



OS202 – Systèmes Parallèles

TD2 : Ensemble de Mandelbrot et Produit Matrice-Vecteur

MENESES GAMBOA Carlos

École Nationale des techniques avancées
Février 2026

Table des matières

1 Résumé	2
2 Machine utilisée	2
3 Parallélisation de l'ensemble de Mandelbrot	2
3.1 Q1 – Partition par blocs contigus	2
3.2 Q2 – Distribution cyclique	3
3.3 Q3 – Stratégie maître-esclave (dynamique)	3
4 Produit matrice-vecteur parallèle	4
4.1 a – Partition par colonnes	4
4.2 b – Partition par lignes	5
4.3 Comparaison des deux approches	5
5 Entraînement pour l'examen écrit	5
5.1 Loi d'Amdahl	5
5.2 Nombre de noeuds raisonnable	6
5.3 Loi de Gustafson	6
6 Conclusion générale	7

1 Résumé

Ce rapport présente la parallélisation MPI de l'ensemble de Mandelbrot selon trois stratégies (blocs contigus, distribution cyclique, maître-esclave) ainsi que le produit matrice-vecteur parallèle par colonnes et par lignes. Une analyse théorique des lois d'Amdahl et Gustafson complète l'étude.

2 Machine utilisée

TABLE 1 – Caractéristiques de la machine.

Champ	Valeur
Architecture	x86_64
Model name	11th Gen Intel Core i5-1135G7 @ 2.40GHz
CPU(s)	8
Thread(s) per core	2
Core(s) per socket	4
L1d / L1i cache	192 KiB / 128 KiB
L2 cache	5 MiB
L3 cache	8 MiB
Hypervisor	Microsoft (WSL2)

3 Parallélisation de l'ensemble de Mandelbrot

L'ensemble de Mandelbrot est défini par la suite récursive :

$$\begin{cases} z_0 = 0 \\ z_{n+1} = z_n^2 + c \end{cases}$$

où c est un paramètre complexe. Si $|z_N| > 2$ pour un certain N , la suite diverge. On fixe $N_{\max} = 50$ itérations et une image de 1024×1024 pixels.

3.1 Q1 – Partition par blocs contigus

Chaque processus calcule un bloc contigu de H/nbp lignes consécutives. Le processus 0 rassemble les résultats via `gather`.

TABLE 2 – Mandelbrot – Partition par blocs contigus.

Processus	Temps (s)	Speedup	Efficacité (%)
1	2.67	1.00	100.0
2	1.62	1.65	82.5
3	1.36	1.96	65.3
4	1.21	2.21	55.3

Interprétation : Le speedup est sous-linéaire car la charge de travail est *déséquilibrée*. Les lignes centrales de l'image (proches du cardioïde de Mandelbrot) nécessitent beaucoup plus d'itérations

que les lignes extérieures qui divergent rapidement. Ainsi, le processus qui hérite des lignes centrales devient un goulot d'étranglement.

3.2 Q2 – Distribution cyclique

Pour équilibrer la charge, on distribue les lignes de façon cyclique : le processus p calcule les lignes $p, p + \text{nbp}, p + 2 \cdot \text{nbp}, \dots$. Cela entrelace les lignes “faciles” et “difficiles” entre tous les processus.

TABLE 3 – Mandelbrot – Distribution cyclique.

Processus	Temps (s)	Speedup	Efficacité (%)
1	2.67	1.00	100.0
2	1.80	1.48	74.0
3	1.44	1.85	61.7
4	1.38	1.93	48.3

Comparaison : De manière surprenante, les deux stratégies donnent des résultats *similaires* (speedup ~ 2 avec 4 processus). Cela s'explique par les **optimisations géométriques** implémentées dans le code :

- Détection du cardioïde principal : $|c|^2 < 0.0625$
- Détection du bulbe secondaire : $|c + 1|^2 < 0.0625$
- Test de la cardioïde : évite les itérations pour les points connus comme convergents

Ces optimisations permettent de *court-circuiter* le calcul pour les points du centre de l'ensemble (qui normalement nécessiteraient 50 itérations). Ainsi, toutes les lignes ont un coût de calcul similaire, et le déséquilibre de charge disparaît.

Vérification expérimentale : En désactivant ces optimisations, on observe une différence significative :

TABLE 4 – Comparaison avec/sans optimisations géométriques (4 processus).

Version	Blocs (s)	Cyclique (s)	Gain cyclique
Avec optimisations	1.21	1.38	-14% (plus lent)
Sans optimisations	2.04	1.57	+23% (plus rapide)

Conclusion : Sans optimisations, le cyclique est effectivement 23% plus rapide grâce à un meilleur équilibrage de charge. Avec optimisations, le déséquilibre disparaît et le cyclique perd son avantage (l'overhead de réordonnancement le rend même légèrement plus lent).

Problème potentiel : Cette stratégie fonctionne bien pour Mandelbrot car les zones de charge élevée sont localisées au centre. Pour d'autres problèmes où la charge est distribuée différemment (par exemple, aléatoirement), la distribution cyclique ne garantirait pas un bon équilibrage.

3.3 Q3 – Stratégie maître-esclave (dynamique)

Le processus 0 (maître) distribue dynamiquement des chunks de lignes aux processus esclaves. Quand un esclave termine un chunk, il en demande un nouveau. Cela permet un équilibrage automatique de la charge.

TABLE 5 – Mandelbrot – Maître-esclave (chunks de 32 lignes).

Processus	Esclaves	Temps (s)	Speedup*	Efficacité (%)
2	1	3.59	0.74	74.0
3	2	2.07	1.29	64.5
4	3	1.83	1.46	48.7

*Speedup calculé par rapport au temps séquentiel de 2.67s.

Analyse : Contrairement aux attentes, la stratégie maître-esclave est *moins performante* que les approches statiques. Cela s'explique par :

1. **Perte d'un processeur** : Le maître ne calcule pas, donc avec 4 processus on n'a que 3 workers.
2. **Overhead de communication** : Chaque chunk nécessite 2 messages (envoi de tâche + réception de résultat), soit ~ 32 aller-retours pour 1024 lignes.
3. **Bon équilibrage naturel du cyclique** : Pour Mandelbrot, la distribution cyclique équilibre déjà bien la charge, rendant le dynamique superflu.

Conclusion : La stratégie maître-esclave serait avantageuse pour des problèmes avec une charge très variable et imprévisible (par exemple, temps de calcul variant de 1 à 1000 selon les données). Pour Mandelbrot avec une charge prévisible, les approches statiques sont préférables.

4 Produit matrice-vecteur parallèle

On considère le produit $v = A \cdot u$ où A est une matrice $N \times N$ avec $A_{ij} = (i + j) \bmod N + 1$ et u un vecteur de taille N . On prend $N = 4800$ (divisible par 1, 2, 3, 4).

Note importante : Pour éviter la contention entre les threads BLAS et les processus MPI, il est essentiel de configurer `OMP_NUM_THREADS=1` avant d'importer NumPy.

4.1 a – Partition par colonnes

Chaque processus détient $N_{\text{loc}} = N/\text{nbp}$ colonnes de A et la partie correspondante de u . Le calcul local produit une contribution partielle au vecteur résultat. Un `Allreduce` avec `MPI_SUM` combine toutes les contributions.

$$N_{\text{loc}} = \frac{N}{\text{nbp}}$$

TABLE 6 – Produit matrice-vecteur par colonnes ($N = 4800$).

Processus	Cómputo (ms)	Comm (ms)	Total (ms)	Speedup
1	8.6	0.08	8.7	1.00
2	6.6	0.40	7.0	1.24
3	5.5	0.57	6.1	1.43
4	5.2	0.87	6.1	1.43

Observation : Le speedup plafonne à 1.43 car le temps de communication (`Allreduce` sur un

vecteur de taille N) augmente avec le nombre de processus, compensant partiellement le gain de calcul.

4.2 b – Partition par lignes

Chaque processus détient $N_{\text{loc}} = N/\text{nbp}$ lignes de A et le vecteur u complet. Le calcul local produit directement une partie du vecteur résultat. Un **Allgather** rassemble toutes les parties.

$$N_{\text{loc}} = \frac{N}{\text{nbp}}$$

TABLE 7 – Produit matrice-vecteur par lignes ($N = 4800$).

Processus	Cómputo (ms)	Comm (ms)	Total (ms)	Speedup
1	9.5	0.07	9.6	1.00
2	7.5	0.14	7.6	1.26
3	6.1	0.22	6.3	1.52
4	6.0	0.57	6.6	1.45

4.3 Comparaison des deux approches

TABLE 8 – Comparaison colonnes vs lignes.

Aspect	Par colonnes	Par lignes
Stockage local de A	$N \times N_{\text{loc}}$	$N_{\text{loc}} \times N$
Stockage de u	N_{loc} (partiel)	N (complet)
Résultat local	Contribution partielle (taille N)	Partie finale (taille N_{loc})
Communication	Allreduce (somme de vecteurs N)	Allgather (concaténation)
Volume communiqué	N doubles par processus	N_{loc} doubles par processus

Conclusion : La partition par lignes est théoriquement plus efficace en communication car **Allgather** transfère moins de données (N_{loc} vs N par processus). En pratique, pour des problèmes de cette taille, les différences sont minimales.

5 Entraînement pour l'examen écrit

5.1 Loi d'Amdahl

Alice a un code où 90% du temps est parallélisable ($p = 0.9$) et 10% est séquentiel ($s = 0.1$).

La loi d'Amdahl donne l'accélération maximale :

$$S(n) = \frac{1}{s + \frac{p}{n}} = \frac{1}{0.1 + \frac{0.9}{n}}$$

Pour $n \rightarrow \infty$:

$$S_{\max} = \frac{1}{s} = \frac{1}{0.1} = 10$$

Réponse : L'accélération maximale est de **10**.

5.2 Nombre de nœuds raisonnable

On cherche n tel que le gain marginal soit acceptable. Calculons l'efficacité $E = S/n$:

TABLE 9 – Accélération et efficacité selon la loi d'Amdahl.

n (nœuds)	$S(n)$	Efficacité (%)
1	1.00	100.0
2	1.82	91.0
4	3.08	77.0
8	4.71	58.8
10	5.26	52.6
16	6.40	40.0
32	7.62	23.8

Réponse : Un nombre raisonnable serait **8 à 10 nœuds**, où l'on obtient environ 50% de l'accélération maximale avec une efficacité encore acceptable ($\sim 50\text{--}60\%$). Au-delà, les ressources sont gaspillées.

5.3 Loi de Gustafson

Alice observe une accélération maximale de 4 (au lieu de 10 théorique). D'après la loi d'Amdahl, si l'accélération se sature à 4, cela implique que la fraction séquentielle *effective* est :

$$s = \frac{1}{S_{\max}} = \frac{1}{4} = 0.25$$

et que la partie parallélisable vaut :

$$p = 1 - s = 0.75$$

Si l'on **double** maintenant la quantité de données et que l'on suppose une complexité parallèle linéaire, alors la partie parallélisable (le travail « scalable ») est doublée tandis que la partie séquentielle reste approximativement constante. Le nouveau temps séquentiel normalisé devient :

$$T'_1 = s + 2 \cdot p = 0.25 + 2 \times 0.75 = 1.75$$

Par conséquent, la nouvelle fraction séquentielle est :

$$s' = \frac{s}{T'_1} = \frac{0.25}{1.75} = \frac{1}{7} \approx 0.143$$

En appliquant la loi de Gustafson pour n nœuds :

$$S_G(n) = n - s'(n - 1) = n - \frac{1}{7}(n - 1) = \frac{7n - (n - 1)}{7} = \frac{6n + 1}{7}$$

Pour n grand :

$$S_G(n) \approx \frac{6}{7}n \approx 0.857n$$

Réponse : En doublant les données, l'accélération maximale suit $S_G(n) = \frac{6n+1}{7}$, soit environ **85.7% d'efficacité parallèle** pour un grand nombre de nœuds. Par exemple, avec 8 nœuds : $S_G(8) = \frac{49}{7} = 7$.

6 Conclusion générale

- **Mandelbrot – Équilibrage de charge :** La distribution cyclique offre un bon compromis entre simplicité et performance. Le maître-esclave, bien que flexible, introduit un overhead significatif pour ce problème.
- **Produit matrice-vecteur :** Les deux partitions (colonnes/lignes) donnent des performances similaires. La partition par lignes est légèrement plus efficace en communication (**Allgather** vs **Allreduce**).
- **Contention BLAS/MPI :** Il est crucial de limiter les threads BLAS (**OMP_NUM_THREADS=1**) pour éviter la surscription avec MPI.
- **Lois de scaling :** Amdahl prédit les limites du speedup pour un problème fixe (ici $S_{\max} = 10$). Gustafson montre qu'en augmentant la taille du problème, on peut maintenir une bonne efficacité.