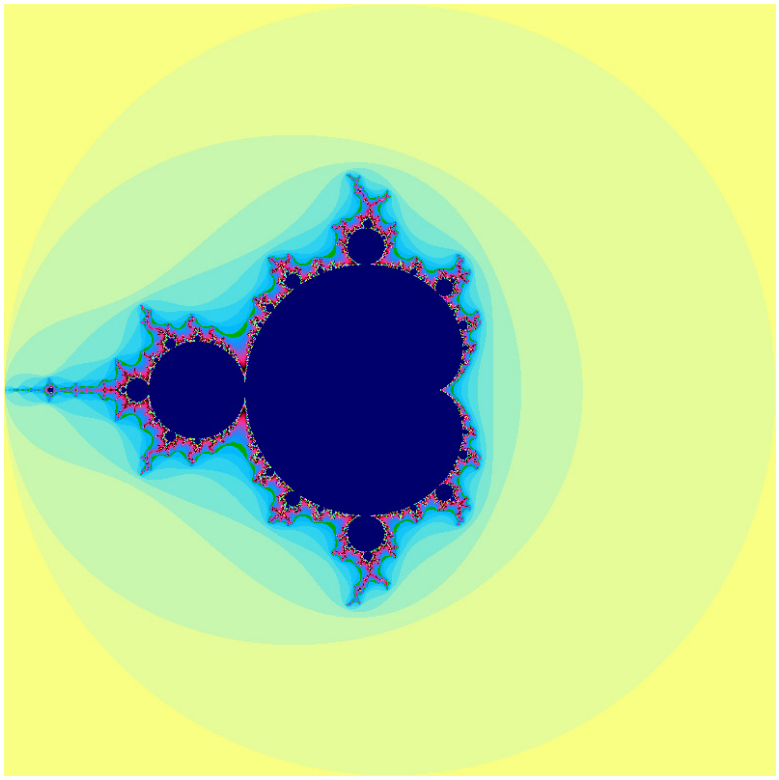
SORBONNE UNIVERSITÉ
CRÉATEURS DE FUTURS DEPUIS 1257

Sorbonne Université

Rapport de TP2 « Mandelbrot », M1 SFPN

Rapport de TP2 « Mandelbrot »



Jia Feiwen, Sanchez Jacobo

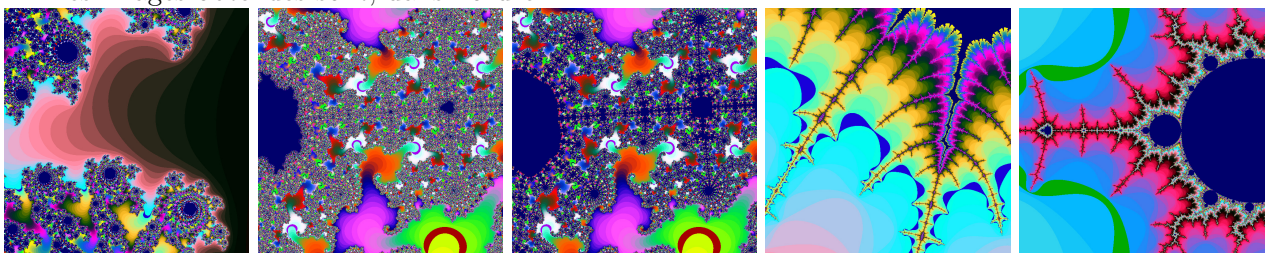
Code disponible sur Github
Encadré par
C. Makassikis
Semestre 2 M1 - 2018

Table des matières

Partie 1 – Version séquentielle.

Nous avons essentiellement testé notre code sur 6 séries de paramètres différentes, en plus de la commande par défaut (dont l'image est en couverture) :
800 800 0.35 0.355 0.353 0.358 200
800 800 -0.736 -0.184 -0.735 -0.183 500
800 800 -0.736 -0.184 -0.735 -0.183 300
800 800 -1.48478 0.00006 -1.48440 0.00044 100
800 800 -1.5 -0.1 -1.3 0.1 10000
800 800 -1.5 -0.1 -1.3 0.1 100000

Les images obtenues sont, dans l'ordre :



Et les temps de calcul correspondants, en secondes, sur un processeur 2,2 GHz Intel Core i7, en fonction de l'optimisation demandée à la compilation.

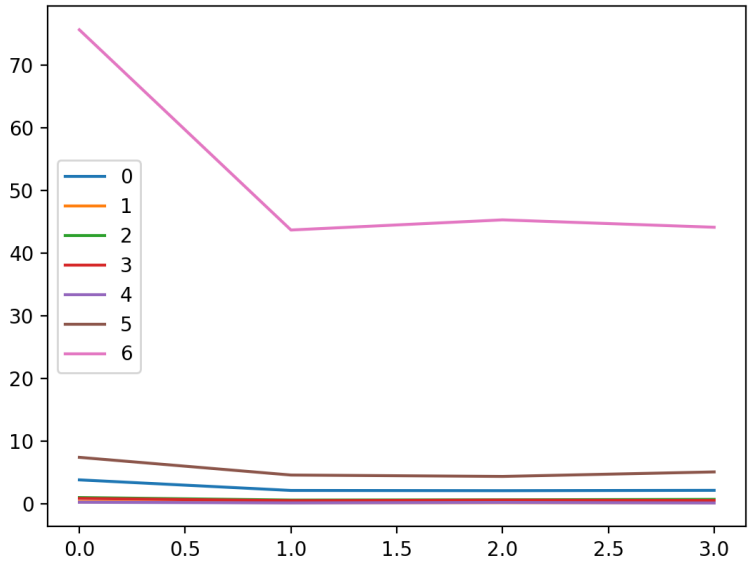


FIGURE 1 – Évolution du temps de calcul pour les commandes en fonction de l'optimisation

On observe un saut important des temps de calcul entre l'absence d'optimisation et celle de degré 1. Les améliorations entre les divers degrés d'optimisation eux-mêmes sont très légères (voire non-significative).

Partie 2 – Algorithme distribué.

Nous avons ensuite restructuré notre code, afin de pouvoir distribuer les tâches sur plusieurs coeurs. Nous avons opté vers une approche de distribution classique, considérant que toutes les machines sur lesquelles on travaille ont une puissance de calcul semblale. De plus, nous travaillerons avec une optisation de degré 3 (maximum sur gcc).

Tout d'abord, nous nous sommes intéressés aux temps de calcul en distribuant les tâches sur les six coeurs physiques de notre machine. Nous nous sommes proposés de tester sur les 12 coeurs logiques aussi. Nous travaillerons sur la 6ème commande, étant donné que le temps de calcul est plus important, et donc la perte de performance liée aux communications sera moins importante.

Nombre de processus	Temps d'éxécution	Speedup	Efficacité
1	42.9326s	1	1
2	22.8403s	$\frac{42.9326}{22.8403} = 1.88$	$\frac{1.88}{2} = 0.94$
3	14.7176s	2.92	0.97
4	10.9652s	3.92	0.98
5	8.75005s	4.91	0.98
6	7.55206s	5.68	0.95
8	5.76346s	7.45	0.93
10	4.72404s	9.08	0.91
12	4.29083	10.01	0.83
16	3.0466	14	0.88
20	2.39778	17.01	0.85
24	1.98515	21.62	0.90
28	1.75037	24.52	0.87
32	1.48895	28.83	0.90
36	1.32895	32.31	0.89
40	1.20698	35.57	0.89

Nous avons aussi expérimenté avec toute la salle de TP. Les résultats obtenus commencent à différer à partir de 12 coeurs. Nous prolongeons donc nos résultats en faisant du calcul sur ces machines.

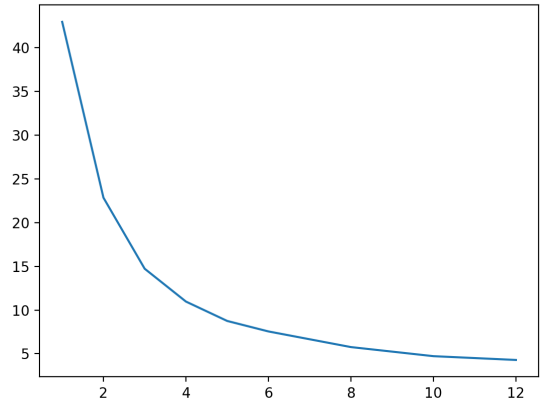


FIGURE 2 – Évolution du temps de calcul en fonction du nombre de coeurs

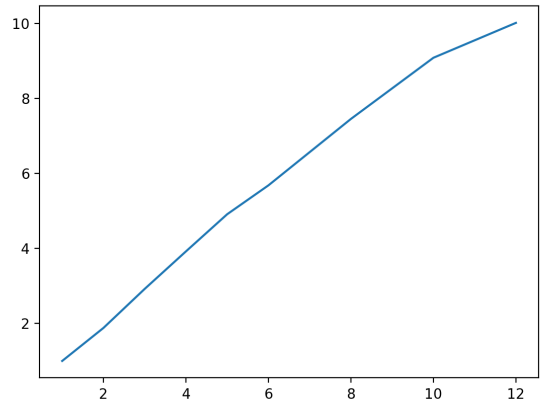


FIGURE 3 – Évolution du speedup en fonction du nombre de coeurs

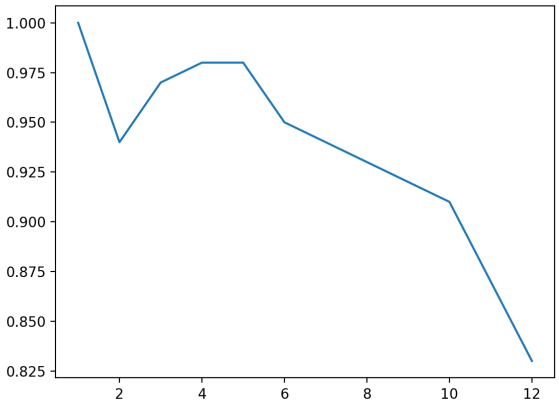


FIGURE 4 – Évolution de l'efficacité en fonction du nombre de coeurs

On observe que le temps de calcul est réduit plus le nombre de coeurs est important, mais avec une tendance exponentielle inverse. Cela accorde avec le fait qu'il y a plus de temps passé à faire des communications et de l'allocation mémoire. Le speedup lui a une allure linéaire et l'efficacité chute d'environ 100% à 90% lorsqu'on ne travaille plus sur une machine locale.