

# TD n°2

*Previously on AMC's OpenMP Videos...*

```
#pragma omp parallel
```

Pour spécifier le nombre de thread :

- Définir une variable l'environnement (OMP\_NUM\_THREAD) est lui affecter un nombre de thread
- Appeler une routine qui définit le nombre de thread.  
omp\_set\_num\_thread(int nb);
- A l'aide de num\_thread directement à l'appel de #pragma omp parallel

Le thread de base a le numéro 0. Les autres sont numérotés séquentiellement.

```
int k; // Variable partagée
#pragma omp parallel num_thread(3) private(i, j) shared (k, l) {
    int i; // Variable privée à chaque thread.
}
// Fin de la parallélisation.
```

```
/* Le "pragma" ci-dessous indique que l'on souhaite paralléliser uniquement la boucle "for"
*/
#pragma omp for
// ou
#pragma omp parallel for // non conseillé
for (...) {
    // Du code ...
}
```

## Méthode statique

```
// le thread n°0 va faire les tours de boucle 0 à 3
// le thread n°1 va faire les tours de boucle 4 à 7
// le thread n°2 va faire les tours de boucle 8 à 11
// ...
#pragma omp for schedule(static, 4)
```

## Méthode dynamique

```
// N'importe quel thread peut faire n'importe quel paquet de 4 itérations.
// le thread n°0 va faire les tours de boucle a,b,c,d
// le thread n°1 va faire les tours de boucle e,f,g,h
// le thread n°2 va faire les tours de boucle h,i,j,k
// ou a,b,c,d,e,f,g,h,i,j,k sont des tours de boucles aléatoires
```

```
#pragma omp for schedule(dynamic, 4)
```

```
// code exécuté uniquement par un thread :  
#pragma omp single
```

```
// Pour creer une barriere explicite ;  
#pragma omp barrier
```

### Exercice 1 :

$S = aX + Y$  avec  $a$  constante.

```
int a = cste;  
  
#pragma omp parallel num_thread(8) shared(a,X,Y) private (i) {  
  
    #pragma omp for  
    for(int i=0; i<x.size(); i++){  
        Y[i] = X[i] * a + Y[i];  
    }  
}
```

### Exercice 2 :

```
#pragma omp parallel num_thread(8) shared(m, q, a, b, m, p, colterm, rowterm) private (i, j)  
{  
  
    #pragma omp for  
    for (i=0 ; i<m ; i++){  
        rowterm[i] = 0.0 ;  
        for (j=0 ; j<p ; j++){  
            rowterm[i] += a[i][2*j] * a[i][2*j+1];  
        }  
  
        #pragma omp for  
        for (i=0 ; i<q ; i++){  
            colterm[i] = 0.0 ;  
            for (j=0 ; j<p ; j++){  
                colterm[i] += b[2*j][i] * b[2*j+1][i];  
            }  
        }  
}
```

### Exercise 3

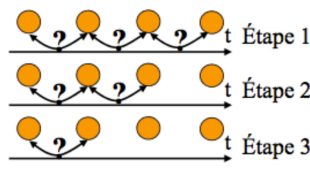
```
#pragma omp parallel private (s) {
    float temp[N] = {100.0, 0.0, 0.0, 0.0, ..., 0.0};
    int t, s, p ;

    p = omp_get_num_threads();

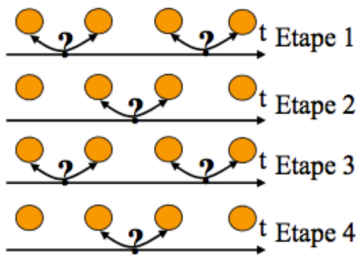
    /* calcul de d'évolution des températures */
    for (t = 0 ; t < TMAX ; t++){
        #pragma omp for schedule (N-2/p)
        for (s = 1 ; s < N-1 ; s++)
            temp[s] = (temp[s-1] + temp[s+1]) / 2.0;
    }
}
```

### Exercise 4

#### Question 1

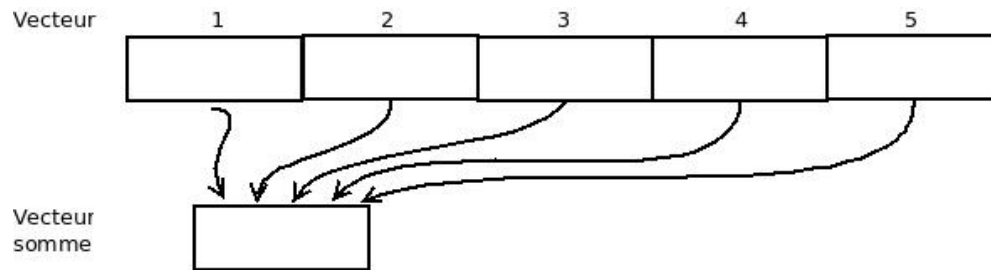
	Non, pas de parallélisme.
---	---------------------------

#### Question 2

	<pre>#pragma omp parallel private (i, temp) {     for(step=N; step &gt; 0; step-- ) {         if (step % 2 == 0) {             #pragma omp for             for (i = 0; i&lt;N-1; i+=2){                 if ( Tab[i] &gt; Tab[i+1]) {                     temp = Tab[i];                     Tab[i] = Tab[i+1]; Tab[i+1] = temp;                 }             }         } else {             #pragma omp for             for (i = 0; i&lt;N-1; i+=2) {                 if ( Tab[i] &gt; Tab[i+1]) {                     temp = Tab[i];                     Tab[i] = Tab[i+1]; Tab[i+1] = temp;                 }             }         }     } }</pre>
---	--

## Exercice 5

### Question 1



stockage de la somme dans un tableau. Chaque thread enregistre à `omp_get_thread_run()`;

```
#pragma omp parallel private (tid, i, v) {  
tid = omp_get_thread_run();  
somme[tid] = 0;  
#pragma omp for schedule(static, n/p)  
for (i=0; i<n; i++)  
    somme[tid] = somme[tid] + v[i];  
#pragma omp single  
...  
}
```

### Question 2

```
#pragma omp parallel private (tid, i, v, step, voisin) shared (somme, global_somme, m, p)  
{  
p = omp_get_num_threads();  
tid = omp_get_thread_run();  
somme[tid] = 0;  
#pragma omp for schedule(static, n/p)  
for (i=0; i<n; i++)  
    somme[tid] += v[i];  
    for(step=1; step<=p; step+=2){  
        voisin = somme[(tid+step)%p];  
        #pragma omp barrier  
        somme[tid] += voisin;  
    }  
#pragma omp single  
global_somme = somme[0];  
}
```