Université Ibn Tofaïl

Faculté des Sciences
كلية العلوم

UNIVERSITÉ IBN TOFAIL

FACULTÉ DES SCIENCES - KENITRA

DÉPARTEMENT INFORMATIQUE

MASTER INFORMATIQUE ET INTELLIGENCE ARTIFICIELLE

COMPTE RENDU

TP 4 : Réduction en OpenMP

Réalisé par :
Anas BOUKHLIJA

Encadré par :
Pr. Nada FAQIR

ANNÉE UNIVERSITAIRE 2024-2025

Table des matières

Table des figures

1 Exercice 1 : Somme parallèle avec OpenMP

1.1 Initialisation du tableau d'entiers

```
    // Initialisation du tableau d'entiers
    // -----------------------------------------
    int *table = (int *) malloc(size * sizeof(int));
    if (table == NULL) {
        printf("Erreur lors de la location de la memoire!\n");
        return 1;
    }
    for (int i = 0; i < size ; i++){
        table[i] = i + 1;
    }
```

1.2 Implémentation de la boucle séquentielle pour calculer la somme

```
    // Implementation de la boucle sequentielle pour calculer la
   somme
    // -----------------------------------------
    int sum_sequential = 0;
    for (int i = 0; i < size; i++) {
        sum_sequential += table[i];
    }
    printf("La somme avec une boucle sequentialle est %d\n",
   sum_sequential);
```

resultat est toujours : La somme avec une boucle séquentialle est 887459712 (size de tableau est 100 000 000)

1.3 Implémentation de la boucle parallèle avec OpenMP

```
    // Implementation de la boucle parallele avec OpenMP
    // -----------------------------------------
    int sum_parallel = 0;
    #pragma omp parallel for
    for (int i = 0; i < size; i++) {
```

```
        sum_parallel += table[i];
    }
    printf("La somme avec une boucle parallele est %d\n",
sum_parallel);
```

Dans cette version parallèle, le résultat est instable en raison des modifications concurrentes de la variable sum-parallele. En effet, plusieurs threads tentent d'accéder à cette variable en même temps, ce qui entraîne des résultats incorrects. Pour résoudre ce problème, il est nécessaire d'utiliser une clause de réduction (reduction) qui permet de combiner les résultats partiels de chaque thread de manière sûre et efficace.

1.4 Utilisation de la clause `reduction(+: sum)` pour la gestion de la somme

```
    // Utilisation de la clause reduction(+: sum)
    // pour la gestion de la somme
    // -----------------------------------------
    int sum_parallel = 0;
    #pragma omp parallel for reduction(+:sum_parallel)
    for (int i = 0; i < size; i++) {
        sum_parallel += table[i];
    }
    printf("La somme avec une boucle parallele est %d\n",
sum_parallel);
```

1.5 Mesure du temps d'exécution de la version séquentielle avec `omp_get_wtime()`

```
    // Mesure du temps d'execution de la version sequentielle
    // avec omp_get_wtime()
    // -----------------------------------------
    int sum_sequential = 0;
    double start_sequential, end_sequential;
    start_sequential = omp_get_wtime();
    for (int i = 0; i < size; i++) {
        sum_sequential += table[i];
    }
    end_sequential = omp_get_wtime();
    printf("La somme avec une boucle sequentialle est %d\n",
sum_sequential);
```

```
12     double execution_time_sequential = end_sequential -
   start_sequential;
13     printf("Le Temps d'Execution Sequentialle: %f seconds\n",
   execution_time_sequential);
```

Résultat de l'exécution

==========

La somme avec une boucle séquentialle est 987459712

Le Temps d'Exécution Séquentialle : 0.218421 seconds

1.6 Mesure du temps d'exécution de la version parallèle avec OpenMP et `omp_get_wtime()`

```
1
2      // Mesure du temps d'execution de la version parallele
3      // avec OpenMP et omp_get_wtime()
4      // ---------------------------------------------
5      int sum_parallel = 0;
6      double start_parallelt, end_parallel;
7      start_parallelt = omp_get_wtime();
8      int num_threads = 0;
9
10     #pragma omp parallel
11     {
12         num_threads = omp_get_num_threads();
13         #pragma omp for reduction(+:sum_parallel)
14         for (int i = 0; i < size; i++) {
15             sum_parallel += table[i];
16         }
17     }
18     end_parallel = omp_get_wtime();
19     printf("La somme avec une boucle parallele est %d\n",
   sum_parallel);
20     double execution_time_parallel = end_parallel -
   start_parallelt;
21     printf("Le Temps d'Execution parallele (%d threads): %f
   seconds\n", num_threads, execution_time_parallel);
```

Résultat de l'exécution

==========

La somme avec une boucle parallele est 987459712
Le Temps d'Exécution parallele (8 threads) : 0.059099 seconds

1.7 Comparaison des temps d'exécution et analyse des résultats

Dans cette section, nous comparons les temps d'exécution entre les versions séquentielle et parallèle, et nous analysons les résultats obtenus.

L'efficacité d'une exécution parallèle est définie comme le rapport entre le temps d'exécution séquentiel et le temps d'exécution parallèle. Plus l'efficacité est proche du nombre de threads utilisés, meilleure est la performance de la version parallèle.

```
1     // Comparaison des temps d'execution
2     //  ----------------------------------------
3     double speed_up = execution_time_sequential /
    execution_time_parallel;
4     printf("L'acceleration avec %d threads est: %f\n", num_threads
    , speed_up);
5     printf("L'efficacite avec %d threads est: %f\n\n", num_threads
    , speed_up / num_threads);
```

Résultat de l'exécution

==========

L'acceleration avec 8 threads est : 3.695833
L'efficacite avec 8 threads est : 0.461979

L'efficacité obtenue est de 3.49 pour 8 threads. Cela montre une accélération notable par rapport à la version séquentielle, mais l'efficacité n'est pas proche de 8, ce qui indique que le programme n'exploite pas pleinement les ressources parallèles.

2 Exercice 2 : Produit parallèle avec OpenMP

2.1 Calcul le produit des éléments

```
1     // Calcul le produit des elements
2     // ----------------------------------------
```

```
    long double product_sequential = 1;
    double start_sequential, end_sequential;
    start_sequential = omp_get_wtime();
    for (int i = 0; i < size; i++) {
        product_sequential *= table[i];
    }
    end_sequential = omp_get_wtime();
    printf("La produit avec une boucle sequentialle est %Lf\n",
product_sequential);
    double execution_time_sequential = end_sequential -
start_sequential;
    printf("Le Temps d'Execution Sequentialle: %f seconds\n",
execution_time_sequential);
```

Résultat de l'exécution

==========

La produit avec une boucle séquentialle est inf

Le Temps d'Exécution Séquentialle : 10.818171 seconds

2.2 Utilisation de la clause `reduction(*: product)` pour le calcul du produit en parallèle

```
    // version parallele de calcul le produit des element
    // ------------------------------------------
    long double product_parallel = 1;
    double start_parallelt, end_parallel;
    start_parallelt = omp_get_wtime();
    int num_threads = 0;

    #pragma omp parallel
    {
        num_threads = omp_get_num_threads();
        #pragma omp for reduction(*:product_parallel)
        for (int i = 0; i < size; i++) {
            product_parallel *= table[i];
        }
    }
```

```
16      end_parallel = omp_get_wtime();
17      printf("Le produit avec une boucle parallele est %Lf\n",
        product_parallel);
18      double execution_time_parallel = end_parallel -
        start_parallelt;
19      printf("Le Temps d'Execution parallele (%d threads): %f
        seconds\n", num_threads, execution_time_parallel);
```

Résultat de l'exécution

==========

Le produit avec une boucle parallele est inf

Le Temps d'Exécution parallele (8 threads) : 1.726439 seconds

2.3 Mesure de l'efficacité de la parallélisation

```
1       // Comparaison des temps d'execution
2       //  ----------------------------------------
3       double speed_up = execution_time_sequential /
        execution_time_parallel;
4       printf("L'acceleration avec %d threads est: %f\n", num_threads
        , speed_up);
5       printf("L'efficacite avec %d threads est: %f\n", num_threads,
        speed_up / num_threads);
```

Résultat de l'exécution

==========

L'acceleration avec 8 threads est : 6.266174

L'efficacite avec 8 threads est : 0.783272

Le rapport d'accélération est plus proche du nombre de threads (8) que dans le programme de somme. Cela s'explique par l'augmentation de la complexité du calcul du produit, qui est généralement plus coûteux que le calcul de la somme, en raison de la multiplication répétée des éléments. Cette différence de complexité se reflète dans le temps d'exécution parallèle, qui reste relativement plus rapide, mais plus sensible à la taille du tableau (la taille utilise est 100 000 000).

3 Exercice 3 : Maximum et Minimum parallèle

3.1 Ajout de la boucle parallèle pour calculer le maximum des éléments

```
    // Calcul le maximum des elements
    // ---------------------------------------------
    int max_sequential = table[0];
    double max_start_sequentialt, max_end_sequential;
    max_start_sequentialt = omp_get_wtime();
    for (int i = 0; i < size; i++) {
        if (table[i] > max_sequential) {
            max_sequential = table[i];
        }
    }
    max_end_sequential = omp_get_wtime();
    printf("Le Max avec une boucle sequentialle est %d\n",
   max_sequential);
    double max_execution_time_sequential = max_end_sequential -
   max_start_sequentialt;
    printf("Le Temps d'Execution Sequentialle: %f seconds\n",
   max_execution_time_sequential);
```

Résultat de l'exécution

==========

Le Max avec une boucle séquentialle est 100000000

Le Temps d'Exécution Séquentialle : 0.257226 seconds

3.2 Ajout de la boucle parallèle pour calculer le minimum des éléments

```
    // Calcul le minimum des elements
    // ---------------------------------------------
    int min_sequential = table[0];
    double min_start_sequentialt, min_end_sequential;
    min_start_sequentialt = omp_get_wtime();
    for (int i = 0; i < size; i++) {
        if (table[i] < min_sequential) {
            min_sequential = table[i];
        }
```

```
    }
    min_end_sequential = omp_get_wtime();
    printf("Le Min avec une boucle sequentialle est %d\n",
min_sequential);
    double min_execution_time_sequential = min_end_sequential -
min_start_sequentialt;
    printf("Le Temps d'Execution Sequentialle: %f seconds\n",
min_execution_time_sequential);
```

Résultat de l'exécution

==========

Le Min avec une boucle séquentialle est 1

Le Temps d'Exécution Séquentialle : 0.232014 seconds

3.3 Utilisation de la clause `reduction(max: max_value)` pour le calcul du maximum

```
    // Version parallele de calcul le maximum des elements
    // ---------------------------------------------
    int max_parallel = table[0];
    double max_start_parallel, max_end_parallel;
    max_start_parallel = omp_get_wtime();
    int max_num_threads = 0;

    #pragma omp parallel
    {
        max_num_threads = omp_get_num_threads();
        #pragma omp for reduction(max: max_parallel)
        for (int i = 0; i < size; i++) {
            if (table[i] > max_parallel) {
                max_parallel = table[i];
            }
        }
    }

    max_end_parallel = omp_get_wtime();
    printf("Le Max avec une boucle Parallele est %d\n",
```

```
    max_parallel);
21   double max_execution_time_parallel = max_end_parallel -
    max_start_parallel;
22   printf("Le Temps d'Execution Parallele: %f seconds\n",
    max_execution_time_parallel);
```

Résultat de l'exécution

==========

Le Max avec une boucle Parallele est 100000000

Le Temps d'Exécution Parallele : 0.054619 seconds

3.4 Utilisation de la clause `reduction(min: min_value)` pour le calcul du minimum

```
1    // Version parallele de calcul le minimum des elements
2    // ------------------------------------------
3    int min_parallel = table[0];
4    double min_start_parallel, min_end_parallel;
5    min_start_parallel = omp_get_wtime();
6    int min_num_threads = 0;
7
8    #pragma omp parallel
9    {
10       min_num_threads = omp_get_num_threads();
11       #pragma omp for reduction(min: min_parallel)
12       for (int i = 0; i < size; i++) {
13           if (table[i] < min_parallel) {
14               min_parallel = table[i];
15           }
16       }
17   }
18   min_end_parallel = omp_get_wtime();
19   printf("Le Min avec une boucle Parallele est %d\n",
    min_parallel);
20   double min_execution_time_parallel = min_end_parallel -
    min_start_parallel;
21   printf("Le Temps d'Execution Parallele: %f seconds\n",
```

```
    min_execution_time_parallel);
```

Résultat de l'exécution

==========

Le Min avec une boucle Parallele est 1

Le Temps d'Exécution Parallele : 0.042414 seconds

3.5 Vérification des résultats par comparaison avec la version séquentielle

```
1     // Comparaison des temps d'ex cution pour le Max
2     //  ----------------------------------------
3     double max_speed_up = max_execution_time_sequential /
    max_execution_time_parallel;
4     printf("L'acceleration cas de Max avec %d threads est: %f\n",
    max_num_threads, max_speed_up);
5     printf("L'efficacite cas de Max avec %d threads est: %f\n",
    max_num_threads, max_speed_up / max_num_threads);
6
7     // Comparaison des temps d'execution pour le Min
8     //  ----------------------------------------
9     double min_speed_up = min_execution_time_sequential /
    min_execution_time_parallel;
10    printf("L'acceleration cas de Min avec %d threads est: %f\n",
    min_num_threads, min_speed_up);
11    printf("L'efficacite cas de Min avec %d threads est: %f\n",
    min_num_threads, min_speed_up / min_num_threads);
```

Résultat de l'exécution

==========

L'acceleration cas de Max avec 8 threads est : 4.709453

L'efficacite cas de Max avec 8 threads est : 0.588682

L'acceleration cas de Min avec 8 threads est : 5.470217

L'efficacite cas de Min avec 8 threads est : 0.683777

4 Exercice 4 : Optimisation et discussion

4.1 Modification de la taille du tableau pour observer l'impact sur les performances

On choiser l'example de le produit et on va changer la taille de tableau de 100 000 000 jusqua 1 000 000 000

```c
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#include <time.h>

#define INITIAL_SIZE 100000000
#define SIZE_INCREMENT 100000000
#define NUM_TESTS 10

int main() {
    for (int test = 0; test < NUM_TESTS; test++) {
        int size = INITIAL_SIZE + test * SIZE_INCREMENT;

        // Initialisation du tableau d'entiers
        // -------------------------------------------
        int *table = (int *) malloc(size * sizeof(int));
        srand(time(NULL));
        if (table == NULL) {
            printf("Erreur lors de la location de la memoire!\n");
            return 1;
        }
        for (int i = 0; i < size; i++) {
            table[i] = i + 1;
        }

        // Mesure du temps d'execution de la version sequentielle
        // -------------------------------------------
        double product_sequential = 1;
        double start_sequentialt, end_sequential;
        start_sequentialt = omp_get_wtime();
```

```
31        for (int i = 0; i < size; i++) {
32            product_sequential *= table[i];
33        }
34        end_sequential = omp_get_wtime();
35        printf("Taille du tableau: %d, Le produit avec une boucle
    sequentielle est %f\n", size, product_sequential);
36        double execution_time_sequential = end_sequential -
    start_sequentialt;
37        printf("Le Temps d'Execution Sequentielle: %f seconds\n",
    execution_time_sequential);
38
39        // Mesure du temps d'execution de la version parallele
    avec OpenMP
40        // -----------------------------------------
41        double product_parallel = 1;
42        double start_parallel, end_parallel;
43        start_parallel = omp_get_wtime();
44        int num_threads = 0;
45
46        #pragma omp parallel
47        {
48            num_threads = omp_get_num_threads();
49            #pragma omp for reduction(*:product_parallel)
50            for (int i = 0; i < size; i++) {
51                product_parallel *= table[i];
52            }
53        }
54        end_parallel = omp_get_wtime();
55        printf("Taille du tableau: %d, Le produit avec une boucle
    parallele est %f\n", size, product_parallel);
56        double execution_time_parallel = end_parallel -
    start_parallel;
57        printf("Le Temps d'Execution parallele (%d threads): %f
    seconds\n", num_threads, execution_time_parallel);
58
59        // Comparaison des temps d'execution
60        //  -----------------------------------------
```

```
61          double speed_up = execution_time_sequential /
    execution_time_parallel;
62          printf("L'acceleration %d threads est: %f\n", num_threads,
     speed_up);
63          printf("L'efficacite %d threads est: %f\n", num_threads,
    speed_up / num_threads);
64
65          // Liberation de la memoire
66          free(table);
67      }
68
69      return 0;
70  }
```

Résultat de l'exécution

==========

Taille du tableau : 100000000, Le produit avec une boucle séquentielle est inf

Le Temps d'Exécution Séquentielle : 0.281788 seconds

Taille du tableau : 100000000, Le produit avec une boucle parallèle est inf

Le Temps d'Exécution parallèle (8 threads) : 0.072148 seconds

L'acceleration 8 threads est : 3.905676

L'efficacite 8 threads est : 0.488210

==========

Taille du tableau : 200000000, Le produit avec une boucle séquentielle est inf

Le Temps d'Exécution Séquentielle : 0.534907 seconds

Taille du tableau : 200000000, Le produit avec une boucle parallèle est inf

Le Temps d'Exécution parallèle (8 threads) : 0.099036 seconds

L'acceleration 8 threads est : 5.401144

L'efficacite 8 threads est : 0.675143

==========

Taille du tableau : 300000000, Le produit avec une boucle séquentielle est inf

Le Temps d'Exécution Séquentielle : 0.895768 seconds

Taille du tableau : 300000000, Le produit avec une boucle parallèle est inf

Le Temps d'Exécution parallèle (8 threads) : 0.189648 seconds

L'acceleration 8 threads est : 4.723317

L'efficacite 8 threads est : 0.590415

==========
Taille du tableau : 400000000, Le produit avec une boucle séquentielle est inf
Le Temps d'Exécution Séquentielle : 1.143215 seconds
Taille du tableau : 400000000, Le produit avec une boucle parallèle est inf
Le Temps d'Exécution parallèle (8 threads) : 0.213992 seconds
L'acceleration 8 threads est : 5.342335
L'efficacite 8 threads est : 0.667792
==========
Taille du tableau : 500000000, Le produit avec une boucle séquentielle est inf
Le Temps d'Exécution Séquentielle : 1.290290 seconds
Taille du tableau : 500000000, Le produit avec une boucle parallèle est inf
Le Temps d'Exécution parallèle (8 threads) : 0.249294 seconds
L'acceleration 8 threads est : 5.175780
L'efficacite 8 threads est : 0.646972
==========
Taille du tableau : 600000000, Le produit avec une boucle séquentielle est inf
Le Temps d'Exécution Séquentielle : 1.541737 seconds
Taille du tableau : 600000000, Le produit avec une boucle parallèle est inf
Le Temps d'Exécution parallèle (8 threads) : 0.292646 seconds
L'acceleration 8 threads est : 5.268268
L'efficacite 8 threads est : 0.658534
==========
Taille du tableau : 700000000, Le produit avec une boucle séquentielle est inf
Le Temps d'Exécution Séquentielle : 1.853374 seconds
Taille du tableau : 700000000, Le produit avec une boucle parallèle est inf
Le Temps d'Exécution parallèle (8 threads) : 0.376670 seconds
L'acceleration 8 threads est : 4.920424
L'efficacite 8 threads est : 0.615053
==========
Taille du tableau : 800000000, Le produit avec une boucle séquentielle est inf
Le Temps d'Exécution Séquentielle : 2.125888 seconds
Taille du tableau : 800000000, Le produit avec une boucle parallèle est inf
Le Temps d'Exécution parallèle (8 threads) : 0.435443 seconds
L'acceleration 8 threads est : 4.882130
L'efficacite 8 threads est : 0.610266

Taille du tableau : 900000000, Le produit avec une boucle séquentielle est inf
Le Temps d'Exécution Séquentielle : 2.439402 seconds
Taille du tableau : 900000000, Le produit avec une boucle parallèle est inf
Le Temps d'Exécution parallèle (8 threads) : 0.442780 seconds
L'acceleration 8 threads est : 5.509281
L'efficacite 8 threads est : 0.688660
==========
Taille du tableau : 1000000000, Le produit avec une boucle séquentielle est inf
Le Temps d'Exécution Séquentielle : 2.693982 seconds
Taille du tableau : 1000000000, Le produit avec une boucle parallèle est inf
Le Temps d'Exécution parallèle (8 threads) : 0.552035 seconds
L'acceleration 8 threads est : 4.880097
L'efficacite 8 threads est : 0.610012

Les résultats obtenus montrent une amélioration notable des temps d'exécution lorsque le calcul du produit est parallélisé à l'aide d'OpenMP. Plus la taille du tableau augmente, plus l'accélération avec 8 threads se rapproche de la valeur optimale, oscillant autour de 5,3 à 5,4. Ces résultats indiquent que des facteurs comme le surcoût de gestion des threads et les accès concurrents à la mémoire limitent les gains de performance.

4.2 Augmentation du nombre de threads OpenMP avec `omp_set_num_threads()`

```c
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#include <time.h>

#define INITIAL_SIZE 100000000
#define SIZE_INCREMENT 100000000
#define NUM_TESTS 3
#define MAX_THREADS 8

int main() {
    for (int test = 0; test < NUM_TESTS; test++) {
        int size = INITIAL_SIZE + test * SIZE_INCREMENT;

        // Initialisation du tableau d'entiers
```

```
16          // ------------------------------------------
17          int *table = (int *) malloc(size * sizeof(int));
18          if (table == NULL) {
19              printf("Erreur lors de la location de la memoire!\n");
20              return 1;
21          }
22
23          // Fill the table with values
24          for (int i = 0; i < size; i++) {
25              table[i] = i + 1;
26          }
27
28          // Mesure du temps d'execution de la version sequentielle
29          // ------------------------------------------
30          double product_sequential = 1;
31          double start_sequentialt, end_sequential;
32          start_sequentialt = omp_get_wtime();
33          for (int i = 0; i < size; i++) {
34              product_sequential *= table[i];
35          }
36          end_sequential = omp_get_wtime();
37          printf("Taille du tableau: %d, Le produit avec une boucle
    sequentielle est %f\n", size, product_sequential);
38          double execution_time_sequential = end_sequential -
    start_sequentialt;
39          printf("Le Temps d'Execution Sequentielle: %f seconds\n",
    execution_time_sequential);
40
41          // Test with different numbers of threads
42          for (int num_threads = 2; num_threads <= MAX_THREADS;
    num_threads+=2) {
43              // Set the number of threads to use in OpenMP
44              omp_set_num_threads(num_threads);
45
46              // Mesure du temps d'execution de la version parallele
     avec OpenMP
47              // ------------------------------------------
```

```
48              double product_parallel = 1;
49              double start_parallel, end_parallel;
50              start_parallel = omp_get_wtime();
51
52              #pragma omp parallel
53              {
54                  #pragma omp for reduction(*:product_parallel)
55                  for (int i = 0; i < size; i++) {
56                      product_parallel *= table[i];
57                  }
58              }
59              end_parallel = omp_get_wtime();
60              printf("Taille du tableau: %d, Le produit avec une
    boucle parallele est %f\n", size, product_parallel);
61              double execution_time_parallel = end_parallel -
    start_parallel;
62              printf("Le Temps d'Execution parallele (%d threads): %
    f seconds\n", num_threads, execution_time_parallel);
63
64              // Comparaison des temps d'execution
65              // ----------------------------------------
66              double speed_up = execution_time_sequential /
    execution_time_parallel;
67              printf("L'acceleration avec %d threads est: %f\n",
    num_threads, speed_up);
68              printf("L'efficacite avec %d threads est: %f\n\n",
    num_threads, speed_up / num_threads);
69          }
70
71          // Liberation de la memoire
72          free(table);
73      }
74
75      return 0;
76  }
```

4.3 Discussion de l'impact du nombre de threads sur les performances

Résultat de l'exécution

==========

Taille du tableau : 100000000, Le produit avec une boucle séquentielle est inf

Le Temps d'Exécution Séquentielle : 0.283210 seconds

Taille du tableau : 100000000, Le produit avec une boucle parallèle est inf

Le Temps d'Exécution parallèle (2 threads) : 0.148243 seconds

L'acceleration avec 2 threads est : 1.910453

L'efficacite avec 2 threads est : 0.955226

==========

Taille du tableau : 100000000, Le produit avec une boucle parallèle est inf

Le Temps d'Exécution parallèle (4 threads) : 0.079169 seconds

L'acceleration avec 4 threads est : 3.577296

L'efficacite avec 4 threads est : 0.894324

==========

Taille du tableau : 100000000, Le produit avec une boucle parallèle est inf

Le Temps d'Exécution parallèle (6 threads) : 0.064591 seconds

L'acceleration avec 6 threads est : 4.384700

L'efficacite avec 6 threads est : 0.730783

==========

Taille du tableau : 100000000, Le produit avec une boucle parallèle est inf

Le Temps d'Exécution parallèle (8 threads) : 0.060951 seconds

L'acceleration avec 8 threads est : 4.646521

L'efficacite avec 8 threads est : 0.580815

==========

Taille du tableau : 200000000, Le produit avec une boucle séquentielle est inf

Le Temps d'Exécution Séquentielle : 0.547276 seconds

Taille du tableau : 200000000, Le produit avec une boucle parallèle est inf

Le Temps d'Exécution parallèle (2 threads) : 0.276995 seconds

L'acceleration avec 2 threads est : 1.975761

L'efficacite avec 2 threads est : 0.987880

==========

Taille du tableau : 200000000, Le produit avec une boucle parallèle est inf

Le Temps d'Exécution parallèle (4 threads) : 0.151035 seconds

L'acceleration avec 4 threads est : 3.623516
L'efficacite avec 4 threads est : 0.905879
==========
Taille du tableau : 200000000, Le produit avec une boucle parallèle est inf
Le Temps d'Exécution parallèle (6 threads) : 0.124961 seconds
L'acceleration avec 6 threads est : 4.379571
L'efficacite avec 6 threads est : 0.729929
==========
Taille du tableau : 200000000, Le produit avec une boucle parallèle est inf
Le Temps d'Exécution parallèle (8 threads) : 0.108868 seconds
L'acceleration avec 8 threads est : 5.026964
L'efficacite avec 8 threads est : 0.628370
==========
Taille du tableau : 300000000, Le produit avec une boucle séquentielle est inf
Le Temps d'Exécution Séquentielle : 0.812966 seconds
Taille du tableau : 300000000, Le produit avec une boucle parallèle est inf
Le Temps d'Exécution parallèle (2 threads) : 0.413046 seconds
L'acceleration avec 2 threads est : 1.968223
L'efficacite avec 2 threads est : 0.984112
==========
Taille du tableau : 300000000, Le produit avec une boucle parallèle est inf
Le Temps d'Exécution parallèle (4 threads) : 0.214019 seconds
L'acceleration avec 4 threads est : 3.798579
L'efficacite avec 4 threads est : 0.949645
==========
Taille du tableau : 300000000, Le produit avec une boucle parallèle est inf
Le Temps d'Exécution parallèle (6 threads) : 0.187181 seconds
L'acceleration avec 6 threads est : 4.343205
L'efficacite avec 6 threads est : 0.723868
==========
Taille du tableau : 300000000, Le produit avec une boucle parallèle est inf
Le Temps d'Exécution parallèle (8 threads) : 0.148373 seconds
L'acceleration avec 8 threads est : 5.479201
L'efficacite avec 8 threads est : 0.684900

L'efficacité tend à diminuer avec l'augmentation du nombre de threads, indiquant une surcharge liée à la gestion de ceux-ci. En somme, l'approche parallèle est particulièrement avantageuse pour le traitement de grands ensembles de données, mais nécessite une optimisation attentive pour maximiser les performances.