TP OpenMP – Méthode de Newton

0. Serveurs disponibles

- **etud** : GNU/Linux, AMD Opteron 6272 (64 cores), gcc 4.4 – OpenMP 3.0
- **omer** : GNU/Linux, AMD Opteron QuadCore 8356 (32 cores), gcc 4.1 – OpenMP 2.5
- **berzet** : Debian, AMD Opteron 8350 (16 cores), gcc 4.9 – OpenMP 4.0
- **master0** : **192.168.100.161**, CentOS 6.7, Quad_Core AMD Opteron 2352 (8 cores) , gcc 4.4 – OpenMP 3.0

1. Prise en main : éditer le programme **hello.c** suivant

```
#include <stdio.h>
#include "omp.h"

int main()
{
   #pragma omp parallel num_threads(4)
   {
      printf("Hello World from TID %d!\n", omp_get_thread_num());
   }
}
```

- Compiler le programme avec : **gcc -fopenmp hello.c -o hello** ; Vous pouvez ajouter les options habituelles de compilation ; puis l'exécuter
- Modifier le nombre de threads en utilisant l'une des méthodes suivantes :
  - la clause `num_threads` de `#pragma omp parallel`
  - la variable d'environnement OMP_NUM_THREADS
  - la fonction `void omp_set_num_threads(int num_threads) ;`

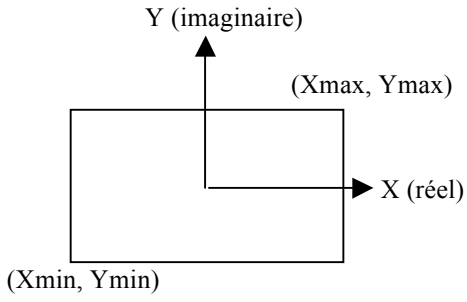
-

2. Génération parallèle d'image fractale (Méthode de Newton)

Etant donnée une fonction *f(x) = x^3 – 1, x* est une variable complexe, le calcul des racines de *f(x)* peut être effectué par la méthode de Newton :

$$\begin{cases} x_{n+1} = x_n + \dfrac{f(x_n)}{f'(x_n)}, n \geq 0 \\ x_0 \text{ est un nombre complexe quelconque} \end{cases}$$

Pour chaque valeur x_0, la méthode de Newton converge vers l'une des 3 racines de *f(x)*.
Nous allons utiliser cette propriété dans la génération d'image fractale définie sur le domaine *[Xmin, Xmax]* x *[Ymin, Ymax]*.



La génération d'image correspond à l'application de la méthode à la fonction f(x), en utilisant chaque point du domaine comme la valeur de x_0. Nous affectons la couleur C_1 (resp. C_2 ou C_3) à un point *P* du domaine, si la méthode de Newton converge vers la racine 1 (resp. la racine 2 ou 3) avec x_0=les coordonnées du *P*.

2.1. Récupérer le programme séquentiel dans un nouveau répertoire – Compiler – Exécuter

2.2. Enlever la ligne « `xhost -` » de votre `.bashrc` s'il y en a.

2.3. 1ère parallélization : Le calcul d'image contient 2 boucles imbriquées. Paralléliser ce calcul à l'aide de la directive OpenMP compacte « #pragma omp parallel for »; La parallélization de laquelle boucle permet d'avoir une meilleure performance ?

Attention :

- Pas d'{ après `#pragma omp parallel for` ou `#pragma omp for`, la ligne suivante doit être une boucle `for`.
- Le nombre de threads dépend non seulement de `num_threads` mais aussi de `chunk_size`.
- Faites attention aux **variables partagées/privées**, à **l'accès de variables partagées** et aux **instructions non préemptives (**qui implique l'**utilisation de section critique)**.

N'oubliez pas de modifier le makefile pour la compilation (ajout de l'option `-fopenmp`).

2.4. Mesurer la performance de la 1ère parallélization en variant le nombre de threads et la taille de l'image, analyser les résultats obtenus. ! on mesure uniquement le temps de calcul d'une image.
Exemple : nombre de threads = 1, 2, 4, 8, 16, 32… ; taille d'image = 1024x1024, 1024x1536, 1536x2048, 2048x2560…

2.5. Que constatez-vous après l'analyse de performance ? Quelle modification peut-on apporter afin d'améliorer la performance du programme parallèle ?

2.6. Ajouter la clause « schedule(static, chunk_size) » et faites varier la taille de bloc traité en une fois par un thread. Refaire la mesure de performance avec une taille de l'image donnée et en variant le nombre de threads.

2.7. Tester l'ordonnancement et la répartition de charge avec « schedule(dynamic) ». Que constatez-vous ?

Annexe : Représentation des mesures de performance d'un programme parallèle

La mesure de la performance d'un programme parallèle se fait en mesurant son temps d'exécution en variant le nombre de threads (ou processus) utilisés et la taille du problème traité.

L'ensemble de mesures sera d'abords stocké dans un tableau, puis tracer en courbes, ce qui facilite l'analyse et l'interprétation des résultats.

Exemple :

Tableau 1 : Temps d'exécution du programme parallèle selon la taille du problème et le nombre de threads employés

Taille du problème \ Nombre de threads	1	2	4	8
1000				
2000				
3000				
4000				

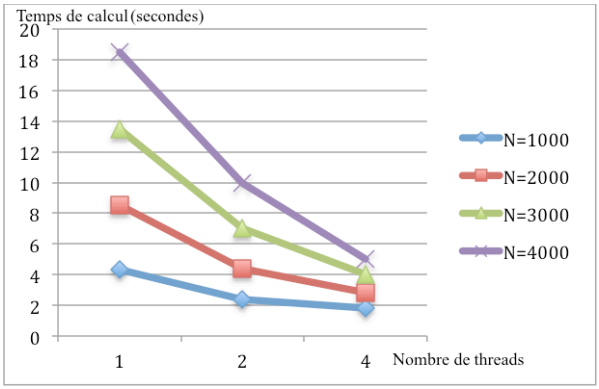


Figure 1 : L'évolution du temps d'exécution du programme parallèle selon la taille du problème et le nombre de threads employés