PROJET
Techniques d'optimisation de la parallélisation
Optimisation d'une simulation

Julien Jaeger
julien.jaeger@cea.fr
Thomas Dionisi
thomas.dionisi@uvsq.fr

1) Objectif

Vous allez devoir optimiser une simulation numérique afin de la rendre la plus scalable possible. La base que nous vous fournissons a été volontairement « désoptimisée » par nos soins. À vous d'utiliser toutes les armes à votre disposition pour repérer et éliminer les différentes contentions de ce programme. Pour aller plus loin, une implémentation hybride MPI+OpenMP sera appréciée.

Afin de vous faire travailler avec un débogueur, la version fournie est également volontairement non fonctionnelle. Il faudra donc dans un premier temps identifier le(s) problème(s) et le(s) corriger.

Ce projet est à faire en binôme.

2) Rapport

Vous fournirez un rapport détaillant la démarche que vous avez suivie :

1. Évaluation de la scalabilité du code d'origine.
2. Description du schéma de communication d'origine. Éventuelles remarques préalables quant aux problèmes de performances que vous pouvez supposer par cette analyse.
3. Explication de la méthode utilisée pour trouver la source du(es) bug(s) et une explication précise du problème.
4. Les différentes contentions que vous avez identifiées et la démarche utilisée pour les identifier. Expliquez les problèmes posés par ce que vous avez repéré.
5. Les tentatives (réussies ou non) de résolution de ces problèmes de performance et l'impact de la correction sur les performances. Si une modification ne porte pas ses fruits en termes de performance, n'hésitez pas à la citer et être critique sur votre travail, c'est tout aussi intéressant.
6. En fin de rapport, à partir des observations que vous aurez réalisées, récapitulez quelques règles qui vous semblent importantes lors de l'optimisation d'une application pour améliorer la scalabilité de votre application.

Le rapport devra être renvoyé par mail et déposé dans un espace prévu à cet effet dans l'ENT au plus tard le **07/05/2020** au format PDF et les sources envoyées sous forme d'une archive. Cette archive devra contenir un unique dossier contenant les sources. Merci de nommer cette archive et le dossier contenu avec la forme NOM_prenom.

Un petit bonus sera attribué pour :

- L'étudiant qui aura la version la plus rapide.
- Les étudiants qui parviendront à faire mieux que notre version optimisée (tous les coups sont permis, soyez imaginatifs). **Attention, elle doit fournir un résultat juste !**

Toute optimisation appliquée devra être décrite de la manière suivante dans votre rapport :

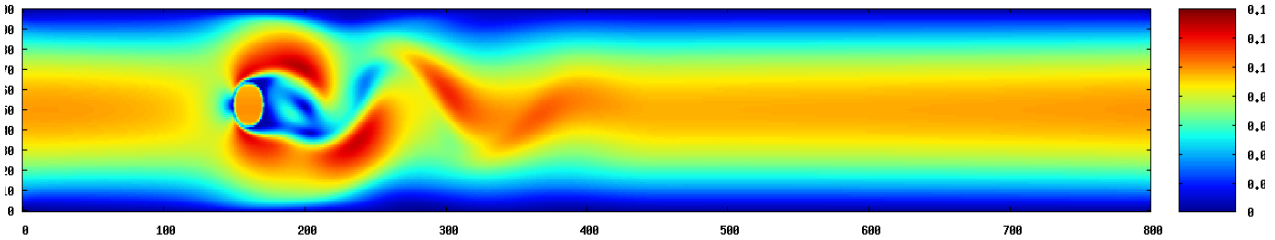
1. **Identification et description du problème d'origine**

2. **Justification de la modification**
3. **Évaluation de l'impact.**

3) La simulation

Le code fourni a pour objectif de simuler une allée de tourbillons de Karman. Ce phénomène s'observe par exemple dans l'atmosphère au passage d'un vent autour d'une île. Suivant les paramètres, une série de vortex se met en place. Historiquement, ces études ont également été mises en place lors des études d'écoulement d'air autour des avions.

Dans notre cas on considérera un fluide parcourant un tube 2D dans lequel on place un obstacle de forme ronde (il est possible de changer la forme de cet obstacle en chargeant une image, les noirs représentant des zones d'obstacles, les blancs les zones de fluide). Ce qui revient sur le principe à simuler une soufflerie.



Pour la résolution on utilise LBM (Lattice Boltzmann Method). La bibliographie suivante aidera les curieux à mieux comprendre les détails de résolution mathématique :

- [1] Description générale et succincte : http://www.cims.nyu.edu/~billbao/report930.pdf
- [2] La thèse de Alexandre Dupuis, voir chapitre 3 : http://cui.unige.ch/~chopard/CA/aDupuisPhD.pdf
- [3] Différentes implémentations exemple : http://wiki.palabos.org/numerics:codes
- http://www.cims.nyu.edu/~billbao/courses.html
- http://wiki.palabos.org/lattice_boltzmann_method

4) Modélisation

Comme expliqué dans les documents précédents, LBM modélise le fluide en le discrétisant spatialement sur un maillage cartésien. Dans les cellules du maillage, le fluide est décomposé au niveau microscopique en particules de fluides pouvant se déplacer dans 9 directions. Modèle dit D2Q9 (dimension : 2, directions : 9), voir figure 1.

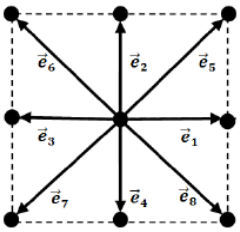


Figure 1
Un nœud du maillage et ses 9

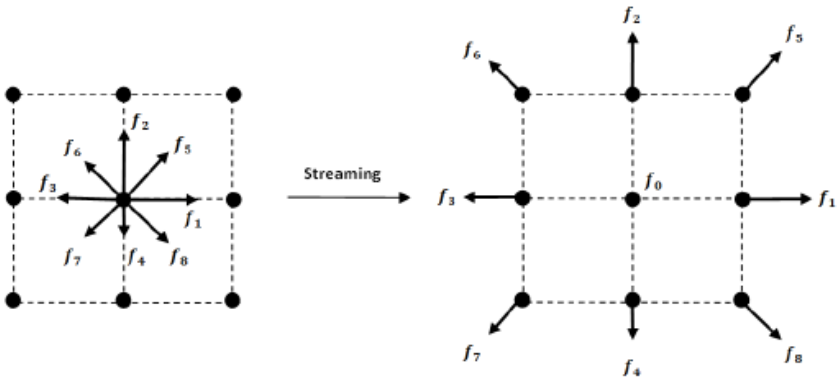


Figure 2
Étape de propagation vers les nœuds voisins.[1]

directions[1].

Chaque nœud porte donc physiquement les quantités :

- f_1, f_2, f_3, f_4, f_5, f_6, f_7, f_8, f_9 : les densités microscopiques de fluide (probabilité) se déplaçant dans la direction associée.
- La densité macroscopique à la position considérée, qui s'obtient en sommant les 9 densités microscopiques :

$$\rho(\vec{x},t)=\sum_{i=0}^{8} f_i(\vec{x},t)$$

- La vitesse macroscopique du fluide, elle aussi construite à partir d'une somme des densités microscopiques :

$$\vec{v}(\vec{x},t)=\frac{1}{\rho}\sum_{i=0}^{8} cf_i(\vec{x},t)\vec{e}_i$$

Lors de l'affichage, on s'intéressera notamment à la norme de la vitesse macroscopique.

4.a) À chaque pas de temps

À chaque pas de temps, il faut :

1. Appliquer les conditions particulières (conditions au bord, obstacle…)
2. Calculer l'effet des collisions sur les particules de fluide au niveau microscopique, voir 3.b.
3. Propager les valeurs suivant chaque direction, comme cela est représenté sur la figure 2.

4.b) Collisions microscopiques des particules de fluides

Pour chaque maille du fluide, à chaque pas de temps il faut mettre à jour les collisions entre les particules de fluide allant dans les différentes directions :

1. Calculer les grandeurs macroscopiques : densité et vitesse
2. Utiliser le calcul précédent et les f_i pour évaluer les collisions entre les 9 particules de fluides situées à la même position.

L'étape 3 utilise une dérivée de l'équation de navier-stokes qui sort du cadre d'intérêt de ce TP, ceux qui voudront plus de détails peuvent se référer aux documents cités comme base de recherche bibliographique.

De manière résumée on applique d'après [1] :

$$f_i=f_i^{*}-\frac{1}{\tau}(f_i^{*}-f_{eq})$$

Où :

- f_i est le nouvel état.
- f_i^* est l'état instable obtenu après propagation à l'étape 1.
- f_{eq} est l'état d'équilibre vers lequel le fluide va tendre, donné en [1]. Attention, l'implémentation retenue rend l'équation adimensionnelle en utilisant le nombre de Reynolds, la constante de vitesse c n'y apparaît donc pas, car absorbée par ce dernier terme.
- Le terme en $\frac{1}{\tau}$ traduit le temps caractéristique que le fluide va mettre pour se mettre à l'équilibre. Cette valeur dépendant de la viscosité du fluide.

On verra à plusieurs reprises intervenir les poids w_i, ces derniers servent à compenser le fait que les 9 vecteurs $\vec{e}_i$ (figure 1) n'ont pas tous la même norme. Les valeurs de ces poids sont donnés dans [1].

4.c) Conditions sur le bord gauche

Le bord gauche constitue le point d'entrée du fluide, on considère donc un état stable avec une vitesse d'écoulement vérifiant une distribution de type Poiseuille (solution d'un écoulement dans un tube). Grossièrement, cette fonction donne un profil de vitesse qui est nulle sur les bords (frottement sur les parois) et maximale au centre.

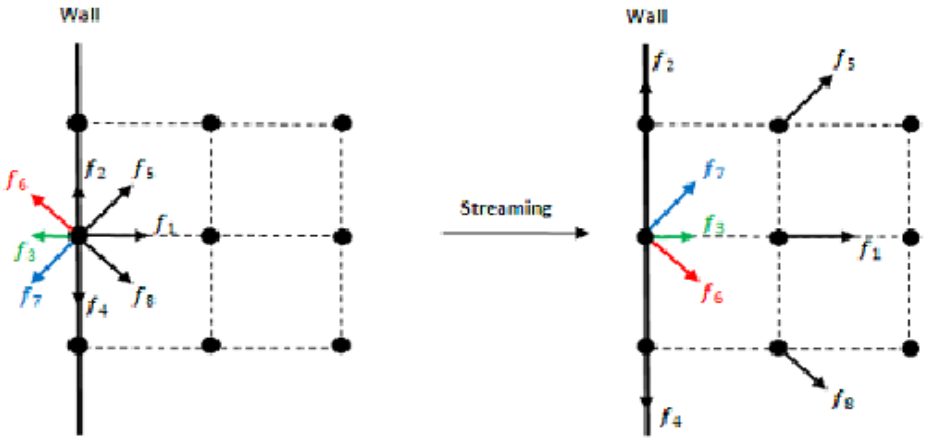
Afin de maintenir l'équilibre du fluide, ce flux est introduit en utilisant la méthode You/Le dont vous trouverez les détails dans [1]. Concrètement il s'agit d'obtenir des f_i cohérents avec la condition de vitesse constante imposée à l'extérieur et l'état actuel de la portion de fluide en contact avec la limite.

4.d) Condition sur le bord droit

Même idée avec You/Le, mais au lieu de maintenir un profil de vitesse, on cherche à maintenir un gradient de densité nulle. Autrement dit, imposer qu'il y ait autant de fluide qui passe la paroi (donc qui sorte du tube) qu'il n'en arrive sur la paroi tout en maintenant une pression non nulle à l'extrémité.

4.e) Conditions au bord haut et bas

Ces bords



représentent des murs, on applique donc une simple réflexion comme on le ferait avec un photon sur un miroir ou une particule contre une plaque. Le frottement sur la paroi impose également une vitesse nulle le long de cette dernière.

4.f) Condition sur l'obstacle

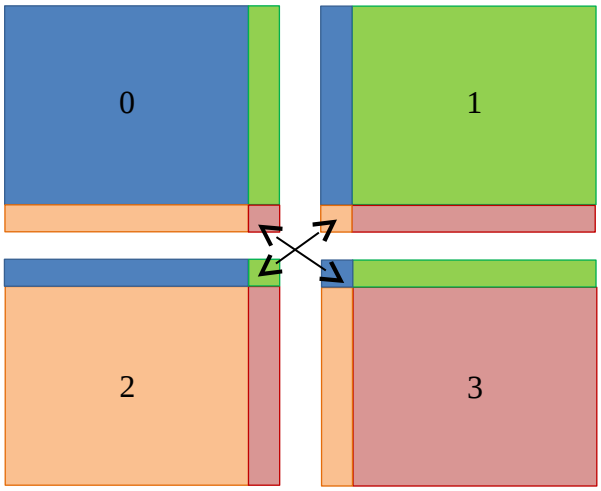
L'obstacle joue le rôle d'un mur, on y applique la même méthode que pour les bords hauts et bas.

4.g) État initial.

L'état initial est considéré comme un écoulement laminaire stable, donc suivant une distribution de vitesse de Poiseuille sur l'ensemble du fluide. La densité est fixée constante à 1 pour l'ensemble des nœuds. À l'instant t_0 on introduit l'obstacle. L'artefact observé lors des premiers pas de temps est lié à cette insertion brutale.

4.h) *Communication avec les voisins*

En mode MPI, le maillage est découpé en sous-domaines de sorte à répartir le travail sur chaque nœud. Dans notre cas, à chaque pas de temps, il faut obtenir les mises à jour des mailles fantômes bordant le domaine local :



4.i) *Fichier de configuration*

La simulation est configurée à l'aide d'un fichier **config.txt** au format :

```
iterations          = 10000
width               = 800
height              = 100
#obstacle_r         =
#obstacle_x         =
#obstacle_y         =
reynolds            = 100
inflow_max_velocity = 0.100000
output_filename     = resultat.raw
write_interval      = 50
#obstacle_filename  = object.png
#obstacle_scale     = 1
#obstacle_rotate    = 0
```

Les paramètres sont :

iterations	Nombre de pas de temps à réaliser
width	Largeur totale du maillage
height	Hauteur totale du maillage
reynolds	Facteur d'échelle de la simulation
inflow_max_velocity	Vitesse maximale du flot entrant
output_filename	Nom du fichier de sortie (Pas d'écriture si non défini)

write_interval	Nombre de pas de temps entre les sorties
obstacle_filename	Fichier image de définition d'obstacles (seulement si compilé avec MagickWand)
obstacle_scale	Facteur de redimensionnement de l'image
obstacle_rotate	Rotation de l'obstacle

4.j) Utilisation

Pour compiler le programme :

```
make
```

Exécution de la simulation :

```
mpirun -np 512 ./lbm
```

4.k) Visualisation des résultats

Nous vous fournissons trois scripts de visualisation s'appuyant sur gnuplot.

Vous devez donc disposer de gnuplot pour les utiliser !

gen_animate_gif.sh :

Ce script utilise les fonctionnalités de gnuplot pour générer une animation gif si votre gnuplot le supporte (suffisamment récent). Si vous obtenez une erreur utilisez **gen_animate_gif_legacy.sh**

Exemple:

```
./gen_animate_gif.sh ./result.raw ./output.gif
```

gen_animate_gif_legacy.sh :

Ce script est identique à gen_animate_gif.sh sauf qu'il utilise convert pour créer le gif. Son utilisation est identique au script précédent. Il peut également être utilisé pour générer les images de tous les pas de temps (situées dans GIF_TMP). **Ce script utilise la commande convert de ImageMagick.**

Exemple:

```
./gen_animate_gif_legacy.sh ./result.raw ./output.gif
```

gen_single_image.sh :

Comme son nom l'indique, ce script sert à générer une seule image extraite du fichier de sortie. Il prend un paramètre supplémentaire qui est le numéro de la sortie (0 à n).

Exemple:

```
./gen_single_image.sh ./result.raw ./output.png 33
```

Remarque :

Vous trouverez dans les sources le programme display qui sert à extraire les données des pas de temps depuis le fichier binaire vers les formats gnuplot et VTK il est utilisé en interne dans les scripts précédents. Il peut vous servir à explorer les données de manière textuelle

Utilisation :

```
./display {--gnuplot|--vtk} {file.raw} {frame_id}
```

Exemple :

```
./display --gnuplot ./result.raw 33
```

4.I) Validation des résultats

À tout moment vous pouvez valider votre résultat pour comparer deux fichiers résultats afin de vérifier leur validité (même checksum). **Pensez à le faire régulièrement !**

```
./display --checksum {file.raw} {frame_id}
```