AMS-I03 - TP3

Quentin Goepfert

1 Introduction

On possède une version MPI d'un code qui résout l'équation de Poisson et souhaite étudier les modifications afin que les calculs soient faits sur GPU. Ainsi, on va détailler toutes les modifications du code qui permettraient de faire un code avec un calcul sur GPU.

Tout d'abord identifions ce qui doit être calculé sur GPU en reprenant la structure du code MPI :

Result: Solution après N étapes u_N de l'équation de poisson
Initialisation :
Récupération des paramètres
Initialisation MPI
Allocation mémoire de u sur chaque processus MPI
Sur chaque processus :
for $i \leftarrow 0$ **to** $itMax$ 1 **do**
  Synchronisation des processus MPI
  Echange des valeurs aux bords aux voisins MPI correspondants
  Calcul de u_i à partir de ses propres valeurs (et des bords des voisins)
end
Reconstruire la solution à partir de tous les processus MPI

Algorithm 1: Structure de l'algorithm MPI

Nous voulons faire les calculs sur le GPU de chaque processus MPI. Ainsi, on doit correctement allouer la mémoire et initialiser les valeurs sur le GPU de chaque processus MPI, faire le calcul sur le processus, renvoyer les valeurs aux bords au CPU du processus MPI, l'échanger avec ses voisins, puis faire redescendre les données : à partir du CPU du processus MPI qui a obtenu la nouvelle valeur des bords, les envoyer à leur GPU, recalculer l'itération...

Ainsi, on va avoir la structure suivante (les étapes modifiées sont en gras) :
Result: Solution après N étapes u_N de l'équation de poisson
Initialisation :
Récupération des paramètres
Allocation mémoire de u sur chaque **GPU de chaque processus MPI**
Sur chaque processus :
for $i \leftarrow 0$ **to** $itMax$ **do**
  Envoi des valeurs aux bords de la mémoire du GPU vers la mémoire du CPU du processus MPI
  Synchronisation des processus MPI
  Echange des valeurs aux bords aux voisins MPI correspondants
  Envoi des nouvelles valeurs aux bords de la mémoire du CPU du processus MPI vers la mémoire du GPU
  Calcul de u_i **sur GPU** à partir de ses propres valeurs (et des bords des voisins)
end
Récupération de la solution calculée sur GPU sur le CPU du processus MPI Reconstruire la solution à partir de tous les processus MPI

Algorithm 2: Structure de l'algorithm MPI/Cuda

2 Détails du code

Regardons maintenant plus en détail dans le code, quelles fonctions doivent être modifiées/créées.
Tout d'abord le $<vector>$ m_u ne sera plus le tableau qui contient les valeurs de u du processus MPI à chaque étape. Ces valeurs seront initialisées sur GPU. Ainsi, on doit rajouter l'initialisation sur GPU des valeurs de u :

$$double * m\_u\_GPU$$

$$cudaMalloc(\&m\_u\_GPU, N * sizeof(double));$$

On peut décider d'initialiser directement les valeur de m_u sur GPU ou alors garder le même code d'initialisation mais alors on doit utiliser un

$$cudaMemcpy(m\_u\_GPU, m\_u, N * sizeof(double), cudaMemcpyHostToDevice);$$

pour remplir les valeurs initiales de u sur le GPU (on effectue la commande précédente après avoir rempli le vecteur $m\_u = u\_o$ que l'on trouve dans le main).

Ensuite, on doit créer la fonction d'itération sur GPU. Elle sera appelée du CPU et exécutée sur GPU et remplacera la fonction *iteration_domaine()*. Nommons-la *iteration_domain_GPU()*. Sa structure sera :

$$\_global\_\_void\ iteration\_domain\_GPU(double*\ m\_u, double*\ m\_v,$$
$$double*\ du\_sum\_local, double\ m\_dt,$$
$$double*m\_dx,\ double*m\_xmin, int\ imin,$$
$$int\ imax, int\ jmin, int\ jmax, int\ kmin, int\ kmax);$$

et elle fera le calcul précédemment fait par *iteration_domain()*, mettra à jour *du_sum_local* (par une réduction sur GPU) qui sera réutilisé pour le calcul de *du* sur un processus MPI après la remontée de valeur de *du_sum_local* sur le CPU. On remarque qu'on doit aussi initialiser *m_xmin* et *m_dx* sur le GPU.

Il faut maintenant rapatrier les valeurs des bords du GPU vers le CPU. Il faut une fonction qui permette de récupérer les valeurs des voisins du GPU et de les rapatrier sur le CPU (grâce à des $cudaMemcpy(\cdot,\cdot,\cdot,$ cudaMemcpyDeviceToHost);).

Sinon plus simplement, on peut rapatrier tout le vecteur *m_u* en entier avec la commande :

$$cudaMemcpy(m\_u, m\_u\_GPU, N*sizeof(double), \text{cudaMemcpyHostToDevice});$$

Remarque : Le coût est légèrement supérieur au rapatriment des données des bords seuls mais cela ne nécessite pas de fonction supplémentaire...

On s'assure que le GPU a fini son calcul par une barrière puis on appelle ensuite la fonction *synchronize()* (qui permet l'échange des bords avec les autres processus MPI, et est déjà présente dans la version MPI seule), puis on rapatrie les valeurs aux bords obtenues des voisins par la commande inverse à la précédente :

$$cudaMemcpy(m\_u\_GPU, m\_u, N*sizeof(double), \text{cudaMemcpyDeviceToHost});$$

On recommence une boucle de l'algorithme en recalculant les valeurs sur GPU.