

### Programmation OpenMP

1 – On a un tableau d'échantillons sur 10bits, c-à-d dont la valeur est comprise entre 0 et 1023 ( $2^{10} = 1024$ ).

On veut en réaliser l'**histogramme**, c-à-d compter le nombre de fois où chaque valeur apparaît :

- ▷ on parcourt les cases du tableau de valeurs ;
- ▷ pour chaque valeur rencontrée on augmente le nombre d'occurrences associé.

tableau échantillons	3   9   0   5   4   4   5   0   6   0   4   4																				
occurrences valeur	<table border="1" style="border-collapse: collapse; width: 100%; text-align: center;"> <tr> <td style="width: 10px;">3</td><td style="width: 10px;">0</td><td style="width: 10px;">0</td><td style="width: 10px;">1</td><td style="width: 10px;">4</td><td style="width: 10px;">2</td><td style="width: 10px;">1</td><td style="width: 10px;">0</td><td style="width: 10px;">0</td><td style="width: 10px;">1</td></tr> <tr> <td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td></tr> </table>	3	0	0	1	4	2	1	0	0	1	0	1	2	3	4	5	6	7	8	9
3	0	0	1	4	2	1	0	0	1												
0	1	2	3	4	5	6	7	8	9												

} histogramme

Une version **correcte** mais **peu efficace** :

```

1 #include <omp.h>
2 #include <stdlib.h>
3 #include <stdio.h>
4 #include <assert.h>
5 #include <omp.h>          96 Méga d'entier ==> 4*6*16 = 384Mio de mémoire utilisée
6
7 #define TAILLE 6*16777216
8 #define SAMPLE 1024
9
10 void lecture(char *nom, int *donnees)
11 {
12     FILE * descripteur = fopen(nom, "r");
13     long int i = 0;
14     for(i = 0 ; i < TAILLE; i++)
15         fread(donnees+i,sizeof(int), 1,descripteur);
16     fclose(descripteur);
17 }
18
19 int main()
20 {
21     long int i = 0;
22     int histo[SAMPLE];
23     int *data = (int*) malloc(TAILLE*sizeof(int));
24     assert(data != NULL);
25 #pragma omp parallel for
26     for(i=0; i < TAILLE; i++)
27         data[i]=0;
28 #pragma omp parallel for
29     for(i=0; i < SAMPLE; i++)
30         histo[i]=0;
31
32     lecture("data.bin", data);
33
34 #pragma omp parallel for schedule(static)
35     for(i = 0; i<TAILLE; i++)
36         #pragma omp atomic Utilisation de l'atomic pour éviter les collisions d'accès
37         histo[data[i]]++;
38
39     for(i=0; i < 30; i++)
40         printf("%d, ", histo[i]);
41 }
```

Une version plus performante avec utilisation d'un tableau intermédiaire propre à chaque cœur :

```
1 #include <omp.h>
2 #include <stdlib.h>
3 #include <stdio.h>
4 #include <assert.h>
5 #include <omp.h>
6
7 #define TAILLE 6*16777216
8 #define SAMPLE 1024
9 int histo[SAMPLE];
10
11 void lecture(char *nom, int *donnees)
12 {
13     FILE * descripteur = fopen(nom, "r");
14     long int i = 0;
15     for(;i < TAILLE; i++)
16         fread(donnees+i,sizeof(int), 1,descripteur);
17     fclose(descripteur);
18 }
19
20 int main()
21 {
22     long int j = 0;
23     double start = omp_get_wtime();
24
25     int *data = (int*) malloc(TAILLE*sizeof(int));
26     assert(data != NULL);
27     lecture("data.bin", data);
28
29 #pragma omp parallel shared(histo)
30 {
31     long int i = 0;
32     int histo_local[SAMPLE];
33     for(i=0; i < SAMPLE; i++)
34         histo_local[i]=0;
35 #pragma omp for
36     for(i=0; i < SAMPLE; i++)
37         histo[i]=0;
38
39 #pragma omp for schedule(static)
40     for(i = 0; i<TAILLE; i++)
41         histo_local[data[i]]++;
42
43     for(i = 0; i<SAMPLE; i++)
44 #pragma omp atomic
45         histo[i]+= histo_local[i];
46     }
47     for(j=0; j < 30; j++)
48         printf("%d, ", histo[j]);
49     printf ("Done. %f\n",omp_get_wtime()-start);
50
51 }
```

tableau local à un cœur pour éviter les accès concurrents

on protège la combinaison des différents tableaux locaux dans le tableau global

Une version avec « réduction » manuelle :

```

1 #include <omp.h>
2 #include <stdlib.h>
3 #include <stdio.h>
4 #include <assert.h>
5 #include <omp.h>
6
7 #define TAILLE 6*16777216
8 #define SAMPLE 1024
9 #define NBCOEURS 4
10    int histo[SAMPLE];
11    int histo_coeurs[NBCOEURS][SAMPLE];
12
13 void lecture(char *nom, int *donnees)
14 {
15     FILE * descripteur = fopen(nom, "r");
16     long int i = 0;
17     for(;i < TAILLE; i++)
18         fread(donnees+i, sizeof(int), 1, descripteur);
19     fclose(descripteur);
20 }
21
22 int main()
23 {
24     long int j = 0;
25     int *data = (int*) malloc(TAILLE*sizeof(int));
26     assert(data != NULL);
27
28     lecture("data.bin", data);
29
30 #pragma omp parallel shared(histo)
31 {
32     long int i = 0;
33     long int k = 0;
34     int histo_local[SAMPLE];
35     int tid = omp_get_thread_num();
36     assert(omp_get_num_threads() == 4);
37     for(i=0; i < SAMPLE; i++)
38         histo_local[i]=0;
39
40 #pragma omp for
41     for(i=0; i < SAMPLE; i++)
42         histo[i]=0;
43
44 #pragma omp for schedule(static)
45     for(i = 0; i<TAILLE; i++)
46         histo_local[data[i]]++;
47
48     for(i = 0; i<SAMPLE; i++)
49         histo_coeurs[tid][i] = histo_local[i];
50
51 #pragma omp barrier... il faut que tous les histogrammes locaux soient finis d'être calculés
52
53 #pragma omp for schedule(static) private(i,k)
54     for(i = 0; i<SAMPLE; i++)
55         for(k = 0; k<NBCOEURS; k++) pour chaque valeur d'échantillons
56             histo[i] } ①
57             += histo_coeurs[k][i];
58     }
59     for(j=0; j < 30; j++)
60         printf("%d, ", histo[j]);
61 }
```

tableau à deux dimensions : chaque ligne correspond à l'histogramme d'un cœur

il faut que tous les histogrammes locaux soient finis d'être calculés

pour chaque valeur d'échantillons

①⇒la « réduction manuelle » calcule la somme des valeurs de chaque cœur par colonne, c-à-d toutes les occurrences de la même valeur d'échantillon pour chaque cœur.

Une version alternative de la version avec **réduction** réalisée par openMP :

```
#include <omp.h>
#include <stdlib.h>
#include <stdio.h>
#include <assert.h>
#include <omp.h>

#define TAILLE 6*16777216
#define SAMPLE 1024
#define NBCOEURS 4
    int histo[SAMPLE];
    int histo_coeurs[NBCOEURS][SAMPLE];

void lecture(char *nom, int *donnees)
{
    FILE * descripteur = fopen(nom, "r");
    long int i = 0;
    for(;i < TAILLE; i++)
        fread(donnees+i,sizeof(int), 1,descripteur);
    fclose(descripteur);
}

int main()
{
    long int j = 0;
    int *data = (int*) malloc(TAILLE*sizeof(int));
    assert(data != NULL);

    lecture("data.bin", data);

#pragma omp parallel shared(histo)
{
    int histo_local[SAMPLE];
    int tid = omp_get_thread_num();
    assert(omp_get_num_threads() == 4);
    for(int i=0; i < SAMPLE; i++)
        histo_local[i]=0;

#pragma omp for
    for(int i=0; i < SAMPLE; i++)
        histo[i]=0;

#pragma omp for schedule(static)
    for(int i = 0; i<TAILLE; i++)
        histo_local[data[i]]++;

    for(int i = 0; i<SAMPLE; i++)
        histo_coeurs[tid][i] = histo_local[i];
} fin de la région parallèle avec « barrier » implicite

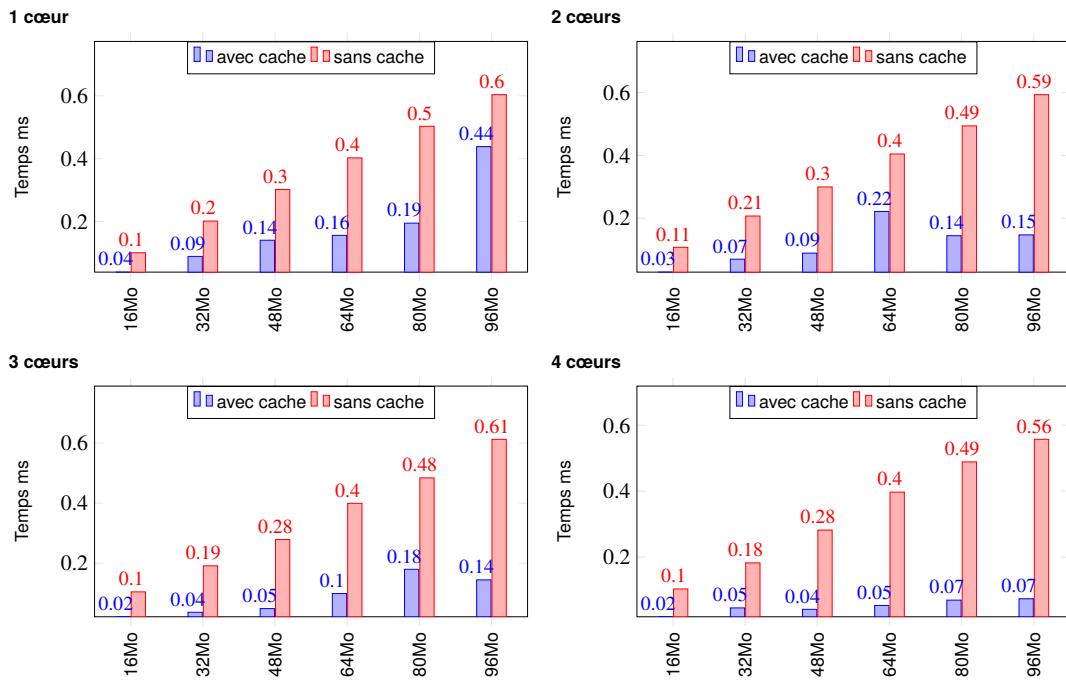
for(int i = 0; i<SAMPLE; i++)
#pragma omp parallel for schedule(static) reduction(+:histo[i])
    for(int k = 0; k<NBCOEURS; k++)
        histo[i] += histo_coeurs[k][i];
    for(j=0; j < 30; j++)
        printf("%d, ", histo[j]);
} La réduction est réalisée dans une région parallèle combinée avec le work-sharing for
```

La réduction se réalisant sur le nombre de coeurs, le gain obtenu est petit.

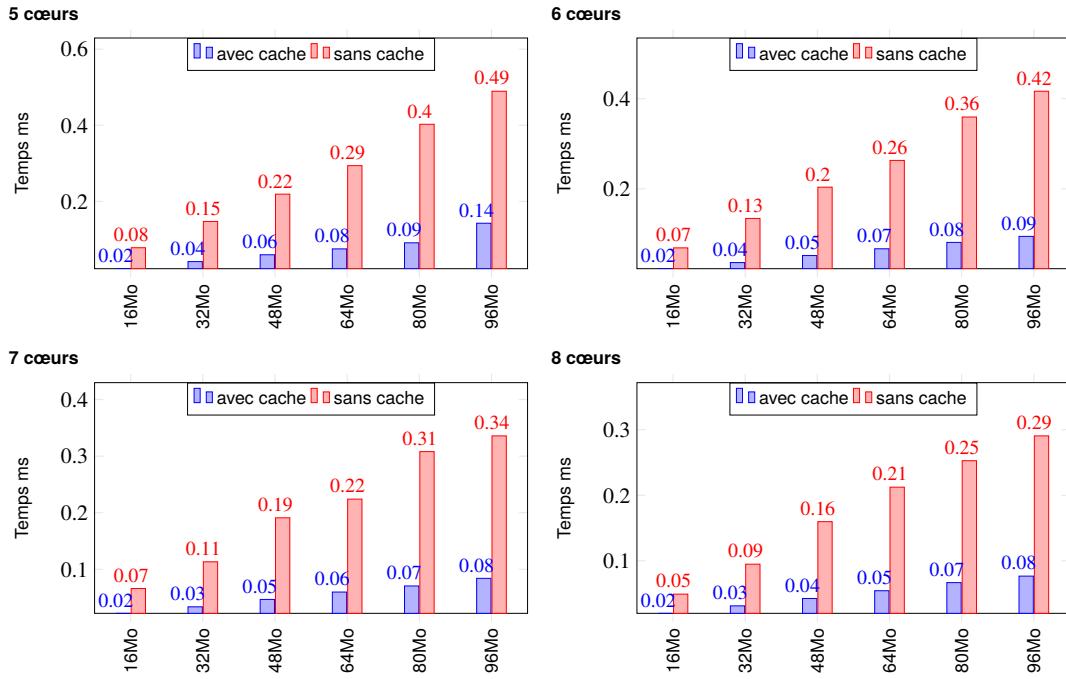
Comparaison de :

- ▷ la méthode avec « *omp atomic* » appelée « *sans cache* » ;
- ▷ la méthode « *avec histogramme local* » appelée « *avec cache* » :

#### OpenMP : Calcul histogramme



#### OpenMP : Calcul histogramme



On fait varier :

- ▷ le nombre de coeurs ;
- ▷ la taille des données manipulées (1 entier  $\Rightarrow$  4 octets).