HPC - TME 3 - Diffusion de la chaleur

Diez Marie

Table des matières

1 Introduction

Ce TP consiste à paralléliser une simulation numérique qui détermine la température atteinte par un CPU récent (un AMD EPYC « rome ») sur lequel on pose une plaque d'aluminium de 15cm×12cm×0.8cm. La face z=0 du dissipateur est maintenue de manière forcée à T= 20°C, et on suppose que l'ensemble du dissipateur est initialement à cette température. On suppose que l'air ambiant est aussi à 20°C. L'idée générale consiste à allumer le processeur au temps t= 0 et à simuler la diffusion de la chaleur dans le dissipateur jusqu'à ce qu'un régime à peu près stationnaire soit atteint.

2 Parrallélisation avec MPI

2.1 Equilibrage de charge statique

J'ai réalisé l'équilibrage de charge statique en prenant en compte que chaque processeur s'occupe de N couches (du dissipateur) de calcules déterminées de la façon suivante :

$$o\_local = o\_global/Nb\_proc$$

Avec o_global étant le nombre totale de couches.

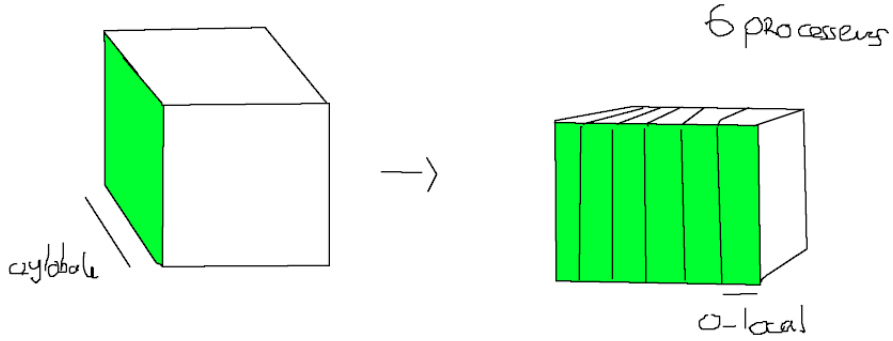

FIGURE 1 – Représentation des couchent

Je n'ai pas mis en place d'équilibrage dynamique comme pour le TME 2 par manque de temps, mais le principe est le même un équilibrage de type Patron/ouvrier : dès qu'un processeur ouvrier à fini son travail il demande plus au patron.

2.2 Les opérations collectives

2.2.1 Reduce

J'ai utilisé l'opération collective *Reduce* pour la variable $Tmax$ qui n'a besoin d'être accessible que par le processeur "père" le processeurs 0 dans mon cas. Tous les threads font appel à cette fonction.

2.2.2 AllReduce

J'ai utilisé l'opérateur *AllReduce* sur la variablea $delta\_T$ car tous les threads ont besoin de sa valeur pour tester leur convergence.Tous les threads font appel à cette fonction.

2.2.3 Gather

J'ai utilisé l'opérateur *Gather* sur les tableaux T_local qui correspondent aux tableaux de chaleur calculé par chaque thread, il faut alors les assemblés dans un tableau que j'ai nommé T_global à l'aide de l'opérateur *Gather*, uniquement le processeur père nécessite de connaître la valeur de T_global pour l'affichage de la carte de chaleur.

2.3 Echange des halos

Les processeurs ont besoin des valeurs adjacentes pour calculé en chaque point la diffusion de la chaleur. Lorsque l'on parrallélise le programme les threads perdent cette information sur les bords de leurs tableaux initiale. Il faut alors mettre en place une communication MPI pour l'échage de ces halos.

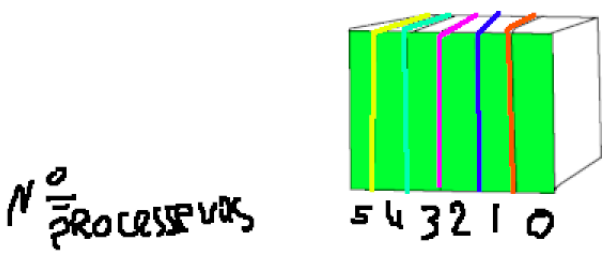

FIGURE 2 – Représentation des halos

Pour préciser le fonctionnement voici une répresentation de l'échange :

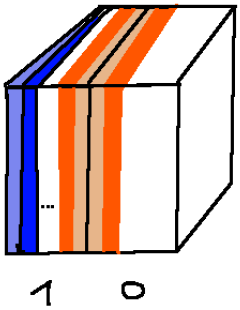

FIGURE 3 – Représentation des halos entres 2 processeurs

Les couleurs pleines foncées représentent les halos des processeurs, les couleurs pleines et claires représentent l'espace créer par le processeur dans son tableau pour acceuillir les halos des voisins.

- Processeur 0
  Il n'a besoin d'envoyer que son halo de gauche avec le processeur 1
  Il n'a besoin de recevoir que le halo de droite du processeur 1
- Processeur 1
  Il doit envoyer son halo de droite avec le processeur 0 et son halo de gauche avec le processeur 2.
  Il doit recevoir le halo de gauche du processeur 0 et le halo de droite du processeur 2.
- ...
- Processeur 5
  Il n'a besoin d'envoyer que son halo de droite avec le processeur 4
  Il n'a besoin de recevoir que le halo de gauche du processeur 4

2.4 Fonctionnement du programme

Les pocesseurs initialisent leurs tableaux de température local T_local et R_local (nécessaire pour le calcule de diffusion de la chaleur de la fonction do_xy_plane pour ne pas écraser les valeurs de T_local) à la taille $n * m * o\_local$ avec $n * m$ la taille d'une couche.
Avant de calcule les nouvelles valeurs de T_local les processeurs échangent leurs halos comme expliqué précédement, avec les fonctions MPI_Send et MPI_Recv. Par la suite ils font la mise à jour de leurs tableaux de températures avec la fonction do_xy_plane, il faut faire attention aux indices données à cette fonction notamment pour aller chercher les données dans T_local et R_local. Une fois la mise à jour faite les processeurs vérifient leur convergence à l'aide de $delta\_T$ calculé par chaque processus et additionné ensemble grâce à l'opération *Allreduce* vu précédemment. Il calcule aussi leur maximum local avant de faire une opération MPI_MAX avec *Reduce* pour trouver le maximum global. A la fin on met en place une opération *Gather* pour assembler tous les tableaux locaux en un tableaux global donnée au processus 0 pour permettre l'écriture dans un fichier *.txt* pour l'affichage de la carte de chaleur.

3 Performance

Les tests sont effecutés sur un processeur *Ryzen 5 3500U* pour cette commande :
mpicc heatsink_para.c -o heatsink_para -lm mpirun –n 30 –oversubscribe –mca opal_war_on_missing_libcuda 0 ./heatsink_para > steady_state.txt 150 8 120

3.1 Version séquentielle

Le programme termine en environ 40min

3.2 Version parallèle

Le programme termine en environ 20min pour $n = 30$ 30 processus lancé par node suivant *mpirun*.
Je n'ai pas pris le temps de faire plus de test convernant la valeur de n. Ce qu'il faudrait faire dans le cardre d'un travail plus pertinant.

4 Résultat

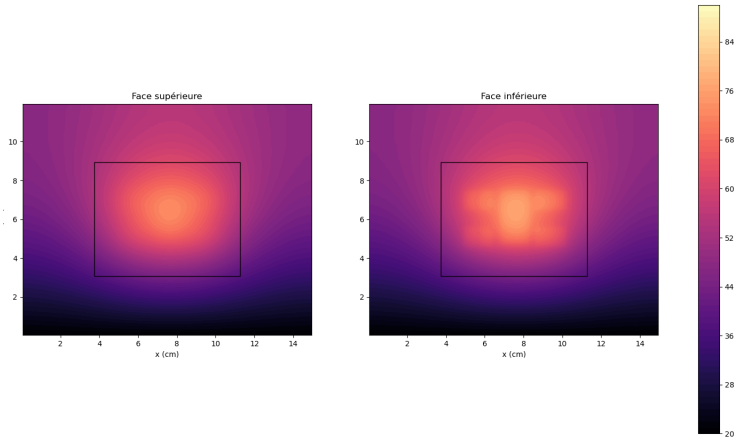

FIGURE 4 – Résultat de la simulation numérique

5 Code

Le code est sur Github : https://github.com/MarieDiez/heatsink