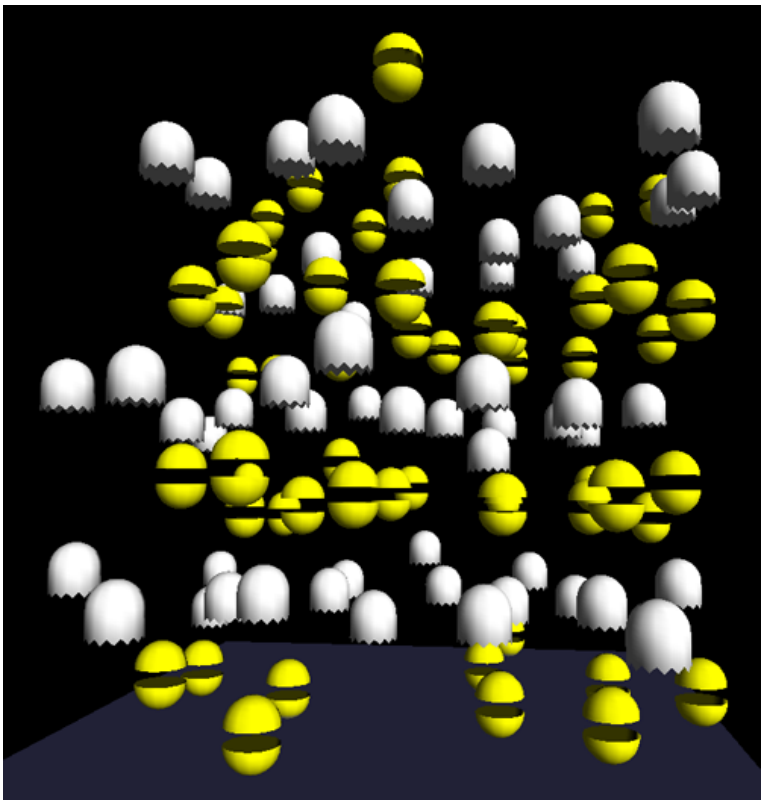
Simulation de particules sur architectures parallèles



Il s'agit de concevoir une application de simulation de particules dans un domaine en trois dimensions, en calculant des interactions à courte distance entre particules. La simulation sera programmée en OpenMP sur machines multicœurs, et en OpenCL sur accélérateurs. 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 OpenGL », de façon à ce que vous puissiez uniquement vous focaliser sur l'accélération des calculs en parallèle.
Rappel : vous trouverez des ressources utiles dans le répertoire `/net/cremi/rnamyst/etudiants/opencl/`. Et notamment, vous trouverez une synthèse des primitives OpenCL utiles dans un « *Quick Reference Guide* » situé ici :

`/net/cremi/rnamyst/etudiants/opencl/Doc/opencl-1.2-quick-reference-card.pdf`

En cas de doute sur le prototype ou le comportement d'une fonction, on pourra se reporter au guide de programmation OpenCL accessible au même endroit :

`/net/cremi/rnamyst/etudiants/opencl/Doc/opencl-1.2.pdf`

1 Premiers pas

1.1 On essaye tout de suite !

Copiez le répertoire `~rnamyst/etudiants/pmg/Projet` sur votre compte. 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 :

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

Si tout va bien, le binaire `bin/atoms` est construit. Affichez l'aide en ligne en tapant `./bin/atoms -h`. Lancez une première exécution avec les options suivantes :

```
./bin/atoms -v -s 0 -o 0
```

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 `q` ou la touche *Escape* pour quitter l'application.

1.2 Structure de la simulation

Le code source de la simulation est organisé au sein d'une bibliothèque rassemblant les fonctions de simulation et de visualisation des particules. Le fichier `main.c` du programme sert à analyser les options de la ligne de ocmmande, à éventuellement charger la configuration initiale depuis un fichier, et à lancer la boucle principale de la simulation.
La bibliothèque est organisée de la façon suivante :

`sotl.c`	Point d'entrée principal de la bibliothèque, ce fichier rassemble les fonctions appelables depuis le programme principal. Il est également en charge de découvrir les accélérateurs disponibles (pour OpenCL) et d'initialiser la bibliothèque.
`atom.c`	Gestion des atomes.
`domain.c`	TODO.
`seq.c`	Implémentation séquentielle de la simulation. C'est probablement le premier fichier à regarder, à modifier pour comprendre son fonctionnement, etc.
`openmp.c`	Fichier dans lequel vous implémenterez la version OpenMP de la simulation. Les fonctions de ce module seront appelées lorsque l'option `--omp` sera utilisée en ligne de commande (voir aide en ligne du programme).
`window.c`	Initialisation d'OpenGL et boucle principale de rafraichissement d'écran.
`vbo.c`	Gestion des points et des triangles destinés à l'affichage OpenGL. Normalement, vous n'aurez pas besoin de consulter/modifier ce module. On peut même très bien vivre sans l'avoir regardé...
`ocl.c`	Initialisation et gestion des différentes structures liées à OpenCL. Il est utile d'y jeter un oeil pour comprendre quels sont les *buffers* alloués sur la carte graphique pour les besoins de cette simulation.
`ocl_kernels.c`	Ensemble des « *wrappers* » permettant d'exécuter les noyaux OpenCL.
`physics.cl`	Code OpenCL des noyaux qui s'exécuteront sur la carte graphique.

Pour vous familiariser avec le cœur de l'application, regardez dans `sotl.c` la fonction `sotl_main_loop` et suivez les fonctions appelées dans `seq.c`.

1.3 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.

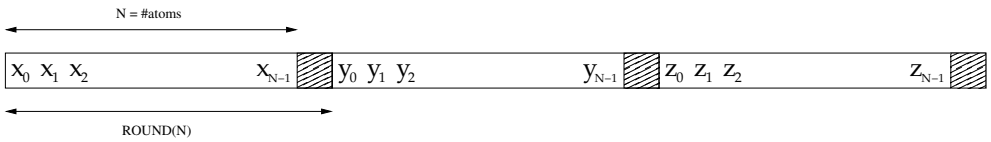


FIGURE 1 – Pour améliorer la performance des accès mémoire sur les accélérateurs, les tableaux `pos_buffer` et `speed_buffer` sont agencés de la manière illustrée ci-dessus : une première tranche contient les coordonnées x (dx pour `speed_buffer`), une seconde contient les y et la troisième contient 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`.

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`).

1.4 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...

1.5 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 [1].
Regardez

1.6 Rebond sur les parois du domaine

Dans chaque fichier de configuration, en plus des coordonnées des atomes, sont définies les coordonnées de deux points `min_ext` et `max_ext` délimitant le parallélépipède rectangle contenant tous les atomes.
Afin de garantir que les atomes restent dans ce parallélépipède, nous allons les faire rebondir sur les parois à chaque fois qu'ils entrent en collision avec l'une d'elles.
Écrivez la fonction `seq_bounce` qui teste, pour chaque atome, la collision avec l'un des bords. Si le centre d'un atome franchit un bord, il faut inverser la composante vitesse qui est orthogonale à ce bord. Par exemple, pour tester le rebond sur le sol, il faut pour chaque atome comparer y et y_{min_ext} et, en cas de collision, multiplier la composante vitesse dy par -1.
Réfléchissez bien à créer un nombre de threads maximisant le parallélisme sur le GPU.

1.7 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

1. Cette façon de stocker les coordonnées n'est pas forcément idéale pour les noyaux OpenCL que nous écrirons ultérieurement, mais elle est imposée par OpenGL.

(effets quantiques). L'intensité F_{ij} de la force exercée par un atome j sur un atome i est donnée par la formule suivante :

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

où 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.

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

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 0 -o 0 conf/choc1.conf
```

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

2 Parallélisation avec OpenMP

2.1 The naive way

Inspirez-vous des fonctions de `seq.c` pour implémenter celles de la version OpenMP. Parallélisez les fonctions [2] au niveau des boucles.
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 0 -i 10 -n 1k
```

Tracez des courbes d'accélération (par rapport à la version séquentielle pure) en faisant varier le nombre de processeurs utilisés (variable d'environnement `OMP_NUM_THREADS`).

TO BE CONTINUED

2. Pas seulement la fonction `seq_force`