
Gravitation universelle

Rapport

ABDELKHALEK RACHED BEN MBARKA MOEZ

Introduction

Le problème à N-corps est un problème mathématique fondamental pour l'astronomie classique. Il s'agit de résoudre les équations du mouvement de Newton pour N corps interagissant gravitationnellement connaissant leurs masses, leurs positions et leurs vitesses initiales.

Nous proposons quatre solutions basées sur différents modèles de programmations parallèle.

CHAPITRE 1

Conception

On va présenter ici les choix de conception communs aux différentes versions. Les détails spécifiques à l'interface utilisée (MPI, MPI communications unilatérales, OpenMP, MPI+OpenMP) seront détaillés dans le chapitre suivant.

Le projet consiste à simuler les interactions gravitationnelles entre un ensemble de particules. Ces particules sont distribuées sur un plan supposé infini. La masse est le seule invariant de toute particule.

1.1 Calcul de la force de gravitation

Le début de la simulation est supposé l'instant 0. Le temps est simulé par une succession de pas séparés d'un délai dt très petit et qui n'est pas forcément constant au cours du temps. A chaque pas, toute particule va subir l'effet de la gravitation causé par toutes les autres particules pendant le délai dt . Cet effet est calculé pour un pas de temps donné. Le calcul des forces de gravitation est basé sur les formules de physique générale. Pour deux particules a et b de masses m_1 et m_2 , distantes de d_{12} , la force appliquée par a sur b est : $F_{21} = G * m_1 * m_2 / d^3 u$ où u est le vecteur unitaire de direction ab . La force finale F_a appliquée sur a est la somme de toutes les force F_{j1} pour j allant jusqu'au nombre total de particules.

Les nouvelles vitesses et positions peuvent se déduire avec les formules suivantes :

$$a_a(t + dt) = F_a / m$$

$$v_a(t + dt) = v_a(t) + dt * a_a(t)$$

$$p_a(t + dt) = p_a(t) + dt * v_a(t) + dt^2 / 2 * a_a(t)$$

Pour calculer sa position suivante, une masse a donc besoin de connaître les masses et les positions de toutes les autres particules. Dans le cas où les données sur les particules sont distribuées sur plusieurs processus, le calcul de la force à l'instant t va nécessiter des échanges de données entre les processus

(2). La simulation doit aussi assurer que les particules passent au même instant de l'instant t à l'instant $t + dt$ ce qui implique une synchronisation entre les particules au début de chaque pas de temps. Le mécanisme de synchronisation dépend de la version et sera abordé dans le chapitre 2.

D'autre part, la norme de la force appliquée par une particule sur une autre croit au fur et à mesure que les deux particules se rapprochent (la norme est en $1/d_3$). Cette force risque donc de diverger et donc amener à de très grandes vitesses. La section suivante détaille ce problème.

1.2 Distance minimale entre deux particules

Bien que dans la vie réelle le temps est continue, la simulation ne peut simuler le temps qu'avec une vision discontinue. En effet, la simulation maîtrise les positions d'une particule aux instants t et $t + dt$ mais n'a pas d'informations sur les positions "intermédiaires" entre les deux instants. Une très grande vitesse peut induire un déplacement très grand entre t et $t + dt$ et donc négliger les effets auxquels serai exposée la particule entre ces deux instants. Un exemple de scénario est qu'une particule avec une très grande vitesse peut dépasser une autre masse sur la direction de sa vitesse sans la toucher.

Pour remédier à ce problème, le sujet suggère d'imposer une contrainte sur le déplacement des masses : une particule ne peut se rapprocher de plus de 10% de la masse la plus proche à l'instant courant. Cela impose, une décision collective entre toutes les particules pour la valeur suivante du pas de temps dt .

Le calcul de la position de la particule la plus proche peut se faire en parallèle du calcul de la force de gravitation puisque ce calcul demande le parcourt des positions de toutes les masses du plan. Si la valeur courante de dt va trop rapprocher la particule de un de ses voisin, il recommence le calcul avec une valeur plus petite de dt . Avant le passage au pas suivant, toutes les particules se mettent d'accord sur la valeur la plus petite de dt parmi celles choisies. L'implémentation de ce mécanisme sera détaillée dans le chapitre 2 selon la version du programme.

D'autre part, le temps de la simulation ne pouvant pas être infini. Nous avons défini des conditions d'arrêt. Cela va être discuté dans la section suivante.

1.3 Conditions de terminaison

Deux paramètres dans la simulation permettent de décrire l'avancement dans le temps :

- Le pas *pas* définit le nombre d'itérations.
- le temps t incrémenté après chaque itération par la valeur courante de dt .

Nous allons donc utiliser deux conditions de terminaison. Il suffit que l'une soit vérifiée pour que la simulation s'arrête.

- Un nombre de pas maximal.
- Un dt très petit.

CHAPITRE 2

Réalisation

2.1 Structures des données

Une particule est définie à un instant donné par : une masse, une position et une vitesse. La section suivante présente les structures de données utilisés.

La structure principale dans la simulation est la masse. Elle est définie par la structure suivante :

```
typedef struct masse_{
    double m ; /* La masse */
    vector p ; /* Le vecteur position */
}masse;
```

Le vecteur est une structure formée par deux réels qui définissent la position de la particule dans le plan.

La structure masse est définie dans le fichier *masse.h*. Le fichier *masse.c* implémente différents accesseurs pour cette structure. La structure *vector* est définie dans le module *plan.h*. Le fichier *plan.c* implémente des opérations divers sur cette structure (norme, calcul de distance, somme...). Ces structures données ont été utilisés dans toutes les versions.

2.2 Versions MPI avec communications par transmission de messages

La première version implémentée est la version MPI avec communication par transmission de messages. On va détailler ici les aspects distribution des particules, échange de données pour le calcul des forces gravitationnelles et mise à jour collective du pas de temps *dt*.

2.2.1 Distribution des particules et échanges de données

En supposant que la mémoire d'un seul processeur ne peut pas stocker les coordonnées de toutes les masses, la charge des calculs doit être distribuée dès le lancement de la simulation. Pour simplifier la distribution, on va faire l'hypothèse que tous les processeurs hébergent le même nombre de particules. Donc si on a N particules et P processeurs, chacun va traiter $N_{loc} = N/P$ particules.

Pour calculer, la force gravitationnelle totale subie par une particule, un processus doit parcourir toutes les masses du plan. Les processus ont donc besoin d'échanger les données sur les particules locales. Volontairement, on n'a pas mis la vitesse dans la structure masse. La raison, est que les communications entre les processus portent seulement sur la position et la masse. Cela permet d'avoir à l'allocation du tableau des masses un buffer local contenant les données à échanger de façon contigue.

Le sujet suggère une topologie en anneau logique. Les buffers des masses sont échangés dans cet anneau dans le sens des aiguilles d'une montre. Les forces gravitationnelles sont accumulées dans un tableau local de vecteurs de forces. Pour optimiser les communications, un nouveau type MPI a été défini pour envoyer ou recevoir un tableau contigu de $3 * N/P$ réels (pour chaque particule locale : un réel pour la masse et deux pour la position).

```
MPI_Type_contiguous(3*N_loc, MPI_DOUBLE, &mpi_masse) ;
```

Quand le processus calcule les contributions de toutes les masses, il calcule pour ses masses locales les nouvelles vitesses et positions avant de passer au pas suivant. Pour tous les pas les schémas de communication sont identiques ce qui suggère l'utilisation des communications persistentes. L'algorithme générale est le suivant :

<pre>Algorithme Schémas des communication Poster les envois. Poster les réceptions. Calculer des contributions des masses reçus à l'échange précédent sur les masses locales. Attendre la fin des communications. Fin</pre>

En plus, on peut utiliser des communications non bloquantes pour espérer faire du recouvrement calculs/communications. A chaque échange, les calculs sont en $O(N_{loc}^2)$ et les communication en $O(N_{loc})$.

D'autre part, on a besoin de synchroniser les différents processus pour que toutes les particules soient à tout moment au même pas. Cela peut être garanti par les communications collectives introduites dans la section suivante.

2.2.2 Décision collective de la valeur de dt

On a vu dans la section 1.2 du chapitre précédent, que la mise à jour du pas de temps peut être nécessaire avant le passage au pas suivant. La décision de la nouvelle valeur de dt doit être fait de façon collective pour garantir la

synchronisation entre les particules de la simulation : il faut qu'à chaque pas de la simulation, tous les processeurs aient la même valeur du temps t . Cette décision est fait avec la routine MPI collective *MPI_Reduce* en utilisant comme opération l'opérateur *MPI_MIN*. Ainsi, la valeur minimale de tous les dt locaux des processus est récupérée sur le processus 0 qui la diffuse avec un *MPI_Bcast*.

2.3 Versions MPI avec communications unilatérales

On a utilisé la même distribution de données que dans la version précédente.

2.3.1 Échange de données :

On garde aussi la topologie en anneau logique. A chaque échange un processus est à la fois :

- un noeud origine : Il récupère les données des masses du processus précédent dans l'anneau.
- un noeud cible en ouvrant une fenêtre pour permettre au processus suivant d'accéder à ses données locales.

Chaque processus ouvre deux fenêtres, une pour chacun des buffers interchangeable.

Les échanges sont aussi non bloquants. Le début de la période d'accès est fait par *MPI_Win_start*. La période d'exposition est déclarée juste après avec *MPI_Win_post* ce qui peut permettre un recouvrement des communications par les calculs.

2.3.2 Mise à jour de dt

Comme dans la version précédente, la mise à jour doit être collective. Pour cela le processus 0 ouvre une fenetre sur la valeur de dt dan laquelle tous les autres processus effectuent une opération "accumulate" avec l'opération *MPI_MIN* ce qui permet d'avoir la valeur minimale de dt dans le processus 0. Les autres viennent ensuite la récupérer de la même fenêtre.

2.4 Version OpenMP

A partir d'une version complètement séquentielle de l'algorithme, une version utilisant OpenMP a été implémentée. L'utilisation de l'outil de *profiling gprof* a permis d'étudier le déroulement de l'algorithme au cours du temps. Cette étude a permis de cibler les parties à paralléliser.

Flat profile:

Each sample counts as 0.01 seconds.

%	cumulative	self	self	self	total	
time	seconds	seconds	calls	ns/call	ns/call	name
43.75	0.07	0.07	10000	7000.00	7000.00	compute_f_loc
18.75	0.10	0.03				add_vector

18.75	0.13	0.03				scal_vector
6.25	0.14	0.01	10000	1000.00	1000.00	update_m_loc
6.25	0.15	0.01				compute_distance
6.25	0.16	0.01				get_p_masse

Nos efforts se sont ainsi concentrés sur le calcul des forces pour chaque masse.

2.4.1 Distribution de la charge

Après plusieurs tests, nous avons constaté que le modèle de distribution statique permettait d'obtenir les meilleures performances. Cela peut être justifié par le fait que pour chaque masse un calcul équivalent aux autres est effectué.

2.4.2 adaptation du modèle séquentiel

Alors que l'algorithme séquentiel permettait d'effectuer le calcul des interactions entre deux masses en un seul tour de boucle (pour les deux masses) La parallélisation du code impose de n'effectuer qu'un seul calcul à la fois sous peine de complexifier considérablement la gestion des variables.

2.5 Version OpenMP + MPI

Il s'agit pour cet algorithme de fusionner les principales idées du premier et du troisième algorithme présentés.

2.6 Visualisation des résultats

Dans de telles simulations il est intéressant de disposer de moyens pour visualiser les résultats. Notre application offre deux méthodes pour exporter les données sous un format qui pourra ensuite être transformé en graphes avec des outils comme *Gnuplot*. Ces deux méthodes sont implémentées dans le fichier *utils.c*.

- Les positions des particules par rapports au temps : L'application crée un fichier par masse auquel est ajouté à une fréquence fixée de pas la position et vitesse courante.
- L'application peut aussi créer un fichier par une fréquence de pas de temps. Ce fichier peut être très utile pour visualiser à instant donné les positions de toutes les particules.

Des exemples de visualisation sont données au chapitre 4

2.7 Importation de fichiers de masses

L'application offre aussi la possibilité d'importer des fichiers de masses. Ceci permet d'effectuer des tests personnalisés. Un fichier de masse est formé par une ligne par particule. Chaque ligne donne la masse, la position et la vitesse de la particule. Un exemple de fichier de masse (*univers.m*) est donnée dans l'archive.

```
# fichier de 3 particules
# m  px  py  vx  vy
100 0  0  0  0
10  0  1  50 50
10  1  0  50 50
```

CHAPITRE 3

Mini-manuel d'utilisation

Chaque version est dans un dossier distinct :

- "seq" : La version séquentielle.
- "mpi" : MPI avec communications par passage de messages.
- "mpi_ul" : MPI avec communications unilatérales.
- "openMP" : Version OpenMP.
- "openMPI" : Version OpenMP + MPI.

Chaque version dispose d'un *Makefile* pour la compilation.

Le fichier *gravitation.h* définit les constantes les plus importantes :

```
/* Parametres de configuration */
int N=15 ; /* Nombre total des particules (par défaut)*/

int Pas_f=100 ; /* Frequence de sauvegarde des positions des particules*/

/* Constantes */

#define PAS_MAX 50000 /* Nombre de pas maximal */
```

3.1 Lancement de l'application :

La compilation donne l'exécutable *gravitation*. La simulation peut être lancée avec les paramètres suivants :

- Lancement avec le nombre de particules par défaut :
./gravitation
- Fixer le nombre de particules :
./gravitation nbParticules

- Donner les paramètres des particules dans un fichier :
(Le format du fichier a été donné dans la section 2.7)
./gravitation nbParticules fichier

Le programme crée des fichier .dat qui pourront être utilisés pour la visualisation avec gnuplot. Des exemples de scripts gnuplot sont aussi fournis dans l'archive. Des exemples de fichiers .m de particules sont également fournis dans le dossier mpi.

CHAPITRE 4

Tests et résultats

On va donner une liste de tests effectués avec les différentes versions . On va donner d'abord des visualisation de tests simples avec un petit nombre de particules. Ensuite, des tests avec un grand nombre pour étude de performances.

4.1 Version MPI avec communications par passage de messages

Fichier de particules :

```
#3 particules : une avec une très grande masse et une vitesse 0  
# et deux particules sur les axes.
```

```
1000000 0 0 0 0  
10 0 1 5 5  
10 1 0 -5 -5
```

La commande à exécuter : (1 seule processus)

```
./go 1 3 univers.m
```

Les résultats sont sur la figure 4.1.

4.2 Etudes de performances

Les mesures suivantes ont été faites avec :

- 100 particules avec les mêmes paramètres pour toutes les versions.
- Un nombre de pas maximal : 100000.

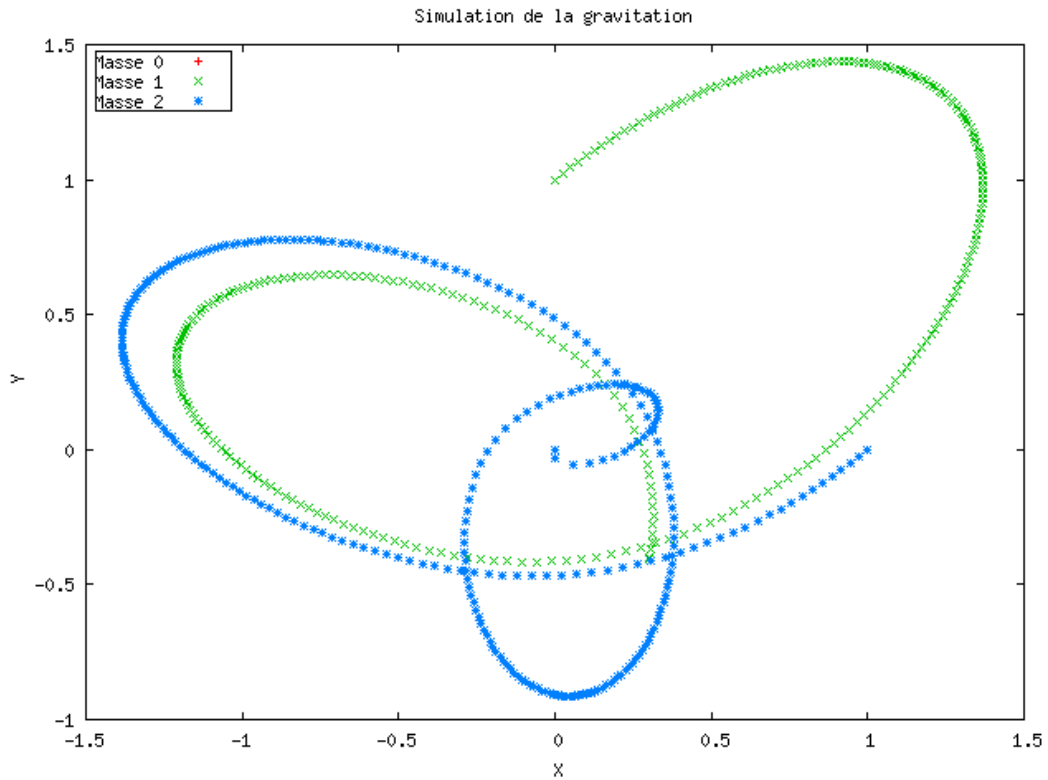


FIG. 4.1 – Graphe 1

- Deux noeuds (Les seul noeuds disponibles au moment des tests sont node2 et node1). Le but étant de comparer les rapports de performances entre les différentes versions ce paramètre peut ne pas être important.
- Pour les versions MPI : deux processus.

Les résultats obtenues :

Version	Temps d'exécution(en s)
OpenMP(4 threads)	182
MPI communications par passage de messages	198
MPI communications unilatérales	284
OpenMP(2 threads)	301
OpenMP + MPI(2 threads)	350
Séquentiel	365
OpenMP + MPI(4 threads)	445

Les résultats sont donnés ordonnés selon le temps d'exécution. A noter que le temps d'exportation des résultats est compté dans tous les tests. On voit que la version la plus rapide est OpenMP avec 4 threads (assez proche de MPI) ce qui peut être expliquée par le faite que pour les versions MPI on a un coût de communication assez important par rapport aux calculs locaux. Cependant MPI est plus rapide si on diminue le nombre de thread à 2 ce qui donne un

parallélisme des charges plus faible.

Conclusion

Ce projet nous a permis de mettre en oeuvre, d'expérimenter et de comparer un ensemble de techniques et de modèles de programmation parallèle en vue de la résolution d'un problème réaliste et important de la physique générale mettant en oeuvre une grande quantité de données et nécessitant des capacités de calcul importantes. Ainsi, l'application réalisée permet la résolution du problème de N-corps en simulant les interactions gravitationnelles.

Les performances obtenues sont meilleures que celles d'un modèle séquentiel naïf qui a été également implémenté. Certaines améliorations peuvent cependant être apportées au modèle de calcul. On pourrait ainsi :

- Répartir les masses sur les processus selon leurs distances. Les particules éloignées (se trouvant sur un processus "éloigné" dans une topologie cartésienne à deux dimensions) seraient alors confondues avec leur centre de gravité ce qui diminuerait les communications et les calculs.
- Pour les versions OpenMP, le nombre de calculs des forces pourrait être divisé par 2 si lors du calcul de la force exercée par une particule sur une autre on mettait à jour les forces associées aux deux particules. Ceci nous imposerait une gestion plus délicate de la répartition des calculs et de la mise à jour des variables.