AMS-I03 Programmation hybride et multi-coeurs
Compte Rendu TP3

El Herichi Hafsa

14 Février 2021

Introduction

Par le biais de ce TP, nous nous intéressons à la programmation hybride MPI-Cuda. Il s'agira donc de créer une version de code prenant en compte l'utilisation de CPU et GPU à partir de codes MPI et Cuda afin que cette version puisse être utilisée sur une machine parallèle dont chaque noeud contient une carte graphique.

Pour ce faire, nous allons nous appuyer sur le code de la version MPI en y intégrant les fonctionnalités de Cuda. Il va donc falloir modifier plusieurs fichiers du code que nous avons.

Fichiers non-modifiés

Même si le but est de modifier le code de la version MPI, il existe quelques fichiers communs aux deux versions qu'on n'aura donc pas besoin de modifier.

Le premier étant le fichier **timer.hxx**, qu'on gardera. Le second est le fichier **user.hxx** qui n'est présent que pour le code MPI et par conséquent n'a pas besoin de modifications. Egalement, le fichier **parameters.cxx** n'a pas besoin de modifications.

Fichiers modifiés

Le reste des fichiers doit être modifié afin d'inclure les fonctionnalités Cuda avec celles de MPI. Pour chaque fichier, nous allons détailler quelles ont été les modifications faites et quel est leur but.

Fichier main.cxx

Concernant le fichier **main.cxx**, on aura plusieurs ajouts légers à faire.

On commence par inclure les AddTimer présents dans le code Cuda et non présents dans celui de MPI, de sorte que le code MPI-Cuda contienne les lignes suivantes :

```
T_AllocId = AddTimer("alloc");
T_CopyId = AddTimer("copy");
T_InitId = AddTimer("init");
T_IterationId = AddTimer("iteration");
T_CommId = AddTimer("comm");

T_VariationId = AddTimer("variation"); // ligne ajoutee au code MPI
T_FreeId = AddTimer("free"); // ligne ajoutee au code MPI

AddTimer("total");
```

Pour le reste, on gardera la version du code MPI, il n'est pas utile de rajouter quoique ce soit.

Fichier values.cxx

Pour ce qui est du fichier **values.cxx**, on aura plusieurs changements à faire puisque la version CUDA utilse un double pointeur afin de réprésenter la solution à un instant donné au lieu d'un simple device pour le code MPI. Il va donc falloir inclure les fonctions qu'utilise le code CUDA au lieu de celui utilisé par MPI.

Notamment, il faudra ajouter le double pointeur utilisé par CUDA, dont un sera sur le GPU et l'autre sur le CPU. Aussi, il faudra inclure la variable $h_synchronize$ au sein du code MPI afin de permettre de vérifier si les deux vecteurs du double pointeur sont égaux.

Pour ce faire, on va devoir inclure dans la fonction **Values(Parameters prm) :** m_p**(prm)**, les quelques lignes utilisées dans CUDA dans le code MPI. Cela nous donne :

```
d_u = allocate(nn); //   ajouter au code MPI

Timer & T = GetTimer(T_AllocId); T.start();

h_u = new double[nn]; //   ajouter au code MPI
T.stop();

h_synchronized = false;
```

au lieu d'avoir :

```
  m_u.resize(nn);
```

Il faudra également ajouter les fonctions **Values()** et **zeros()** présentes dans la version CUDA au sein du code MPI pour assurer le bon fonctionnement des fonctionnalités CUDA.

Pour ce qui est des fonctions **init()** et **boundaries()**, on va modifier les triples boucles utilisées par MPI et les remplacer par celles utilisées par CUDA et qui font appel à des fonctions Wrapper. Il faut donc également inclure le fichier cuda dans lequel sont répertoriées toutes les fonctions auxquelles fait appel le code CUDA.

Et enfin, pour toutes les fonctions qui restent, notamment **print(std : :ostream f) const , swap(Values other), plot(int order) const** et **operator = (const Values other)** , on utilise celles du code CUDA puisqu'elles traitent toutes le double pointeur au lieu du simple device de MPI.

Fichier scheme.cxx

En ce qui concerne le fichier **scheme.cxx**, il ya quelques fonctions auxquelles on ne touchera pas puisqu'elles sont identiques dans les deux versions CUDA et MPI. Ces fonctions sont : **present(), getOutput()** et **setInput(const Values u)**.

Cependant, pour les fonctions qui restent, nous allons également procéder à quelques changements puisque CUDA utilise des structures de données permettant de calculer la différence entre les GPU et donc il va falloir inclure cela au sein du code MPI.

Tout d'abord, la fonction **Scheme()** du code MPI doit être remplacée par celle du code CUDA puisqu'à chaque variation, il est essentiel de calculer les nouveaux vecteurs diff et partialdiff pour que les calculs ne soient pas faussés.

La difficulté pour les fonctions qui restent est de calculer la différence entre les variations sur un GPU, les partialDiff sur un autre mais également de calculer la somme totale des partialDiff pour chaque variation sur un CPU.

Pour cela, nous allons utiliser la fonction

```
reduce(const double *u, double *partialDiff, int n)
```

dans cuda qui va permettre de calculer la somme des partialDiff simplement.