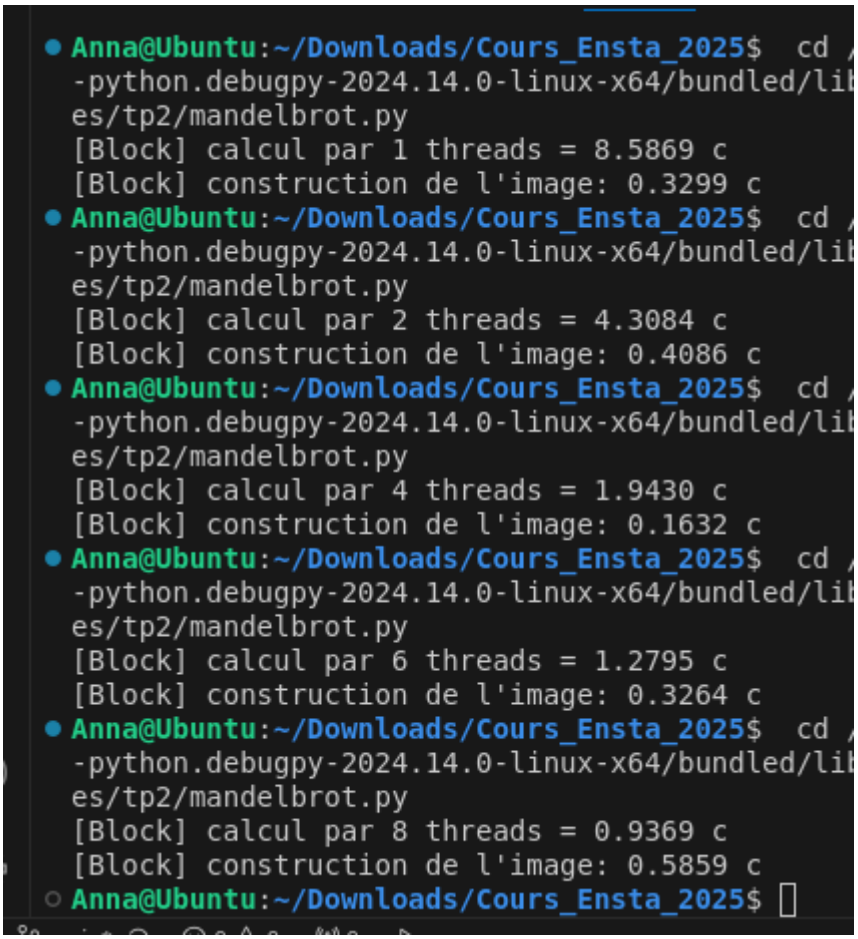
1.1.Partition statique par blocs



Speedup (2th) = $\frac{8.58}{4.31} = 1.99$

Speedup (4th) = $\frac{8.58}{1.94} = 4.42$

Speedup (6th) = $\frac{8.58}{1.28} = 6.71$

Speedup (8th) = $\frac{8.58}{0.94} = 9.13$

Les résultats montrent qu'à mesure que le nombre de processus augmente, le temps d'exécution diminue considérablement et l'accélération s'approche d'une croissance linéaire. Cela indique une bonne efficacité de parallélisation de la tâche. Avec 8 processus, le temps d'exécution a été réduit d'environ 9 fois, ce qui est un excellent résultat. Cette accélération peut être due à des erreurs matérielles ou de mesure, et à mesure que le nombre de processus augmente encore, l'accélération peut devenir moins linéaire en raison de la surcharge de communication interprocessus.

1.2. Répartition statique intercalée (cyclic)

L'idée ici est d'attribuer à chaque processus les lignes d'index i telles que i % nbp == rank. Par exemple, pour nbp = 4 :

Rank 0 calcule les lignes 0, 4, 8, …

Rank 1 calcule les lignes 1, 5, 9, …, etc.

Le code est similaire, à la différence de la manière dont on détermine les indices des lignes à calculer.

Speedup (2th) = $\frac{6.72}{3.57} = 1.88$

Speedup (4th) = $\frac{6.72}{3.02} = 2.22$

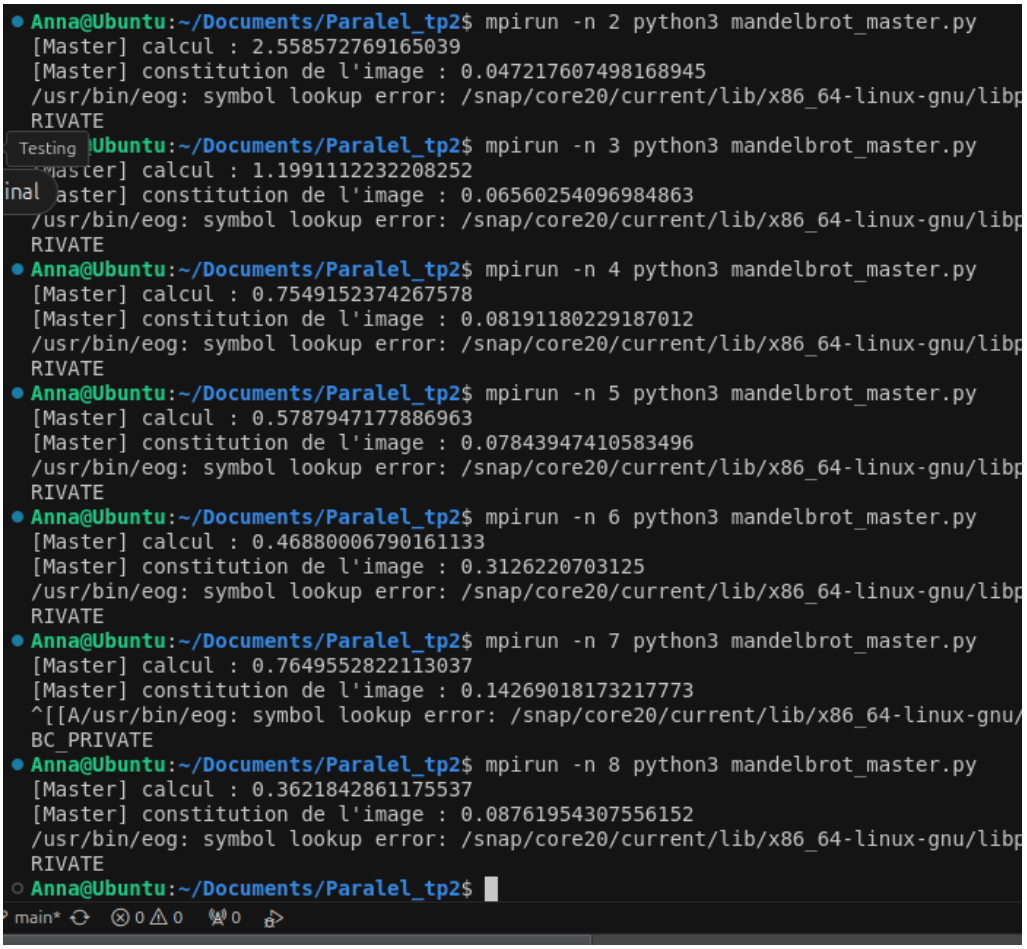
Speedup (6th) = $\frac{6.72}{1.29} = 5.21$

Speedup (8th) = $\frac{6.72}{1.09} = 6.12$

Pour cet exemple, la distribution de lignes par blocs s'est avérée plus efficace en termes d'accélération que la distribution à tour de rôle. Cela peut être dû au fait qu'avec l'allocation de blocs, chaque processus reçoit un bloc de lignes contigu, ce qui améliore la localité des données et permet une utilisation optimale du cache du processeur. La distribution circulaire peut en fait être utile si la charge de calcul sur les lignes est très hétérogène (c'est-à-dire que certaines lignes nécessitent un nombre d'itérations beaucoup plus important). Cependant, dans notre cas, la surcharge associée à la transmission de nombreux petits blocs de données puis à leur collecte a entraîné une diminution de la vitesse globale.

1.3. Maître-esclave

Dans cette approche, le processus maître distribue les tâches (ici, le calcul d'une ligne) à la volée aux processus esclaves qui, dès qu'ils terminent leur tâche, demandent une nouvelle ligne à traiter. Cette méthode permet d'équilibrer la charge de façon dynamique.



Speedup (3th) = $\frac{2,6}{1,2} = 2,17$

Speedup (4th) = $\frac{2,6}{0,75} = 3,46$

Speedup (5th) = $\frac{2,6}{0,57} = 4,56$

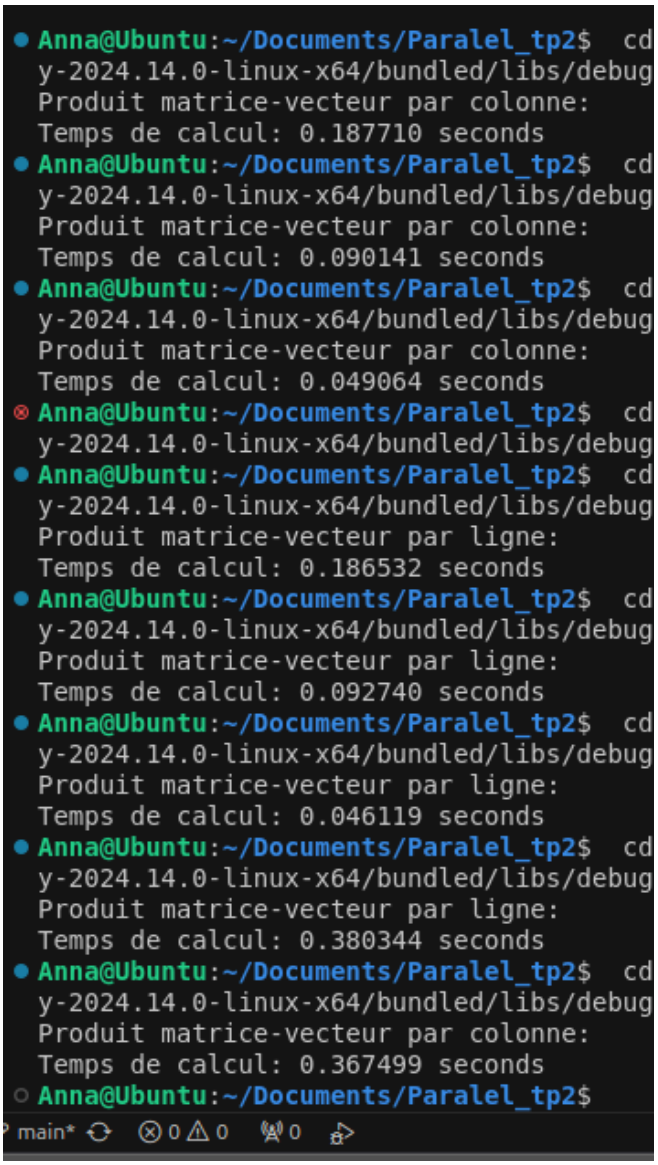
Speedup (6th) = $\frac{2,6}{0,46} = 5,65$

Speedup (7th) = $\frac{2,6}{0,76} = 3,42$

Speedup (8th) = $\frac{2,6}{0,36} = 7.22$

L'approche maître-esclave, qui distribue les tâches de manière dynamique pour équilibrer la charge, offre un compromis intéressant en adaptant le traitement aux variations du coût des tâches. Néanmoins, ses résultats (par exemple, 7,22 avec 8 processus) révèlent une performance irrégulière, probablement liée à l'overhead de communication fréquent entre le maître et les esclaves. Ainsi, bien que le modèle maître-esclave présente une flexibilité appréciable en cas de déséquilibre, il demeure moins performant que la répartition par blocs lorsque la charge est homogène.

2. Produit matrice-vecteur par ligne et par colonne



Colonne (500):

Speedup (2th) = $\frac{0.38}{0,19} = 2$

Speedup (4th) = $\frac{0.38}{0,09} = 4.22$

Speedup (8th) = $\frac{0.38}{0.05} = 7.6$

Ligne(500):

Speedup (2th) = $\frac{0.37}{0,18} = 2.05$

Speedup (4th) = $\frac{0.37}{0,09} = 4.11$

Speedup (8th) = $\frac{0.37}{0,05} = 7.4$

Les deux approches montrent une amélioration quasi linéaire avec l'augmentation du nombre de tâches. La version par colonne atteint un speedup légèrement supérieur pour 4 et 8 tâches, probablement dû à des différences de gestion de la mémoire et d'efficacité dans l'opération collective (Allreduce vs. Allgather). Cependant, les différences restent minimes, indiquant que dans les deux cas, la parallélisation est efficace mais limitée par les coûts de communication.