OS202

Rapport de projet

Clément Bastaert et Albin Joyeux

Palaiseau, France

Mars 2023

Contents

1 Séparation interface-graphique et calcul

On souhaite séparer l'interface graphique du calcul. On va réaliser cette paraléllisation à l'aide de MPI dans le main du fichier vortexSimulation.cpp.

On introduit nbp le nombre de processerurs que l'on souhaite utiliser (ici 2) ainsi que leur rang (0 et 1).

Le processeur 0 s'occupe de l'affichage, il tourne tant que le fenêtre est ouverte, c'est-à-dire tant que myScreen.isOpen() est true. Le processeur 1 calcule en parallèle tant que run calcul est true.

Pour chaque touche du clavier touchée, on envoie avec MPI Send la demande de calcul au processeur 1. A ce moment là, le processeur 1 a déjà commencé les calculs et va les envoyer au processeur 0. Cependant, MPI ne peut transmettre que des listes alors que l'on a des tableaux. On crée donc une liste dataVect dans laquelle on ajoute les éléments des tableaux grid, cloud et vortices. On les ordonne en rangeant d'abord l'élément x d'une ligne du tableux puis l'élément y. Au niveau du MPI Recv dans le processeur 0, on reconstitue ces tableaux de la même manière.

Dans la réception des envois dans le processeur qui gère le calcul, on utilise la fonction MPI IProbe, qui permet de prévenir MPI Recv que l'on a reçu quelque chose, pour ne pas être bloquant.

Pour vérifier que notre parallélisation fonctionne, on regarde les fps avant et après parallélisation. On observe une augmentation de 5% à 10% sur les différentes fichiers du répertoire "data". On peut expliquer cette faible augmentation du fait que le calcul prend beaucoup plus de temps que l'affichage, ainsi en parallélisant on n'a plus que le temps de calcul, mais qui reste élevé, d'où l'intérêt plus tard de paralléliser le calcul.

2 Parallélisation en mémoire partagée

On décide de paralléliser les calculs à l'aide d'OpenMP. On ajoute la directive MPI "#pragma omp parallel for" pour les boucles parallélisables, c'est à dire celles qui ne contiennent pas d'interdépendance.

Pour vérifier que la parallélisation fonctionne, on va comparer les fps avant et après parallélisation. Cependant, les valeurs de fps fluctuent beaucoup donc on prend la valeur moyenne. Le tableau en Figure 1 résume nos résultats. On constate une amélioration pouvant atteindre un facteur 3.

Data	cornertest	manyvortices	onevortex	simplesimulation	triplevortex
FPS moyen sans OMP	136	19	14	38	37
FPS moyen avec OMP	481	30	47	88	58

Table 1: Comparaison des FPS moyen ($temps^{-1}$) avec et sans OpenMP (OMP)

3 Parallélisation en mémoire distribuée et partagée des calculs

Nous avons distribué les particules parmis les processus ayant pout tâche le calcul afin de paralléliser le calcul du déplacement des particules à l'aide de MPI.

Data	cornertest	manyvortices	onevortex	simplesimulation	triplevortex
FPS moyen : 1p	136	19	14	38	37
FPS moyen : 4p	235	40	33	79	71
FPS moyen : 6p	251	42	37	80	75
FPS moyen : 8p	268	45	37	88	83
FPS moyen : 10p	271	49	43	89	85

Table 2: Comparaison des FPS moyen ($temps^{-1}$) pour différents nombres de processus MPI

Les FPS s'améliorent lorsqu'on augmente le nombre de processus MPI. On remarque qu'on tend vers une asymptote pour chaque data. Ceci peut s'expliquer par le temps de communication et la synchronisation des processus mais aussi par une partie séquentielle du code (notamment le calcul du champ de vitesse des vortex fait par le rang 0).

4 Réflexions sur l'approche mixte Eulérienne–Lagrangienne

Le problème principal qui semble se poser est la cohérence des données. La parallélisation par sous-domaine nécessite des échanges de données entre les processus voisins pour garantir la cohérence des résultats. Cependant, lors de la parallélisation lagrangienne, les particules peuvent traverser les frontières des sous-domaines. Il est donc nécessaire de synchroniser les échanges de données entre les processus et les déplacements des particules pour éviter des incohérences.

Ainsi, il y a la mise en place d'une communication inter-processus. Cependant, ces échanges de données peuvent entrainer des coûts élevés et ralentir les performances. Il est donc nécessaire de minimiser le nombre d'échanges.

Pour un maillage très grand, les temps de communication peuvent devenir très importants et donc devenir un facteur plus important que le temps de calcul, ce qui limmiterait les performances.

Dans le cas d'un très grand nombre de particules, il peut y avoir des limites de stockage des données calculées pour chaque particule.

Enfin, dans le cas d'un maillage de très grande taille et un très grand nombre de particules, les deux problèmes peuvent se cumuler.