AMSI03-3

Nikita Matckevich

14.02.2021

CUDA et MPI peuvent être assez facilement combinés et utilisés ensemble.

Pour ce faire, dans le code, nous devons d'abord initialiser le contexte MPI. Ensuite, dans chacun des processus MPI, on peut appeler les fonctions déjà écrites en PoissonCUDA `...Wrapper`. Des variables supplémentaires sont nécessaires pour utiliser ces fonctions, telles que `double* partialDiff, diff` le champ mutable `h_synchronized` et l'addresse `d_u` de vecteur de données sur GPU dans le class `values`. Nous allons simplement les copier dans le code PoissonMPI de code CUDA.

Nous allons ajouter tous les fichiers supplementaires du code CUDA: `dim.hxx, iteration.hxx, variation.hxx`, ainsi que le dossier `/cuda`.

Les calculs déplacés vers GPU sont:

1. les boucles 3d pour initialiser les champs dans `values.cxx:init...`
2. les boucles 2d pour définir les valeurs du bord `values.cxx:boundaries`. La boucle externe d'une taille 3, ainsi que les expressions conditionnelles `if ...` ne sont pas concernées.
3. la boucle principale 3D dans la fonction `scheme.cxx:iteration_domaine`

Dans cette configuration, presque toutes les boucles d'une taille $\neq O(1)$ sont effectuées avec GPU. Il nous reste seulement des boucles $O(N^2)$ dans la fonction `synchronize` à traiter. La fonction de ces boucles est d'arranger les valeurs aux limites de sous-domaines dans les tableaux `bufferIn` et `bufferOut` et ensuite échanger les données entre dexu processus à l'aide de MPI. Théoriquement on peut effectuer cette copie des données sur GPU, parce que les opérations sont massivement concurrentes et très facilement parallélisables. Mais il s'avère que pour effectuer ces opérations sur GPU il faut (pour la plupart d'architectures) effectuer une copie de memoire host vers la mémoire graphique et vice-versa. Ces transferts (et surtout le transfert inverse) peuvent subir d'un lenteur, donc c'est à mésurer si la parallélisation de ces boucles est nécessaire.

Sauf les boucles $O(N^2)$ et $O(N^3)$ que nous avons déjà discuté, les autres parties du code restent les memes comme dans le code MPI.

Finalement, nous obtenons le code de l'algorithme hybride. Tous les transferts de données entre 2 processus (entre 2 CPUs) sont effectués à l'aide de MPI. Les transferts directes de GPU vers un autre GPU où un CPU autre que le host

ou vice versa ne sont jamais effectués. Les seules procedures faites sont les transferts entre le GPU vers son host (à l'aide des fonctions `copyDeviceToHost` et `copyHostToDevice`) et le transfert entre 2 CPUs si besoin (MPI). La fonction `copyDeviceToDevice` est en fait appelée seulement lors de copie de données dans le meme processus. Donc, les deux vecteurs de données dans ce cas "appartiennent" au meme GPU (en termes d'addressage).