TDP 5 - Problème à N corps avec une méthode hiérarchique de type Barnes-Hut

Etienne THIERY - Youssef SEDRATI

5 janvier 2017

Dans ce projet nous revisitons le problème à N corps, cette fois-ci en utilisant une méthode hiérarchique de type Barnes-Hut d'abord optimisée localement, puis distribuée

1 Notice

1.1 Compilation

La compilation du projet nécessite la présence d'Intel MKL et d'une implémentation d'MPI sur la machine (modules `intel/mkl/64/11.2/2016.0.0`, `intel-tbb-oss/intel64/43_20150424oss` et `mpi/openmpi/gcc/1.10.0-tm` sur PlaFRIM).

Les 5 exécutables du projet peuvent être compilés via la commande :

```
make
```

1.2 Exécution

`tests` ne prend aucun argument et exécute des tests unitaires, et des tests de régression comparant les résultats obtenus avec le code du TP 2 et ceux obtenus avec celui de ce TP (validant les méthodes `P2P_in`, `P2P_ext`, `M2P` puis le calcul complet).

`tuning` ne prend aucun argument et mesure l'impact du paramètre utilisé pour le critère d'acceptation de développement multipolaire sur la précision des approximations. Voir partie 4.1 pour plus de détails. Un script est à disposition dans le répertoire `Plots` pour tracer un graphe des résultats, avec les commandes suivantes :

```
tuning > Outputs/tuning.dat
gnuplot Plots/tuning.p
eog Plots/tuning.png
```

`benchNaive` prend en argument un nombre de particules, et mesure le temps de calcul des intéractions entre celles-ci en utilisant une implémentation naive et séquentielle, pour référence.

`benchLocal` prend en argument un nombre de particules et une hauteur de quadtree, et mesure le temps de calcul des intéractions entres particules en utilisant une implémentation d'une méthode hiérarchique de type Barnes-Hut, vectorisée et parallelisé sur les coeurs d'une unique machine.

`benchDistribued` prend en argument un nombre de particules et une hauteur de quadtree, et mesure le temps de calcul des intéractions entres particules en utilisant une implémentation d'une méthode hiérarchique de type Barnes-Hut, vectorisée et parallelisée sur les coeurs de plusieurs machines à l'aide de MPI. Un script de job Slurm est à disposition dans le répertoire `Jobs` pour l'exécuter sur 200k, 400k, 800k et 1M6 particules générées aléatoirement, en utilisant 8 noeuds de 20 coeurs sur Plafrim. Il peut être utilisé comme suit :

```
sbatch Jobs/job8.sh
```

2 Vue d'ensemble

Dans notre implémentation, le quadtree est implémenté par un tableau de multipoles, ordonnés par profondeur croissante puis par indice de Morton de la cellule approximée croissant. Ainsi, le multipole 0 est le centre de masse de l'ensemble de particules. Il est subdivisé en 4 multipoles d'indices 1, 2, 3 et 4. Le multipole d'indice 1 est lui même subdivisé en multipoles d'indices 5, 6, 7 et 8, et ainsi de suite. Cette représentation a pour avantage d'être contigue (contrairement à une structure arborescente avec pointeurs), et de permettre de déterminer pour tout multipole i son multipole parent ($i/4$) et ses multipoles fils ($4 * i + 1$ à $4 * i + 4$) facilement. Les feuilles de l'arbre (multipoles les plus profonds) sont doublés par les celulles qu'ils approximent.

Le calcul des indices de Morton est fait de trois manières différentes selon que l'architecture utilisée supporte des instructions spécialisées (très performant, Intel Haswell ou plus récent, non présent sur PlaFRIM), le jeu d'instruction AVX2 (performant, présent sur PlaFRIM), ou pas (pas performant).

Le calcul de la force exercée par une particule (ou multipole) sur l'ensemble des particules d'une cellule (un appel dans `M2P`, un appel par particule de la cellule source dans `P2P`) a été vectorisée à l'aide de BLAS et de méthodes vectorielles offertes par MKL. Cela a été motivé par un profiling qui a montré que ce calcul représentait plus de 95% du temps de calcul initialement.

Nous avons implémenté une parallélisation à gros grain en répartissant les cellules feuilles entre les noeuds d'un cluster MPI, puis en répartissant les cellules attribuées à un noeud entre ses processeurs avec openMP. Pour avoir un bon équilibre de charge sur les 20 coeurs de 8 noeuds de PlaFRIM, il est nécessaire que le nombre de cellules feuilles soit un multiple de $20 * 8 = 160$, nous préconisons donc une hauteur d'arbre de 5 ou plus.

Nous n'avons pas implémenté de scatter/gather d'un ensemble de particules initialement présent sur un seul noeud. Les particules sont initialisées aléatoirement directement sur tous les noeuds à partir d'une même graine pour assurer la cohérence (duplication totale de l'arbre).

3 Structure du projet

Les fichiers `Multipole.h|c` définissent une simple structure contenant la masse et la position d'un centre de masse, ainsi que les cordonnées de la cellule qu'il approxime.

Les fichiers `Cell.h|c` définissent une structure de cellule (tableau de particules) et tous les opérateurs de la méthode hiérarchique dans une version naïve de référence et dans une version optimisée. `pOnP` applique les forces d'une particule (ou multipole) sur les particules d'une cellule, `P2P` les forces de toutes les particules d'une cellule sur toutes celles d'une autre cellule et `M2P` celle d'un centre de masse sur les particules d'une cellule. `P2M` calcule le centre de masse des particules d'une cellule et `M2M` le centre de masse de 4 centres de masse. Ils définissent également une méthode vérifiant si le critère d'acceptation de développement multipolaire (critère de Barnes Hut simplifié présenté en TD) est vérifié par une cellule et un centre de masse.

Les fichiers `WorkingVecs.h|c` définissent une structure contenant des tampons réutilisables, nécessaires pour le calcul vectoriel de `pOnP` et permettant de limiter le nombre d'allocations.

Les fichiers `Morton.h|c` définissent les méthodes de calcul des indices de Morton.

Les fichiers `Quadtree.h|c` définissent une structure de quadtree, ainsi que les opérations de calcul des centres de masse en remontant dans l'arbre, et d'applications des forces en redescendant (version locale et version distribuée).

Les fichiers `utils.h|c` définissent des fonctions utilitaires de base, et des fonctions de test. L'une d'entre elle permet de calculer des statistiques sur les erreurs relatives commises par la méthode hiérarchique comparé à la méthode naïve (minimum, maximum et quartiles).

Enfin, les fichiers `tests.c`, `tuning.c`, `benchNaive.c`, `benchLocal.c` et `benchDistributed.c` définissent les fonctions principales des exécutables correspondants.

4 Performances

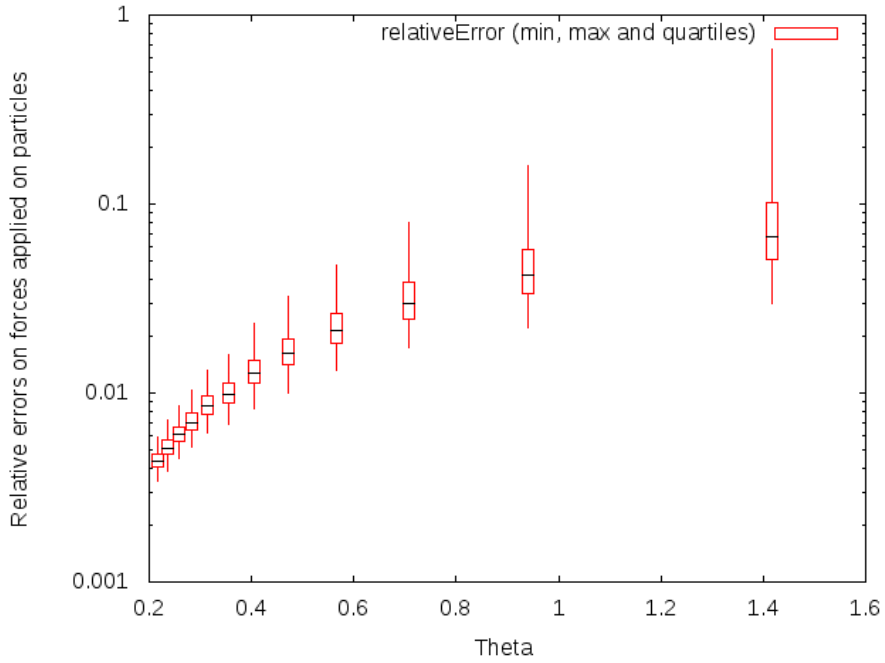
4.1 Précision

On considère qu'un centre de masse approximant une cellule de taille l est dans le champ lointain des particules si une distance $d > \frac{l}{\theta}$ les sépare.

Une fois l'approximation d'un ensemble de cellules par son centre de masse en place, nous avons voulu mesurer l'erreur effectuée selon θ.

La figure suivante montre que la valeur $\theta = 0.707$ mentionée dans le cours permet d'effectuer une erreur médiane inférieure à 2% et d'au maximum 8% sur la somme des forces effectuées sur chaque cellule. (Les extrémités des bougies représentent le minimum et le maximum d'une série, les extrémités supérieures et inférieures des boites représentent les premier et troisième quartiles, la barre horizontale centrale représente la médiane)

FIGURE 1 – Impact du paramètre d'un critère d'acceptation dérivé du critère de Barnes-Hut sur la précision de l'approximation



Il est intéressant de remarquer que le maximum d'une série est bien plus elevé que la valeur médiane. Cela s'explique en fait par le fait que la somme de plusieurs petites erreurs relatives peut donner une bien plus grosse erreur relative globale. Par exemple, une force de $8.58 \times 10^{-40}N$ approximée par une force de $7.50 \times 10^{-40}N$ (erreur relative de 12%) sommée avec une force exacte de $-9.73 \times 10^{-40}N$ donne une force totale $-1.145 \times 10^{-40}N$ approximée par une force de $-2.219 \times 10^{-40}N$ soit une erreur relative de 94%. Plus le nombre d'approximations effectuées est grand, plus la probabilité d'un tel évènement augmente, d'où la taille croissante des bougies.

4.2 Vitesse

L'exécution du script Slurm fourni sur 8 noeuds de PlaFRIM montre que notre implémentation est capable de calculer les intéractions dans un ensemble d'1.6 millions de particules en une dizaine de secondes, contre seulement 10000 avec la méthode naïve dans le même temps !

Nous n'avons pas tracé de courbe de speedup pour ce projet. En effet, puisqu'on s'est focalisé sur un seul pas de temps de la simulation, et que l'arbre est entièrement reproduit aléatoirement à partir d'une même graine sur chaque noeud (en un temps négligeable par rapport au temps calcul des intéractions), il n'y a aucune synchronisation entre noeuds. De ce fait, le speedup est logiquement quasi linéaire. Il faudrait implémenter les communications entre deux pas de temps successifs pour que la courbe de speedup prenne un sens.