**Travaux Pratiques : Réduction en OpenMP**

**Objectif**

Dans ce TP, vous allez explorer l'utilisation des réductions en OpenMP pour différentes opérations (somme, produit, maximum, minimum). Vous allez également mesurer l'efficacité de la parallélisation avec OpenMP pour ces opérations.

**Exercice 1 : Somme parallèle avec OpenMP**

1. Créez un programme qui initialise un tableau d'entiers de taille 10 000 avec des valeurs aléatoires.
2. Implémentez une boucle parallèle avec OpenMP pour calculer la somme des éléments du tableau.
3. Utilisez la clause reduction(+: sum) pour gérer la somme en parallèle.
4. Comparez le temps d'exécution de la version séquentielle et de la version parallèle avec OpenMP en mesurant le temps avec omp\_get\_wtime().

**Exercice 2 : Produit parallèle avec OpenMP**

1. Modifiez votre programme pour calculer le produit des éléments du tableau.
2. Utilisez la clause reduction(\*: product) pour paralléliser le calcul du produit.
3. Assurez-vous que la variable de produit est correctement initialisée à 1.
4. Mesurez l'efficacité de la parallélisation.

**Exercice 3 : Maximum et Minimum parallèle**

1. Ajoutez deux nouvelles boucles parallèles pour calculer le maximum et le minimum des éléments du tableau.
2. Utilisez les clauses reduction(max: max\_value) et reduction(min: min\_value) respectivement.
3. Vérifiez les résultats en comparant avec la version séquentielle.

**Exercice 4 : Optimisation et discussion**

1. Modifiez la taille du tableau pour observer l'impact sur les performances.
2. Augmentez le nombre de threads OpenMP avec la fonction omp\_set\_num\_threads().
3. Discutez de l'impact du nombre de threads sur les performances. Y a-t-il une amélioration pour toutes les opérations ?

**Exemples de code pour démarrer**

**Exercice 1 : Somme parallèle**

#include <stdio.h>

#include <stdlib.h>

#include <omp.h>

int main() {

int i, n = 10000;

double sum = 0.0;

double \*array = (double\*) malloc(n \* sizeof(double));

// Initialisation du tableau avec des valeurs aléatoires

for (i = 0; i < n; i++)

array[i] = rand() % 100;

double start\_time = omp\_get\_wtime();

// Parallélisation de la somme

#pragma omp parallel for reduction(+: sum)

for (i = 0; i < n; i++) {

sum += array[i];

}

double end\_time = omp\_get\_wtime();

printf("La somme des éléments est : %f\n", sum);

printf("Temps d'exécution : %f secondes\n", end\_time - start\_time);

free(array);

return 0;

}

**Exercice 2 : Produit parallèle**

#include <stdio.h>

#include <stdlib.h>

#include <omp.h>

int main() {

int i, n = 10000;

double product = 1.0;

double \*array = (double\*) malloc(n \* sizeof(double));

// Initialisation du tableau avec des valeurs aléatoires

for (i = 0; i < n; i++)

array[i] = (rand() % 10) + 1; // On évite les 0 pour le produit

double start\_time = omp\_get\_wtime();

// Parallélisation du produit

#pragma omp parallel for reduction(\*: product)

for (i = 0; i < n; i++) {

product \*= array[i];

}

double end\_time = omp\_get\_wtime();

printf("Le produit des éléments est : %f\n", product);

printf("Temps d'exécution : %f secondes\n", end\_time - start\_time);

free(array);

return 0;

}

**Conclusion** La réduction en OpenMP est un outil puissant pour paralléliser efficacement des opérations qui accumulent des résultats partiels à travers des boucles. Ce TP vous permet d'expérimenter avec différentes opérations et d'observer l'impact de la parallélisation sur les performances.