AMS-I03 TP3
Modèles et techniques en programmation parallèle hybride et multi-cœurs

Dongshu LIU

1 Rappelle du problème

Chercher une solution $u : (x,t) - - > u(x,t)$, où $x \in \Omega = [0,1]^3$ et $t \geq 0$ qui vérifie :

$$\frac{\partial u}{\partial t} = \nabla u + f(x,t) \tag{1}$$
$$u(x,0) = g(x) \qquad x \in \Omega \tag{2}$$
$$u(x,t) = g(x) \qquad x \in \partial\Omega, t > 0 \tag{3}$$

où f et g sont des fonctions donées.

2 Version hybride de MPI et Cuda

2.1 Structure de MPI_Cuda

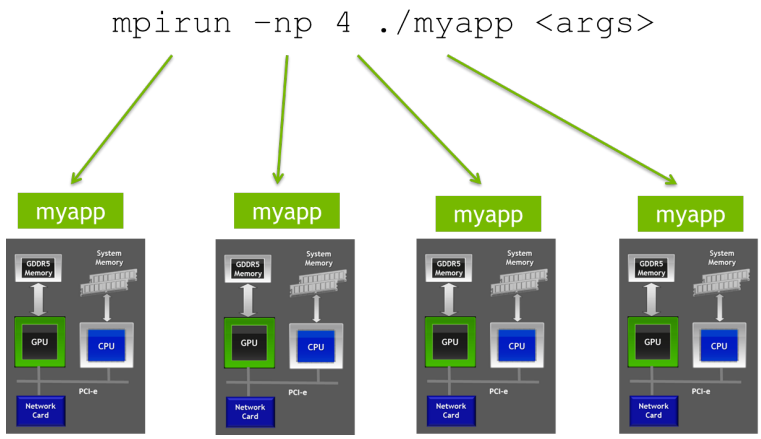

FIGURE 1 – Structure de MPI_Cuda[1]

[1] NIVIDA Developer Blog

2.2 Fonction concernée

Pour réaliser la version hybride de MPI-Cuda, il faut que la partie cuda puisse cuculer dans chaque processus de MPI.

parameters.cxx

On garde "parameters.cxx" de la version de MPI. Dans cette partie MPI sépare le nombre de calcul selon le nombre de processus.

Dans sous-domaine courant, *imin(i)* et *imax(i)* sont l'indice de premier point intérieur et l'indice de dernière point intérieur dans la direction i réspectivement ; *xmin(i)* et *xmax(i)* sont le coordonnée minimale et maximale dans la direction.

imin_global(i) et *imax_global(i)* sont le premier indice et le dernier indice dans toute la domaine.

values.cxx

On garde "values.cxx" de la version de Cuda.

Chaque processus va demander GPU à initialiser une mémoire de taille "nn" $(imax(1) - imin(1) - 3) * (imax(2) - imin(2) - 3) * (imax(3) - imin(3) - 3)$, dont le pointer est nomé par "d_u".

Pour printer les valeurs dans GPU, d'abord on récupere le data de GPU "d_u" à CPU h_u, en utilisant la fonction copyDeviceToHost, et après les printer dans CPU. (*Values : :print*)

scheme.cxx

```
Scheme::Scheme{
double lx[3];
...
lx[i] = 1.0/(m_dx[i]*m_dx[i]);
setDims(m_n, m_xmin, m_dx, lx);
diff = NULL;
partialDiff = NULL;
...
}
```

— Copier les valeurs de "m_n", "m_xmin","m_dx","lx" de CPU à GPU (constante memory) ;

— Initialiser la valeur de variation partiale (sum de tous les blocks), et la valeur de variation totale dans le processus (sum de tous les grids) à "NULL".

```
Scheme::iteration::domaine{
double du_sum, du_sum_local = 0.0;
...
iterationWrapper(m_v, m_u, m_dt, m_n,
                 imin, imax, jmin, jmax, kmin, kmax);
m_v.synchronized(false);
du_sum_local = variationWrapper(m_u, m_v,
                       diff, partialDiff,
                       m_n[0]*m_n[1]*m_n[2]);
MPI_Allreduce(&du_sum_local, &du_sum, 1, MPI_DOUBLE, MPI_SUM, m_P.comm());
return du_sum;
}
```

— Initialiser la valeur de variation partitiale (dans chaque processus courant) et la valeur de variation totale (dans tous les processus) à 0 ;

— "iterationWrapper" Calcul dans GPU avec les valeurs initialisé dans "values.cxx" ;
— "synchronized" Échanger les valeux aux frontières de chaque processus ;
— "variationWrapper" Calculer la variation dans GPU, et envoyer le résultat calculé à CPU ;
— "$MPI\_Allreduce$" Faire l'addition de tous les processus, et rentrer par la valeur de "du_sum".