Simulation de particules sur architectures parallèles

Il s'agit de paralléliser à l'aide d'OpenMP une application de simulation de particules dans un domaine en trois dimensions, en calculant des interactions à courte distance entre particules. Le déroulement de la simulation pourra être visualisé en « temps réel » grâce à un rendu OpenGL des particules. Une version séquentielle naïve du code vous est fournie, ainsi que la partie visualisation, de façon à ce que vous puissiez uniquement vous focaliser sur l'accélération des calculs en parallèle.

1 Premiers pas

1.1 On essaye tout de suite !

Dans le répertoire `fichiers/`, il vous faut d'abord générer le Makefile automatique à l'aide de `cmake` :

```
mkdir build
(cd build ; cmake ..)
```

Ensuite vous pouvez compiler pour construire le binaire `bin/atoms` :

```
(cd build ; make)
```

Affichez l'aide en ligne en tapant :

```
./bin/atoms -h
```

Puis lancez une première exécution avec les options suivantes :

```
./bin/atoms -v -s 1 -o 1
```

Normalement, un ensemble d'une centaine d'atomes apparaît dans une fenêtre OpenGL. Vous pouvez alors :

- changer l'angle de vue à la souris (cliquer-déplacer) ;
- taper `>` (resp. `<`) pour zoomer (resp. dézoomer) ;
- taper `+` (resp. `-`) pour accélérer (resp. ralentir) la simulation ;
- taper `m` pour activer/désactiver le mouvement des atomes ;
- taper `f` pour activer/désactiver le potentiel de Lennard-Jones ;
- taper `q` ou la touche *Escape* pour quitter l'application.

2 Parallélisation avec OpenMP

Le fichier à modifier est `libsotl/src/openmp.c`. Parallélisez les fonctions `omp_force`, `omp_move` et `omp_update_vbo` au niveau de leur boucle principale. Dans un second temps on essayera de réduire les appels au pragma `parallel` en créant les threads uniquement dans la fonction `omp_one_step_move`. Testez visuellement votre nouvelle version (avec le fichier `choc1.conf` par exemple), puis mesurez les performances en mode *batch*. Par exemple :

```
./bin/atoms -v --omp 1 -i 10 -n 1k
```

Modifiez les couleurs des atomes de façon à colorier les atomes selon le threads qui les a traités. Visualiser l'effet des différentes politiques de scheduling.

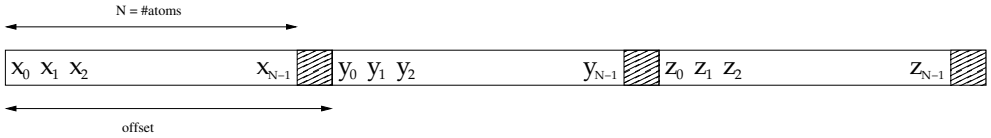

FIGURE 1 – Pour améliorer la performance des accès mémoire sur les accélérateurs, les tableaux `pos_buffer` et `speed_buffer` sont agencés de la manière illustrée ci-dessus : une première tranche contient les coordonnées x (dx pour `speed_buffer`), une seconde contient les y et la troisième contient les z. La taille totale de chaque tranche est agrandie de manière à correspondre à un multiple de 16 éléments. Le nombre d'atomes d'un ensemble `set` d'atomes est `set.natoms`. La taille totale d'une tranche est contenue dans `set.offset`.

3 Pour en savoir plus

3.1 Positions et coordonnées des atomes

Pour calculer le résultat des interactions entre atomes, on mémorise pour chaque atome sa position (x, y, z) et sa vitesse (dx, dy, dz). Pour simplifier, on considère que la vitesse d'un atome est calibrée de manière à ce qu'à chaque itération de la simulation, (dx, dy, dz) représente le vecteur qu'il faut ajouter à la position d'un atome pour obtenir sa position à l'itération suivante.
L'application mémorise la position des atomes et leur vitesse dans deux tableaux distincts, respectivement nommés `pos` et `speed` dans la structure `atom_set` (définie dans `atom.h`).

3.2 Mouvement des particules

Regardez le code de la fonction `seq_move` (dans `seq.c`) : c'est celle-ci qui met à jour, à chaque itération, les positions de tous les atomes en fonction de leur vitesse. Notez qu'il s'agit d'une addition de vecteurs...

3.3 Visualisation

Pour les besoins de la visualisation, un *buffer* OpenGL nommé `vbo_buffer` (« *Vertex Buffer Object* ») contient les coordonnées de chaque point utilisé pour afficher un atome. Ce tableau contient une suite de triplets (x, y, z) (chaque coordonnée étant de type `float`). Les `vertices_per_atom` $\times 3$ floats forment donc les coordonnées du premier atome, etc.

3.4 Un peu de culture : le potentiel de Lennard Jones

De nombreux phénomènes physiques entrent en jeu lorsqu'il s'agit de modéliser les interactions entre atomes au sein d'un gaz, d'un solide ou d'un liquide (cf `http://fr.wikipedia.org/wiki/Potentiel_interatomique`).
Nous nous intéresserons ici au potentiel de Lennard-Jones, qui capture à la fois les phénomènes d'attractions entre atomes lorsqu'ils sont distants, et les phénomènes de répulsion lorsqu'ils sont trop proches (effets quantiques). L'intensité F_{ij} de la force exercée par un atome j sur un atome i est donnée par la formule suivante :

$$F_{ij} = \begin{cases} 24\frac{\epsilon}{r}\left(\left(\frac{\sigma}{r}\right)^6 - \left(\frac{\sigma}{r}\right)^{12}\right) & \text{si } r \leq r_c \\ 0 & \text{sinon.} \end{cases} \quad (1)$$

ou r est la distance entre i et j. σ et ϵ sont des constantes choisies en fonction des caractéristiques physiques du matériau simulé. Notez que σ représente la distance à laquelle l'interaction entre les atomes est nulle.
Lorsque la distance entre deux atomes excède un seuil nommé *rayon de coupure* (r_c), les forces sont négligées.

$$\vec{F_{i*}} = \sum_{j \neq i} F_{ij}.\hat{u}_{ij} \text{ avec } \hat{u}_{ij} = \frac{\vec{ij}}{r} \tag{2}$$

L'implémentation des interactions entre atomes est effectuée dans la fonction `seq_force`. Notez que les distances entre atomes sont effectuées pour tous les couples d'atomes, d'où un temps d'exécution en $O(n^2)$. Notez aussi que, bien qu'il aurait suffit de calculer qu'une seule fois l'intensité de la force pour chaque paire d'atomes, elle est ici calculée deux fois, une fois par atome. Pourquoi selon vous ?
Essayez cette implémentation séquentielle avec un fichier de configuration contenant un nombre modéré d'atomes, tel que `choc1.conf` :

```
./bin/atoms -v -s 1 -o 1 conf/choc1.conf
```

Appuyez sur `f`, puis sur `m`...