TD 02 - Parallélisation ensemble de Mandelbrot

Luiza Gonçalves Soares
École Nationale Supérieure des Techniques Avancées – ENSTA Paris, Palaiseau, France
luiza.goncalves@ensta-paris.fr

I. Question 01

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 ?

a) Objectif.: À partir du code séquentiel `mandelbrot.py`, l'objectif est de distribuer le calcul de l'ensemble de Mandelbrot sur `nbp` processus en réalisant une *partition équitable par blocs de lignes* (bandes horizontales), puis de *rassembler* l'image sur le processus de rang 0 afin de la sauvegarder/afficher. Enfin, on mesure le temps d'exécution pour plusieurs valeurs de `nbp` et on calcule le *speedup*.

b) Stratégie de partition (blocs de lignes).: L'image comporte H lignes et W colonnes. Le calcul est parallélisé en découpant l'ensemble des lignes en intervalles contigus $[y_{\text{start}}, y_{\text{end}})$, attribués à chaque processus de rang $r \in \{0, \ldots, \texttt{nbp} - 1\}$. Pour assurer une répartition équitable, on utilise la règle suivante :

$$\text{base} = \left\lfloor \frac{H}{\texttt{nbp}} \right\rfloor, \qquad \text{rem} = H \bmod \texttt{nbp}.$$

Les `rem` premiers processus reçoivent base+1 lignes, les autres base lignes. Cette stratégie garantit que la différence de charge entre deux processus est au plus d'une ligne.

c) Calcul local et rassemblement.: Chaque processus calcule la convergence uniquement pour ses lignes, et stocke localement une matrice

$$\texttt{convergence\_local} \in \mathbb{R}^{(y_{\text{end}} - y_{\text{start}}) \times W}.$$

Les blocs sont ensuite envoyés au processus 0 (collecte), qui reconstruit la matrice globale par empilement vertical :

$$\texttt{convergence} = \texttt{vstack}(\texttt{blocs}) \in \mathbb{R}^{H \times W}.$$

La constitution et la sauvegarde/affichage de l'image (via la colormap) sont réalisées uniquement sur le rang 0.

d) Mesure du temps d'exécution.: Le temps mesuré T_p correspond au *temps de calcul de l'ensemble* (boucle principale), et non au temps de constitution de l'image (qui n'est pas représentatif du parallélisme car effectué sur un seul processus). Chaque processus mesure son temps local t_r, puis on retient :

$$T_p = \max_r(t_r),$$

car le temps total est dicté par le processus le plus lent.

e) Speedup et efficacité.: Le *speedup* est défini par

$$S(p) = \frac{T_1}{T_p},$$

et l'*efficacité* par

$$E(p) = \frac{S(p)}{p} = \frac{T_1}{p\,T_p}.$$

f) Résultats expérimentaux.: Les temps ci-dessous correspondent au calcul de l'ensemble de Mandelbrot. Lorsque deux mesures étaient disponibles, la valeur reportée est la moyenne.

nbp	**Temps T_p (s)**	**Speedup $S(p)$**	**Efficacité $E(p)$**
1	1.865	1.00	1.00
2	1.043	1.79	0.90
4	0.564	3.31	0.83
8	0.368	5.08	0.64

TABLE I: Temps d'exécution, speedup et efficacité pour différents nombres de processus.

g) Interprétation des résultats.: On constate que le temps de calcul diminue fortement lorsque le nombre de processus augmente, ce qui confirme que le problème se parallélise bien : le calcul de la convergence est indépendant pour chaque pixel (et donc facilement répartissable par lignes). Les speedups obtenus sont significatifs ($S(4) \approx 3.31$ et $S(8) \approx 5.08$), mais restent inférieurs au cas idéal $S(p) = p$. L'efficacité décroît avec p (de 0.90 à 0.64), ce qui traduit des rendements décroissants à mesure que le nombre de processus augmente.

Cette perte d'efficacité s'explique principalement par :

- **Surcoûts de communication et de synchronisation :** l'utilisation de barrières et de l'opération de rassemblement (collecte des blocs) ajoute un temps qui ne diminue pas proportionnellement avec p.
- **Partie séquentielle incompressible :** l'assemblage final des blocs et la constitution/sauvegarde de l'image sur le rang 0 ne sont pas parallélisés. Selon la loi d'Amdahl, ces sections séquentielles limitent le speedup maximal.
- **Déséquilibre de charge :** même avec une partition équitable en nombre de lignes, certaines zones proches de la frontière de l'ensemble de Mandelbrot nécessitent plus d'itérations et peuvent rendre certains processus plus lents. Comme $T_p = \max_r(t_r)$, le temps total est imposé par le processus le plus lent.

En résumé, la parallélisation par blocs de lignes apporte un gain important pour des valeurs modérées de `nbp`, mais l'efficacité diminue lorsque `nbp` devient plus grand à cause des surcoûts et des parties séquentielles.

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 ?

Dans la répartition par blocs (Q1-A), certaines bandes de lignes peuvent contenir davantage de points proches de la frontière de l'ensemble de Mandelbrot et nécessiter plus d'itérations. Cela crée un déséquilibre : le temps parallèle est alors imposé par le processus le plus lent.

Pour améliorer l'équilibrage tout en restant dans une stratégie statique, on adopte une répartition *entrelacée* (round-robin) des lignes :

$$y = r,\ r + \texttt{nbp},\ r + 2\texttt{nbp}, \dots$$

Chaque processus calcule ainsi des lignes réparties sur toute la hauteur de l'image, ce qui permet de mieux distribuer les zones coûteuses.

Les temps mesurés montrent que cette stratégie peut améliorer les performances pour certaines valeurs de `nbp` (par exemple pour $\texttt{nbp} = 4$), grâce à un meilleur équilibrage de charge. Cependant, le gain n'est pas systématique et dépend du compromis entre équilibrage et surcoût.

h) Limites.: Cette stratégie présente plusieurs inconvénients possibles :

- l'assemblage est plus complexe, car les lignes ne sont plus contiguës ;
- la localité mémoire est moins bonne qu'avec des blocs contigus ;
- la stratégie reste statique : si la zone étudiée change (zoom), la répartition des lignes coûteuses peut être différente et la méthode peut devenir moins efficace.

Ainsi, la répartition entrelacée permet un meilleur équilibrage dans certains cas, mais n'assure pas systématiquement de meilleures performances que la répartition par blocs.

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 ?

Une stratégie maître–esclave est utilisée pour répartir dynamiquement les lignes de l'image. Le rang 0 distribue les indices de lignes à calculer, tandis que les autres processus demandent une ligne, la calculent et renvoient le résultat avant de redemander du travail. Cette méthode permet d'adapter la charge en fonction du coût réel de chaque ligne.

Speedup et comparaison : Le speedup est défini par $S(p) = T_1/T_p$. Par rapport aux répartitions statiques (blocs contigus et entrelacée), la stratégie maître–esclave améliore l'équilibrage lorsque le temps de calcul varie fortement selon les lignes. En revanche, pour un petit nombre de processus, le speedup est plus faible à cause du surcoût de communication (requêtes et envois fréquents au maître).

La stratégie maître–esclave est la plus robuste face à l'hétérogénéité du coût des lignes, mais son overhead de communication et la centralisation sur le maître limitent les performances lorsque le nombre de processus augmente. Elle est donc avantageuse en présence d'un fort déséquilibre de charge, mais pas toujours plus rapide que les stratégies statiques.

II. Question 2

A. Produit matrice–vecteur : stratégie par colonnes

a) Valeur de N_{loc}.: Avec un découpage par blocs de colonnes et N divisible par `nbp`, on a :

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

Pour $N = 120$:

$$\texttt{nbp} = 1 \Rightarrow N_{\text{loc}} = 120, \quad \texttt{nbp} = 2 \Rightarrow N_{\text{loc}} = 60, \quad \texttt{nbp} = 4 \Rightarrow N_{\text{loc}} =$$

b) Parallélisation.: Chaque tâche r assemble uniquement les colonnes

$$j \in [rN_{\text{loc}}, (r+1)N_{\text{loc}}[$$

et calcule une somme partielle :

$$v_i^{(r)} = \sum_{j=rN_{\text{loc}}}^{(r+1)N_{\text{loc}}-1} A_{ij} u_j.$$

Le vecteur final est obtenu par :

$$v = \sum_{r=0}^{\texttt{nbp}-1} v^{(r)},$$

ce qui est réalisé par une opération `Allreduce`, de sorte que toutes les tâches possèdent v à la fin.

c) Speed-up.: Avec les temps mesurés :

$$T_1 = 2.13 \times 10^{-4}, \quad T_2 = 9.63 \times 10^{-5},$$

$$T_4 = 1.91 \times 10^{-4}, \quad T_8 = 1.53 \times 10^{-2},$$

on obtient :

$$S(2) = \frac{T_1}{T_2} \approx 2.22, \quad S(4) = \frac{T_1}{T_4} \approx 1.12,$$

$$S(8) = \frac{T_1}{T_8} \approx 1.4 \times 10^{-2}.$$

Produit parallèle matrice - vecteur par lignes

d) Valeur de N_{loc}.: Avec un découpage par blocs de lignes et N divisible par `nbp` :

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

Pour $N = 120$:

$$\texttt{nbp} = 1 \Rightarrow N_{\text{loc}} = 120, \quad \texttt{nbp} = 2 \Rightarrow N_{\text{loc}} = 60,$$

$$\texttt{nbp} = 4 \Rightarrow N_{\text{loc}} = 30, \quad \texttt{nbp} = 8 \Rightarrow N_{\text{loc}} = 15.$$

e) Parallélisation par lignes.: Chaque tâche r construit uniquement les lignes

$$i \in [rN_{\text{loc}}, (r+1)N_{\text{loc}}[$$

et calcule un sous-vecteur :

$$v^{(r)} = A^{(r)}u.$$

Le vecteur final est obtenu par concaténation des sous-vecteurs locaux à l'aide d'une opération `Allgather`, de sorte que toutes les tâches possèdent le vecteur v complet.

f) Speed-up.: Avec les temps mesurés :

$$T_1 = 3.08 \times 10^{-5}, \quad T_2 = 4.17 \times 10^{-5},$$
$$T_4 = 6.79 \times 10^{-5}, \quad T_8 = 1.23 \times 10^{-4},$$

le speed-up est :

$$S(2) = \frac{T_1}{T_2} \approx 0.74, \quad S(4) = \frac{T_1}{T_4} \approx 0.45,$$
$$S(8) = \frac{T_1}{T_8} \approx 0.25.$$

III. Entraînement pour l'examen écrit

On suppose que la partie parallélisable du programme représente $p = 90\%$ du temps d'exécution séquentiel.

a) Accélération maximale (loi d'Amdahl).: La loi d'Amdahl donne :

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

Pour $n \gg 1$:

$$S_{\max} = \frac{1}{1-p} = \frac{1}{0.1} = 10.$$

L'accélération maximale théorique est donc de 10.

b) Nombre raisonnable de nœuds de calcul.: Au-delà de quelques nœuds, le gain devient faible car la partie séquentielle (10%) limite fortement l'accélération. Il est donc raisonnable de ne pas dépasser un nombre de nœuds de l'ordre de 10, afin d'éviter un gaspillage de ressources CPU.

c) Accélération observée.: Alice observe expérimentalement une accélération maximale de 4 pour ce jeu de données, ce qui est inférieur à la borne théorique donnée par la loi d'Amdahl. Cela s'explique par les surcoûts de communication et de synchronisation.

d) Loi de Gustafson (données doublées).: Si la taille des données est doublée et si la complexité de la partie parallèle est linéaire, la loi de Gustafson donne :

$$S_G(n) = n - (1-p)(n-1).$$

Avec $p = 0.9$ et une accélération observée maximale $n = 4$:

$$S_G(4) = 4 - 0.1 \times (4-1) = 4 - 0.3 = 3.7.$$

Alice peut donc espérer une accélération d'environ 3.7 avec la loi de Gustafson.