### TP3 : Génération de l'Ensemble de Mandelbrot avec OpenMP

**Objectifs**

* Apprendre à utiliser les tâches avec OpenMP.
* Paralléliser un problème complexe et irrégulier.
* Générer une image de l'ensemble de Mandelbrot en utilisant des tâches OpenMP.

**Matériel Requis**

* Un compilateur compatible OpenMP.
* Un éditeur de texte ou un IDE.

Pour le TP3, nous allons aborder un concept avancé d'OpenMP : la gestion des tâches et la parallélisation des sections de code indépendantes. Ce TP portera sur l'utilisation de la directive task d'OpenMP pour paralléliser un problème de calcul fractal, tel que la génération d'un ensemble de Mandelbrot.

#### Introduction à la Gestion des Tâches

La gestion des tâches dans OpenMP permet de paralléliser des sections de code qui ne peuvent pas être facilement exprimées sous forme de boucles. Elle est particulièrement utile pour des algorithmes récursifs ou pour des calculs qui ont des dépendances complexes.

## Directives OpenMP Utilisées

* #pragma omp task : Utilisée pour définir une nouvelle tâche. Les tâches sont des unités de travail qui peuvent être exécutées de manière parallèle.
* #pragma omp taskwait : Utilisée pour indiquer qu'un thread doit attendre que toutes les tâches enfants qu'il a créées soient terminées.

##### **Exemple de Base**

***#include <omp.h>***

***#include <stdio.h>***

***void task\_example(int n) {***

***#pragma omp task***

***{***

***printf("Task %d executed by thread %d\n", n, omp\_get\_thread\_num());***

***}***

***}***

***int main() {***

***omp\_set\_num\_threads(4);***

***#pragma omp parallel***

***{***

***#pragma omp single***

***{***

***for (int i = 0; i < 10; i++) {***

***task\_example(i);***

***}***

***}***

***}***

***return 0;***

***}***

## Étapes du TP

**1. Introduction à l'Ensemble de Mandelbrot**

L'ensemble de Mandelbrot est un ensemble de points dans le plan complexe dont les coordonnées sont définies par une condition de convergence d'une suite itérative. Il est visualisé en traçant les points dans le plan qui appartiennent à cet ensemble.

**2. Code Séquentiel pour Générer l'Ensemble de Mandelbrot**

Voici un exemple de code séquentiel pour générer une image en niveaux de gris de l'ensemble de Mandelbrot.

#include <stdio.h>

#include <stdlib.h>

#include <complex.h>

#define WIDTH 800

#define HEIGHT 800

#define MAX\_ITER 1000

int mandelbrot(double complex c) {

double complex z = 0 + 0\*I;

int iter;

for (iter = 0; iter < MAX\_ITER; iter++) {

if (cabs(z) > 2.0) break;

z = z\*z + c;

}

return iter;

}

void generate\_mandelbrot(unsigned char image[HEIGHT][WIDTH]) {

for (int y = 0; y < HEIGHT; y++) {

for (int x = 0; x < WIDTH; x++) {

double complex c = (x - WIDTH/2.0)\*4.0/WIDTH + (y - HEIGHT/2.0)\*4.0/HEIGHT\*I;

int value = mandelbrot(c);

image[y][x] = (value == MAX\_ITER) ? 0 : (255 \* value / MAX\_ITER);

}

}

}

int main() {

unsigned char image[HEIGHT][WIDTH];

generate\_mandelbrot(image);

FILE \*fp = fopen("mandelbrot.pgm", "wb");

fprintf(fp, "P5\n%d %d\n255\n", WIDTH, HEIGHT);

fwrite(image, 1, WIDTH\*HEIGHT, fp);

fclose(fp);

printf("Image générée : mandelbrot.pgm\n");

return 0;

}

**3. Parallélisation avec OpenMP**

Parallélisez ce code en utilisant des tâches OpenMP.

**4. Analyse des Performances**

1. **Comparer les temps d'exécution séquentiel et parallèle.**
2. **Tester avec différents nombres de threads en utilisant omp\_set\_num\_threads().**
3. **Analyser l'impact de la gestion des tâches sur la performance.**

**Conclusion**

Ce TP permet d'explorer les concepts avancés d'OpenMP, en particulier la gestion des tâches, et d'appliquer ces concepts à un problème complexe et intéressant : la génération de l'ensemble de Mandelbrot. En utilisant des tâches, vous pouvez paralléliser efficacement des calculs irréguliers et améliorer les performances de votre code.

**Exercices Supplémentaires (optionnels)**

1. **Optimisation des Tâches :**
   * Expérimenter avec la taille des tâches pour voir comment cela affecte les performances.
   * Utiliser taskloop au lieu de task pour simplifier la parallélisation des boucles.
2. **Amélioration de la Visualisation :**
   * Générer une image en couleurs en utilisant une palette de couleurs pour les différentes itérations.
3. **Gestion Dynamique des Tâches :**
   * Implémenter une version où les tâches sont créées dynamiquement en fonction de la complexité des calculs (zones de l'ensemble de Mandelbrot qui nécessitent plus d'itérations).

**Explications des Exercices Supplémentaires**

1. **Optimisation des Tâches :**
   * **Blocs de Tâches :** En regroupant les lignes en blocs, vous réduisez le nombre de tâches, ce qui diminue le surcoût de gestion.
   * **Taskloop :** La directive taskloop simplifie la parallélisation des boucles et permet de définir la granularité des tâches avec grainsize.
2. **Amélioration de la Visualisation :**
   * **Image en Couleurs :** En utilisant une palette de couleurs, vous améliorez la visualisation de l'ensemble de Mandelbrot en fonction du nombre d'itérations nécessaires pour déterminer la convergence.
3. **Gestion Dynamique des Tâches :**
   * **Subdivision Dynamique :** En subdivisant dynamiquement les régions en fonction de la complexité, vous équilibrez mieux la charge de travail entre les threads et optimisez l'utilisation des ressources.