ÉCOLE NATIONALE SUPÉRIEURE DE TECHNIQUES AVANCÉES

OS202

Projet ENSTA : Modélisation de tourbillons sur un tore en deux dimensions

Réalisé par :

Gustavo Jodar Soares

Vanessa López Noreña

03/2023

Configurations d' ordinateur qui a fait les tests

Arquitetura: x86_64

Modo(s) operacional da CPU: 32-bit, 64-bit

Address sizes: 39 bits physical, 48 bits virtual

Ordem dos bytes: Little Endian

CPU(s): 8

Lista de CPU(s) on-line: 0-7

ID de fornecedor: GenuineIntel

- Nome do modelo: Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz
- Família da CPU: 6
- Modelo: 142
- Thread(s) per núcleo: 2
- Núcleo(s) por soquete: 4
- Soquete(s): 1
- Step: 10
- CPU MHz máx.: 3400,0000
- CPU MHz mín.: 400,0000
- BogoMIPS: 3600.00
- Opções: fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov

Division du Projet:

Le projet est divisé en 3 fichier:

- Projet_ENSTA2023_Q1 , le code qui comprend juste la spécifique à la partie 1 de parcellisation du projet
- Projet_ENSTA2023_Q2 , le code qui comprend juste la spécifique à la partie 2 de parcellisation du projet
- Projet_ENSTA2023_Q3 , le code qui comprend juste la spécifique à la partie 3 de parcellisation du projet

Pour compiler les code il faut entrer dans ces fichier et utiliser la commande << **$make all >>** . Pour exécuter les codes (on peut changer les archives des données):

- Q1 = **$mpirun -n 2 ./vortexSimulation.exe data/<**l'archive .dat de votre préférence> **1280 1024**

- Q2 = **$mpirun -n 2 ./vortexSimulation.exe data/<**l'archive .dat de votre préférence> **1280 1024**
- Q3 = **$mpirun -n <** num_process >=2 **> ./vortexSimulation.exe data/<**l'archive .dat de votre préférence> **1280 1024**

1 - Séparation interface-graphique et calcul

Dans la parallélisation du code par rapport à la séparation interface-graphique (processus 0) et calcul (processus 1), les opérations suivantes sont mises en œuvre : On ajoute la librairie MPI et on initialise le communicateur. Ensuite, on procède à la parallélisation des processus.

Processus 0 :

Pendant que l'interface graphique est ouvert, le processus 0 est chargé de l'affichage à l'écran avec Graphisme::Screen et la gestion des évènements liés au clavier/fermeture de fenêtre, cette action est effectué lorsque vous souhaitez saisir une action de commande telle que l'arrêt de l'animation, l'augmentation de la vitesse, la fermeture de la fenêtre, etc. Il effectue cette inspection de l'écran pendant toute la durée du processus 0, tant que la commande n'est pas X (terminer ce processus) ou l'utilisateur ferme la fenêtre.
Pour la gestion des évènements, on fait juste le changement de mot clé pour chaque commande et, par la suite, envoyé au processus 1, ce qui effectue les calculs.

Enfin, il vérifie s' il y a des messages du processus 1, si oui chaque type message a un tag spécifique le tag=1 -> cloud, tag=2->grid et tag=3->vortices, il reçoit ses messages et les mets dans les bonnes variables, ensuite les affiche. Pour terminer, il faut un action de fermeture, après p0 envoi key=T à p1, qui reçoit, envoi k=x au p0 et termine, seulement après recevoir key= "X", le p0 ferme l'interface graphique et met fin au processus, pour éviter l'interblocage.

Processus 1:

Le processus 1 vérifie s'il y a des messages à recevoir de manière interbloquant (MPI_Iprobe), en fonction du mot-clé reçu, le processus change les valeurs des variables, e.g. augmenter la vitesse (dt *= 2), et enfin faire le calcul et envoyer les valeurs au processus d'affichage (processus 0).

Speed Up (FPS - Frames Per Second):

Data Used	Non-Paralized (teacher's code)	Paralized	Speed Up
onevortexsimulation.dat	10.3	12	16.5%
simpleSimulation.dat	27.3	31	13.55%
triplevortex.dat	25.4	30	18.11%

2 - Parallélisation en mémoire partagée

Dans cette partie du projet on a vérifié le temps que chaque boucle prend dans le calcul pour décider lesquelles appliquer la parallélisation par OpenMP.

Pour le calcul de vortices movables (function solve_RK4_mouvable_vortices - boucles 1, 2 et 3) et updateVelocityField (boucle 4) on a utilisé les données de simpleSimulation.dat.

Pour le calcul de vortices fixes, fonction solve_RK4_fixed_vortices (boucle 1) on a utilisé onevortexsimulation.dat.

Le reste du code reste le même que le code développé pour la partie 1 du projet.

solve_RK4_mouvable_vortices			solve_RK4_fixed_vortices		
	without OpenMP	with OpenMP		without OpenMP	with OpenMP
first for	0.0440365	0.025942	first for	0.101245	0.0712086
second for	1.87e-06	non parallélisable	-	-	-
third for	7.1e-08	non parallélisable	-	-	-
forth for	0.00389872	0.00377572	-	-	-

Table 1: Comparaison entre l'usage ou non du openMP dans les boucles du code.

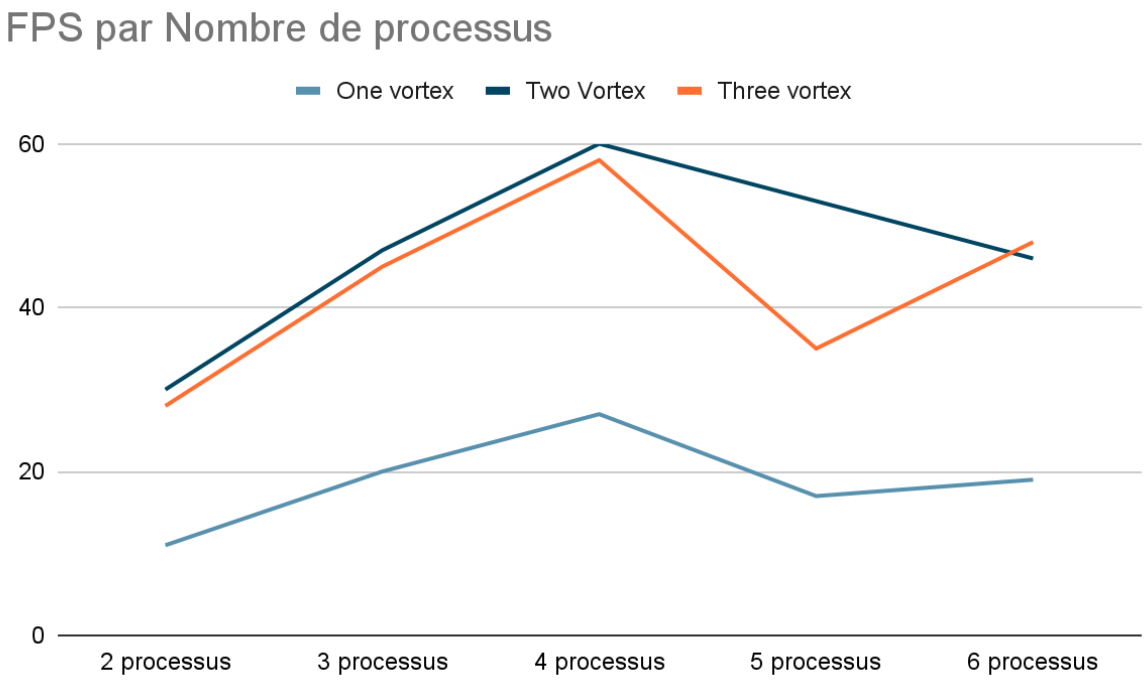
Speed Up (FPS - Frames Per Second):

Data Used	Non-Paralyzed (teacher's code)	Paralyzed affichage/calcul + OpenMP (first boucle)	Speed Up
onevortexsimulation.dat	10.3	16	55.33%
simpleSimulation.dat	27.3	42	13.55%
triplevortex.dat	25.4	40	57.48%

3-Parallélisation en mémoire distribuée et partagée des calculs

Pour le programme de la partie 3 du projet, il faut paralléliser le calcul entre plus qu'un processus, alors on a divisé la matrice CloudOfPoints entre les processus qui s'occupent de faire ce calcul. Cette tâche est plus ou moins simple, une fois que cette matrice garde la position (x,y) oú se retrouve une particule. De cette manière, il n'y a pas de conflits de calcul entre les processus.
La manière qu'on a décidé de le faire est la suivante:

- Chaque processus responsable pour le calcul fait une allocation de mémoire pour garder une partie de CloudOfPoints respective à son numéro de processus.
- Le processus 0 (encore responsable pour l'affichage) demande du calcul aux processus de calcul.
- Après avoir reçu la demande, chaque processus fait le calcul qui lui a été désigné de la partie CloudOfPoints, les variables de vortices et grid non pas été parallélisées, alors chaque processus a une copie de ses deux variables. Ensuite, avec une Gatherv, ses données sont envoyées au processus 0 pour l'affichage et que le processus 1 envoie les données du grid et vortices (ils sont les mêmes dans tous les processus).
- Dans le Gatherv du processus 0 les données de CloudOfPoints sont groupées et les données des vortices et grid sont reçues normalement par un Recv. Les processus 0 affiche et peuvent envoyer une autre demande de calcul ou des ordres de clavier/fermeture de fenêtre.



Graphique 1: FPS par nombre des processus, c'est possible d' observer une amélioration de la performance jusqu'à 4 processus (nombre de cours physiques de la machine), plus que 4 le programme n'est pas trop performant.

Speed Up (FPS - Frames Per Second):

Data Used	Non-Paralyzed (teacher's code)	Paralyzed par 4 processus	Speed Up
onevortexsimulation.dat	10.3	27	162.13%
simpleSimulation.dat	27.3	60	119.68%
triplevortex.dat	25.4	58	128.34%

4-Réflexions sur l'approche mixte Eulérienne--Lagrangienne

Un approche par sous-domaine pourrait être faite par le calcul d'une moyenne de mouvements des particules proches, en utilisant un nombre fixe des particules par région. Si on fait comme ça on peut traiter un cas où le nombre des particules est trop grand. Cette approche fonctionnerait car les particules qui sont proches entre elles sont plus probables à avoir le même comportement. Ensuite, il faut envoyer le comportement du sous-domaine au processus 0, qui affiche le même mouvement pour toutes les particules du même sous-domaine. Il pourrait être fait ou par division équitable des sous-domaines ou par attribution par un maître esclave.

Que se passe-t-il dans le cas d'un maillage de très grande dimension avec la ou les solutions que vous proposez ?

Par l'approche proposée, l'usage d'un maillage trop grand laisserait le calcul lent si les sous-domaines sont petits car on aurait un nombre trop grand de sous-domaines à calculer. Une manière de contourner ce problème serait de trouver l'équilibre entre un sous-domaine plus grand et le vrai comportement des particules.

Que se passe-t-il dans le cas d'un très grand nombre de particules ?

Dans le cas où le nombre de particules est trop grand le programme aurait une bonne performance, car il prendrait en compte un nombre fixe des particules par sous-domaine.

Et dans le cas d'un maillage de très grande taille ET un très grand nombre de particules ?

Dans ce cas, la réponse serait la même que la première question.