الجـــمهوريـــة الجزائريـــة الديمـــقراطيـــــة الشـعبيـة
ⵜⴻⵣⴰⵢⵉⵔ ⵜⴰⵣⵣⴰⵢⵔⵜ ⵜⴰⵎⴰⵣⴰⵖⵜ ⵜⴰⵡⵉⵔⵉⵜ
République Algérienne Démocratique et Populaire
وزارة التعليـــم العـــالي والبـــحث العلمـــي
ⴰⵡⵔⴻⵔ ⵏ ⵜⵓⵙⵎⵎⵉⵔⵉⵏ ⵜⵉⵏⴻⵣ ⴰ ⵜⵏⴰⴻ ⵜⵓⵓⵏⴰ
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

ECOLE NATIONALE SUPÉRIEURE D'INFORMATIQUE

المدرسة الوطنية العليا للإعلام الآلي
ⴰⵡⵜⵓⴰⴽ ⴰⵡⵎⵏⴰⵍ ⵜⵉⵏⴻⵣ ⵏ ⵜⵓⵜⵏⵎⵔⴷⵜ
École nationale Supérieure d'Informatique

Rapport de Travail Pratique (TP)

Module : Calcul intensif (HPC – High Performance Computing)

2ème année Cycle Supérieur (2CS)

Option : Systèmes Intelligents et Données (SID)

Groupe : 2SD1

Thème :

Programmation parallèle avec Pthreads

Application à l'ensemble de Mandelbrot

Réalisé par :

- ABOUD Ibrahim
- BOUYAKOUB Rayane

Proposé par :

- Mme HAICHOUR Amina Selma.

Année universitaire : 2024-2025

Tables des matières

Tables des matières I

Liste des figures II

Liste des tableaux II

1. Introduction 1

2. Objectifs 1

3. Présentation du problème traité 2

4. Présentation de l'algorithme 4

5. Intérêt de la parallélisation 6

6. Stratégie de parallélisation 7

7. Solution de la parallélisation 8

8. Résultats 10

9. Interprétations 13

   9.1. Impact du nombre d'itérations sur les performances 13

   9.2. Dégradation des performances et limitations matérielles 14

   9.3. Nombre optimal de threads 15

10. Conclusion 15

Références bibliographiques 16

Liste des figures

Figure 1: Représentation visuelle de l'ensemble de Mandelbrot. 2

Figure 2: Impact du nombre d'itérations sur la précision de la représentation lors d'un zoom profond sur une bordure de l'ensemble de Mandelbrot. 3

Figure 3: Répartition du temps d'exécution par fonction selon le profilage gprof. 7

Figure 4: Schéma de la parallélisation de l'algorithme de Mandelbrot. 9

Figure 5: Évolution du temps d'exécution en fonction du nombre de threads pour différents nombres d'itérations maximales. 11

Figure 6: Accélération obtenue en fonction du nombre de threads pour différents nombres d'itérations maximales. 11

Figure 7: Efficacité de la parallélisation en fonction du nombre de threads pour différents nombres d'itérations maximales. 12

Liste des tableaux

Tableau 1: Mesure du temps d'exécution des différentes tâches du programme de génération de l'ensemble de Mandelbrot. 6

Tableau 2: Mesures de performance de l'implémentation parallèle de l'ensemble de Mandelbrot en fonction du nombre de threads et du nombre maximal d'itérations. 10

1. Introduction

Dans l'ère numérique actuelle, le calcul haute performance (HPC - High Performance Computing) est devenu un pilier fondamental de l'innovation technologique et scientifique. Face à des volumes de données toujours croissants et des applications de plus en plus complexes, les besoins en puissance de calcul n'ont cessé d'augmenter. Le HPC répond à ces exigences en permettant de traiter rapidement des problèmes computationnels complexes grâce à l'utilisation efficace des architectures parallèles et distribuées.

L'importance du HPC se manifeste dans de nombreux domaines : de la simulation météorologique à la recherche en génomique, en passant par l'intelligence artificielle et la modélisation financière. Ces applications modernes nécessitent non seulement une exécution rapide mais aussi une utilisation optimale des ressources matérielles disponibles. La parallélisation des calculs apparaît alors comme une solution incontournable pour atteindre les performances requises.

Dans ce contexte, nous nous intéressons à la parallélisation d'un problème mathématique fascinant : l'ensemble de Mandelbrot. Cette fractale, découverte par Benoît Mandelbrot dans les années 1980, présente un excellent cas d'étude pour l'application des techniques de parallélisation, car son calcul est à la fois intensif et naturellement parallélisable.

Ce rapport est structuré de la manière suivante : nous commencerons par présenter le problème de l'ensemble de Mandelbrot et l'algorithme séquentiel permettant de le générer. Nous analyserons ensuite l'intérêt de la parallélisation en identifiant les points critiques de l'algorithme. Nous détaillerons alors notre stratégie de parallélisation et sa mise en œuvre. Enfin, nous présenterons et analyserons les résultats obtenus, en comparant les performances des versions séquentielles et parallèles.

2. Objectifs

Les objectifs de ce travail pratique sont nombreux :

1. Identifier les éléments de parallélisme dans un programme séquentiel donné, en vue de son exécution sur une architecture à mémoire partagée.

2. Implémenter de manière efficace les fonctions de la bibliothèque Pthreads pour paralléliser un programme séquentiel, visant à une exécution efficace sur une architecture parallèle à mémoire partagée.

3. Analyser et comparer les performances d'une exécution séquentielle et d'une exécution parallèle du même programme afin d'évaluer l'accélération obtenue.

3. Présentation du problème traité

L'ensemble de Mandelbrot, découvert dans les années 1980 par le mathématicien Benoît Mandelbrot, est une fractale célèbre en mathématiques. Il est défini comme l'ensemble des points $c \in \mathbb{C}$ pour lesquels la suite récurrente suivante reste bornée :

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

Il est prouvé mathématiquement que si la valeur du module $|z_n| \geq 2$, la suite va diverger vers l'infini, et le point c sera alors considéré comme étant en dehors de l'ensemble de Mandelbrot.

Pour représenter cet ensemble visuellement, on parcourt une région du plan complexe, chaque point de cette zone représentant un complexe c. On applique l'équation de récurrence définie précédemment pour chaque point, jusqu'à ce que la suite diverge ($|z_n| \geq 2$) ou jusqu'à un nombre maximal d'itérations. Si ce nombre maximal est atteint sans que $|z_n|$ dépasse 2, on conclut que la suite est bornée et donc le point c appartient à l'ensemble. Chaque pixel de l'image représente un point c et est coloré fonction du comportement de la suite récurrente :

- Les points qui appartiennent à l'ensemble de Mandelbrot sont colorés en noir.
- Les points en dehors de l'ensemble sont colorés en fonction du nombre d'itérations nécessaires avant que la suite diverge.

Une illustration de cette représentation visuelle de l'ensemble de Mandelbrot est fournie dans la figure ci-dessous, mettant en évidence les points de divergence rapide en couleurs vives et les points appartenant à l'ensemble en noir.

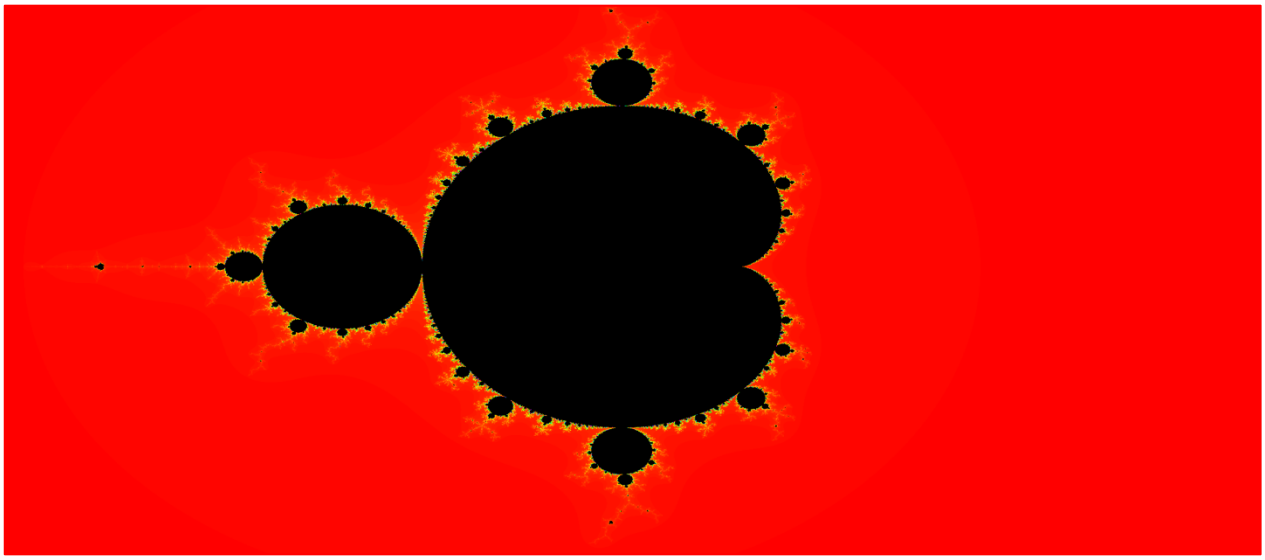
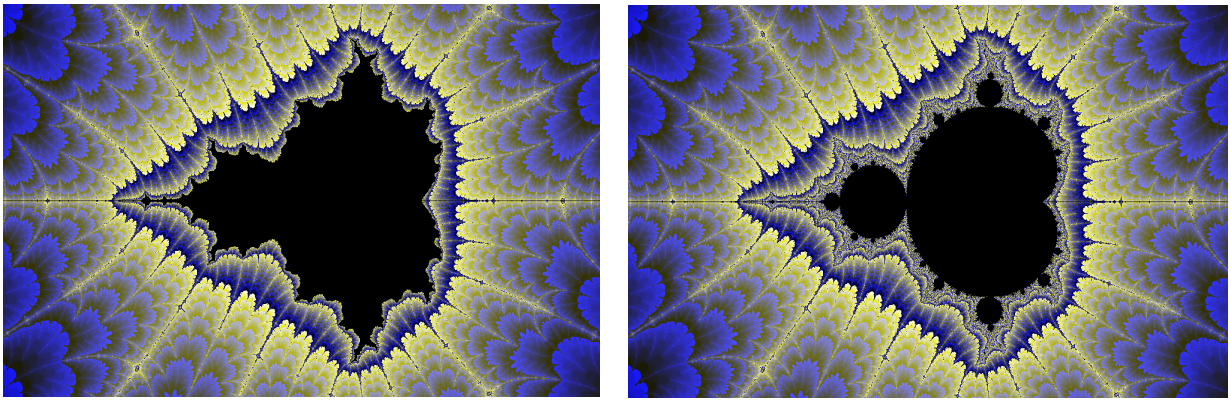


Figure 1: Représentation visuelle de l'ensemble de Mandelbrot.

L’une des propriétés fascinantes de l’ensemble de Mandelbrot réside dans le fait qu’il génère une fractale infiniment complexe : chaque fois qu’on zoome sur les bordures de sa représentation, on découvre de nouvelles structures fractales, de plus en plus petites, mais similaires dans leur apparence. Ce phénomène d’autosimilarité crée une beauté visuelle unique qui explique son attrait dans l’art génératif et la visualisation mathématique. Pour explorer davantage cette propriété fascinante de l'ensemble, on vous recommande de consulter le lien suivant.

La précision de cette représentation dépend de deux facteurs principaux : la résolution de l’image (le nombre de pixels) et le nombre d’itérations utilisé pour chaque point. Plus la résolution est élevée, plus les détails sont fidèlement rendus. De même, augmenter le nombre d’itérations permet d’identifier les points qui appartiennent à l’ensemble avec plus de précision, notamment dans les zones complexes où la limite de divergence est difficile à déterminer.

La figure ci-dessous illustre l’effet du nombre d’itérations sur la précision de la représentation lors d’un zoom profond sur une bordure de l’ensemble de Mandelbrot.



Représentation avec 1000 itérations. Représentation avec 10000 itérations.

Figure 2: Impact du nombre d’itérations sur la précision de la représentation lors d’un zoom profond sur une bordure de l’ensemble de Mandelbrot.

En pratique, la génération de l’ensemble de Mandelbrot est très gourmande en calculs. Une optimisation par parallélisation est donc nécessaire pour produire des images haute résolution, en particulier lors des zooms profonds où chaque pixel et chaque itération ajoutent des détails cruciaux à la structure.

4. Présentation de l'algorithme

Après avoir présenté les concepts fondamentaux de l'ensemble de Mandelbrot, nous allons désormais aborder l'algorithme permettant de générer une représentation visuelle de cet ensemble.

Algorithme 1 : Représentation Mandelbrot

Entrées : $maxIterations$. // nombre maximal d'itération.
$Largeur\ et\ hauteur$. // dimensions de l'image.

Sortie : Représentation visuelle de l'ensemble de Mandelbrot.

1. **Pour** chaque point de coordonnées $(x; y)$ du plan délimité par les dimensions de l'image
2. $\quad c \leftarrow x + yi$
3. $\quad z \leftarrow 0$
4. $\quad iterations \leftarrow 0$
5. $\quad$ **Tant que** $(|z| \leq 2)$ **et** $(iterations < maxIterations)$ // Vérifier l'appartenance.
6. $\quad\quad z \leftarrow z^2 + c$
7. $\quad\quad iterations \leftarrow iterations + 1$
8. $\quad$ **Fin Tant que**
9. $\quad$ **Si** $iterations = maxIterations$ // Le point appartient à l'ensemble.
10. $\quad\quad Couleur\ du\ pixel \leftarrow noir$
11. $\quad$ **Sinon** // Le point n'appartient pas à l'ensemble.
12. $\quad\quad Couleur\ du\ pixel \leftarrow couleur\ en\ fonction\ du\ nombre\ d'itérations$
13. $\quad$ **Fin Si**
14. **Fin Pour**

Dans certains environnements où l'arithmétique complexe n'est pas supportée, on peut séparer les parties réelle et imaginaire pour effectuer les calculs nécessaires à la représentation de l'ensemble de Mandelbrot. C'est pour cela que l'on définit deux variables pour le nombre z, une pour la partie réelle z_r, et une pour la partie imaginaire z_i, on va faire de même pour c. Le calcul $z^2 + c$ devient donc :

$$z = z^2 + c$$
$$z = (z_r + iz_i)^2 + (c_r + ic_i)$$
$$z = z_r^2 + 2iz_r z_i + (iz_i)^2 + c_r + ic_i$$
$$z = (z_r^2 - z_i^2 + c_r) + i(2z_r z_i + c_i)$$

Il ne reste plus qu'à séparer la partie entière de la partie imaginaire

$$z_r = z_r^2 - z_i^2 + c_r$$
$$z_i = 2z_r z_i + c_i$$

De plus au lieu de calculer le module de z et de le comparer à 2, on va juste calculer la somme des carrés de ses composantes et comparer le résultat à 4, en effet :

$$|z| \leq 2 \iff |z|^2 \leq 4 \iff z_r^2 + z_i^2 \leq 4$$

Algorithme 2 : Représentation Mandelbrot Alternative.

Entrées : $maxIterations$. // nombre maximal d'itération
$Largeur\ et\ hauteur$. // dimensions de l'image

Sortie : Représentation visuelle de l'ensemble de Mandelbrot.

1. **Pour** chaque point de coordonnées (x ; y) du plan délimité par les dimensions de l'image
2. $c_r \leftarrow x$
3. $c_i \leftarrow y$
4. $z_r \leftarrow 0$
5. $z_i \leftarrow 0$
6. $iterations \leftarrow 0$
7. **Tant que** ($z_r * z_r + z_i * z_i \leq 4$) et ($iterations < maxIterations$) // Vérifier l'appartenance.
8. $temp \leftarrow z_r$
9. $z_r \leftarrow z_r * z_r - z_i * z_i + c_r$
10. $z_i \leftarrow 2 * z_r * z_i + c_i$
11. $iterations \leftarrow iterations + 1$
12. **Fin Tant que**
13. **Si** $iterations = maxIterations$ // Le point appartient à l'ensemble
14. $Couleur\ du\ pixel \leftarrow noir$
15. **Sinon** // Le point n'appartient pas à l'ensemble
16. $Couleur\ du\ pixel \leftarrow couleur\ en\ fonction\ du\ nombre\ d'itérations$
17. **Fin Si**
18. **Fin Pour**

5. Intérêt de la parallélisation

Pour analyser les performances de notre programme de génération de l'ensemble de Mandelbrot, nous avons utilisé deux méthodes de mesure du temps d'exécution. La première méthode consiste à utiliser la fonction $clock()$ pour mesurer le temps CPU des différentes parties du code, et a produit les résultats suivants :

Tâche	Temps d'exécution (s)	Proportion du temps d'exécution
Détermination des points qui appartiennent à l'ensemble de Mandelbrot.	7.957663	98,98 %
Sauvegarde de la représentation visuelle de l'ensemble de Mandelbrot	0.081316	1.02 %
Total	8.038979	100 %

Tableau 1: Mesure du temps d'exécution des différentes tâches du programme de génération de l'ensemble de Mandelbrot.

Ces mesures montrent que la détermination des points appartenant à l'ensemble de Mandelbrot consomme presque tout le temps de calcul (98,98 %), tandis que la sauvegarde de la visualisation ne représente qu'une fraction infime du temps d'exécution (1,02 %).

Pour confirmer ces résultats et obtenir plus de détails, nous avons également utilisé l'outil de profilage **gprof**, un outil performant pour analyser le temps passé dans chaque fonction d'un programme en C, C++ ou Fortran. Cet outil génère un rapport qui facilite l'identification des goulets d'étranglement dans le code. Les étapes pour générer ce profil sont les suivantes :

1. gcc -pg -o main main.c
2. ./main
3. gprof main gmon.out > analysis.txt

Le profil généré est présenté dans la figure ci-dessous :

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 98.87      7.87     7.87        1     7.87     7.87  computePixelValues
  0.75      7.93     0.06                             __sfp_handle_exceptions
  0.13      7.94     0.01                             __divtf3
  0.13      7.95     0.01                             __subtf3
  0.13      7.96     0.01                             __trunctfdf2
  0.00      7.96     0.00        1     0.00     0.00  savePPM
```

Figure 3: Répartition du temps d'exécution par fonction selon le profilage gprof.

Le profil gprof confirme que la fonction **computePixelValues**, chargée de déterminer l'appartenance des points à l'ensemble et de leur attribuer une couleur, constitue le "hotspot" principal du code, consommant 98,87 % du temps d'exécution. À l'inverse, la fonction **savePPM**, qui enregistre la visualisation en image, a un impact négligeable sur le temps total.

Cette analyse montre que le calcul des valeurs de pixels pour l'ensemble de Mandelbrot est une opération très coûteuse, car elle nécessite de nombreuses itérations pour chaque point, ce qui en fait un excellent candidat pour l'optimisation par parallélisation. Puisque chaque point peut être traité indépendamment, il est possible de répartir ces calculs sur plusieurs processeurs ou threads, permettant ainsi de réduire le temps d'exécution total en tirant parti des ressources de calcul parallèles.

6. Stratégie de parallélisation

Pour paralléliser notre programme de génération de l'ensemble de Mandelbrot, nous avons opté pour un partitionnement de domaine (ou un partitionnement de données). Cette stratégie consiste à diviser l'ensemble des points de l'image en sous-domaines qui peuvent être traités indépendamment, chaque sous-domaine étant assigné à un thread distinct. Le calcul pour chaque point de l'ensemble de Mandelbrot est indépendant des autres, ce qui rend cette approche particulièrement adaptée, car chaque tâche peut être exécutée sans attendre les résultats des autres.

Le choix de diviser l'image en sous-domaines s'explique par le fait que cela minimise la communication entre threads, puisque chaque thread calcule les points de son sous-domaine sans interagir avec les autres threads. Cette indépendance réduit la nécessité de synchronisation entre threads, ce qui améliore les performances.

Dans notre implémentation, chaque thread traite une portion de l'image en parcourant les lignes de façon cyclique. Par exemple, le thread 0 calcule les lignes 0, $Nbthreads$, $2Nbthreads$, etc., le thread 1 s'occupe des lignes 1, $Nbthreads + 1$, $2Nbthreads + 1$, et ainsi de suite. Ce partitionnement cyclique assure une bonne répartition des tâches et un bon équilibre de charge.

7. Solution de la parallélisation

La solution parallèle résultant de cette stratégie de partitionnement est présentée dans l'algorithme suivant

Algorithme 3 : Représentation Mandelbrot Parallélisée.

Entrées : $maxIterations$. // Le nombre maximal d'itérations.
$Largeur\ et\ hauteur$. // Dimensions de l'image.
$NbThreads$. // Nombre de threads pour la parallélisation.

Sortie : Représentation visuelle de l'ensemble de Mandelbrot.

1. Diviser l'espace de travail en sous-régions pour chaque thread.
2. Pour chaque sous-région, assigner un thread distinct.
3. **Dans chaque** thread
4. **Pour** chaque point de coordonnées $(x; y)$ dans la sous-région du thread
5. $c_r \leftarrow x$
6. $c_i \leftarrow y$
7. $z_r \leftarrow 0$
8. $z_i \leftarrow 0$
9. $iterations \leftarrow 0$
10. **Tant que** ($z_r * z_r + z_i * z_i \leq 4$) et ($iterations < maxIterations$) // Vérifier l'appartenance.
11. $temp \leftarrow z_r$
12. $z_r \leftarrow z_r * z_r - z_i * z_i + c_r$
13. $z_i \leftarrow 2 * z_r * z_i + c_i$
14. $iterations \leftarrow iterations + 1$
15. **Fin Tant que**
16. **Si** $iterations = maxIterations$ // Le point appartient à l'ensemble
17. $Couleur\ du\ pixel \leftarrow noir$
18. **Sinon** // Le point n'appartient pas à l'ensemble
19. $Couleur\ du\ pixel \leftarrow couleur\ en\ fonction\ du\ nombre\ d'itérations$
20. **Fin Si**
21. **Fin Pour**
22. **Fin tâche du thread**
23. Synchroniser tous les threads une fois les calculs terminés.
24. Générer l'image finale à partir des données calculées par chaque thread.

Le schéma ci-dessous illustre la solution parallèle de l'algorithme de Mandelbrot, présentant la répartition des tâches entre les threads, la décomposition du plan en sous-régions pour chaque thread, ainsi que les communications et synchronisations essentielles à la coordination des calculs pour la génération finale de l'image.

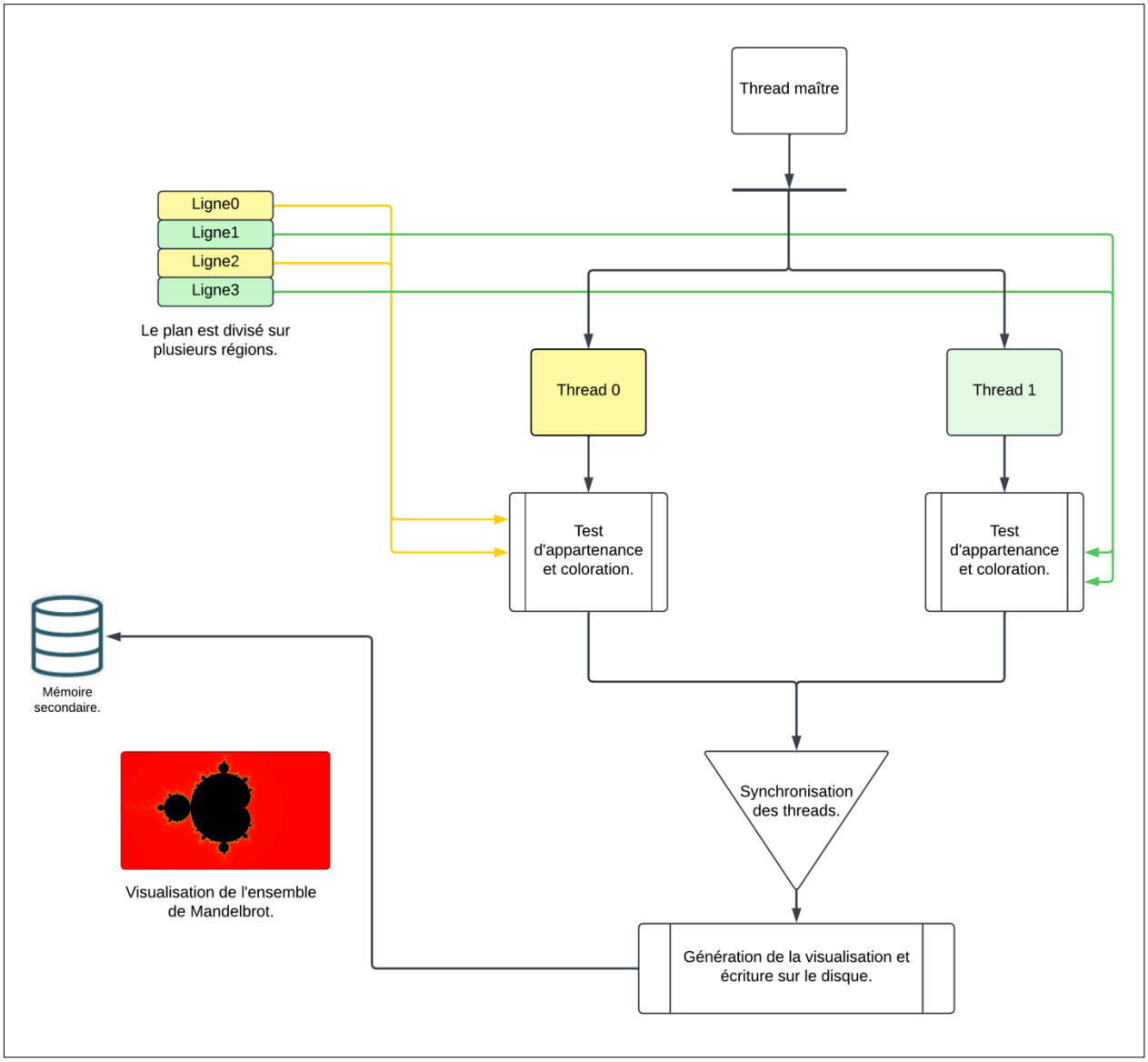


Figure 4: Schéma de la parallélisation de l'algorithme de Mandelbrot.

8. Résultats

Dans le but d'évaluer les performances de notre implémentation parallèle de l'algorithme de génération de l'ensemble de Mandelbrot, nous avons réalisé une série de tests en faisant varier deux paramètres clés : le nombre de threads (1 à 32) et le nombre maximal d'itérations (1000 et 10000). Pour chaque configuration, nous avons mesuré le temps d'exécution, calculé l'accélération (rapport entre le temps séquentiel et le temps parallèle) et l'efficacité de la parallélisation (accélération divisée par le nombre de threads). Les résultats de ces mesures sont présentés dans le tableau ci-dessous.

Nombre de threads	Nombre maximal d'itérations	Temps d'exécution (s)	Accélération	Efficacité
1	1000	2,90	0,00	0,00
2	1000	1,66	1,75	0,87
4	1000	1,13	2,57	0,64
8	1000	0,95	3,06	0,38
16	1000	1,01	2,88	0,18
32	1000	1,08	2,68	0,08
1	10000	10,31	0,00	0,00
2	10000	4,83	2,13	1,07
4	10000	3,08	3,35	0,84
8	10000	2,78	3,71	0,46
16	10000	2,77	3,72	0,23
32	10000	3,56	2,90	0,09

Tableau 2: Mesures de performance de l'implémentation parallèle de l'ensemble de Mandelbrot en fonction du nombre de threads et du nombre maximal d'itérations.

Pour mieux visualiser l'évolution des performances et faciliter l'analyse des résultats, on a représenté graphiquement les différentes métriques mesurées en fonction du nombre de threads, pour les deux configurations du nombre maximal d'itérations (1000 et 10000). Les résultats de cette visualisation sont présentés dans les figures suivantes.

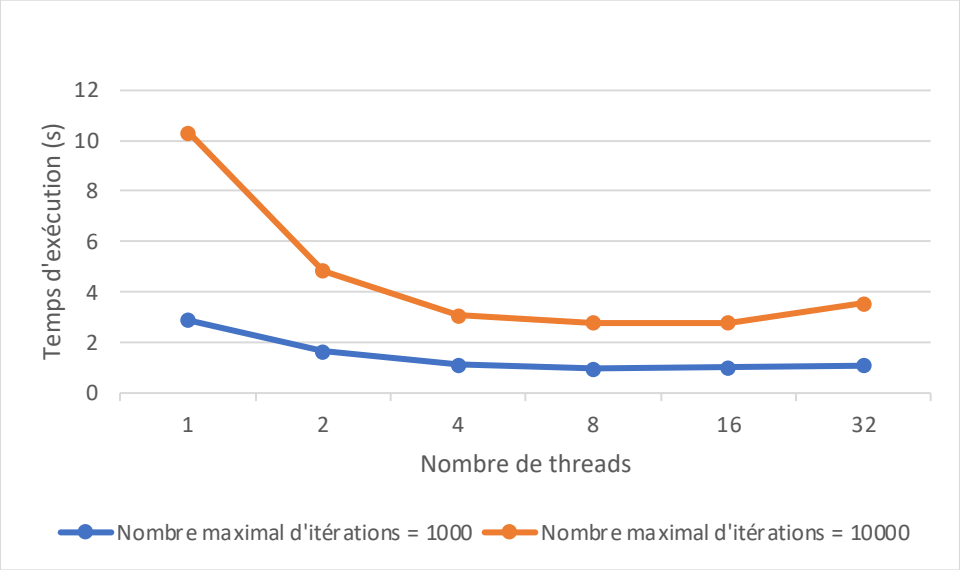


Figure 5: Évolution du temps d'exécution en fonction du nombre de threads pour différents nombres d'itérations maximales.

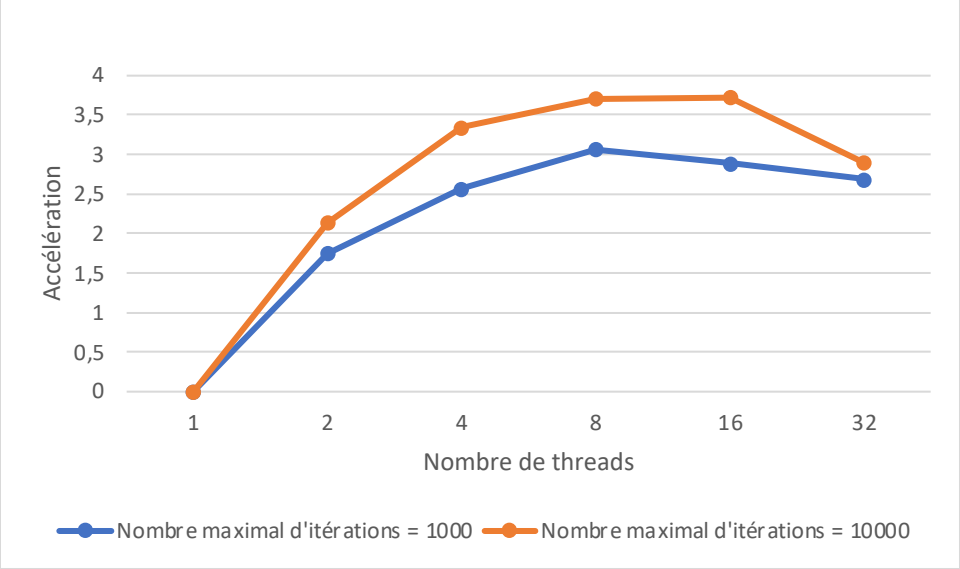


Figure 6: Accélération obtenue en fonction du nombre de threads pour différents nombres d'itérations maximales.

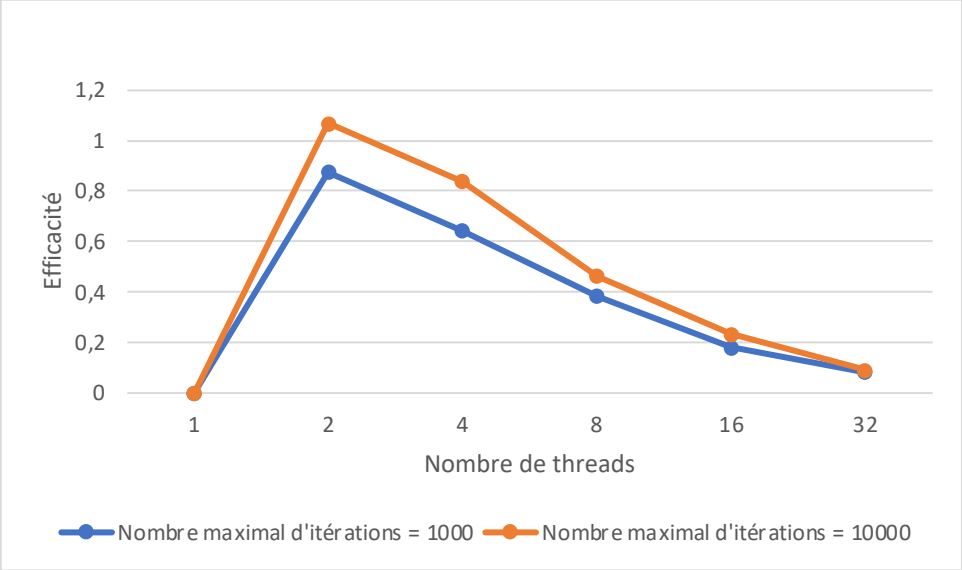


Figure 7: Efficacité de la parallélisation en fonction du nombre de threads pour différents nombres d'itérations maximales.

9. Interprétations

Dans cette section, nous interprétons les résultats des tests pour évaluer l'impact de la parallélisation sur les performances, en fonction du nombre d'itérations et de threads.

9.1. Impact du nombre d'itérations sur les performances

Les résultats montrent une différence significative dans l'impact de la parallélisation selon la charge de calcul. Avec 10000 itérations, on observe une chute spectaculaire du temps d'exécution entre 1 et 2 threads (de 10,31s à 4,83s, soit une réduction de 53%), tandis qu'avec 1000 itérations, la réduction est moins prononcée (de 2,90s à 1,66s, soit 43%).

Cette différence de performance se confirme sur l'ensemble des mesures, comme le montrent les courbes d'accélération et d'efficacité où les résultats pour 10000 itérations (courbes orange) sont systématiquement supérieurs à ceux pour 1000 itérations (courbes bleues). Cette amélioration s'explique par plusieurs facteurs :

- Un meilleur ratio entre temps de calcul et surcoût de parallélisation.
- Une utilisation plus optimale des ressources de calcul avec une charge plus importante.
- Une meilleure répartition de la charge entre les threads.

Ces observations suggèrent que la parallélisation devient plus efficace lorsque la quantité de données à traiter est plus importante. Ce principe peut être extrapolé à d'autres paramètres du problème :

- L'augmentation du nombre d'itérations maximales, comme démontré dans nos tests.
- L'augmentation de la résolution de l'image (plus de pixels à calculer).
- La combinaison des deux paramètres.

En effet, augmenter la résolution de l'image fournirait plus de points à calculer, augmentant ainsi la charge de calcul par thread. Cette augmentation de la quantité de données à traiter permettrait, comme pour l'augmentation du nombre d'itérations, de mieux amortir les coûts liés à la parallélisation (création, synchronisation et destruction des threads).

9.2. Dégradation des performances et limitations matérielles

On observe une dégradation des performances au-delà de 8-16 threads, qui s'explique par plusieurs facteurs clés. Il est important de noter que la machine de test dispose de 4 cœurs physiques avec 8 threads logiques au total.

- **Limitations matérielles et planification**
    - La machine ne disposant que de 8 threads matériels, toute utilisation au-delà nécessite une planification par le système d'exploitation.
    - Le système doit gérer la répartition de 16 ou 32 threads sur seulement 8 cœurs logiques disponibles.
    - Cette situation force certains threads à attendre leur tour d'exécution.

- **Surcoût de gestion**
    - Changements de contexte fréquents : le système doit sauvegarder et restaurer l'état des threads qui alternent sur les cœurs disponibles.
    - Ces opérations de changement de contexte consomment du temps et des ressources supplémentaires.
    - Plus le nombre de threads dépasse le nombre de cœurs disponibles, plus ce surcoût devient important.

- **Autres facteurs de dégradation :** plus le nombre de threads augmente, plus le coût lié à leur création, synchronisation et destruction s'élève, ce qui peut entraîner des dégradations significatives des performances et limiter l'accélération que l'on pourrait obtenir grâce à la parallélisation.

9.3. Nombre optimal de threads

Compte tenu de l'architecture de la machine (8 threads disponibles), les résultats montrent qu'un nombre optimal se situe entre 4 et 8 threads :

- **À 4 threads :** excellent compromis avec une accélération de 2,57 (1000 itérations) et 3,35 (10000 itérations), et une efficacité respective de 0,64 et 0,84.
- **À 8 threads :** accélération maximale (3,06 et 3,71) mais efficacité réduite (0,38 et 0,46).
- **Au-delà de 8 threads :** le dépassement du nombre de threads matériels disponibles entraîne une dégradation des performances.

10. Conclusion

Ce rapport a présenté notre démarche de parallélisation de l'algorithme de génération de l'ensemble de Mandelbrot, depuis l'analyse du problème jusqu'à l'évaluation des performances. Les objectifs fixés ont été atteints avec succès.

L'analyse initiale du programme séquentiel, appuyée par des outils de profilage, nous a permis d'identifier que 98,98% du temps d'exécution était consacré au calcul des points de l'ensemble. Cette observation a confirmé le potentiel de parallélisation du problème.

Notre stratégie de partitionnement de domaine, distribuant les calculs entre plusieurs threads, s'est révélée efficace, comme en témoignent nos résultats expérimentaux. Nous avons obtenu une accélération significative, atteignant un facteur de 3,71 avec 8 threads pour 10000 itérations, démontrant ainsi l'efficacité de notre approche.

Les tests de performance ont également mis en évidence l'importance du choix du nombre de threads en fonction de l'architecture matérielle. Nous avons constaté qu'un nombre optimal de threads se situe entre 4 et 8 sur notre machine de test, correspondant aux ressources matérielles disponibles.

En perspective, ce travail pourrait être étendu en explorant d'autres stratégies de parallélisation, comme l'utilisation de GPU ou d'architectures distribuées, pour améliorer davantage les performances de la génération de l'ensemble de Mandelbrot.

Références bibliographiques

Support de cours

- HAICHOUR, Amina Selma. (2024). Cours High Performance Computing (HPC). École Supérieure d'Informatique (ESI), Alger, 2ème année cycle supérieur.

Ressources en ligne

- Zeste de Savoir. Dessiner la fractale de Mandelbrot.
- Wikipédia. Ensemble de Mandelbrot.

Ressources multimédias

- TheMathemagiciansGuild. The Mandelbrot Set Explained [Vidéo en ligne]. YouTube.