

UNIVERSITÉ IBN TOFAIL

FACULTÉ DES SCIENCES - KENITRA

DÉPARTEMENT INFORMATIQUE

MASTER INFORMATIQUE ET INTELLIGENCE ARTIFICIELLE

COMPTE RENDU

TP 1 : Introduction à OpenMP avec une Boucle Parallèle (sous langage C)

Réalisé par :

Anas BOUKHLIJA

ANNÉE UNIVERSITAIRE 2023-2024

Table des matières

1	Exercice 1 : Parallélisation d'une Boucle Simple	4
1.1	Écrivez un programme C qui initialise un tableau de 1 million d'éléments, chacun ayant la valeur de 1.0	4
1.2	Implémentez une boucle séquentielle pour calculer la somme des éléments du tableau.	4
1.3	Utilisez la clause reduction(+ :sum) pour éviter les conditions de course.	6
1.4	Comparez et affichez les temps d'exécution séquentiel et parallèle.	8
2	Exercice 2 : Modification du Nombre de Threads	8
2.1	Utilisez le programme de l'exercice 1.	8
2.2	Modifiez le nombre de threads en utilisant la fonction <code>omp_set_num_threads()</code> au début de votre programme.	8
2.3	Comparez les temps d'exécution avec différents nombres de threads (par exemple, 1, 2, 4, 8).	10
3	Exercice 3 : Sections Parallèles	12
3.1	Ajoutez une tâche supplémentaire pour calculer la moyenne des éléments du tableau.	12
3.2	Utilisez la directive <code>#pragma omp parallel sections</code> pour paralléliser la somme et la moyenne.	14
4	Exercice 4 : Utilisation des Barrières	16
4.1	Ajoutez une section de code où les threads calculent la somme partielle de sections du tableau.	17

4.2	Utilisez la directive <code>#pragma omp barrier</code> pour synchroniser les threads avant de calculer la somme totale.	19
5	Exercice 5 : Comparaison avec OpenMP Disabled	23
5.1	Ajoutez des directives de préprocesseur pour conditionnellement compiler les directives OpenMP.	23
5.2	Compilez et exécutez le programme avec OpenMP activé et désactivé. .	26
5.3	Comparez les performances.	26

Table des figures

1	Temps d'exécution séquentiel	6
2	Temps d'exécution parallèle	7
3	Temps d'exécution séquentiel et parallèle	8
4	Utilisant la fonction <code>omp_set_num_threads</code> avec 4 Threads	10
5	Utilisant la fonction <code>omp_set_num_threads</code> avec 1, 2, 4 et 8 Threads	12
6	tâche supplémentaire pour calculer la moyenne des éléments du tableau	14
7	tâche supplémentaire pour calculer la moyenne des éléments du tableau avec <code>#pragma omp parallel</code>	16
8	Section de code où les threads calculent la somme partielle de sections du tableau	19
9	Utilisation la directive <code>#pragma omp barrier</code> pour synchroniser les threads . . .	22
10	Programme avec OpenMP activé	26
11	Programme avec OpenMP désactivé	26

1 Exercice 1 : Parallélisation d'une Boucle Simple

✓ **Objectif** : Paralléliser une boucle de sommation simple.

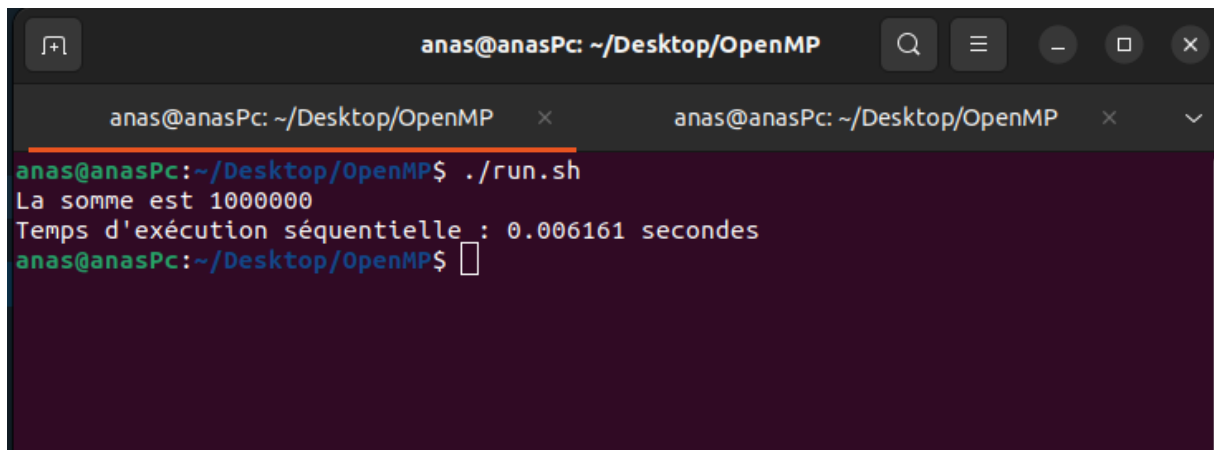
1.1 Écrivez un programme C qui initialise un tableau de 1 million d'éléments, chacun ayant la valeur de 1.0

```
1 #include <omp.h>
2 #include <stdio.h>
3 #include <stdlib.h>
4
5 int main() {
6
7     int size = 1000000;
8
9     int *array = (int *)malloc(size * sizeof(int));
10
11     // Initialisation du tableau
12     for (int i = 0; i < size; i++){
13         array[i] = 1.0;
14     }
15
16     free(array);
17     return 0;
18 }
```

1.2 Implémentez une boucle séquentielle pour calculer la somme des éléments du tableau.

```
1 #include <omp.h>
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <time.h>
5
6 int main() {
7
8     int size = 1000000;
```

```
9     int sum = 0;
10
11     int *array = (int *)malloc(size * sizeof(int));
12
13     // Mesurer le temps de debut
14     clock_t start_time = clock();
15
16     // Initialisation du tableau
17     for (int i = 0; i < size; i++){
18         array[i] = 1.0;
19     }
20
21     // Calcul de la somme
22     for (int i = 0; i < size; i++){
23         sum += array[i];
24     }
25
26     // Mesurer le temps de fin
27     clock_t end_time = clock();
28
29     // Calculer le temps d'execution en secondes
30     double time_spent = (double)(end_time - start_time) /
CLOCKS_PER_SEC;
31
32     printf("La somme est %d\n", sum);
33     printf("Temps d'execution sequentielle : %f secondes\n",
time_spent);
34
35     free(array);
36     return 0;
37 }
```



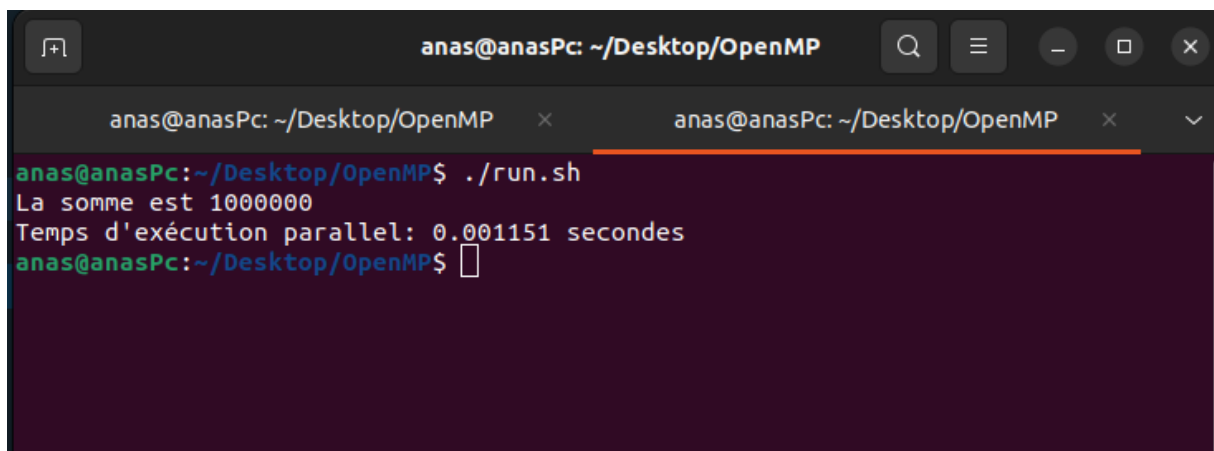
```
anas@anasPc: ~/Desktop/OpenMP
anas@anasPc:~/Desktop/OpenMP$ ./run.sh
La somme est 1000000
Temps d'exécution séquentielle : 0.006161 secondes
anas@anasPc:~/Desktop/OpenMP$
```

FIGURE 1 – Temps d'exécution séquentiel

1.3 Utilisez la clause `reduction(+ :sum)` pour éviter les conditions de course.

```
1 #include <omp.h>
2 #include <stdio.h>
3 #include <stdlib.h>
4
5 int main() {
6     int size = 1000000;
7     int sum = 0;
8
9     int *array = (int *)malloc(size * sizeof(int));
10    if (array == NULL) {
11        fprintf(stderr, "Erreur d'allocation de memoire\n");
12        return 1;
13    }
14
15    // Mesurer le temps de debut
16    double start_time = omp_get_wtime();
17
18    // Initialisation du tableau avec OpenMP
19    #pragma omp parallel for
20    for (int i = 0; i < size; i++) {
21        array[i] = 1;
22    }
23
```

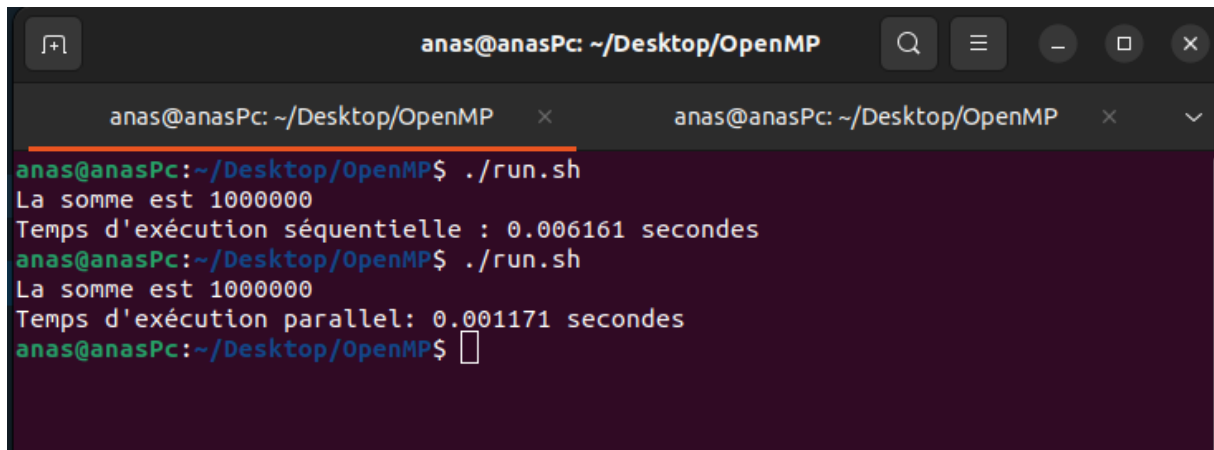
```
24 // Calcul de la somme avec reduction OpenMP
25 #pragma omp parallel for reduction(+:sum)
26 for (int i = 0; i < size; i++) {
27     sum += array[i];
28 }
29
30 // Mesurer le temps de fin
31 double end_time = omp_get_wtime();
32
33 printf("La somme est %d\n", sum);
34 printf("Temps d'exécution parallèle: %f secondes\n", end_time -
35 start_time);
36
37 free(array);
38 return 0;
39 }
```



The screenshot shows a terminal window titled 'anas@anasPc: ~/Desktop/OpenMP'. The user has executed the command './run.sh'. The output of the program is displayed in two lines: 'La somme est 1000000' and 'Temps d'exécution parallèle: 0.001151 secondes'. The prompt 'anas@anasPc:~/Desktop/OpenMP\$' is visible at the bottom of the terminal.

FIGURE 2 – Temps d'exécution parallèle

1.4 Comparez et affichez les temps d'exécution séquentiel et parallèle.



```
anas@anasPc: ~/Desktop/OpenMP
anas@anasPc: ~/Desktop/OpenMP x anas@anasPc: ~/Desktop/OpenMP x
anas@anasPc:~/Desktop/OpenMP$ ./run.sh
La somme est 1000000
Temps d'exécution séquentielle : 0.006161 secondes
anas@anasPc:~/Desktop/OpenMP$ ./run.sh
La somme est 1000000
Temps d'exécution parallèle: 0.001171 secondes
anas@anasPc:~/Desktop/OpenMP$
```

FIGURE 3 – Temps d'exécution séquentiel et parallèle

2 Exercice 2 : Modification du Nombre de Threads

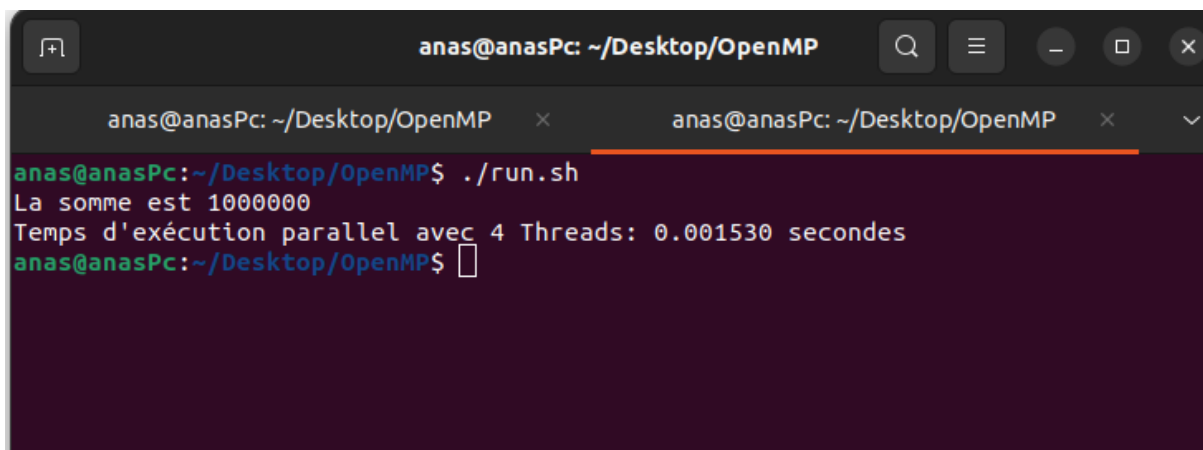
✓ **Objectif** : Observer l'impact du nombre de threads sur les performances.

2.1 Utilisez le programme de l'exercice 1.

2.2 Modifiez le nombre de threads en utilisant la fonction `omp_set_num_threads()` au début de votre programme.

```
1 #include <omp.h>
2 #include <stdio.h>
3 #include <stdlib.h>
4
5 int main() {
6     int size = 1000000;
7     int sum = 0;
8     int nb_threads = 4;
9
10    int *array = (int *)malloc(size * sizeof(int));
11    if (array == NULL) {
12        fprintf(stderr, "Erreur d'allocation de memoire\n");
13        return 1;
14    }
```

```
14     }
15
16     // Définir le nombre de threads
17     omp_set_num_threads(nb_threads);
18
19
20     // Mesurer le temps de debut
21     double start_time = omp_get_wtime();
22
23     // Initialisation du tableau avec OpenMP
24     #pragma omp parallel for
25     for (int i = 0; i < size; i++) {
26         array[i] = 1;
27     }
28
29     // Calcul de la somme avec reduction OpenMP
30     #pragma omp parallel for reduction(+:sum)
31     for (int i = 0; i < size; i++) {
32         sum += array[i];
33     }
34
35     // Mesurer le temps de fin
36     double end_time = omp_get_wtime();
37
38     printf("La somme est %d\n", sum);
39     printf("Temps d'exécution parallèle avec %d Threads: %f\n", nb_threads, end_time - start_time);
40
41     free(array);
42     return 0;
43 }
```



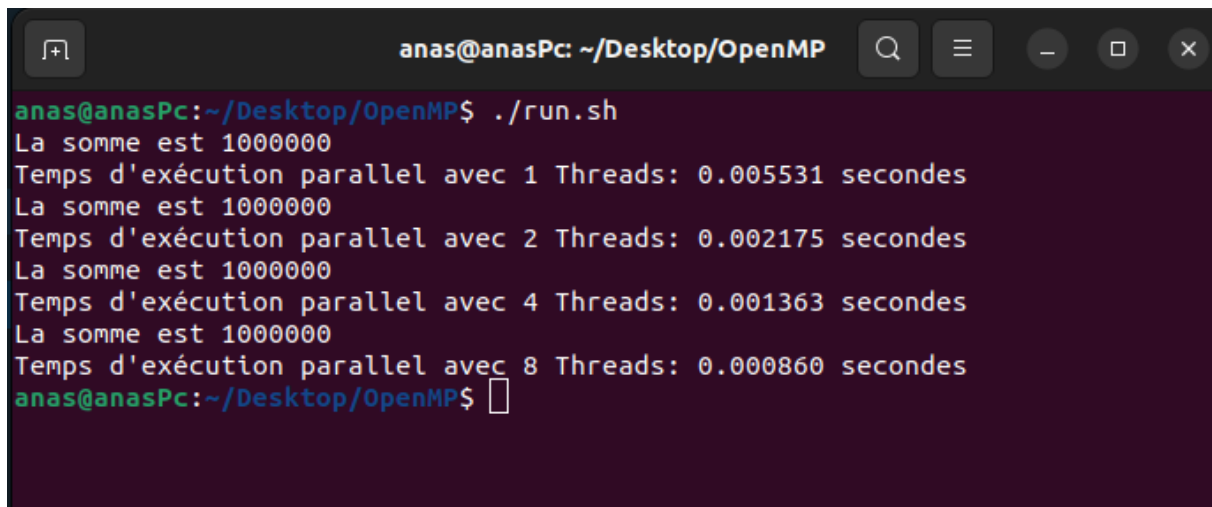
The screenshot shows a terminal window titled 'anas@anasPc: ~/Desktop/OpenMP'. The prompt is 'anas@anasPc:~/Desktop/OpenMP\$'. The user has entered './run.sh'. The output of the program is: 'La somme est 1000000' followed by 'Temps d'exécution parallèle avec 4 Threads: 0.001530 secondes'. The prompt is now 'anas@anasPc:~/Desktop/OpenMP\$' with a cursor.

FIGURE 4 – Utilisant la fonction `omp_set_num_threads` avec 4 Threads

2.3 Comparez les temps d'exécution avec différents nombres de threads (par exemple, 1, 2, 4, 8).

```
1 #include <omp.h>
2 #include <stdio.h>
3 #include <stdlib.h>
4
5 int main()
6 {
7     int size = 1000000;
8
9     int *array = (int *)malloc(size * sizeof(int));
10    if (array == NULL)
11    {
12        fprintf(stderr, "Erreur d'allocation de memoire\n");
13        return 1;
14    }
15
16    int nb_threads_table[4] = {1, 2, 4, 8};
17
18    for (int j = 0; j < 4; j++)
19    {
20        int sum = 0;
21
22        int nb_threads = nb_threads_table[j];
```

```
23
24 // Définir le nombre de threads
25 omp_set_num_threads(nb_threads);
26
27 // Mesurer le temps de debut
28 double start_time = omp_get_wtime();
29
30 // Initialisation du tableau avec OpenMP
31 #pragma omp parallel for
32 for (int i = 0; i < size; i++)
33 {
34     array[i] = 1;
35 }
36
37 // Calcul de la somme avec reduction OpenMP
38 #pragma omp parallel for reduction(+ : sum)
39 for (int i = 0; i < size; i++)
40 {
41     sum += array[i];
42 }
43
44 // Mesurer le temps de fin
45 double end_time = omp_get_wtime();
46
47 printf("La somme est %d\n", sum);
48 printf("Temps d'execution parallel avec %d Threads: %f
49 secondes\n", nb_threads, end_time - start_time);
50 }
51
52 free(array);
53 return 0;
54 }
```



```
anas@anasPc: ~/Desktop/OpenMP
anas@anasPc:~/Desktop/OpenMP$ ./run.sh
La somme est 10000000
Temps d'exécution parallèle avec 1 Threads: 0.005531 secondes
La somme est 10000000
Temps d'exécution parallèle avec 2 Threads: 0.002175 secondes
La somme est 10000000
Temps d'exécution parallèle avec 4 Threads: 0.001363 secondes
La somme est 10000000
Temps d'exécution parallèle avec 8 Threads: 0.000860 secondes
anas@anasPc:~/Desktop/OpenMP$
```

FIGURE 5 – Utilisant la fonction `omp_set_num_threads` avec 1, 2, 4 et 8 Threads

3 Exercice 3 : Sections Parallèles

✓ **Objectif** : Diviser une tâche en sections parallèles.

3.1 Ajoutez une tâche supplémentaire pour calculer la moyenne des éléments du tableau.

```
1 #include <omp.h>
2 #include <stdio.h>
3 #include <stdlib.h>
4
5 int main()
6 {
7     int size = 1000000;
8
9     int *array = (int *)malloc(size * sizeof(int));
10    if (array == NULL)
11    {
12        fprintf(stderr, "Erreur d'allocation de mmoire\n");
13        return 1;
14    }
15
16    int nb_threads_table[4] = {1, 2, 4, 8};
```

```
17
18     for (int j = 0; j < 4; j++)
19     {
20         int sum = 0;
21         double average = 0.0;
22
23         int nb_threads = nb_threads_table[j];
24
25         // Définir le nombre de threads
26         omp_set_num_threads(nb_threads);
27
28         // Mesurer le temps de debut
29         double start_time = omp_get_wtime();
30
31         // Initialisation du tableau avec OpenMP
32         #pragma omp parallel for
33         for (int i = 0; i < size; i++)
34         {
35             array[i] = 1;
36         }
37
38         // Calcul de la somme avec reduction OpenMP
39         #pragma omp parallel for reduction(+ : sum)
40         for (int i = 0; i < size; i++)
41         {
42             sum += array[i];
43         }
44
45         // Calcul de la moyenne
46         average = (double)sum / size;
47
48         // Mesurer le temps de fin
49         double end_time = omp_get_wtime();
50
51         printf("La somme est %d\n", sum);
52         printf("La moyenne est %f\n", average);
```

```

53     printf("Temps d'exécution parallèle avec %d Threads: %f
secondes\n", nb_threads, end_time - start_time);
54 }
55
56 free(array);
57 return 0;
58 }

```

```

anas@anasPc: ~/Desktop/OpenMP
anas@anasPc:~/Desktop/OpenMP$ ./run.sh
La somme est 1000000
La moyenne est 1.000000
Temps d'exécution parallèle avec 1 Threads: 0.005401 secondes
La somme est 1000000
La moyenne est 1.000000
Temps d'exécution parallèle avec 2 Threads: 0.002254 secondes
La somme est 1000000
La moyenne est 1.000000
Temps d'exécution parallèle avec 4 Threads: 0.001380 secondes
La somme est 1000000
La moyenne est 1.000000
Temps d'exécution parallèle avec 8 Threads: 0.001668 secondes
anas@anasPc:~/Desktop/OpenMP$

```

FIGURE 6 – tâche supplémentaire pour calculer la moyenne des éléments du tableau

3.2 Utilisez la directive `#pragma omp parallel sections` pour paralléliser la somme et la moyenne.

```

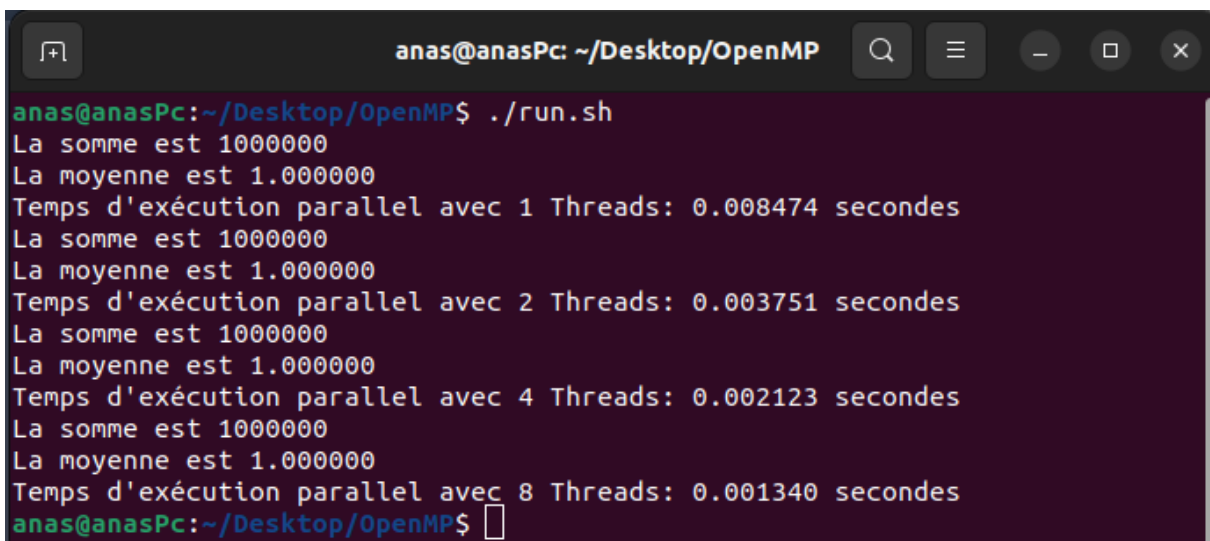
1 #include <omp.h>
2 #include <stdio.h>
3 #include <stdlib.h>
4
5 int main()
6 {
7     int size = 1000000;
8
9     int *array = (int *)malloc(size * sizeof(int));
10    if (array == NULL)
11    {

```

```
12     fprintf(stderr, "Erreur d'allocation de mmoire\n");
13     return 1;
14 }
15
16 int nb_threads_table[4] = {1, 2, 4, 8};
17
18 for (int j = 0; j < 4; j++)
19 {
20     int sum = 0;
21     double average = 0.0;
22     int nb_threads = nb_threads_table[j];
23
24     // Définir le nombre de threads
25     omp_set_num_threads(nb_threads);
26
27     // Mesurer le temps de debut
28     double start_time = omp_get_wtime();
29
30     // Initialisation du tableau avec OpenMP
31     #pragma omp parallel for
32     for (int i = 0; i < size; i++)
33     {
34         array[i] = 1;
35     }
36
37     // Calcul de la somme avec reduction OpenMP
38     #pragma omp parallel for reduction(+ : sum)
39     for (int i = 0; i < size; i++)
40     {
41         sum += array[i];
42     }
43
44     // Moyenne en parallele
45     #pragma omp parallel for reduction(+ : average)
46     for (int i = 0; i < size; i++)
47     {
48         average += (double)array[i];
```



```
49     }
50
51     // Calcul de la moyenne
52     average /= size;
53
54     // Mesurer le temps de fin
55     double end_time = omp_get_wtime();
56
57     printf("La somme est %d\n", sum);
58     printf("La moyenne est %f\n", average);
59     printf("Temps d'exécution parallèle avec %d Threads: %f
secondes\n", nb_threads, end_time - start_time);
60 }
61
62 free(array);
63 return 0;
64 }
```



```
anas@anasPc: ~/Desktop/OpenMP
anas@anasPc:~/Desktop/OpenMP$ ./run.sh
La somme est 1000000
La moyenne est 1.000000
Temps d'exécution parallèle avec 1 Threads: 0.008474 secondes
La somme est 1000000
La moyenne est 1.000000
Temps d'exécution parallèle avec 2 Threads: 0.003751 secondes
La somme est 1000000
La moyenne est 1.000000
Temps d'exécution parallèle avec 4 Threads: 0.002123 secondes
La somme est 1000000
La moyenne est 1.000000
Temps d'exécution parallèle avec 8 Threads: 0.001340 secondes
anas@anasPc:~/Desktop/OpenMP$
```

FIGURE 7 – tâche supplémentaire pour calculer la moyenne des éléments du tableau avec `#pragma omp parallel`

4 Exercice 4 : Utilisation des Barrières

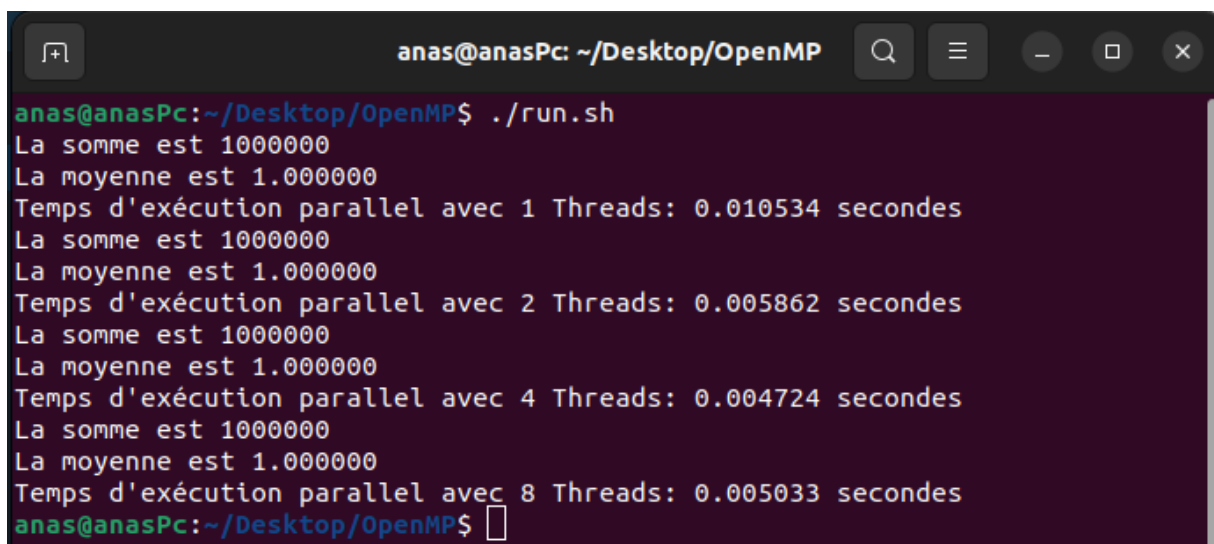
✓ **Objectif** : Comprendre l'utilisation des barrières en OpenMP.

4.1 Ajoutez une section de code où les threads calculent la somme partielle de sections du tableau.

```
1 #include <omp.h>
2 #include <stdio.h>
3 #include <stdlib.h>
4
5 int main()
6 {
7     int size = 1000000;
8     int num_sections = 8; // Nombre de sections    diviser le
    tableau
9
10    int *array = (int *)malloc(size * sizeof(int));
11    if (array == NULL)
12    {
13        fprintf(stderr, "Erreur d'allocation de m moire\n");
14        return 1;
15    }
16
17    int nb_threads_table[4] = {1, 2, 4, 8};
18
19    for (int j = 0; j < 4; j++)
20    {
21        int sum = 0;
22        double average = 0.0;
23        int nb_threads = nb_threads_table[j];
24
25        // D finir le nombre de threads
26        omp_set_num_threads(nb_threads);
27
28        // Mesurer le temps de d but
29        double start_time = omp_get_wtime();
30
31        // Initialisation du tableau avec OpenMP
32        #pragma omp parallel for
33        for (int i = 0; i < size; i++)
```

```
34     {
35         array[i] = 1;
36     }
37
38 // Somme en parallèle
39 #pragma omp parallel for reduction(+ : sum)
40     for (int i = 0; i < size; i++)
41     {
42         sum += array[i];
43     }
44
45 // Moyenne en parallèle
46 #pragma omp parallel for reduction(+ : average)
47     for (int i = 0; i < size; i++)
48     {
49         average += (double)array[i];
50     }
51
52 // Calcul de la moyenne
53 average /= size;
54
55 // Somme partielle en parallèle
56 int partial_sum[num_sections];
57
58 #pragma omp parallel for
59     for (int i = 0; i < num_sections; i++)
60     {
61         int start_index = i * (size / num_sections);
62         int end_index = (i + 1) * (size / num_sections);
63
64         partial_sum[i] = 0;
65         for (int j = start_index; j < end_index; j++)
66         {
67             partial_sum[i] += array[j];
68         }
69     }
70
```

```
71 // Somme des sommes partielles
72 int total_sum = 0;
73 for (int i = 0; i < num_sections; i++)
74 {
75     total_sum += partial_sum[i];
76 }
77 // Mesurer le temps de fin
78 double end_time = omp_get_wtime();
79
80 printf("La somme est %d\n", sum);
81 printf("La moyenne est %f\n", average);
82 printf("Temps d'ex cution parallel avec %d Threads: %f
83 secondes\n", nb_threads, end_time - start_time);
84 }
85
86 free(array);
87 return 0;
88 }
```



```
anas@anasPc: ~/Desktop/OpenMP
anas@anasPc:~/Desktop/OpenMP$ ./run.sh
La somme est 1000000
La moyenne est 1.000000
Temps d'exécution parallel avec 1 Threads: 0.010534 secondes
La somme est 1000000
La moyenne est 1.000000
Temps d'exécution parallel avec 2 Threads: 0.005862 secondes
La somme est 1000000
La moyenne est 1.000000
Temps d'exécution parallel avec 4 Threads: 0.004724 secondes
La somme est 1000000
La moyenne est 1.000000
Temps d'exécution parallel avec 8 Threads: 0.005033 secondes
anas@anasPc:~/Desktop/OpenMP$
```

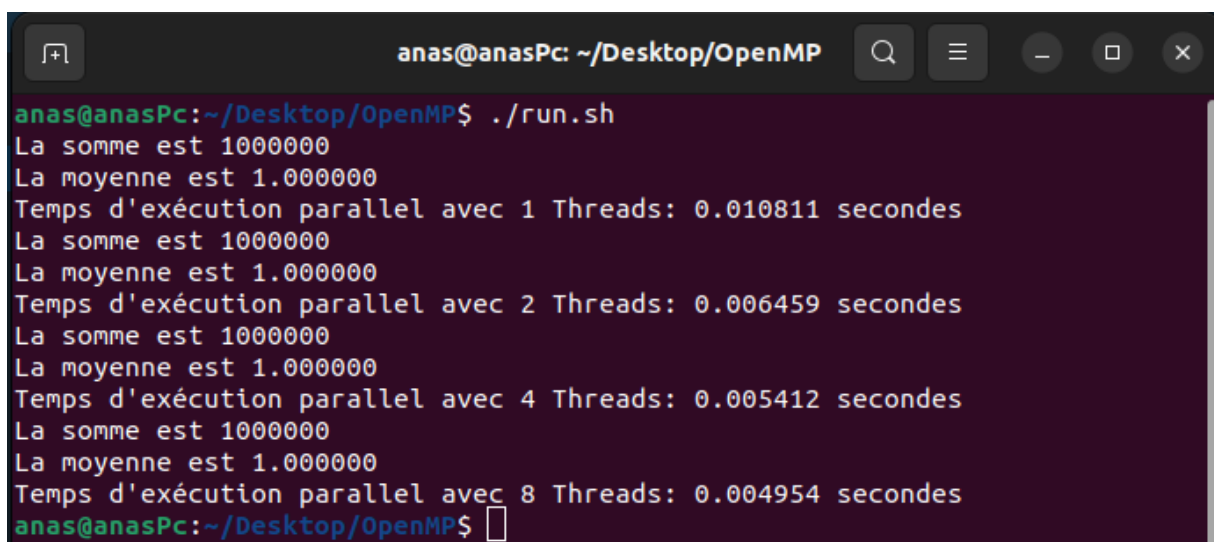
FIGURE 8 – Section de code où les threads calculent la somme partielle de sections du tableau

4.2 Utilisez la directive `#pragma omp barrier` pour synchroniser les threads avant de calculer la somme totale.

```
1 #include <omp.h>
2 #include <stdio.h>
3 #include <stdlib.h>
4
5 int main()
6 {
7     int size = 1000000;
8     int num_sections = 8; // Nombre de sections    diviser le
    tableau
9
10    int *array = (int *)malloc(size * sizeof(int));
11    if (array == NULL)
12    {
13        fprintf(stderr, "Erreur d'allocation de mmoire\n");
14        return 1;
15    }
16
17    int nb_threads_table[4] = {1, 2, 4, 8};
18
19    for (int j = 0; j < 4; j++)
20    {
21        int sum = 0;
22        double average = 0.0;
23        int nb_threads = nb_threads_table[j];
24
25        // D finir le nombre de threads
26        omp_set_num_threads(nb_threads);
27
28        // Mesurer le temps de d but
29        double start_time = omp_get_wtime();
30
31        // Initialisation du tableau avec OpenMP
32        #pragma omp parallel for
33        for (int i = 0; i < size; i++)
34        {
35            array[i] = 1;
36        }
```

```
37
38 // Somme en parallèle
39 #pragma omp parallel for reduction(+ : sum)
40 for (int i = 0; i < size; i++)
41 {
42     sum += array[i];
43 }
44
45 // Moyenne en parallèle
46 #pragma omp parallel for reduction(+ : average)
47 for (int i = 0; i < size; i++)
48 {
49     average += (double)array[i];
50 }
51
52 // Calcul de la moyenne
53 average /= size;
54
55 // Somme partielle en parallèle
56 int partial_sum[num_sections];
57
58 #pragma omp parallel for
59 for (int i = 0; i < num_sections; i++)
60 {
61     int start_index = i * (size / num_sections);
62     int end_index = (i + 1) * (size / num_sections);
63
64     partial_sum[i] = 0;
65     for (int j = start_index; j < end_index; j++)
66     {
67         partial_sum[i] += array[j];
68     }
69 }
70
71 // Synchroniser les threads avant de calculer la somme
72 // totale
73 #pragma omp barrier
```

```
73
74     // Somme des sommes partielles
75     int total_sum = 0;
76     for (int i = 0; i < num_sections; i++)
77     {
78         total_sum += partial_sum[i];
79     }
80     // Mesurer le temps de fin
81     double end_time = omp_get_wtime();
82
83     printf("La somme est %d\n", total_sum);
84     printf("La moyenne est %f\n", average);
85     printf("Temps d'exécution parallèle avec %d Threads: %f
86     secondes\n", nb_threads, end_time - start_time);
87 }
88
89 free(array);
90 return 0;
91 }
```



```
anas@anasPc: ~/Desktop/OpenMP
anas@anasPc:~/Desktop/OpenMP$ ./run.sh
La somme est 1000000
La moyenne est 1.000000
Temps d'exécution parallèle avec 1 Threads: 0.010811 secondes
La somme est 1000000
La moyenne est 1.000000
Temps d'exécution parallèle avec 2 Threads: 0.006459 secondes
La somme est 1000000
La moyenne est 1.000000
Temps d'exécution parallèle avec 4 Threads: 0.005412 secondes
La somme est 1000000
La moyenne est 1.000000
Temps d'exécution parallèle avec 8 Threads: 0.004954 secondes
anas@anasPc:~/Desktop/OpenMP$
```

FIGURE 9 – Utilisation la directive #pragma omp barrier pour synchroniser les threads

5 Exercice 5 : Comparaison avec OpenMP Disabled

✓ **Objectif** : Mesurer l'impact d'OpenMP en désactivant les directives.

5.1 Ajoutez des directives de préprocesseur pour conditionnellement compiler les directives OpenMP.

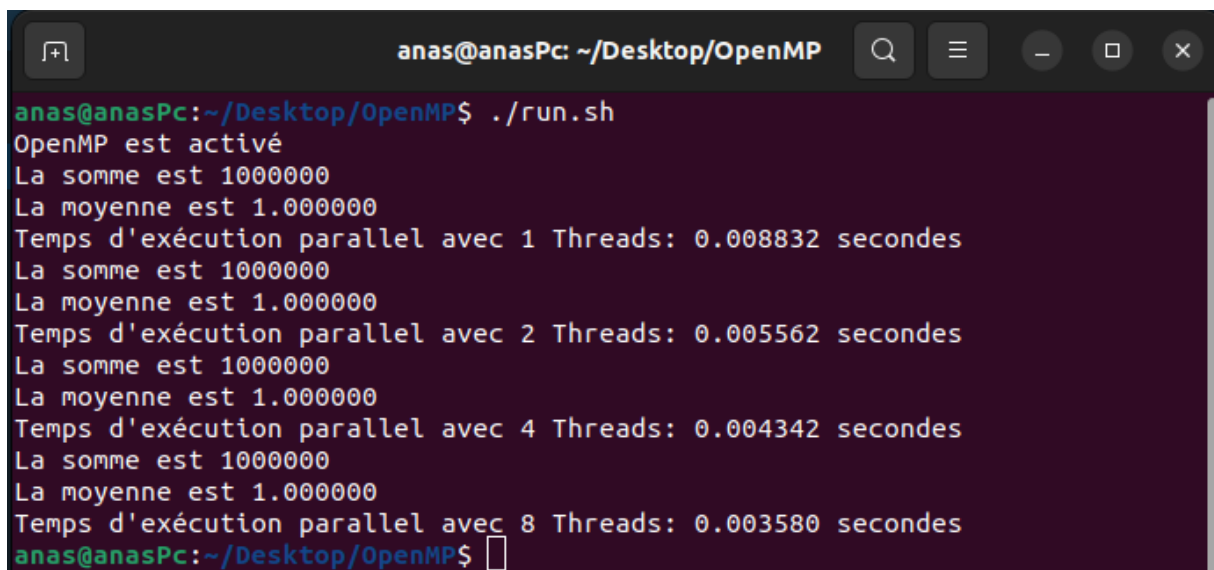
```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 // Define a macro to control OpenMP compilation
5 #define USE_OPENMP 0
6
7 #ifndef USE_OPENMP
8 #include <omp.h>
9 #endif
10
11 int main()
12 {
13     printf("OpenMP est %s\n", USE_OPENMP ? "active" : "desactive");
14     ;
15
16     int size = 1000000;
17     int num_sections = 8; // Nombre de sections a diviser le
18     tableau
19
20     int *array = (int *)malloc(size * sizeof(int));
21     if (array == NULL)
22     {
23         fprintf(stderr, "Erreur d'allocation de memoire\n");
24         return 1;
25     }
26
27     int nb_threads_table[4] = {1, 2, 4, 8};
28
29     for (int j = 0; j < 4; j++)
30     {
```



```
29     double average = 0.0;
30     int nb_threads = nb_threads_table[j];
31
32     // Mesurer le temps de debut
33     double start_time = omp_get_wtime();
34
35     // Initialisation du tableau avec ou sans OpenMP
36     #if USE_OPENMP
37     omp_set_num_threads(nb_threads);
38     #pragma omp parallel for
39     #endif
40     for (int i = 0; i < size; i++)
41     {
42         array[i] = 1;
43     }
44
45     // Moyenne en parallele avec ou sans OpenMP
46     #if USE_OPENMP
47     #pragma omp parallel for reduction(+ : average)
48     #endif
49     for (int i = 0; i < size; i++)
50     {
51         average += (double)array[i];
52     }
53
54     // Calcul de la moyenne
55     average /= size;
56
57     // Somme partielle en parallele avec ou sans OpenMP
58     int partial_sum[num_sections];
59
60     #if USE_OPENMP
61     #pragma omp parallel for
62     #endif
63     for (int i = 0; i < num_sections; i++)
64     {
65         int start_index = i * (size / num_sections);
```

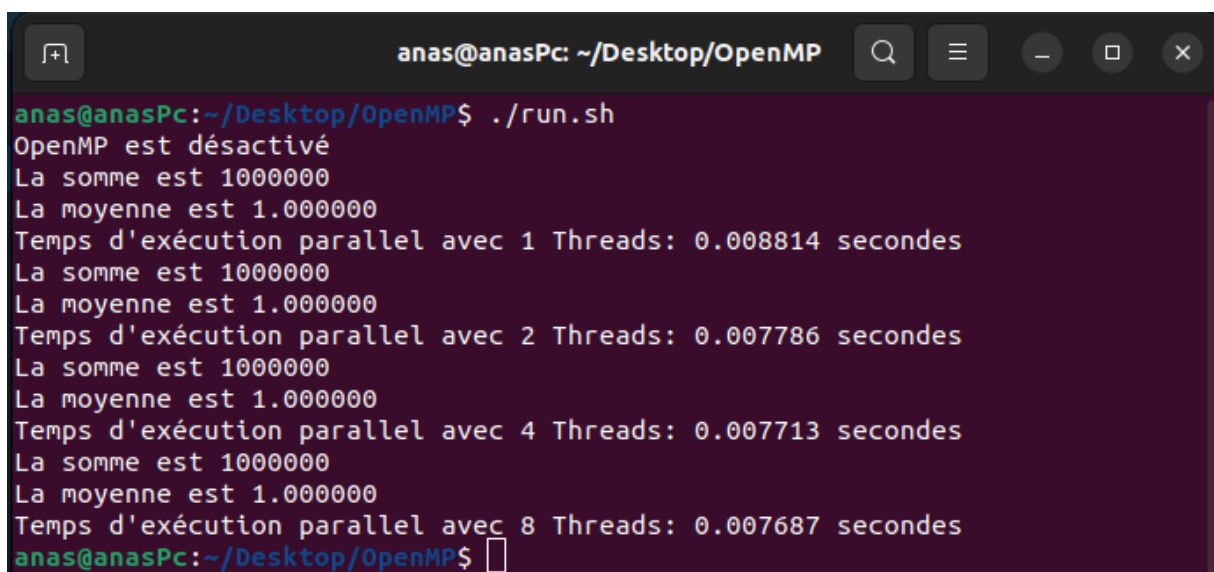
```
66         int end_index = (i + 1) * (size / num_sections);
67
68         partial_sum[i] = 0;
69         for (int j = start_index; j < end_index; j++)
70         {
71             partial_sum[i] += array[j];
72         }
73     }
74
75     // Synchroniser les threads avant de calculer la somme
76     // totale avec ou sans OpenMP
77     #if USE_OPENMP
78     #pragma omp barrier
79     #endif
80
81     // Somme des sommes partielles avec ou sans OpenMP
82     int total_sum = 0;
83     for (int i = 0; i < num_sections; i++)
84     {
85         total_sum += partial_sum[i];
86     }
87
88     // Mesurer le temps de fin
89     double end_time = omp_get_wtime();
90
91     printf("La somme est %d\n", total_sum);
92     printf("La moyenne est %f\n", average);
93     printf("Temps d'exécution parallèle avec %d Threads: %f
94     secondes\n", nb_threads, end_time - start_time);
95 }
96
97 free(array);
98 return 0;
99 }
```

5.2 Compilez et exécutez le programme avec OpenMP activé et désactivé.



```
anas@anasPc: ~/Desktop/OpenMP
anas@anasPc:~/Desktop/OpenMP$ ./run.sh
OpenMP est activé
La somme est 1000000
La moyenne est 1.000000
Temps d'exécution parallèle avec 1 Threads: 0.008832 secondes
La somme est 1000000
La moyenne est 1.000000
Temps d'exécution parallèle avec 2 Threads: 0.005562 secondes
La somme est 1000000
La moyenne est 1.000000
Temps d'exécution parallèle avec 4 Threads: 0.004342 secondes
La somme est 1000000
La moyenne est 1.000000
Temps d'exécution parallèle avec 8 Threads: 0.003580 secondes
anas@anasPc:~/Desktop/OpenMP$
```

FIGURE 10 – Programme avec OpenMP activé



```
anas@anasPc: ~/Desktop/OpenMP
anas@anasPc:~/Desktop/OpenMP$ ./run.sh
OpenMP est désactivé
La somme est 1000000
La moyenne est 1.000000
Temps d'exécution parallèle avec 1 Threads: 0.008814 secondes
La somme est 1000000
La moyenne est 1.000000
Temps d'exécution parallèle avec 2 Threads: 0.007786 secondes
La somme est 1000000
La moyenne est 1.000000
Temps d'exécution parallèle avec 4 Threads: 0.007713 secondes
La somme est 1000000
La moyenne est 1.000000
Temps d'exécution parallèle avec 8 Threads: 0.007687 secondes
anas@anasPc:~/Desktop/OpenMP$
```

FIGURE 11 – Programme avec OpenMP désactivé

5.3 Comparez les performances.

En comparant les performances entre OpenMP activé et désactivé, on peut observer une différence significative dans les temps d'exécution parallèle avec différents nombres de threads.

En conclusion, l'activation d'OpenMP a considérablement amélioré les performances en réduisant les temps d'exécution, ce qui montre l'avantage de l'utilisation de la parallélisation pour ce type de traitement.