# TP1 : OpenMP 1

Version du 11 janvier 2022

## Exercice 1 – Echauffement

1. (**Accès à distance aux machines**) En TP nous allons travailler sur les machines d'une salle du bâtiment M5 du FIL, par exemple la salle A13 dans la suite. Avec le VPN activé, vous pouvez y accéder à distance via (sous Linux) : `ssh loginfil@a13pXX.fil.univ-lille1.fr` avec XX compris entre 1 et 24.

   Sous Windows, vous pouvez utiliser les outils `PuTTY` ou `OpenSSH`.

   Au besoin, vous pouvez vérifier que vous êtes seul avec votre binôme sur une machine avec les commandes `who` et `htop`.

   Pour éditer des fichiers/répertoires à distance, vous pouvez au choix :
   — utiliser un éditeur en mode texte : `emacs`, `nano`, `vim`, `vi` ...
   — éditer vos fichiers en local (chez vous) et transférer les fichiers modifiés par `scp`, `winscp` (Windows), `rsync`, `git` ... (pas très pratique pour des modifications fréquentes)
   — monter votre système de fichier du M5 en local (sur votre machine sous Linux) via `sshfs` :

   ```
   $ mkdir M5
   $ sshfs -o idmap=user loginfil@a13pXX.fil.univ-lille1.fr: M5
   ```

   Vous pouvez alors créer des répertoires et éditer des fichiers en local sur votre machine dans le répertoire M5, et les modifications sont répercutées sur votre système de fichier au M5.
   Important : à la fin de votre session de travail vous devez "démonter" le système de fichier monté via `sshfs` (risque de perte de données sinon) avec la commande :

   ```
   $ fusermount -u M5
   ```

   En cas de problèmes d'accès aux machines du M5 (merci de me prévenir), vous pouvez faire les TP et les tests de performance sur votre propre ordinateur à condition que vous ayez un processeur avec au moins 4 coeurs physiques et des unités AVX2, que vous ayez installé une version "récente" de `gcc`, et que vous n'utilisiez pas de machine virtuelle.

2. (**Nombre de cœurs disponibles**) Vérifiez à l'aide de la commande : `cat /proc/cpuinfo` (ou : `lscpu`), le nombre de cœurs disponibles sur votre machine.

   L'accélération parallèle est le ratio entre le temps séquentiel et le temps parallèle : dans la suite de ce TP, et pour les prochains TP, on mesurera par défaut les performances en considérant l'accélération parallèle obtenue avec autant de threads que de coeurs physiques sur la machine.

3. (**Premier programme**) Utilisez un des programmes du cours pour tester votre environnement. N'oubliez pas de positionner la variable `OMP_NUM_THREADS`.

   Rappel pour la compilation : `gcc -fopenmp toto.c -o toto`

4. (**Chronométrage**) Pour toutes les mesures de temps, on pourra utiliser la fonction suivante qui retourne le temps en secondes depuis le 1er janvier 1970.

```c
#include <sys/time.h>
double my_gettimeofday(){
  struct timeval tmp_time;
  gettimeofday(&tmp_time, NULL);
  return tmp_time.tv_sec + (tmp_time.tv_usec * 1.0e-6L);
}
```

## Exercice 2 – Variables partagées et variables privées

1. Pour chacune des variables a,b,c,d,e du code ci-dessous, déterminez si elle est partagée ou privée en OpenMP au niveau de l'exécution du printf .

```
#include <stdio.h>
int a;
int f(int x){  int e = x+1;  return e*e; }

int main(){
  int b,c;

#pragma omp parallel private(c)
  {
    int d; printf("%d\n", f(a+b+c+d));
  }
  return 0;
}
```

## Exercice 3 – Puissance K

On considère l'algorithme séquentiel (non efficace) suivant qui élève à la puissance $K$ les $N$ éléments d'un tableau.

**Entrées :** $N, K$ : constantes entières
1: Allocation mémoire d'un tableau $tab$ de $N$ nombres à virgule flottante en simple précision
2: Initialisation du contenu de $tab$ : aléatoirement (avec par exemple : $drand48()/srand48()$) ou de façon arbitraire (avec par exemple : $tab[i] = i$)
3: Démarrage du chronométrage
4: **Pour** $i = 0$ à $N - 1$ **faire**
5: float result = 1.0
6: **Pour** $j = 0$ à $K - 1$ **faire**
7: result = result * tab[i]
8: **Fin pour**
9: tab[i] = result
10: **Fin pour**
11: Arrêt du chronométrage et affichage du temps consommé
12: Libération mémoire de $tab$

1. Implémentez en C cet algorithme séquentiel.
2. En supposant que le nombre de threads $T$ divise $N$, parallélisez le code séquentiel en OpenMP à l'aide des fonctions OpenMP (i.e. en mode SPMD) : à savoir, après avoir récupéré dans la région parallèle OpenMP la valeur de $T$ et le numéro du thread courant, faites en sorte que le premier thread traite les $N/T$ premières itérations, le deuxième thread les $N/T$ suivantes, et ainsi de suite.
3. Comment pourrait-on gérer le cas où $T$ ne divise pas $N$ ? (implémentation facultative)
4. Comparez et analysez les accélérations parallèles obtenues pour $N = 16 * 1024^2$ avec $K = 2$ et $K = 4096$.

## Exercice 4 – Calcul de $\pi$

1. Ecrire un programme C qui calcule une valeur approchée de $\pi$ à l'aide de la formule suivante (pour une valeur de $N$ fixée, par exemple $N = 1000000000$, et avec des nombres à virgule flottante en double précision) :

$$\pi \approx \frac{1}{N} \sum_{i=0}^{N} \frac{4}{1 + (\frac{i}{N})^2}$$

2. Avec vos connaissances actuelles en OpenMP, parallélisez ce code en OpenMP de trois façons différentes, en utilisant à chaque fois une technique de synchronisation différente. On pourra supposer que $N$ est divisible par le nombre de threads. Observez-vous un gain en performance ?
3. Quelle parallélisation alternative pouvez-vous proposer en utilisant un tableau intermédiaire de $T$ cases pour $T$ threads ? Implémentez la, et indiquez l'accélération parallèle ainsi obtenue.

## Exercice 5 – Les premiers nombres premiers

On considère l'algorithme séquentiel suivant ((extrêmement) non efficace) qui permet de déterminer les nombres premiers entre 2 et un entier $N$ donné.

**Entrées :** $N$ : entier
**Sorties :** les nombres premiers entre 2 et N

```
 1: Allouer un tableau tab de taille N-1
 2: cpt=0
 3: Pour i = 2 à N faire
 4:    est_premier=true
 5:    j=2
 6:    Tant que j^2 <= i faire
 7:       Si i%j = 0 Alors
 8:          est_premier=false
 9:       Fin si
10:       j++
11:    Fin tant que
12:    Si est_premier Alors
13:       tab[cpt++] = i
14:    Fin si
15: Fin pour
16: Afficher les cpt premiers éléments de tab
```

1. Après avoir implémenté en C cet algorithme, parallélisez-le en OpenMP en conservant un stockage contigu (mais pas nécessairement trié dans l'ordre croissant) des nombres premiers dans le tableau $tab$.

## Exercice 6 – Calcul de fractales

L'ensemble de Mandelbrot est constitué des points $c$ du plan complexe $\mathbb{C}$ pour lesquels le schéma itératif suivant :

$$\begin{cases} z_0 = 0 \\ z_{n+1} = z_n^2 + c \end{cases} \tag{1}$$

ne diverge pas. En posant $z = x + iy$ et $c = a + ib$, l'équation (1) se réécrit :

$$\begin{cases} x_{n+1} = x_n^2 - y_n^2 + a \\ y_{n+1} = 2x_n y_n + b \end{cases}$$

avec les conditions initiales $x_0 = y_0 = 0$.

On peut montrer que s'il existe un entier $n$ tel que $|z_n| > 2$ (soit $|z_n|^2 = (x_n^2 + y_n^2) > 4$), alors le schéma (1) diverge. Pour en savoir plus, on pourra consulter les références suivantes :

— *The Fractal Geometry of the Mandelbrot Set*, R. L. Devaney, `http://math.bu.edu/DYSYS/FRACGEOM/`

— vidéo montrant l'aspect fractal de cet ensemble : `https://www.youtube.com/watch?v=jm_Q1FO9bP4`

Nous allons représenter l'ensemble de Mandelbrot entre les points $(x_{min}, y_{min})$ et $(x_{max}, y_{max})$ (avec par défaut : $(x_{min}, y_{min}) = (-2, -2)$ et $(x_{max}, y_{max}) = (2, 2)$) de $\mathbb{R}^2$ (identifié à $\mathbb{C}$) sur une image informatique de $h \times w$ pixels (avec par défaut : $h = w = 800$) : voir figure 1. Pour cela, nous allons devoir procéder à deux approximations :

— approximation en temps : nous n'allons pas pouvoir itérer à l'infini la suite des $(z_n)$, et nous allons donc fixer un nombre d'itérations maximal (une « profondeur ») pour déterminer si la suite des $(|z_n|)$ dépasse ou non 2 ;

— approximation en espace : nous n'allons pas déterminer pour chaque point de l'espace entre $(x_{min}, y_{min})$ et $(x_{max}, y_{max})$ s'il appartient à l'ensemble car il y a une infinité de tels points. Nous allons donc faire le calcul pour le centre de chacun des $h \times w$ pixels, chaque pixel ayant une taille de $inc_x \times inc_y$ dans $\mathbb{R}^2$ (voir figure 1).

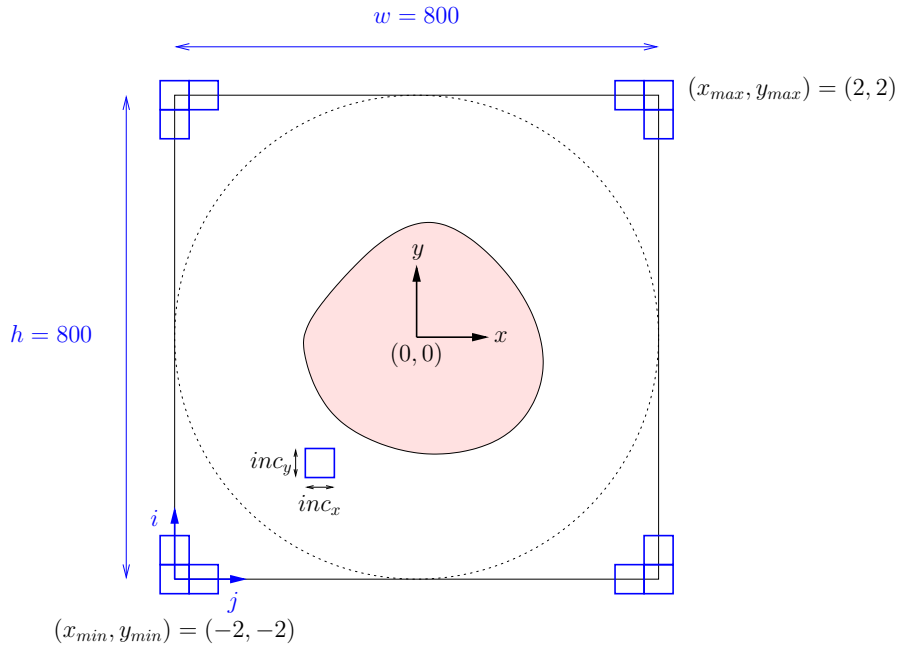


FIGURE 1 – Systèmes de coordonnées dans $\mathbb{R}^2$ (en noir) et pour les pixels (en bleu) avec les valeurs par défaut pour la représentation de l'ensemble de Mandelbrot (indiqué approximativement en rose). Les pixels dans les coins de l'image sont indiqués (carrés bleus), ainsi que le cercle de rayon 2 et centré en (0,0).

Nous manipulerons des images en couleur au format Sun Rasterfile. Chaque pixel est codé sur un octet, et la valeur de chaque pixel (donc un entier compris entre 0 et 255) représente un index dans une table de couleur. Contrairement à la convention en imagerie, le pixel de coordonnées $(0, 0)$ sera situé sur le coin inférieur gauche de l'image, et c'est donc aussi le premier élément du tableau des pixels en mémoire. Ceci permet de faire croître les $i$ et les $y$ dans le même sens.

L'algorithme séquentiel est donc le suivant :

1. Pour le centre de chacun des $h \times w$ pixels de l'image, faire (fonction `xy2color`) :
   (a) calculer le nombre d'itérations pour lequel le schéma itératif diverge (i.e. $|z_n| > 2$)
   (nombre maximal d'itérations limité à une «profondeur» donnée) ;
   (b) mettre à jour la valeur du pixel correspondant suivant :
   — si profondeur atteinte : couleur_pixel ← 255
   (soit la couleur correspondant à l'ensemble de Mandelbrot ; c'est un bleu foncé dans la palette actuelle)
   — sinon : couleur_pixel ← NombreIterations % 255
2. Sauver l'image (fonction `sauver_rasterfile`).

Le fichier `mandel.c` contient une implémentation de cet algorthme séquentiel. Des paramètres par défaut sont fournis pour $w$, $h$, $x_{min}$, $y_{min}$, $x_{max}$, $y_{max}$ et pour la profondeur $prof$ : ils peuvent aussi être indiqués en ligne de commande. On pourra utiliser le programme `display` de la suite logicielle `ImageMagick` (sous Linux) pour visualiser l'image résultat. On pourra se contenter d'étudier les fonctions `main` et `xy2color` pour la parallélisation.

1. Quelles sont les boucles parallèles de l'algorithme ?
   Comment paralléliser ce calcul en OpenMP en mode SPMD ?
   Vérifier que le résultat de votre exécution parallèle est correct en utilisant le programme `compare` de la suite logicielle `ImageMagick`.
   Quelles sont les efficacités parallèles obtenues pour différents nombres de threads (1 thread par cœur physique) ?
   Analysez vos performances.
2. Comment améliorer les performances obtenues ? (toujours en mode SPMD)
   Présentez et analysez les nouvelles performances ainsi obtenues.