TP1 - OS202: Systèmes Distribués

ENSTA

February 3, 2026

1 Github Link - codes

Link: https://github.com/matbs/SO202/tree/main/TP2

2 Disclaimer

Mon laptop n'a que 4 coeurs, donc les tests de performance sont limités à 2 processus. Cependant, les résultats obtenus sont suffisants pour observer les tendances de performance et les effets de la parallélisation en quelquers situations.

3 Exercice 1: Mandelbrot.py

3.1 Partition Equitable

Le speedup est obtenu par l'equation:

$$S = \frac{T_{sequentiel}}{T_{parallele}}$$

où $T_{sequentiel}$ est le temps d'exécution du programme séquentiel et $T_{parallele}$ est le temps d'exécution du programme parallèle.

Donc, pour calculer le speedup, on mesure les temps d'exécution des deux versions du programme et on applique la formule ci-dessus. Ainsi, les resultats obtenus montrent que le programme parallèle est significativement plus rapide que le programme séquentiel, démontrant l'efficacité de la parallélisation pour ce type de calcul.

Nombre de Processus	Temps Séquentiel (s)	Temps Parallèle (s)	Speedup
1	4.397 s	4.397 s	1.0
2	4.923 s	2.487 s	1.97

Table 1: Temps d'exécution et Speedup pour Mandelbrot.py

3.2 Repartition Statique par Blocs

Une manière d'améliorer la répartition du travail est d'utiliser une répartition que considère la complexite des calculs pour chaque partie de l'image. Par exemple, pour fair le calcul de l'ensemble de Mandelbrot, certaines régions de l'image nécessitent plus de calculs que d'autres. Ainsi, une répartition statique par blocs qui prend en compte cette complexité peut être plus efficace.
Le problème de cette approche est qu'elle nécessite une connaissance préalable de la complexité de chaque région de l'image, ce qui peut ne pas être toujours possible. De plus, si la répartition

Nombre de Processus	Temps Séquentiel (s)	Temps Parallèle (s)	Speedup
1	5.223 s	5.223 s	1.0
2	5.300 s	2.677 s	1.98

Table 2: Temps d'exécution et Speedup pour Mandelbrot.py

n'est pas bien équilibrée, certains processus peuvent finir leur travail plus tôt que d'autres, ce qui entraîne une inefficacité.

En comparaison avec la répartition anterieure, cette approche n'est pas significativement meilleure. Cela peut être dû au fait que la complexité des calculs n'a pas été correctement estimée, ou que la répartition n'était pas suffisamment équilibrée.

3.3 Repartition Maitre - Esclaves

Le problème principal de cette approche est l'absence de plusieurs processus esclaves car je peux utiliser que 2 processus sur mon laptop. Ainsi, le gain de performance est limité. Cependant, on peut observer que cette approche permet une meilleure répartition du travail, car le processus maître peut assigner des tâches aux esclaves en fonction de leur disponibilité.

Avec 2 processus les resultats obtenus sont:

Nombre de Processus	Temps Séquentiel (s)	Temps Parallèle (s)	Speedup
2	5.223 s	5.223 s	1.0

Table 3: Temps d'exécution et Speedup pour Mandelbrot.py

Je ne peux pas faire de comparaison puisque il sera necessaire d'avoir plus de 2 processus pour observer un gain de performance significatif je crois.

4 Mandelbrot matrice - vecteur (colonnes)

Les adaptations pour utiliser une approche matrice-vecteur sur les colonnes de l'image ont rendu les resultats suivants:

Nombre de Processus	Temps Séquentiel (s)	Temps Parallèle (s)	Speedup
1	0.000056 s	0.000056 s	1.0
2	0.000097 s	0.000056 s	0.57

Table 4: Temps d'exécution et Speedup pour Mandelbrot matrice-vecteur

5 Mandelbrot matrice - vecteur (lignes)

Les resultats obtenus en utilisant une approche matrice-vecteur sur les lignes de l'image sont les suivants:

Nombre de Processus	Temps Séquentiel (s)	Temps Parallèle (s)	Speedup
1	0.000064 s	0.000064 s	1.0
2	0.000064 s	0.000089 s	0.73

Table 5: Temps d'exécution et Speedup pour Mandelbrot matrice-vecteur lignes

6 Entrainement pour l'examen écrit

La loi d'Amdahl peut être écrit comme suit par les valeurs de n » 1:

$$S = \frac{1}{f}$$

où f est la fraction du programme qui est séquentielle (non parallélisable). Ainsi, si 10% du programme est séquentielle, le speedup maximum théorique est:

$$S = \frac{1}{0.1} = 10$$

Pour cette raison, le nombre de noeuds de calcul raisonable pour n'est trop gaspiller les ressouces de calcul sont entre 5 et 10 noeuds puisque au delà de ce nombre, le speedup additionnel devient marginal. On peut calculer le speedup pour différents nombres de noeuds de calcul en utilisant la loi d'Amdahl:

- Pour 5 noeuds:
$$S = \frac{1}{0.1 + \frac{0.9}{5}} = 4.17$$
- Pour 10 noeuds:
$$S = \frac{1}{0.1 + \frac{0.9}{10}} = 5.26$$
- Pour 20 noeuds:
$$S = \frac{1}{0.1 + \frac{0.9}{20}} = 5.88$$

Ainsi, on peut voir que le speedup augmente avec le nombre de noeuds de calcul, mais l'augmentation devient de plus en plus marginale à mesure que le nombre de noeuds augmente.

Pour cette raison, l'accélération maximale en utilisant la loi de Gustafson peut être calculée par:

D'abord, une accélération maximale de 4 correspond selon la loi d'Amdahl à:

$$S = 4 = \frac{1}{f}$$

où f est la fraction séquentielle du programme.

En réarrangeant cette équation, on peut trouver la fraction séquentielle f:

$$f = \frac{1}{S} = \frac{1}{4} = 0.25$$

Ainsi, pour obtenir une accélération de 4, la fraction séquentielle du programme doit être de 25%.

Le temps séquentiel total peut être exprimé comme:

$$T_{\text{total}} = T_{\text{sequentiel}} + T_{\text{parallele}}$$

avec $T_{\text{sequentiel}} = f \cdot T_{\text{total}} = 0.25\,T_{\text{total}}$ et $T_{\text{parallele}} = (1 - f) \cdot T_{\text{total}} = 0.75\,T_{\text{total}}$.

En doublant la quantité de données et en supposant la complexité de l'algorithme parallèle linéaire, le nouveau temps parallèle double tandis que le temps séquentiel reste constant:

$$T'_{\text{parallele}} = 2 \times T_{\text{parallele}} = 1.5\,T_{\text{total}}$$

$$T'_{\text{sequentiel}} = T_{\text{sequentiel}} = 0.25\,T_{\text{total}}$$

Le nouveau temps séquentiel total devient:

$$T'_{\text{total}} = T'_{\text{sequentiel}} + T'_{\text{parallele}} = 0.25\,T_{\text{total}} + 1.5\,T_{\text{total}} = 1.75\,T_{\text{total}}$$

La nouvelle fraction séquentielle f' est donc:

$$f' = \frac{T'_{\text{sequentiel}}}{T'_{\text{total}}} = \frac{0.25\,T_{\text{total}}}{1.75\,T_{\text{total}}} = \frac{1}{7}$$

En utilisant la loi de Gustafson, l'accélération maximale devient:

$$S'_{\text{max}} = \frac{1}{f'} = 7$$

Ainsi, en doublant les données, Alice peut espérer une accélération maximale de **7**.