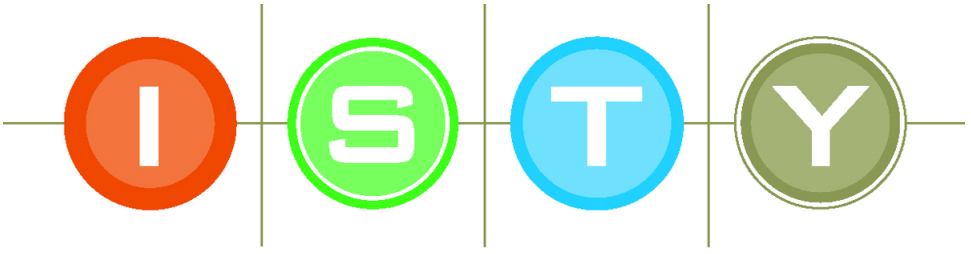
Analyse de performance et optimisation de code

Pierre AYOUB

25 janvier 2019



INSTITUT DES SCIENCES ET TECHNIQUES DES YVELINES

Résumé

La simulation numérique est un procédé informatique visant à modéliser un phénomène par ordinateur, s'agissant le plus souvent d'un phénomène physique. Cette modélisation prend forme par des systèmes d'équations décrivant l'état du système physique représenté à chaque instant. De nombreux domaines scientifiques convergent vers la simulation informatique, tel que certaines branches de la physique, de l'analyse et de l'optimisation mathématique, ou encore le calcul haute performance en informatique. Enfin, la simulation trouve naturellement de nombreuses applications concernant des sujets variés, tel que la simulation du climat et des évènements météorologiques, la simulation d'essais nucléaires, de l'effet d'un médicament sur un corps, ou encore des astres et de l'univers. Ce rapport s'articulera donc autour de la simulation de fumée, phénomène impliquant les lois de la mécanique des fluides. Notre travail portera sur l'aspect du calcul haute performance de cette simulation.

Table des matières

1 Introduction

Le projet que nous vous présentons aujourd'hui consiste à analyser puis, grâce à nos mesures, optimiser un code de simulation numérique. Ce dernier nous offre une interface graphique permettant d'ajouter de la fumée dans un espace confiné et, ainsi, d'en observer le comportement. Nous pouvons influencer la quantité de fumée et sa vélocité dans l'espace. De plus, l'application nous donne le contrôle sur la résolution de la simulation, cela revient à dire sur sa précision, qui détermine principalement la performance du programme.

Le déroulement du projet s'est effectué en plusieurs étapes distinctes :

Analyse du code Cette phase consiste à analyser le programme d'un point de vue mathématique et informatique. De cette première approche, il s'agira de comprendre les opérations du programme sur les équations qui régissent le système physique. De l'autre approche, il convient d'étudier l'architecture logicielle de l'application, ainsi que les choix mis en œuvres afin d'implémenter le ou les algorithmes nécessaires.

Protocole expérimental Une fois l'analyse effectuée, nous pouvons en déduire le moyen le plus adapté afin de mesurer les performances de notre implémentation. Nous allons donc mettre en avant les critères théoriques à atteindre dans nos mesures, puis nous exposerons la manière dont nous avons mis ceci en pratique.

Optimisations et mesures Grâce au protocole mis en place, nous pouvons quantifier la performance du programme. De ce fait, serons en mesure d'expérimenter différentes techniques d'optimisation sur le programme et d'en calculer l'accélération.

2 Analyse du code

fluid.cflow

```
+-advect <void  at fluid.c:183>
  +-build_index <int  at fluid.c:10>
  \-setBoundry <void  at fluid.c:52>
    \-build_index <int  at fluid.c:10>
+-buoyancy <float  at fluid.c:143>
  \-build_index <int  at fluid.c:10>
+-c_densitySolver <void  at fluid.c:319>
  +-addSource <void  at fluid.c:37>
  +-swap <void  at fluid.c:20>
  +-diffuse <void  at fluid.c:117>
   \-linearSolver <void  at fluid.c:92>
     +-build_index <int  at fluid.c:10>
     \-setBoundry <void  at fluid.c:52>
       \-build_index <int  at fluid.c:10>
  \-advect <void  at fluid.c:183>
    +-build_index <int  at fluid.c:10>
    \-setBoundry <void  at fluid.c:52>
      \-build_index <int  at fluid.c:10>
+-c_velocitySolver <void  at fluid.c:341>
  +-addSource <void  at fluid.c:37>
  +-vorticityConfinement <void  at fluid.c:232>
   +-build_index <int  at fluid.c:10>
   +-abs
   +-calculate_curl <float  at fluid.c:132>
    \-build_index <int  at fluid.c:10>
   \-sqrt
  +-buoyancy <float  at fluid.c:143>
   \-build_index <int  at fluid.c:10>
  +-swap <void  at fluid.c:20>
  +-diffuse <void  at fluid.c:117>
   \-linearSolver <void  at fluid.c:92>
     +-build_index <int  at fluid.c:10>
     \-setBoundry <void  at fluid.c:52>
       \-build_index <int  at fluid.c:10>
  +-project <void  at fluid.c:288>
   +-build_index <int  at fluid.c:10>
   +-setBoundry <void  at fluid.c:52>
    \-build_index <int  at fluid.c:10>
   \-linearSolver <void  at fluid.c:92>
     +-build_index <int  at fluid.c:10>
     \-setBoundry <void  at fluid.c:52>
       \-build_index <int  at fluid.c:10>
  \-advect <void  at fluid.c:183>
    +-build_index <int  at fluid.c:10>
    \-setBoundry <void  at fluid.c:52>
      \-build_index <int  at fluid.c:10>
+-calculate_curl <float  at fluid.c:132>
  \-build_index <int  at fluid.c:10>
+-diffuse <void  at fluid.c:117>
  \-linearSolver <void  at fluid.c:92>
    +-build_index <int  at fluid.c:10>
```

```
    \-setBoundry <void  at fluid.c:52>
      \-build_index <int  at fluid.c:10>
+-linearSolver <void  at fluid.c:92>
  +-build_index <int  at fluid.c:10>
  \-setBoundry <void  at fluid.c:52>
    \-build_index <int  at fluid.c:10>
+-project <void  at fluid.c:288>
  +-build_index <int  at fluid.c:10>
  +-setBoundry <void  at fluid.c:52>
   \-build_index <int  at fluid.c:10>
  \-linearSolver <void  at fluid.c:92>
    +-build_index <int  at fluid.c:10>
    \-setBoundry <void  at fluid.c:52>
      \-build_index <int  at fluid.c:10>
+-setBoundry <void  at fluid.c:52>
  \-build_index <int  at fluid.c:10>
+-vorticityConfinement <void  at fluid.c:232>
  +-build_index <int  at fluid.c:10>
  +-abs
  +-calculate_curl <float  at fluid.c:132>
   \-build_index <int  at fluid.c:10>
  \-sqrt
```

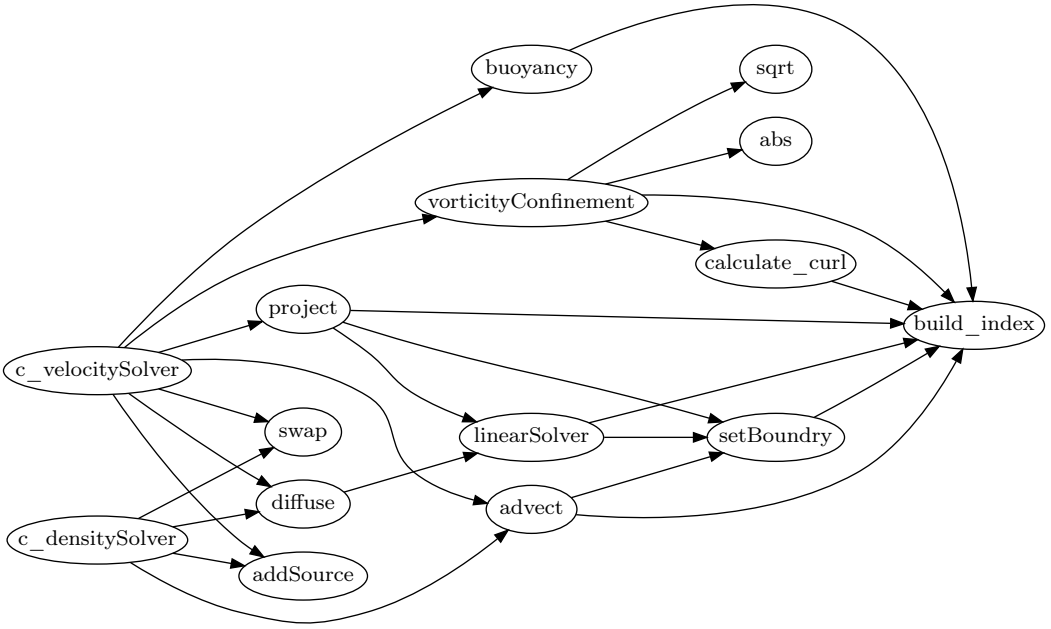


FIGURE 1 – Graphe d'appel du fichier *fluid.c*

3 Protocole expérimental

3.1 Théorie

3.2 Pratique

4 Optimisations et mesures

4.1 Déroulage de boucle

4.2 Vectorisation

4.3 Inlining

5 Conclusion

Acronymes

CPU Central Processing Unit, processeur central de l'ordinateur
RAM Random Access Memory
HT Hyper-Threading