Projet HPC : le modèle *shallow water*

Version du 21 février 2018

1 Présentation

1.1 Équations de *shallow water*

Le modèle "*shallow water*" ou "équations de Saint-Venant" permet de représenter l'écoulement d'un fluide homogène sur la verticale. Le système d'équation est relativement simple à poser mais leur résolution est toujours aujourd'hui un sujet de recherche. C'est pourquoi il est nécessaire de faire appel à un code informatique qui résout de façon approchée les solutions de ces équations. On parle alors de modèle numérique.

Malgré son apparente simplicité, ce modèle permet de représenter des phénomènes d'une grande diversité. C'est ce modèle qui permet de modéliser les ondes concentriques à la surface de l'eau lorsque vous y jetez un caillou. Le modèle est également utilisé pour étudier les courants dans les bassins d'aquacultures, les effets d'une rupture de barrage hydroélectrique ou d'une inondation. A plus grande échelle, c'est ce même modèle qui permet de représenter les courants dûs aux marées dans l'océan, ou la propagation dans l'océan d'un tsunami. Pour des exemples en vidéo, on pourra consulter par exemple : `https://www.youtube.com/watch?v=FbZBR-FjRwY`

Plus précisément, les équations de *shallow water* [1] décrivent la relation entre la vitesse d'un fluide dans le plan horizontal et la hauteur de ce fluide. Notre modèle est linéarisé, et défini avec les trois équations aux dérivées partielles suivantes :

$$\begin{aligned}
\frac{\partial u}{\partial t} &= -g^*.\frac{\partial h}{\partial x} + f.v - \gamma.u \\
\frac{\partial v}{\partial t} &= -g^*.\frac{\partial h}{\partial y} - f.u - \gamma.v \\
\frac{\partial h}{\partial t} &= -H.(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y})
\end{aligned} \tag{1}$$

Les champs u et v sont les composantes de la vitesse, et h, la variation de la hauteur de la colonne de fluide.

Les paramètres sont :

— g^*, la gravité réduite ;
— f, la force de Coriolis ;
— γ, un facteur de dissipation ;
— H, la hauteur moyenne de la colonne.

Le domaine 2D est une grille régulière de points pour lequels les champs sont alignés sur la grille Arakawa-C suivante :

```
h(i,j)     –   u(i,j)     –   h(i+1,j)
  |                            |
v(i,j)                       v(i+1,j)
  |                            |
h(i,j-1)   –   u(i,j-1)   –   h(i+1,j-1)
```

Sur cette grille, les équations continues sont discrétisées sous la forme :

$$\begin{aligned}
\frac{u_{i,j}^t - u_{i,j}^{t-1}}{\Delta t} &= -g^*.\frac{h_{i+1,j}^{t-1} - h_{i,j}^{t-1}}{\Delta x} + f.\frac{v_{i,j}^{t-1} + v_{i+1,j}^{t-1} + v_{i,j+1}^{t-1} + v_{i+1,j+1}^{t-1}}{4} - \gamma.u_{i,j}^{t-1} \\
\frac{v_{i,j}^t - v_{i,j}^{t-1}}{\Delta t} &= -g^*.\frac{h_{i,j}^{t-1} - h_{i,j-1}^{t-1}}{\Delta y} + f.\frac{u_{i-1,j-1}^{t-1} + u_{i,j-1}^{t-1} + u_{i-1,j}^{t-1} + u_{i,j}^{t-1}}{4} - \gamma.v_{i,j}^{t-1} \\
\frac{h_{i,j}^t - h_{i,j}^{t-1}}{\Delta t} &= -H.(\frac{u_{i,j}^{t-1} - u_{i-1,j}^{t-1}}{\Delta x} + \frac{v_{i,j+1}^{t-1} - v_{i,j}^{t-1}}{\Delta y})
\end{aligned} \tag{2}$$

1. https ://en.wikipedia.org/wiki/Shallow_water_equations

Pour attenuer les instabilités numériques de ce schéma, on applique un filtre faisant une moyenne pondérée aux champs u, v et h. Chaque point de grille correspondra donc à 6 champs.

1.2 Programme séquentiel

Le programme séquentiel qui vous est fourni calcule l'évolution de l'état d'une cloche d'eau centrée sur un nombre de pas de temps fixé par l'utilisateur.

Il peut prendre en argument les options ci-dessous.

— `-x`, première dimension de la grille
— `-y`, deuxième dimension de la grille
— `-t`, nombre de pas de temps
— `-dt`, pas de temps
— `-dx`, taille physique du domaine
— `-dy`, taille physique du domaine
— `-pcor`, force de Coriolis
— `-grav`, force de gravité
— `-dissip`, facteur de dissipation
— `-alpha`, coefficient du filtre
— `-hmoy`, hauteur moyenne du milieu
— `-hinit`, hauteur relative de la cloche initiale
— `-export`, active l'export de la hauteur filtrée au format binaire
— `-export-path`, chemin pour le fichier d'export
— `-h`, message d'aide et valeurs par défaut des options

L'export binaire produit un fichier `shalw_<x>x<y>_T<t>.sav`, où `<x>` et `<y>` sont les dimensions de la grille et `<t>` est le nombre de pas de temps. La visualisation de la simulation à partir du fichier binaire pourra être effectuée avec le script python `visu.py` :

```
$ ./visu.py shalw_<x>x<y>_T<t>.sav
```

Pour rappel, votre répertoire `HOME` est accessible depuis toutes les machines mais limité par votre quota d'espace disponible. Le répertoire `/tmp` est par contre local à votre machine courante, et correspond à des entrées-sorties sur le disque dur de la machine. Pour ce projet, on s'intéressera donc **uniquement** aux cas tests suivants.

— Cas 1, uniquement pour visualiser une simulation complète (en séquentiel uniquement, et avec écriture dans le répertoire `/tmp` du disque dur) :
```
$ ./bin/shalw --export --export-path /tmp/votre_login
```
Veillez à bien effacer le fichier généré dans `/tmp/votre_login` après utilisation.
— Cas 2, pour les mesures de performances en parallèle :
```
$ ./bin/shalw -x 8192 -y 8192 -t 20
```
Il est strictement interdit de générer un export pour ce cas. Si nécessaire ou pertinent, on pourra aussi s'intéresser à des grilles de dimensions plus petites (mais toujours sans export).
— Cas 3, pour la partie entrées-sorties parallèles uniquement :
```
$ ./bin/shalw -x 512 -y 512 -t 40 --export --export-path ~/.
```

Si vous souhaitez tester d'autres cas, vous devrez en discuter au préalable avec votre chargé de TDTP.

2 Travail à effectuer

Le travail à effectuer se décompose en 2 parties.

2.1 Partie 1 : parallélisation MPI

Dans un premier temps, on considère que votre code ne fait pas de sauvegarde de la grille dans un fichier.

1. Deux décompositions sont a priori possibles pour distribuer la grille sur les processus :
   — une décomposition par bandes,
   — une décomposition par blocs (rectangulaires).
   Donner les avantages et les inconvénients de chaque décomposition. Ecrire un programme MPI qui effectue cette simulation en parallèle avec les deux décompositions, et déterminer quelle est en pratique la meilleure décomposition pour les ressources matérielles dont nous disposons.
   On s'appuiera ici sur le cas 2 et on utilisera au plus 16 noeuds.
2. Améliorer la solution retenue en introduisant un recouvrement des communications par le calcul.
3. On considère désormais que votre code doit pouvoir faire des sauvegardes de la grille dans un fichier. Pour cela, on ne considérera que 4 processus et uniquement le cas 3 : il est strictement interdit de travailler avec des cas plus gros (qui risqueraient de causer des problèmes sur le NFS des salles de TP).
   Par ailleurs, vous devrez utiliser votre propre compte pour sauvegarder le fichier d'export (de taille < 100 Mo), comme indiqué dans la ligne de commande du cas 3.
   Utiliser MPI-IO pour simuler des entrées/sorties parallèles afin que le fichier résultat soit écrit directement par tous les processus.
   Les fonctionalités MPI-IO sont clairement présentées et détaillées avec des exemples (en Fortran 90) dans le document de l'IDRIS disponible à cette adresse : `http://www.idris.fr/data/cours/parallel/mpi/IDRISMPI.pdf` (section 8).
   On pourra aussi s'intéresser au recouvrement des entrées/sorties par du calcul, et observer le gain obtenu en pratique (en faisant éventuellement varier la fréquence de sauvegarde des grilles sur disque : à savoir sauvegarder une grille tous les X pas de temps).

2.2 Partie 2 : parallélisation MPI+OpenMP et SIMD

2.2.1 Prise en compte de la hiérarchie mémoire

Vérifier les accès mémoire effectués par le code. Sont-ils adaptés à la hiérarchie mémoire de vos machines ?

2.2.2 Parallélisation MPI+OpenMP

Mettre en place une parallélisation hybride MPI+OpenMP permettant de mieux exploiter les processeurs multicœur dont vous disposez. On pourra alors comparer les performances d'une implémentation "MPI+OpenMP" et d'une implémentation "MPI pur" (sans multi-threading), et ce pour un même nombre de cœurs CPU.

2.2.3 Parallélisation SIMD

A l'aide d'*intrinsèques* et/ou de directives OpenMP, vectorisez votre code.

On pourra d'abord mesurer le gain de performance offert par les unités SIMD pour un code séquentiel, avant de le mesurer pour votre meilleur code parallèle.

Si vous avez implémenté la vectorisation avec les *intrinsèques* et avec les directives OpenMP, quelle est la différence de gain de performance (par rapport à une exécution scalaire) entre ces deux implémentations ?

3 Travail à remettre

Pour chacune des deux parties, vous devrez remettre le code source, sous la forme d'une archive `tar` compressée et nommée suivant le modèle `projet_HPC_nom1_nom2.tar.gz`. L'archive ne doit contenir aucun exécutable, et les différentes versions demandées devront être localisées dans des répertoires différents. Chaque répertoire devra contenir un fichier `Makefile` : la commande `make` devra permettre de lancer la compilation, et la commande `make exec` devra lancer une exécution parallèle représentative avec des paramètres appropriés. Un fichier `Makefile` situé à la racine de votre projet devra permettre (avec la commande `make`) de lancer la compilation de chaque version.

A la fin du projet, vous devrez remettre un rapport au format `pdf` (de 5 à 10 pages, nommé suivant le modèle `rapport_HPC_nom1_nom2.pdf`) présentant vos algorithmes, vos choix d'implémentation (sans code source), vos résultats (notamment vos efficacités parallèles) et vos conclusions pour les deux parties. L'analyse du comportement de vos programmes sera particulièrement appréciée. Les machines n'étant pas strictement identiques d'une salle à l'autre, on précisera dans le rapport la salle utilisée pour les tests de performance.

4 Quelques précisions importantes

— Le projet est à réaliser par binôme.
— Pour les étudiants en master informatique, vous **devez** lire le document intitulé « Projet HPC : conditions d'utilisation d'OpenMPI ». Vous veillerez notamment à n'utiliser qu'une salle à la fois pour vos tests, et vous n'oublierez pas de tuer **tous** vos processus sur **toutes** les machines utilisées à la fin de vos tests.
— Le code de la première partie («Parallélisation MPI»), accompagné des slides de votre soutenance au format `pdf` (et donc sans animations) (prévoir une soutenance de 10 minutes, suivie de 5 minutes de questions), est à remettre au plus tard le dimanche 08 / 04 / 2018 à 23h59 (heure locale). Les soutenances de présentation de la première partie auront lieu lors de la séance de TDTP du lundi 09 / 04 / 2018 ou du mardi 10 / 04 / 2018.
Le code de la seconde partie («Parallélisation MPI+OpenMP et SIMD») et le rapport final sont à remettre au plus tard le dimanche 20 / 05 / 2018 à 23h59 (heure locale).
— Les remises se feront par courriel à :
  — `szaidi@aneo.fr` et `wkirschenmann@aneo.fr` pour les étudiants du master informatique,
  — `cmakassikis@aneo.fr` et `smoustafa@aneo.fr` pour les étudiants de MAIN.
— En cas d'imprévu ou de problème technique commun, n'hésitez pas à nous contacter pour que nous puissions vous proposer une solution ou une alternative.