Rapport Mandelbrot TP1

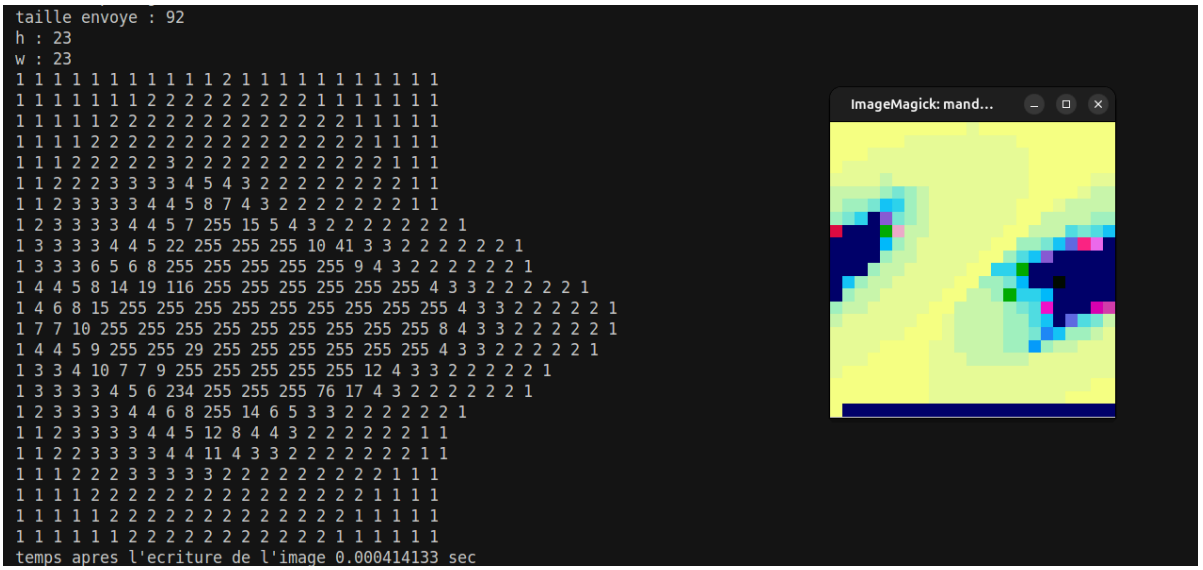
Simon Sepiol-Duchemin Joshua Setia

Parallélisation

Idée

Pour paralléliser l'algorithme fourni, on commence par trouver le nombre de core disponibles n. Ensuite on augmente la hauteur de l'image pour qu'elle soit divisible par n. On augmente aussi ymax pour que les lignes venant du padding ne contiennent pas une partie de l'image qu'on voulait calculer.
Puis chaque processus va calculer sa portion de l'image, sauf le dernier processus qui ne calcul pas les données non voulues en vérifiant si le numéro de sa ligne n'est pas trop grand. Puis on fait un gather au processus 0 qui va écrire l'image de taille h*w donc sans prendre en compte la partie de l'image qui vient de l'augmentation de la taille.

Difficulté



Dans le screenshot, on voit un affichage du contenu du tableau image après le gather de taille padded_h*w et l'affichage par display qui ne correspond pas, nous n'avons pas réussi à corriger ce bug. Ce bug arrive aussi avec le code fourni sur moodle sans parallélisation sans l'exécuter avec mpi.

Ici on voit une image 50*50, calculée sur un pc avec 6 core, le padding a donc été utilisé et l'image est correcte.

Analyse des performances

Conditions

Les expériences sont conduites sur le nœud de rennes de grid 5000. Elles sont lancées avec "oarsub -l core=150 ./mpi_script.sh"

Le code a été compilé avec "mpicc mandel.c rasterfile.h -o mandel -lm -O3"

La commande d'exécution est : "mpiexec --mca pml ^ucx --hostfile $OAR_NODEFILE ./mandel 12000 12000 0.35 0.355 0.353 0.358 1000", le nombre de processus est réglé avec l'option –n.

Les expériences sont lancées par un script bash qui fait 5 itérations de mandel.c pour chaque nombre de processus et qui récupère le contenu de $OAR_NODEFILE.

Nous avons fait 3 expériences, une en calculant le temps avec le script bash, une en calculant le temps dans le programme c et une pour voir l'écart de temps de calcul entre les différents processus.

Les machines utilisé sont :

paranoia-1.rennes.grid5000.fr
paranoia-2.rennes.grid5000.fr

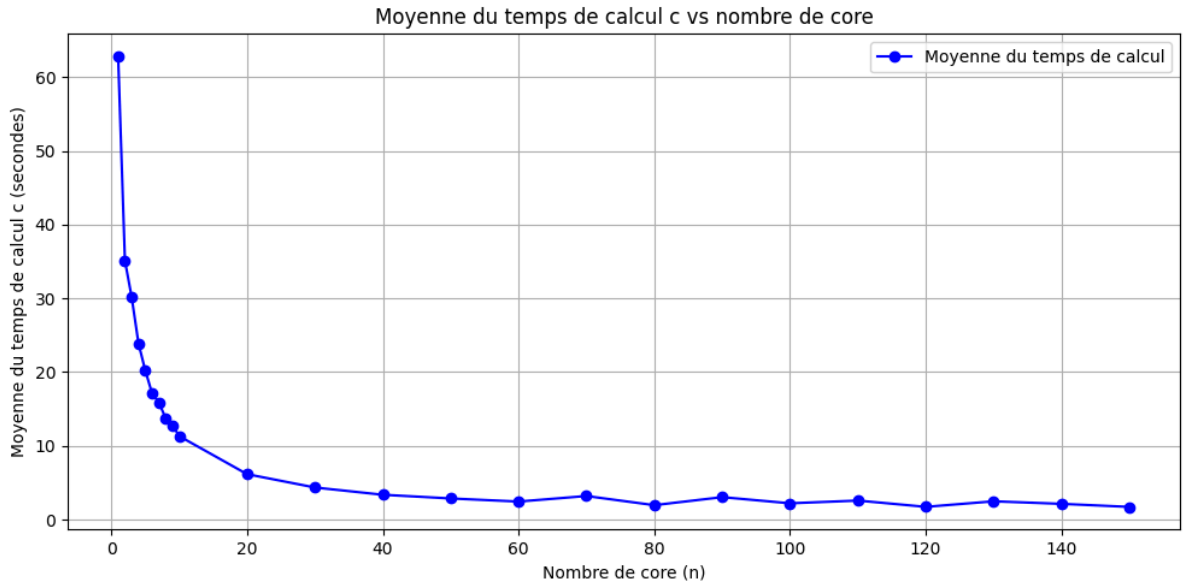
paranoia-3.rennes.grid5000.fr
paranoia-3.rennes.grid5000.fr
paranoia-3.rennes.grid5000.fr
paranoia-3.rennes.grid5000.fr
parasilo-4.rennes.grid5000.fr
paravance-14.rennes.grid5000.fr
paravance-16.rennes.grid5000.fr
paravance-18.rennes.grid5000.fr
paravance-37.rennes.grid5000.fr
paravance-46.rennes.grid5000.fr

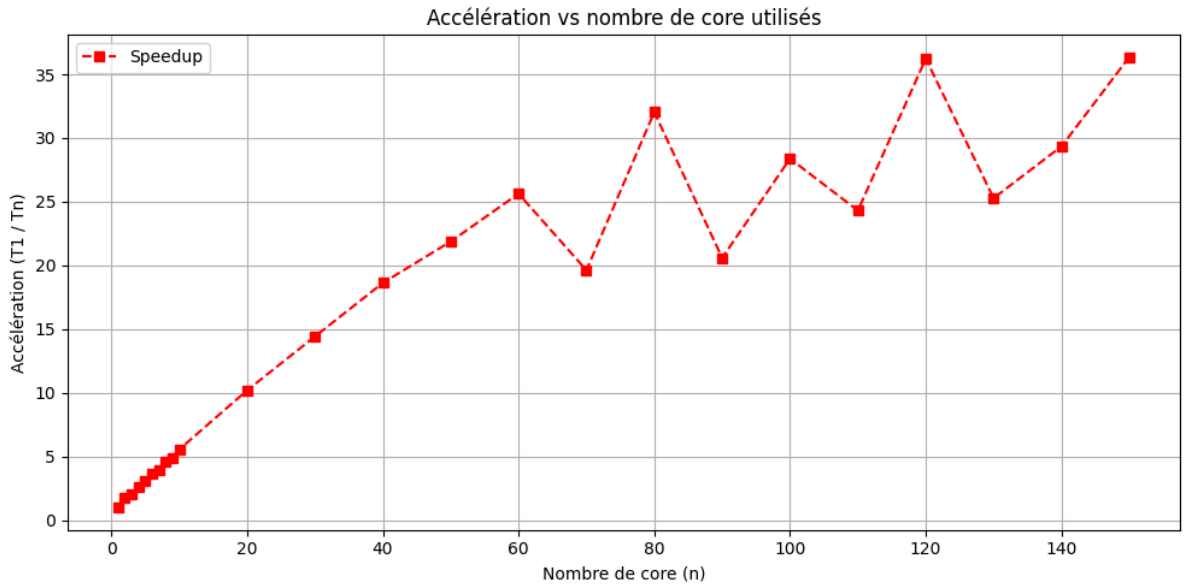
pour le calcul dans bash et le calcul de l'écart de temps et

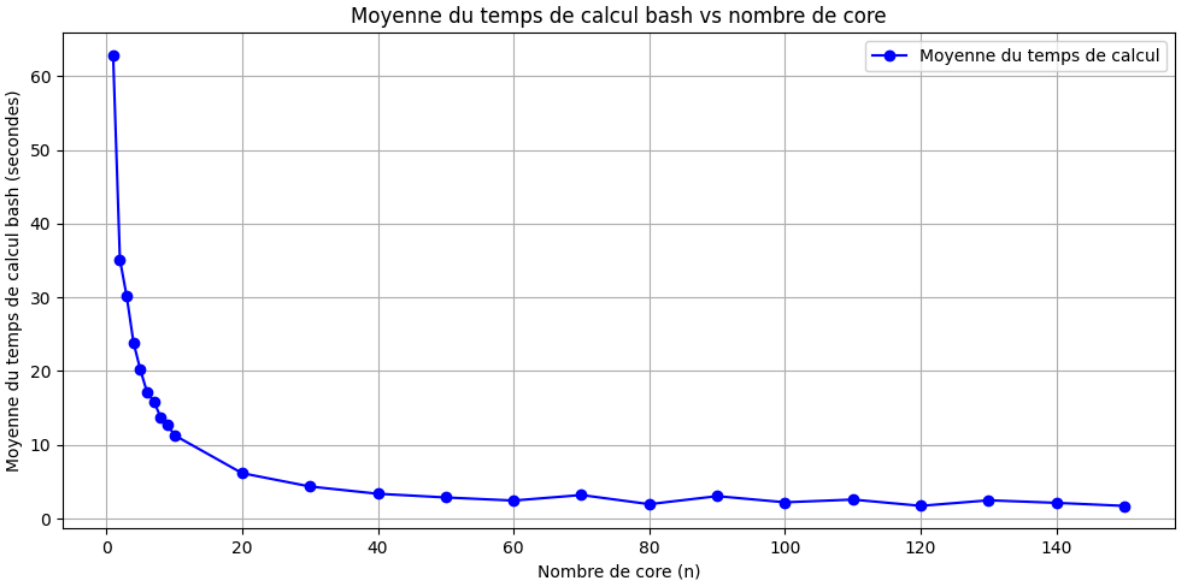
paranoia-1.rennes.grid5000.fr
paranoia-2.rennes.grid5000.fr
paranoia-3.rennes.grid5000.fr
paranoia-4.rennes.grid5000.fr
paravance-14.rennes.grid5000.fr
paravance-16.rennes.grid5000.fr
paravance-18.rennes.grid5000.fr
paravance-27.rennes.grid5000.fr
paravance-46.rennes.grid5000.fr

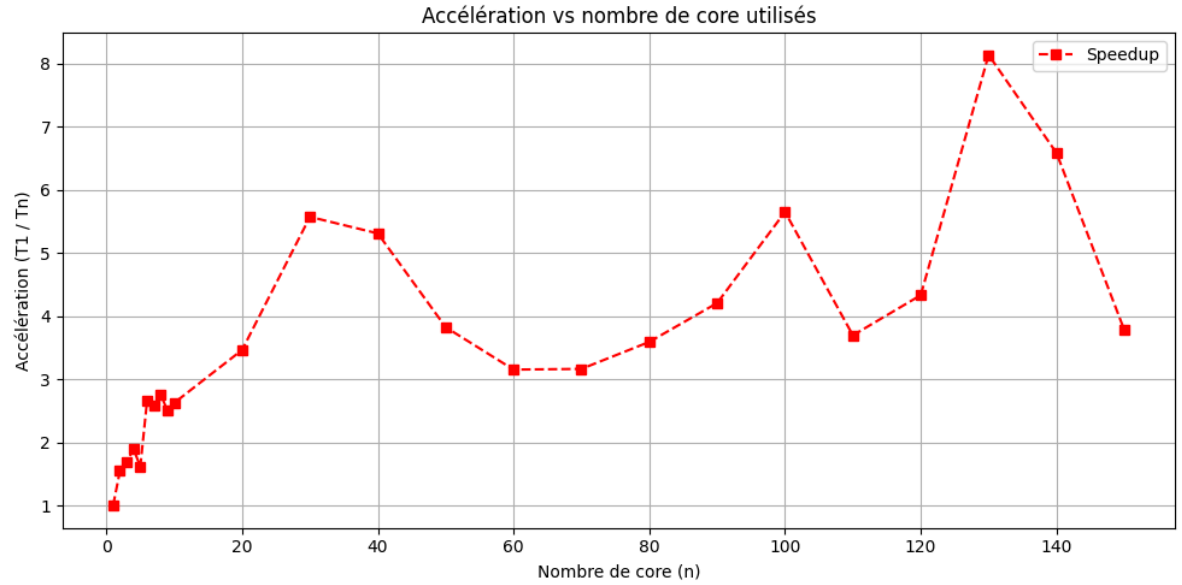
pour le calcul dans mandel.c

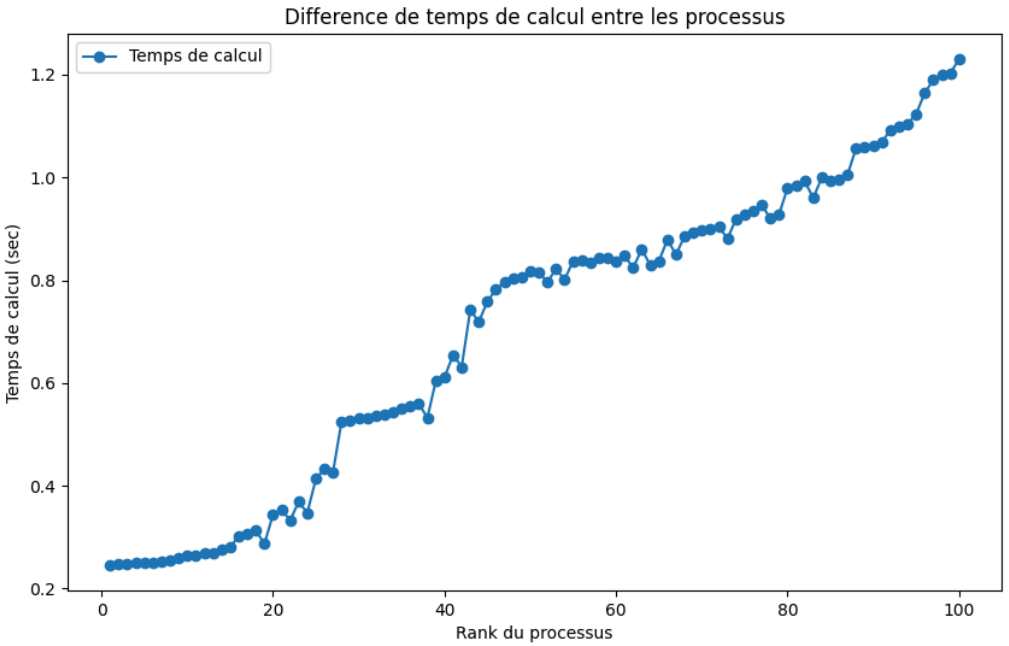
Résultats


Moyenne du temps de calcul c vs nombre de core
Moyenne du temps de calcul
Moyenne du temps de calcul c (secondes)
Nombre de core (n)
0
10
20
30
40
50
60
0
20
40
60
80
100
120
140


Accélération vs nombre de core utilisés
Speedup
Accélération (T1 / Tn)
Nombre de core (n)
0
5
10
15
20
25
30
35
0
20
40
60
80
100
120
140


Moyenne du temps de calcul bash vs nombre de core
Moyenne du temps de calcul
Moyenne du temps de calcul bash (secondes)
Nombre de core (n)
0
10
20
30
40
50
60
0
20
40
60
80
100
120
140


Accélération vs nombre de core utilisés
Speedup
Accélération (T1 / Tn)
Nombre de core (n)
1
2
3
4
5
6
7
8
0
20
40
60
80
100
120
140



Conclusion

L'accélération est linéaire jusqu'à 60 core, puis est plus difficilement lisible. On remarque que le temps de calcul entre les différents core est très variable, une piste d'amélioration est de faire en sorte que les core ayant fini en premier puissent faire d’autre calculs.