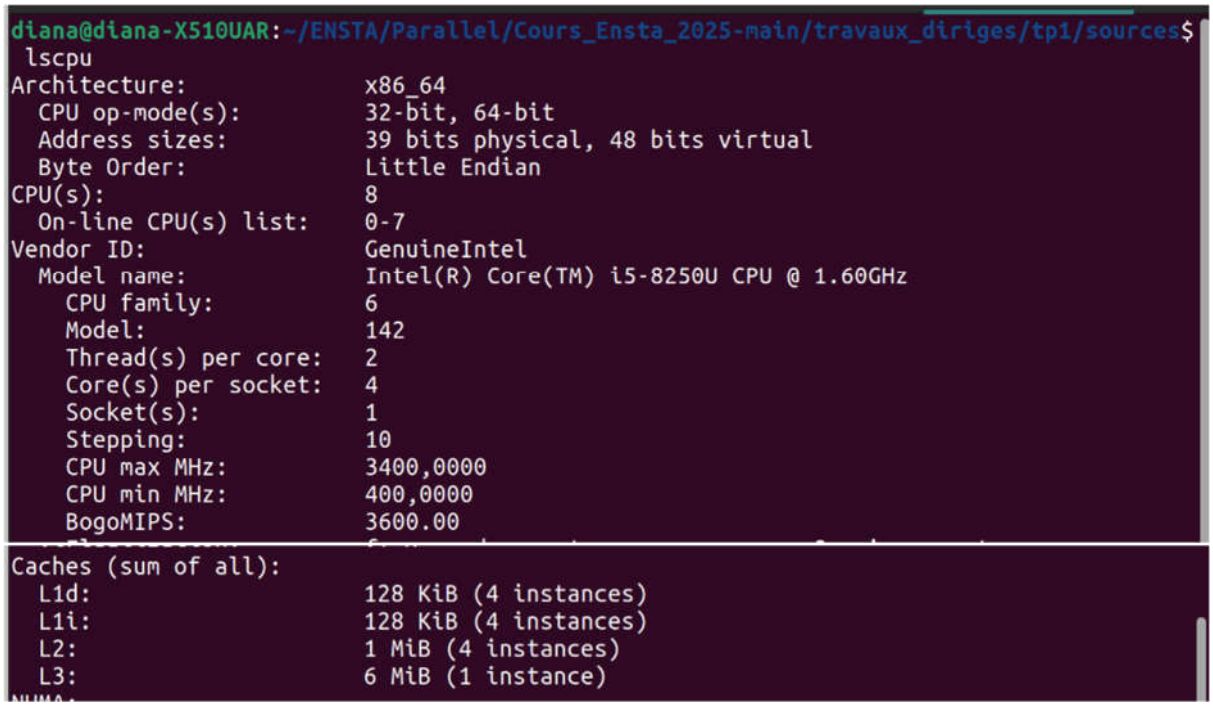
Systèmes parallèles et distribués

Travaux dirigés n°2

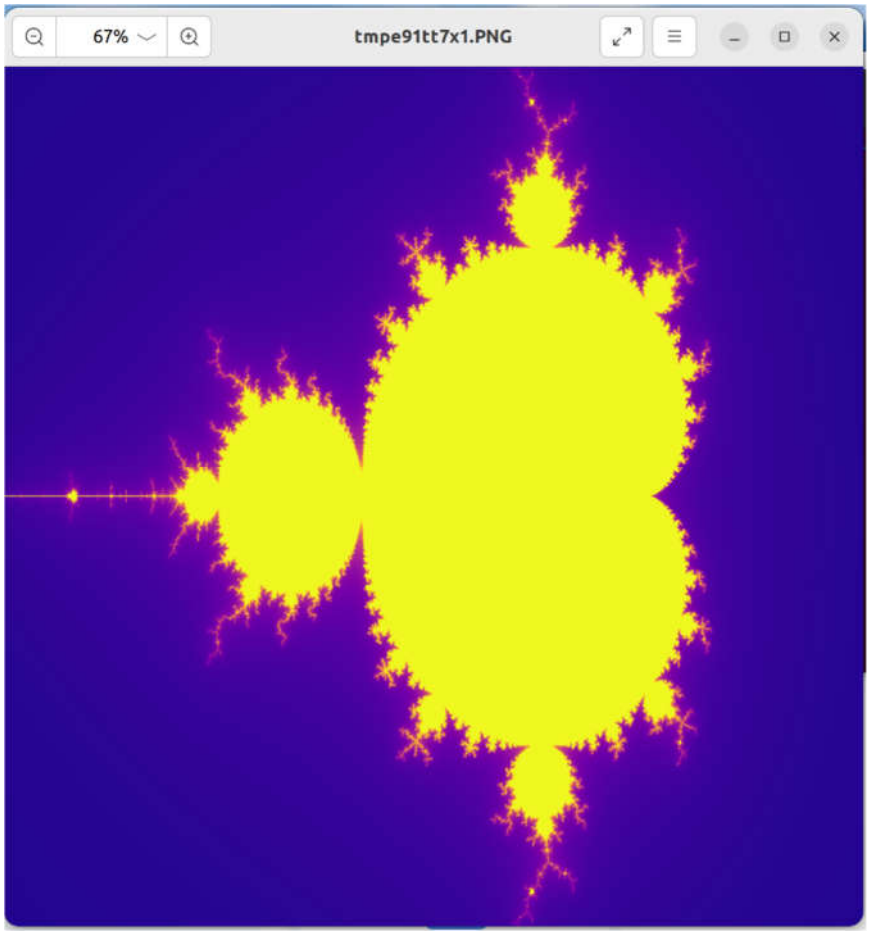
Diana Tymoshenko



1. Parallélisation ensemble de Mandelbrot

1. À partir du code séquentiel mandelbrot.py, faire une partition équitable par bloc suivant les lignes de l'image pour distribuer le calcul sur nbp processus puis rassembler l'image sur le processus zéro pour la sauvegarder. Calculer le temps d'exécution pour différents nombre de tâches et calculer le speedup. Comment interpréter les résultats obtenus ?

   mandelbrot_1.py




```
mpirun -n 3 python mandelbrot_1.py
diana@diana-X510UAR:~/ENSTA/Parallel/Cours_Ensta_2025-main/travaux_diriges/tp2$
mpirun -n 1 python mandelbrot_1.py
Temps du calcul de l'ensemble de Mandelbrot : 3.83638858795166, rank : 0
Temps de constitution de l'image : 0.03729605674743652
diana@diana-X510UAR:~/ENSTA/Parallel/Cours_Ensta_2025-main/travaux_diriges/tp2$
mpirun -n 2 python mandelbrot_1.py
Temps du calcul de l'ensemble de Mandelbrot : 1.833141803741455, rank : 0
Temps du calcul de l'ensemble de Mandelbrot : 1.9612514972686768, rank : 1
Temps de constitution de l'image : 0.048431396484375
diana@diana-X510UAR:~/ENSTA/Parallel/Cours_Ensta_2025-main/travaux_diriges/tp2$
mpirun -n 4 python mandelbrot_1.py
Temps du calcul de l'ensemble de Mandelbrot : 0.9035611152648926, rank : 1
Temps du calcul de l'ensemble de Mandelbrot : 0.9510128498077393, rank : 2
Temps du calcul de l'ensemble de Mandelbrot : 1.0517938137054443, rank : 3
Temps du calcul de l'ensemble de Mandelbrot : 1.1078681945800781, rank : 0
Temps de constitution de l'image : 0.08032512664794922
diana@diana-X510UAR:~/ENSTA/Parallel/Cours_Ensta_2025-main/travaux_diriges/tp2$
mpirun --oversubscribe -n 8 python mandelbrot_1.py
Temps du calcul de l'ensemble de Mandelbrot : 0.9063098430633545, rank : 3
Temps du calcul de l'ensemble de Mandelbrot : 0.9309577941894531, rank : 4
Temps du calcul de l'ensemble de Mandelbrot : 0.9917323589324951, rank : 7
Temps du calcul de l'ensemble de Mandelbrot : 1.0279603004455566, rank : 5
Temps du calcul de l'ensemble de Mandelbrot : 1.085416316986084, rank : 0
Temps du calcul de l'ensemble de Mandelbrot : 1.0862693786621094, rank : 2
Temps du calcul de l'ensemble de Mandelbrot : 1.243807077407837, rank : 6
Temps du calcul de l'ensemble de Mandelbrot : 1.2472972869873047, rank : 1
Temps de constitution de l'image : 0.1177973747253418
```

№ P	T	Speedup
1	3.8364	1
2	1.9613	1.956
4	1.1079	3.4628
8	1.2473	3.0758

Comme on peut le voir dans le tableau, l'accélération maximale se produit avec 4 processus, car c'est à ce moment-là que le parallélisme est le plus efficace (la machine possède 4 cœurs physiques). Avec 8 processus, le programme s'exécute légèrement plus lentement.

On peut également remarquer que les différents processus ont des temps d'exécution inégaux, ce qui signifie que la charge de travail entre les processus n'est pas équilibrée (la plus grande différence de temps entre les processus est de 1.24729 - 0.90631 = 0.34098 sec).

2. Réfléchissez à une meilleur répartition statique des lignes au vu de l'ensemble obtenu sur notre exemple et mettez la en œuvre. Calculer le temps d'exécution pour différents nombre de tâches et calculer le speedup et comparez avec l'ancienne répartition. Quel problème pourrait se poser avec une telle stratégie ?

   mandelbrot_2.py

```
diana@diana-X510UAR:~/ENSTA/Parallel/Cours_Ensta_2025-main/travaux_diriges/tp2$
mpirun -n 1 python mandelbrot_2.py
Temps du calcul de l'ensemble de Mandelbrot : 3.4900739192962646, rank : 0
Temps de constitution de l'image : 0.03599405288696289
diana@diana-X510UAR:~/ENSTA/Parallel/Cours_Ensta_2025-main/travaux_diriges/tp2$
mpirun -n 2 python mandelbrot_2.py
Temps du calcul de l'ensemble de Mandelbrot : 1.7527847290039062, rank : 1
Temps du calcul de l'ensemble de Mandelbrot : 1.7584927082061768, rank : 0
Temps de constitution de l'image : 0.04842257499694824
diana@diana-X510UAR:~/ENSTA/Parallel/Cours_Ensta_2025-main/travaux_diriges/tp2$
mpirun -n 4 python mandelbrot_2.py
Temps du calcul de l'ensemble de Mandelbrot : 0.9171745777130127, rank : 3
Temps du calcul de l'ensemble de Mandelbrot : 0.9256172180175781, rank : 1
Temps du calcul de l'ensemble de Mandelbrot : 0.9271910190582275, rank : 0
Temps du calcul de l'ensemble de Mandelbrot : 0.9292032718658447, rank : 2
Temps de constitution de l'image : 0.0735619068145752
diana@diana-X510UAR:~/ENSTA/Parallel/Cours_Ensta_2025-main/travaux_diriges/tp2$
mpirun --oversubscribe -n 8 python mandelbrot_2.py
Temps du calcul de l'ensemble de Mandelbrot : 0.9582774639129639, rank : 0
Temps du calcul de l'ensemble de Mandelbrot : 0.9615447521209717, rank : 5
Temps du calcul de l'ensemble de Mandelbrot : 0.9622189998626709, rank : 3
Temps du calcul de l'ensemble de Mandelbrot : 0.963956356048584, rank : 6
Temps du calcul de l'ensemble de Mandelbrot : 0.9747917652130127, rank : 1
Temps du calcul de l'ensemble de Mandelbrot : 0.9804868698120117, rank : 2
Temps du calcul de l'ensemble de Mandelbrot : 0.9839749336242676, rank : 7
Temps du calcul de l'ensemble de Mandelbrot : 0.9848976135253906, rank : 4
Temps de constitution de l'image : 0.1313459873199463
```

№ P	T	Speedup
1	3.49	1
2	1.75849	1.9847
4	0.9292	3.7559
8	0.9849	3.5435

Ici, nous répartissons les lignes entre les processus de manière séquentielle (la première ligne au premier processus, la deuxième ligne au deuxième...) afin de leur attribuer une quantité de travail à peu près égale, réduisant ainsi les écarts de temps d'exécution (la plus grande différence de temps entre les processus est de 0.02662 sec). L'accélération maximale est également observée avec 4 processus. Cette répartition donne de meilleurs résultats par rapport à la méthode précédente.

3. Mettre en œuvre une stratégie maître-esclave pour distribuer les différentes lignes de l'image à calculer. Calculer le speedup avec cette approche et comparez avec les solutions différentes. Qu'en concluez-vous ?

   mandelbrot_3.py

```
diana@diana-X510UAR:~/ENSTA/Parallel/Cours_Ensta_2025-main/travaux_diriges/tp2$
mpirun -n 2 python mandelbrot_3.py
Temps du calcul de l'ensemble de Mandelbrot : 3.513826370239258, rank : 0
Temps de constitution de l'image : 0.04695844650268555
diana@diana-X510UAR:~/ENSTA/Parallel/Cours_Ensta_2025-main/travaux_diriges/tp2$
mpirun -n 3 python mandelbrot_3.py
Temps du calcul de l'ensemble de Mandelbrot : 1.7672414779663086, rank : 0
Temps de constitution de l'image : 0.06405234336853027
diana@diana-X510UAR:~/ENSTA/Parallel/Cours_Ensta_2025-main/travaux_diriges/tp2$
mpirun --oversubscribe -n 5 python mandelbrot_3.py
Temps du calcul de l'ensemble de Mandelbrot : 1.126852035522461, rank : 0
Temps de constitution de l'image : 0.09293985366821289
diana@diana-X510UAR:~/ENSTA/Parallel/Cours_Ensta_2025-main/travaux_diriges/tp2$
mpirun --oversubscribe -n 9 python mandelbrot_3.py
Temps du calcul de l'ensemble de Mandelbrot : 1.2214875221252441, rank : 0
Temps de constitution de l'image : 0.13185548782348633
diana@diana-X510UAR:~/ENSTA/Parallel/Cours_Ensta_2025-main/travaux_diriges/tp2$
diana@diana-X510UAR:~/ENSTA/Parallel/Cours_Ensta_2025-main/travaux_diriges/tp2$
diana@diana-X510UAR:~/ENSTA/Parallel/Cours_Ensta_2025-main/travaux_diriges/tp2$
mpirun -n 4 python mandelbrot_3.py
Temps du calcul de l'ensemble de Mandelbrot : 1.2230498790740967, rank : 0
Temps de constitution de l'image : 0.07579851150512695
diana@diana-X510UAR:~/ENSTA/Parallel/Cours_Ensta_2025-main/travaux_diriges/tp2$
mpirun --oversubscribe -n 8 python mandelbrot_3.py
Temps du calcul de l'ensemble de Mandelbrot : 1.1741924285888672, rank : 0
Temps de constitution de l'image : 0.12976622581481934
diana@diana-X510UAR:~/ENSTA/Parallel/Cours_Ensta_2025-main/travaux_diriges/tp2$
```

№ P	T	Speedup
2	3.51383	1
3	1.76724	1.9883
5	1.12685	3.1183
9	1.22149	3.13134

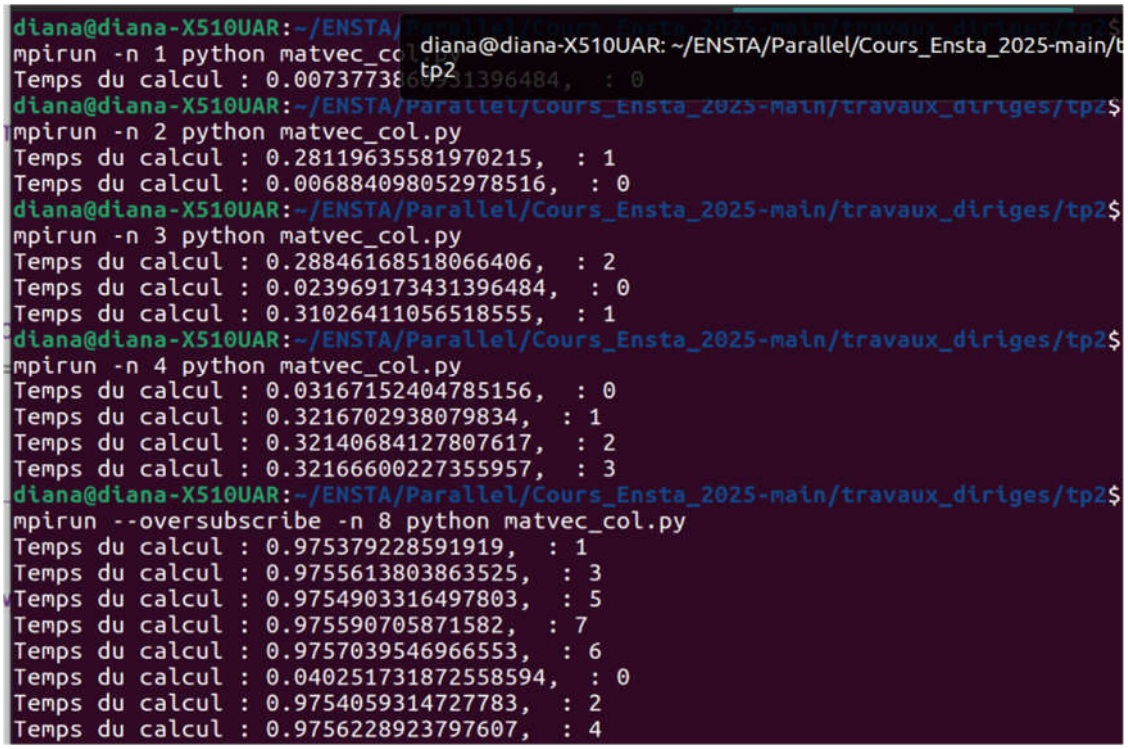
Dans ce cas, un processus supplémentaire a été utilisé, car le processus principal joue le rôle de maître. L'accélération par rapport aux méthodes précédentes est légèrement inférieure, ce qui signifie qu'elle n'apporte pas d'avantages notables, et un processus supplémentaire est également impliqué.

2. Produit matrice-vecteur

a - Produit parallèle matrice-vecteur par colonne

- Calculer en fonction du nombre de tâches la valeur de Nloc
- Paralléliser le code séquentiel matvec.py en veillant à ce que chaque tâche n'assemble que la partie de la matrice utile à sa somme partielle du produit matrice-vecteur. On s'assurera que toutes les tâches à la fin du programme contiennent le vecteur résultat complet.
- Calculer le speed-up obtenu avec une telle approche

matvec_col.py



N = 1200

№ P	T	Speedup
1	0.007377	1
2	0.281196	0.02623
4	0.32167	0.0229
8	0.9757	0.00756

Nous voyons que le programme s'est exécuté le plus rapidement avec un seul processus, ce qui signifie qu'avec plusieurs processus, beaucoup de temps est perdu pour la distribution des données entre eux, ce qui n'est pas avantageux dans ce cas. Puisque le processus zéro n'attend pas les données transmises via le réseau, mais les possède immédiatement, il a donc le temps d'exécution le plus court.

b - Produit parallèle matrice-vecteur par ligne

- Calculer en fonction du nombre de tâches la valeur de Nloc
- paralléliser le code séquentiel matvec.py en veillant à ce que chaque tâche n'assemble que la partie de la matrice utile à son produit matrice-vecteur partiel. On s'assurera que toutes les tâches à la fin du programme contiennent le vecteur résultat complet.
- Calculer le speed-up obtenu avec une telle approche

matvec_row.py

```
diana@diana-X510UAR:~/ENSTA/Parallel/Cours_Ensta_2025-main/travaux_diriges/tp2$
mpirun -n 1 python matvec_row.py
Temps du calcul : 0.00722050666809082,  : 0
diana@diana-X510UAR:~/ENSTA/Parallel/Cours_Ensta_2025-main/travaux_diriges/tp2$
mpirun -n 2 python matvec_row.py
Temps du calcul : 0.00681614875793457,  : 0
Temps du calcul : 0.275562047958374,  : 1
diana@diana-X510UAR:~/ENSTA/Parallel/Cours_Ensta_2025-main/travaux_diriges/tp2$
mpirun -n 3 python matvec_row.py
Temps du calcul : 0.009821653366088867,  : 0
Temps du calcul : 0.28928232192993164,  : 1
Temps du calcul : 0.28923654556274414,  : 2
diana@diana-X510UAR:~/ENSTA/Parallel/Cours_Ensta_2025-main/travaux_diriges/tp2$
mpirun -n 4 python matvec_row.py
Temps du calcul : 0.28587913513183594,  : 2
Temps du calcul : 0.013500213623046875,  : 0
Temps du calcul : 0.2970430850982666,  : 1
Temps du calcul : 0.30917930603027344,  : 3
diana@diana-X510UAR:~/ENSTA/Parallel/Cours_Ensta_2025-main/travaux_diriges/tp2$
mpirun --oversubscribe -n 8 python matvec_row.py
Temps du calcul : 0.9519734382629395,  : 5
Temps du calcul : 0.9520938396453857,  : 1
Temps du calcul : 0.9522292613983154,  : 7
Temps du calcul : 0.9522554874420166,  : 3
Temps du calcul : 0.9525966644287109,  : 4
Temps du calcul : 0.9527668952941895,  : 6
Temps du calcul : 0.036466121673583984,  : 0
Temps du calcul : 0.9530630111694336,  : 2
```

№ P	T	Speedup
1	0.00722	1
2	0.27556	0.0262
4	0.30918	0.02335
8	0.95277	0.00757

Comme on le voit, la multiplication par lignes donne des résultats légèrement meilleurs que par colonnes. Cela peut être lié au fait qu'ici, il n'y a pas d'opérations de sommation à la fin (Allreduce) ou que le vecteur u est transmis en entier sans être divisé entre les processus.

3. Entraînement pour l'examen écrit

1. Alice a parallélisé en partie un code sur machine à mémoire distribuée. Pour un jeu de données spécifiques, elle remarque que la partie qu'elle exécute en parallèle représente en temps de traitement 90% du temps d'exécution du programme en séquentiel.

En utilisant la loi d'Amdhal, pouvez-vous prédire l'accélération maximale que pourra obtenir Alice avec son code (en considérant n □ 1) ?

La loi d'Amdhal :

$$S(n) = \frac{1}{(1-p) + \frac{p}{n}}$$

n - le nombre de processeurs, p - la fraction parallélisable du programme.

p = 0.9

$$S_{max} = \lim_{n\to\infty} S(n) = \frac{1}{1-p} = \frac{1}{1-0.9} = 10$$

2. À votre avis, pour ce jeu de donné spécifique, quel nombre de nœuds de calcul semble-t-il raisonnable de prendre pour ne pas trop gaspiller de ressources CPU ?

On peut calculer S(n) et l'efficacité de l'utilisation (cela montre à quel point les processeurs sont chargés.) E(n) = S(n) / n pour les valeurs différentes de n.

n	S(n)	E(n)
2	1.81	0.905
4	3.077	0.769
6	4	0.67

8	4.706	0.588
10	5.263	0.5263
16	6.4	0.4
32	7.805	0.244
64	8.767	0.137

Je pense qu'une bonne option serait 6 à 8 processeurs, car l'accélération est significative (4 - 4,7) et les processeurs ne restent pas trop inactifs (leur occupation est de 58 % à 67 %).

3. En effectuant son cacul sur son calculateur, Alice s'aperçoit qu'elle obtient une accélération maximale de quatre en augmentant le nombre de nœuds de calcul pour son jeu spécifique de données.

En doublant la quantité de donnée à traiter, et en supposant la complexité de l'algorithme parallèle linéaire, quelle accélération maximale peut espérer Alice en utilisant la loi de Gustafson ?

S(n) = 4, p = 0.9, n = ?

$$4 = \frac{1}{(1-0.9) + \frac{0.9}{n}}$$
$$\mathrm{n} = 6$$

La loi de Gustafson :

$$S(n) = n - (1-p)(n-1)$$

n - le nombre de processeurs, p - la fraction parallélisable du programme.

Puisque la partie parallèle a augmenté de 2 fois et que la partie séquentielle est restée inchangée, la valeur de p a changé aussi.

$$\mathrm{p}' = \frac{2p}{1+p} = \frac{2 * 0.9}{1 + 0.9} = 0.947$$

p = 0.947, n = 6

$$S(6) = 6 - (1 - 0.947)(6 - 1) = 5.735$$