpu au premier tour donne son adresse en suivant puis l'adresse de son precedent ...



adr est la liste des adresse des cpu où adr[i] : l'adresse du $i^{ième}$ cpu.

I e)

si defaut send attend que la destination attrape, cependant la destination est peut être aussi en "send" etc ... de sorte que le dernier attend le premier et infinité ...

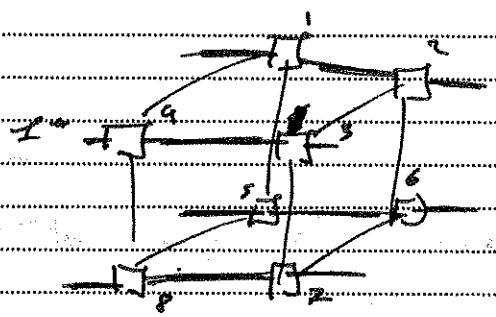
Send ne doit pas être en exclusif to receave, c'est a dire non bloquant.

~~Ex~~ I 3

$(t_s + L \times t_w)$ le coût d'un envoi d'adresse

le tout ~~est~~ multiplié par le nombre de tours p

$p \times (t_s + t_w \times L)$, à chaque tour nous avons $p \times L$ octets.



avec 8 cpu

1er tour : horizontal

passage

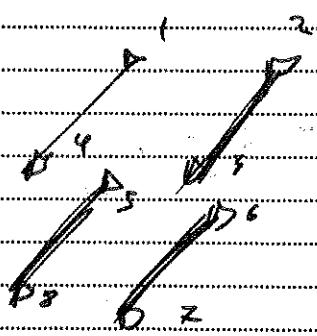
1 connait 2 et vice versa

4 " 3 "

8 " 7 "

5 " 6 "

5/5



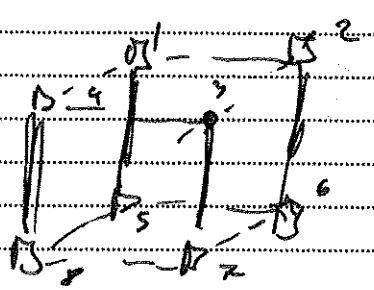
2ème

1 " 4 "

2 " 3 "

5 " 8 "

6 " 7 "



3ème

1 " 5 "

4 " 8 "

3 " 7

2 " 6 "

sans avoir obligé précédent plus

passage

Intercalaire N°

en ? "tour", tous les tableaux sont formés
nous avons consommés log (n) tour,
coût total
$(L \times t_w + t_s) \times \log(n)$

bien meilleur

la mémoire ~~totale~~ circulé est la même que la première ~~solution~~ Pas à chaque step

---

Exercice 4

I1

1) Le nombre de cpu est ~~de~~ égale le carré d'un entier

I2)

c = mpi_comt.

```
~~Nord~~
int nord()
    if (!i - √c < 0)
        return i - √c else return null
    fin if
fin nord

int sud ()
    if (!i + √c} > c)
        return i + √c
    else
        return null
    fin else
fin sud
```