Projet OS202

Billy WEYNANS - Quentin LINCOLN-RABOUIN

Configuration de la machine utilisée pour ce projet

CPU

Processeur(s) : 8
Thread(s) par cœur : 2
Nom de modèle : Intel(R) Core(TM) i5-9300H CPU @ 2.40GHz
Vitesse du processeur en MHz : 2400.000
Cache L1d : 128 KiB
Cache L1i : 128 KiB
Cache L2 : 1 MiB
Cache L3 : 8 MiB

RAM

8Go

Toutes les exécutions ont été lancées avec une grille de taille 1280x1024.

Q1. Séparation interface-graphique et calcul

La séparation interface-graphique et calcul se fait simplement en définissant deux processus MPI :

- Le processus 0 qui gère l'interface et la partie graphique
- Le processus 1 qui réalise les calculs

Après un profiling des configurations initiale et post-parallélisation, on obtient les résultats suivants :

Dans le cas séquentiel ; FPS = 32

	Temps affichage	Temps calcul	Total
Temps en secondes	0,002377	0,0283963	0,0307733
Part dans le temps total	7,72%	92,28%	100,00%

Temps d'affichage et de calcul avant séparation interface-graphique et calcul

Dans le cas MPI ; FPS = 35

Processus d'affichage (0)	**Temps affichage**	**Temps communication**	**Total**
Temps en secondes	0,002983	0,006236	0,009219
Part dans le temps total	32,36%	67,64%	100,00%
Processus de calcul (1)	**Temps calcul**	**Temps communication**	**Total**
Temps en secondes	0,029313	0,000005	0,029318
Part dans le temps total	99,98%	0,02%	100,00%

Temps d'affichage et de calcul après séparation interface-graphique et calcul

Comme on s'y attend, les temps d'affichage et de calcul sont quasi-identiques.

On remarque quand même une amélioration de 3 FPS entre les deux configurations. Cela est possible car le temps de communication dans les deux processus est inférieur au temps de calcul de la configuration initiale.

Q2. Parallélisation en mémoire partagée

N.B : Nous n'avons malheureusement pas pu essayer de lancer les calculs en utilisant deux ordinateurs.

Après inspection du code, on remarque que les fonctions principales de calcul sont :

- Numeric::solve_RK4_fixed_vortices
  - Est appelée lorsque les vortex sont immobiles
  - Comporte une boucle for
- Numeric::solve_RK4_movable_vortices
  - Est appelée lorsque les vortex sont mobiles
  - Comporte 3 boucles for

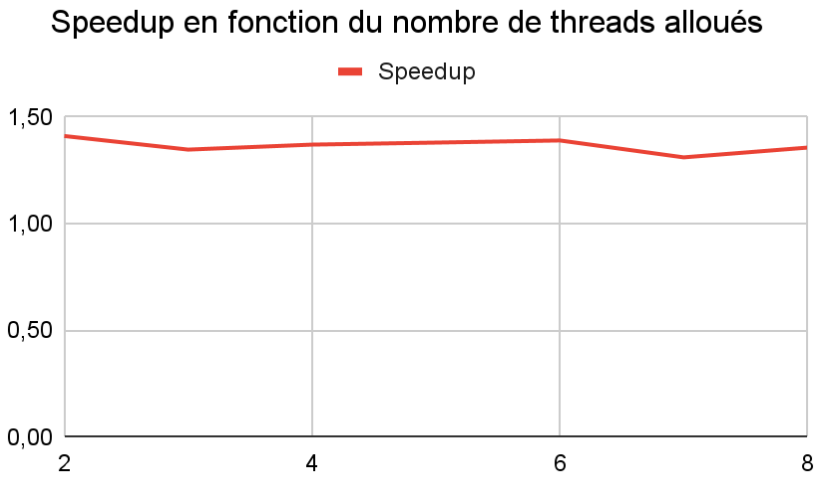
On commence simplement par paralléliser la boucle for de la fonction **Numeric::solve_RK4_fixed_vortices** dans le fichier **runge_kutta.cpp** dont boucle for est assez conséquente. À l'intérieur de cette boucle for, il n'y a pas de fonction appelée ayant des structures parallélisables (du moins simplement) par OpenMP.

On applique donc la directive "**#pragma omp parallel for**" car elle ne nécessite pas de gestion particulière.

On obtient les résultats suivants :

Parallélisation de la boucle 1 de **solve_RK4_movable_vortices** avec onevortexsimulmation.dat			
OMP_NUM_THREADS	**Temps d'exécution de la boucle 1 (s)**	**Speedup boucle 1**	**FPS**
séquentiel	0,026185	XX	32
1	0,027486	0,95	33
2	0,019543	1,34	42
3	0,021586	1,21	41
4	0,019477	1,34	43
5	0,020519	1,28	42
6	0,021173	1,24	41
7	0,02011	1,30	42
8	0,019735	1,33	42

Temps d'exécution de la boucle 1 après parallélisation OpenMP en fonction de OMP_NUM_THREADS



On remarque que le speedup n'évolue que peu à partir de 2 threads, on utilisera donc OMP_NUM_THREADS=2 par la suite.

Après avoir parallélisé la fonction responsable des vortex fixes, il faut maintenant que nous traitions la version avec vortex mobiles: **Numeric::solve_RK4_movable_vortices** dans le même fichier.

Cette fonction comporte principalement trois boucles for dont voici le profiling.

Temps d'exécution de la boucle 1 (s)	Pourcentage du temps total boucle 1	Temps d'exécution de la boucle 2 (s)	Pourcentage du temps total boucle 2	Temps d'exécution de la boucle 3 (s)	Pourcentage du temps total boucle 3
0,026185	99,996%	0,000001	0,004%	0	0,000%

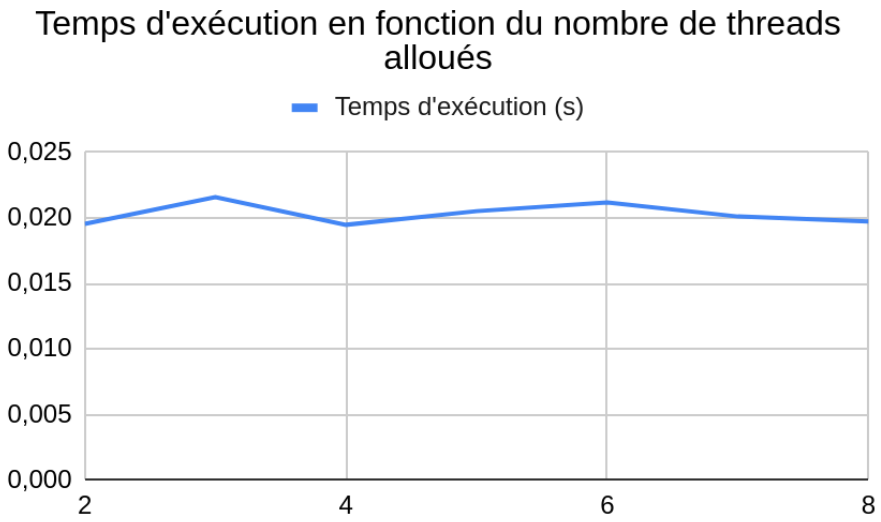
Temps d'exécution des différents boucles en séquentiel

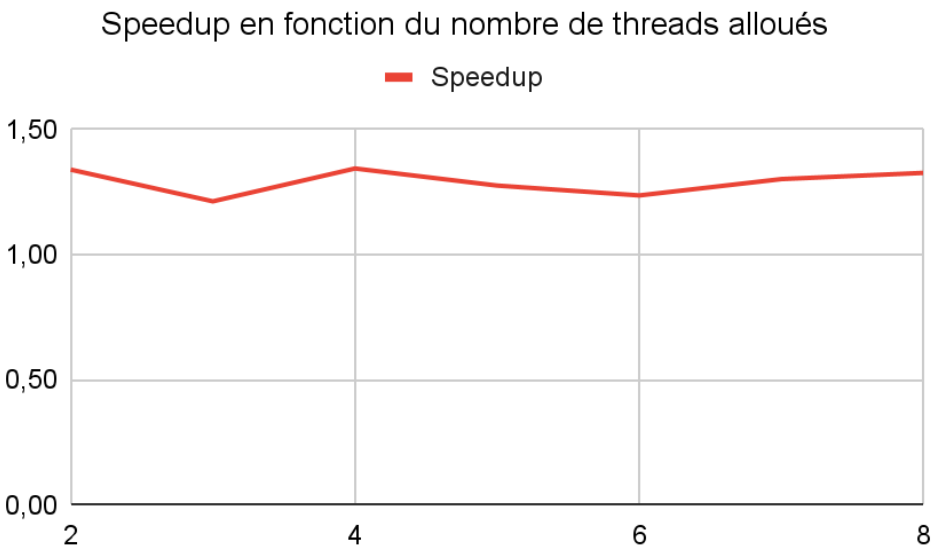
On remarque que la première boucle prend l'écrasante majorité du temps. Ainsi il n'est pas utile de chercher à améliorer les temps d'exécution des deux autres boucles pour le moment.

Après parallélisation de cette boucle, on obtient :

Parallélisation de la boucle 1 de **solve_RK4_movable_vortices** avec onevortexsimulmation.dat		
OMP_NUM_THREADS	**Temps d'exécution de la boucle 1 (s)**	**Speedup boucle 1**
séquentiel	0,026185	XX
1	0,027486	0,95
2	0,019543	1,34
3	0,021586	1,21
4	0,019477	1,34
5	0,020519	1,28
6	0,021173	1,24
7	0,02011	1,30
8	0,019735	1,33

Temps d'exécution de la boucle 1 après parallélisation OpenMP en fonction de OMP_NUM_THREADS





Même après parallélisation et en prenant la meilleure moyenne de temps, qui est de 0.019 s (speedup de 1.35), ce temps représente plus de 99% du temps d'exécution de la fonction. En effet les deux autres boucles ont des temps d'exécution de l'ordre de la microseconde ce qui est négligeable en comparaison avec la boucle 1 qui est de l'ordre de la milliseconde.

On en conclut que l'amélioration des performances de la simulation par OpenMP ne se fera pas sur les boucles 2 et 3 ni sur les fonctions qu'elles contiennent :

- **Vortices::computeSpeed**
- **CartesianGridOfSpeed::updateVelocityField**

qui n'ont donc pas grand intérêt à être parallélisées.

Q3. Parallélisation en mémoire distribuée et partagée des calculs

Afin de faciliter le codage des communications et la clarté du code, un communicateur propre aux processus de calcul a été créé.

Ainsi, on a :

- Le processus 0 qui gère l'affichage et les entrées clavier. Il communique uniquement avec 1 et est le seul qui n'est pas dans le communicateur de calcul
- Le processus 1 qui communique avec 0 et les autres processus de calcul
- Les processus > 1 qui attendent les instructions de 1 et calculent lorsque nécessaire

On suppose que toutes les machines exécutant des processus ont accès au fichier de configuration, ce qui évite un envoi d'initialisation.

Après répartition entre les processus de calcul et plusieurs exécutions avec OMP_NUM_THREADS = 2 (car meilleur compromis d'après la Q2), on obtient :

- Pour le processus d'affichage

Nombre de processus MPI	FPS	Temps d'affichage (s)	% Temps d'affichage	Temps de communication	% Temps de communication	Temps total itération (s)
2	23	0,002784	6,73%	0,038579	93,27%	0,041363
3	43	0,002718	12,17%	0,019608	87,83%	0,022326
4	53	0,002964	16,22%	0,015311	83,78%	0,018275

Temps d'affichage et de communication du processus d'affichage après parallélisation en mémoire distribuée et partagée des calculs

Le temps d'affichage reste constant comme on pourrait attendre car le processus d'affichage n'a pas été modifié.

Cependant, le % du temps total passé en communication - dont attente de communication - baisse au fur et à mesure que le nombre de processus de calcul augmente. C'est logique car les informations sont traitées plus rapidement et arrivent donc plus rapidement également.

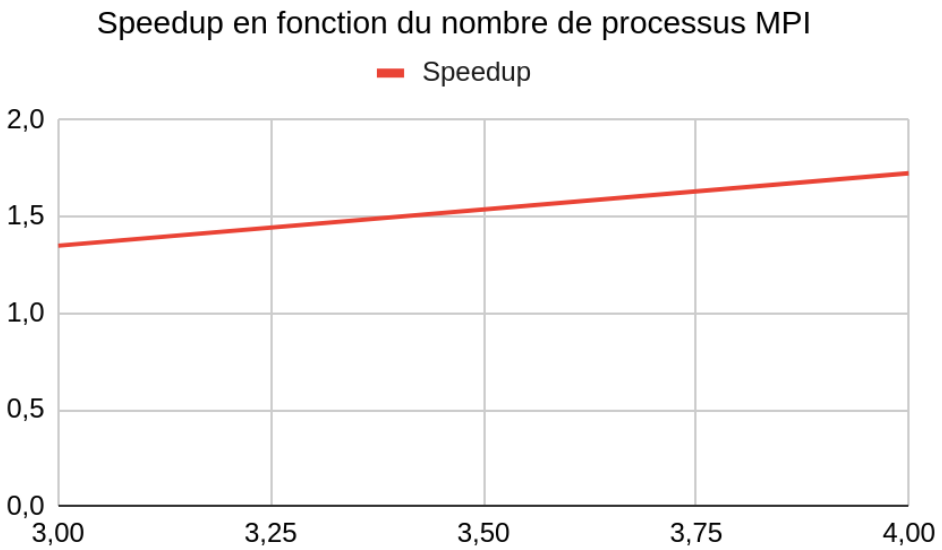
- Pour les processus de calcul

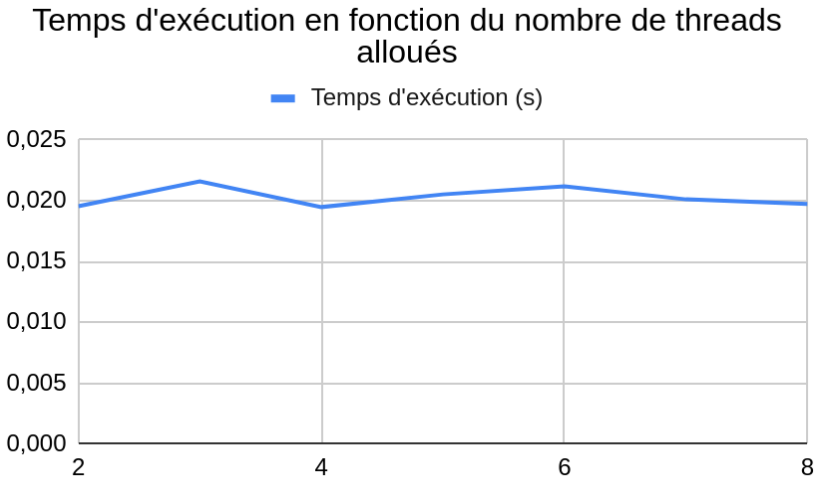
Nombre de processus MPI	Max Temps de calcul moyen parmi les processus de calcul	% Max Temps de calcul	Speedup calcul	Temps de communication	% Temps de communication	Temps total itération (s)
2	0,040631	98,65%	0,7278924959	0,000557	1,35%	0,041188
3	0,021907	79,77%	1,350025106	0,0055541	20,23%	0,0274611
4	0,017142	91,69%	1,725294598	0,001553	8,31%	0,018695

Temps d'affichage et de communication des processus de calcul après parallélisation en mémoire distribuée et partagée des calculs

On notera que la ligne avec deux processus MPI est dégénérée et ne représente pas fidèlement le comportement du programme car il n'est pas fait pour avoir deux processus. Aussi, la machine sur laquelle le programme a été exécuté ne permettait pas de mettre cinq processus MPI. En essayant avec l'option --oversubscribe de mpirun, les performances ont largement été réduites par rapport au cas avec quatre processus MPI, c'est pourquoi cette configuration n'a pas été retenue dans les résultats.

Le speedup a été calculé relativement à une valeur moyenne obtenue du temps de calcul d'une itération avec la parallélisation OpenMP de la Q2 et OMP_NUM_THREADS=2.





Enfin, on peut remarquer que le temps de calcul prend le pas sur les temps de communication au fur et à mesure que le nombre de processus MPI augmente. Cette tendance est sûrement modifiée à partir d'un certain nombre de processus MPI qui entraînerait des problèmes de communication.

Finalement, on aura réussi à atteindre 53 fps sur une machine alors que la version séquentielle tournait à 32 fps sur cette même machine.

Q4. Réflexions sur l'approche mixte Eulérienne-Lagrangienne

L'approche Eulérienne consiste à distribuer à chaque processus une portion du domaine que l'on aurait précédemment partitionné.

L'approche Lagrangienne se focalise sur la parallélisation du déplacement des particules. Bien que l'on ait un avantage sur le suivi et les informations disponibles individuellement pour chaque particule, il est naturel qu'à grande échelle cela ralentit considérablement les calculs.

Maintenant, si l'on souhaitait combiner les deux, cela reviendrait à partitionner le domaine de calcul et à le distribuer parmi les threads dans un premier temps.
Deuxièmement, il faudrait suivre les particules individuellement en les distribuant entre les différents processus.

- Dans le cas d'un partitionnement à grande échelle:
  Un premier obstacle est rapidement identifiable. En effet, ces nombreuses partitions impliquent un nombre important de communications. Cela est problématique car les communications ont un coût en temps constant, le temps d'exécution sera donc fonction du nombre d'opérations de parallélisation. Cette méthode n'est donc pas efficace d'un point de vue du runtime. Il faut aussi considérer qu'à très grande échelle les communications risquent de saturer le réseau et entraîner des blocages et/ou des erreurs de transmission.
  On doit également prendre en compte que puisque l'approche Lagrangienne suit la trajectoire des particules individuellement, en augmentant le maillage, on augmente le nombre de coordonnées possibles pour chaque particule. Cela alourdit considérablement les calculs.

- Dans le cas d'un nombre très important de particules:
  L'approche Lagrangienne alloue beaucoup de ressources (en comparaison à Euler) à chaque particule, le coût de stockage y est par conséquent très élevé. Il semble alors naturel qu'en augmentant le nombre de particules, le suivi individuel des particules alourdit le programme. Il y a également le problème de la répartition des particules entre les processeurs qui devient plus coûteux lorsque le nombre de ces dernières augmente considérablement.

- Si le maillage est effectué à grande échelle et que l'on a un nombre important de particules, les problèmes soulignés dans les cas précédents seront combinés. Il faudra trouver une méthode plus efficace pour distribuer les tâches. De plus, la mise en place de la parallélisation deviendra très fastidieuse et le risque d'erreur deviendra très important en plus de nuire à la lisibilité du code.

FIN