Sigal Nicolas

Rapport de projet d'IN203 : Programmation parallèle

Sujet : expansion au sein d'une galaxie

L'objectif de ce rapport est de résumer de manière exhaustive les différentes améliorations et tentatives d'amélioration de code que j'ai effectuées à partir des fichiers sources obtenus sur GitHub. Il abordera lesdites améliorations de manière chronologique, ce qui correspond aux avancées proposées dans l'aide du sujet : dans une première partie, je détaillerai donc les améliorations en mémoire partagée obtenues grâce à OpenMP, et détaillerai les problèmes de code, compilation, et exécution que cette librairie a entraîné, avant de faire de même pour la librairie MPI, qui améliore le code en parallélisant en mémoire distribuée. Enfin, la dernière partie portera sur les problèmes rencontrés quant à la mise en place d'une parallélisation double (à la fois en mémoire partagée et en mémoire distribuée) sur Mac OS, ainsi que les divers problèmes d'affichage ou de calculs pour paralléliser (en MPI notamment).

I) Parallélisation en mémoire partagée : les avantages d'OpenMP.

L'utilisation de la librairie OpenMP permet de choisir directement dans le shell le nombre de threads que l'on souhaite attribuer à la tâche de calcul. Pour ce faire, une fois la librairie importée, j'ai commencé par mettre en place un espace pragma parallel dans lequel je souhaitais faire travailler séparément mes threads sur des boucles de calcul. J'ai donc dans un premier temps utilisé des variables entières de type private (et donc propres à chaque thread). Ceci a eu pour effet d'accélérer (de manière peu significative cependant) mes calculs, et j'ai ensuite effectué une comparaison avec le programme « basique », issu du repository GitHub qui nous était donné.

Pour pouvoir observer un changement de comportement, j'ai alors implémenté une nouvelle méthode de probabilité, en utilisant une graine de génération aléatoire (qui dépendait notamment du thread), afin d'observer s'il y avait une différence. Le graphique suivant présente le résultat, à savoir les temps de calculs obtenus (une simple commande entrée permettait de figer la ligne du terminal, donnant alors les valeurs de temps de calcul, trop lourdes à stocker pour moyenner).

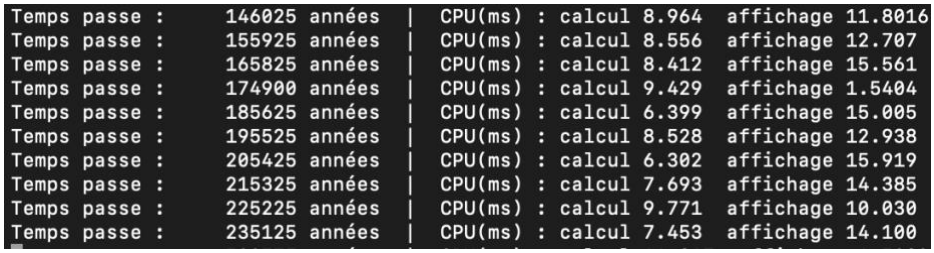

Fig. 1 : Affichage de différentes valeurs au terminal

On suppose pour le « temps idéal » que le processus accélère de manière linéaire par rapport au nombre de threads. La conclusion est que cette idée est illusoire, et que rajouter un nombre même grand de threads (non indiqué sur le graphe) n'apporte que relativement peu. Le second graphe indique le « speed up », quotient du temps initial divisé par le temps obtenu. En orange on voit les courbes réelles, et en bleu et gris, les projections d'un algorithme qui serait linéaire par rapport au nombre de threads.

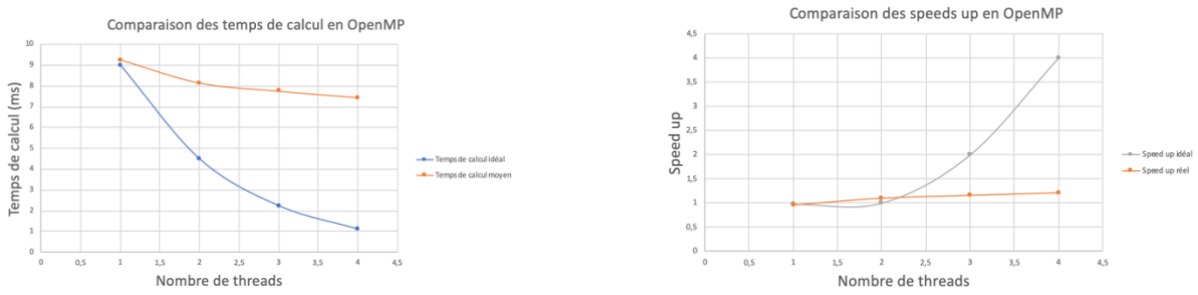

Fig.2 : Temps de calcul et speed up par OpenMP.
(Les temps de calcul sont ici la moyenne de dix temps mesurés, pour différents nombres de threads).

II) Parallélisation en mémoire distribuée : les avantages de MPI.

Dans un deuxième temps, j'ai utilisé la librairie MPI afin de paralléliser le calcul en mémoire distribuée. Pour ce faire, après une modification du Make_osx.inc afin de changer de CC et de CCX (on utilise ici mpic++), on effectue un découpage de la galaxie en sous-galaxies selon des lignes de longueur width, auxquelles on rajoute (en fonction de si elles sont au bord au nom) entre une et deux lignes de planètes dites fantômes.

Cela permet d'assurer une continuité du calcul de propagation : en effet, sans elles, les planètes de la dernière ligne (resp. première suivant la sous-galaxie considérée) ne sont sujettes qu'à une propagation des planètes du dessus (resp. du dessous), et ne se propagent pas en-dessous (resp. au-dessus) ! C'est donc un gros problème de continuité du problème initial, car une planète a la possibilité de s'étendre dans 4 directions, et non seulement 3 ! Une fois le calcul effectué à l'aide d'une galaxie copiée et actualisée, on remet en place la partie centrale de la bande, et on échange les informations des cellules fantômes pour poursuivre le résultat.

Une erreur de compréhension pouvait entraîner un calcul double, d'où l'apparition de lignes de planètes aux frontières du découpage (non traité ici car seulement évoqué avec des camarades, ce n'est pas un problème que j'ai rencontré).

Le résultat obtenu est surprenant en fonction du nombre de processus demandés à MPI : en effet, on obtient jusqu'à un speed up sept fois plus important que le programme basique !

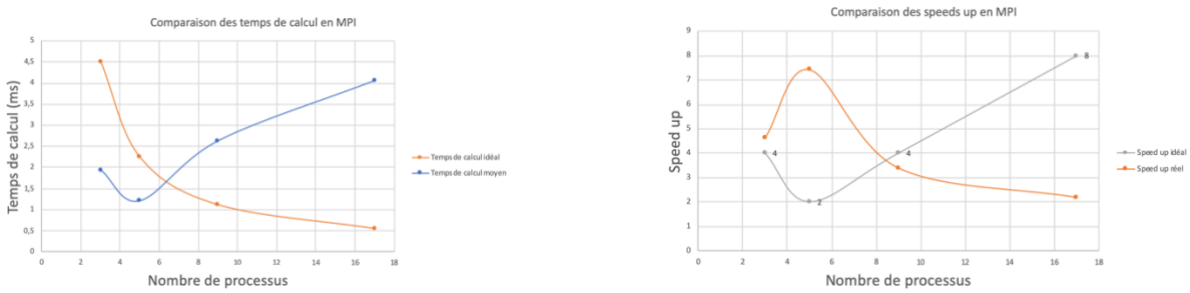

Fig.3 : Temps de calcul et speed up par MPI.
(Les temps de calcul sont ici la moyenne de dix temps mesurés, pour différents nombres de threads).

Ici on peut voir sur la courbe que le maximum d'efficacité est obtenu pour 5 processus, i.e. quatre processus de calcul et un d'affichage. Les valeurs choisies sont 3, 5, 9, 17, (soit 2, 4, 8, 16 processus de calcul) car les autres valeurs ne permettaient pas de diviser la fenêtre (la hauteur de l'image, fixe, n'étant pas multiple de trois par exemple). On utilisera notamment « -oversubscribe » afin de dépasser les limites physiques de l'ordinateur, jusqu'à un certain point : à partir de 5, on observe une décroissance rapide du speed up, i.e. du gain de vitesse.

L'utilisation d'un découpage cartésien n'a pas pu être faite, car j'essayais ensuite (sans grande réussite) d'utiliser les deux procédés vus précédemment de manière simultanée : OpenMP et MPI.

III) Divers écueils et problèmes de compilation, code, affichage lors de la parallélisation double.

Le principal écueil auquel je me suis heurté est … ma configuration d'ordinateur portable : travaillant sur Mac OS, j'ai dû faire face à des erreurs d'importation de librairies. En effet, la librairie « omp.h » n'est plus disponible dès lors que le compilateur n'utilise plus g++ mais mpic++, et ce, même en précisant « -fopenmp », « -qopenmp », et bien d'autres solutions proposées en ligne, (telles que passer par brew afin de l'importer localement) ne permettent même pas de faire tourner un simple HelloWorld doublement parallèle. J'ai donc décidé de m'attaquer aux problèmes de code, qui, une fois résolus, m'ont permis de faire des essais sur l'affichage de l'image (changer sa taille n'était cependant pas une bonne idée, car j'ai failli perdre une partie du programme en modifiant le code sans être revenu à la taille initiale, ce qui générait des erreurs que je ne comprenais plus).

L'objectif de lancer le programme sur ma machine virtuelle n'a été que peu concluant, car non habitué à cet environnement et au compilations associées, ajouté au fait qu'elle est assez lente, ne permettait pas l'obtention de résultat. Je me suis donc contenter d'écrire dans quatre fichiers différents (Basique, OMP, MPI, et Final), les différents Makefile correspondant à des compilations séparées, afin de faciliter l'accès depuis un terminal. Les dossiers, autonomes, par conséquent, permettent de faire tourner respectivement le programme initial, trouvé sur GitHub, l'amélioration OpenMP, et celle MPI. Le quatrième dossier contient ce que mes essais de parallélisation double ont permis d'obtenir, mais ne compile pas (« fatal error : <omp.h> file not found »…).

Une amélioration du code consisterait à résoudre ce problème afin d'observer les temps moyens et les speed up suivant un tableau double entrée : nombre de threads x nombre de processus. Les limitations physiques de la machine risqueraient cependant d'entraîner des diminutions rapides dès lors que les nombres deviendraient trop grands.

Au cours de ce projet, j'ai été confronté à divers problèmes : segmentation fault, procédés de data race résolus par des séparations entre threads et données privées, qui ont permis de mener plus loin ma réflexion sur la programmation parallèle. Dans la partie MPI, en mémoire partagée, il n'y a pas de conflit mémoire, ni d'accès aux données ni de partage de ressources, par définition. Pour conclure, je vous indique en annexe la manière de suivre l'arborescence de mon repository, afin de pouvoir suivre le chemin qu'a été le mien tout au long de mes essais concluants et de mes tentatives, qui aboutissent finalement à de belles choses quant au peuplement de notre galaxie.

IV) ANNEXE

Organisation de l'arborescence de mon repository :

Basique : Issu directement du repository https://github.com/JuvignyEnsta/Promotion_2022
OMP : contient les fichiers nécessaires à la compilation de « colonisation_OMP.exe », travail de code OpenMP uniquement.
MPI : contient les fichiers nécessaires à la compilation de « colonisation_MPI.exe », travail de code MPI uniquement
Final : essais de parallélisation double, ne compile malheureusement pas sur Mac OS.
Screens Excel : diverses figures utilisées dans ce rapport.

Voici pour finir le tableau résumé des mesures faites suivant les différentes améliorations, afin que vous puissiez y voir les valeurs réelles de speed up et de temps moyen de calcul, méthode par méthode (en jaune).

Méthode	Basique	OMP	OMP	OMP	OMP	MPI 3	MPI 5	MPI 9	MPI 17
Threads/Processus	1	Non précisé	1	2	4	3	5	9	17
dont calculant	1	?	1	2	4	2	4	8	16
Tps calcul 1	8,664	10,172	8,964	10,353	7,697	1	0,635	2,16	5,263
Tps calcul 2	8,591	12,402	8,556	7,639	6,925	0,432	0,589	1,962	5,331
Tps calcul 3	9,013	8,367	8,412	7,956	7,51	3,792	0,607	1,999	2,94
Tps calcul 4	8,949	8,147	9,429	7,194	5,714	0,428	0,609	2,865	2,884
Tps calcul 5	8,946	9,037	6,399	7,652	8,981	3,242	2,474	3,549	4,297
Tps calcul 6	9,092	9,139	8,528	8,046	9,092	2,818	0,639	2,738	4,971
Tps calcul 7	8,912	8,551	6,302	6,931	7,7	2,332	1,185	2,189	4,394
Tps calcul 8	9,301	8,989	7,693	9,298	6,969	0,343	2,401	2,721	3,001
Tps calcul 9	9,387	9,663	9,771	6,931	8,023	2,823	2,323	2,963	4,539
Tps calcul 10	9,155	7,997	7,453	5,62	5,93	2,119	0,635	3,163	2,963
Moyenne (ms)	9,001	9,246	8,151	7,762	7,454	1,933	1,21	2,631	4,058
Modèle de temps idéal	9	9	4,5	2,25	1,125	4,5	2,25	1,125	0,5625
Speed up	1	0,974	1,104	1,16	1,208	4,656	7,439	3,421	2,218
Idéal (1 d'affichage)	1	0,975	1	2	4	2	4	8	16