Parallélisme

—

TP5

Ce travail pratique est évalué, le rendu doit se faire au plus tard le dimanche 13 Janvier 2019 sur la plateforme moodle → cours Parallélisme (13X007) → TP5.

Répartition de charge

Dans ce TP nous étudierons des stratégies de répartition de charge (appelé aussi équilibrage de charge ou *load balancing* en anglais). La répartition de charge est une technique utilisée en informatique pour distribuer un travail entre plusieurs ressources (dans notre cas il s'agit de coeurs de calcul) afin d'obtenir une utilisation optimale de ces ressources. Votre travail consistera à calculer en parallèle l'ensemble de Julia avec différentes techniques de répartition de charge. Comme l'algorithme est complètement local (c.-à-d. sans communications), le problème de répartition de charges se réduit à un problème de partitionnement de l'espace de calcul.

Ensemble de Julia

L'ensemble de Julia est une fractale définie comme l'ensemble des points z du plan complexe $\mathbb{C}$ pour lesquels la suite

$$z_{n+1} = z_n^2 + c \tag{1}$$

ne diverge pas. Avec $z_0 = z$ et c choisi.

Pour effectuer le calcul, on prends un nombre d'itérations maximum fini et on considère que la suite diverge lorsque $||z_n|| > 2$. On fait ensuite corresponde le nombre d'itération pour que la suite diverge en z pour produire une image. Vous pouvez effectuer le calcul sur la portion du plan complexe $[-2, 2] \times [-2i, -2i] \in \mathbb{C}$. Ce domaine est discrétisé sur une grille dont la finesse du maillage détermine la précision du calcul.

Travail à accomplir

Vous devez implémenter les trois stratégies suivantes pour calculer l'ensemble de Julia à l'aide de threads et en MPI. Vous implémenterez les trois stratégies à l'aide de thread, et seulement les deux premières en MPI. Vos implémentations généreront des images au format PGM[1].

La parallélisation triviale. C'est la manière la plus simple de paralléliser le calcul de l'ensemble. Elle consiste à attribuer à chaque coeur de calcul une bande (horizontale ou verticale) du domaine. Si le domaine de dimensions LH est partagé entre P processeurs, chaque processeur p_i va recevoir une bande de hauteur H/P et de largeur L.

1. `https://fr.wikipedia.org/wiki/Portable_pixmap`

Répartition ou équilibrage statique. Le domaine est partagé en bandes horizontales d'un "point" de hauteur. Avec cette stratégie, chaque noeud ou processeur p_i va recevoir H/P bandes de hauteur 1 et de largeur L. Pour équilibrer la charge entre les noeuds, les bandes d'indice j tel que $j \equiv i \bmod P$ sont attribué à p_i.[2].

Répartition ou équilibrage dynamique. Dans cette stratégie, on considère que le domaine de dimension LH est découpé en $n = H/l$ bandes horizontales de hauteur l, l pouvant être égal à 1 dans le cas le plus simple. Chaque tâche vient ensuite demander une bande à calculer, effectue le calcul sur cette bande et va cherche une nouvelle bande, ceci jusqu'à ce que tout le travail soit effectué. Il faut donc écrire une fonction permettant à un *thread* de demander une portion de travail à effectuer (les information sur la bande de domaine à calculer). Cette fonction doit être protégée contre les accès concurrents.

Vous devez mesurer les performances des différentes stratégies que vous avez implémentées. Vous devez, pour chaque stratégie, mesurer le speedup avec 1, 2, 4, 8, 10, 12, 16 et 20 coeurs de calcul sur la machine *Baobab* pour les versions utilisant des *threads* et avec 1, 10, 20, 40, 60, 80, 100 et 120 coeurs de calcul sur la machine *Baobab* pour les versions MPI. Tracez les courbes de speedup obtenues et discutez vos résultats.

Vous devez rédiger un rapport contenant une courte introduction expliquant ce qui est recherché, une présentation de vos résultats (courbes de *speedup* des différentes stratégies) ainsi qu'une discussion traitant de vos résultats.

1 Consignes

- Vos implémentations doivent être réalisées en C/C++ doivent fonctionner avec un nombre arbitraire de processus / threads.
- Vous devez rendre votre code et votre rapport dans une archive au format **zip** portant votre nom d'utilisateur qui doit être structurée de la façon suivante :
  - un répertoire `src` qui contient vos fichiers sources
  - un fichier `Makefile` à la racine de l'archive
  - le rapport à la racine de l'archive
- Le fichier `Makefile` doit contenir au moins les deux cibles suivantes :
  - la cible `all` compile vos applications dans le répertoire courant et produit des exécutables nommés :
    - `julia_thread_simple` pour l'implémentation avec des threads triviale
    - `julia_thread_static` pour l'implémentation avec des threads utilisant un équilibrage de charge statique
    - `julia_thread_dynamic` pour l'implémentation avec des threads utilisant un équilibrage de charge dynamique
    - `julia_mpi_simple` pour l'implémentation avec MPI triviale
    - `julia_mpi_static` pour l'implémentation avec MPI utilisant un équilibrage de charge statique
  - la cible clean supprime les fichiers compilés

2. Par exemple pour $P = 4$ et $H = 20$ le noeud p_0 traitera les bandes $0, 4, 8, 12$ et 16. Le noeud p_1 traitera les bandes $1, 5, 9, 13$ et 17. Le noeud p_2 traitera les bandes $2, 6, ...$

— Vos applications doivent prendre les arguments suivants, dans l'ordre : point inférieur gauche, point suppérieur droit, c, nombre maximum d'itérations, taille du domaine, nom du fichier de sortie, plus le nombre de threads pour la version avec des threads. Vous pouvez prendre exemple sur le fichier `julia.cpp` sur moodle.
— L'archive produite doit être déposée sur moodle dans

Parallélisme → TP5