Modèles et techniques en programmation parallèle hybride et multi-cœurs

Travail pratique 3

Marc Tajchman

CEA - DEN/DM2S/STMF/LDEI

19/12/2024

Travail pratique 3

On part de deux codes qui calculent une solution approchée du problème suivant :

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

$$\begin{aligned} \frac{\partial u}{\partial t} &= \Delta u + f(x,t) \\ u(x,0) &= g(x) && x \in \Omega \\ u(x,t) &= g(x) && x \in \partial\Omega, t > 0 \end{aligned}$$

où f *et* g *sont des fonctions données.*

On utilise des différences finies pour approcher les dérivées partielles et on découpe Ω en $n_0 \times n_1 \times n_2$ subdivisions.

Codes de départ

Récupérer et décompresser le fichier `TP3_incomplet.tar.gz`.

Cette archive contient 2 sous-répertoires :

- la version séquentielle (`PoissonSeq`),
- une version incomplète, à paralléliser avec Cuda (`PoissonCuda`),
  L'étudiant pourra choisir de compléter le code avec des instructions Cuda, ou un mélange d'instructions Cuda et Thrust.

La version séquentielle sert à produire des résultats et temps d'exécution de référence.

Travail à réaliser

Dans les fichiers sources de `PoissonCuda`, vous trouverez des commentaires qui commencent par "A completer" et qui se trouvent dans les fichiers `scheme.cu` et `values.cu`.

Ces commentaires indiquent ce par quoi il faut les remplacer :

- un noyau Cuda dans `scheme.cu` qui calcule une solution à partir de la solution au temps précédent,
- la réservation/libération mémoire pour la classe `Values` sur CPU et GPU dans `values.cu`.

On pourra s'aider des sources C++ dans le répertoire PoissonSeq et d'autres noyaux Cuda existants dans `PoissonCuda`.

Travail à réaliser (2)

Cette partie s'adresse à ceux qui choisissent d'utiliser Thrust dans le code (le travail est plus conséquent)

En plus de ce qui est demandé, il faudra utiliser

- à la place des pointeurs double * sur CPU, des variables de type thrust::host_vector, et
- à la place des pointeurs double * sur GPU, des variables de type thrust::device_vector

D'autre part, on pourra utiliser les fonctions de sommation de thrust pour simplifier la fonction qui calcule la variation entre 2 solutions (fichier variation.cu)

On pourra s'aider des exemples thrust vus en cours et de la référence en ligne sur thrust, par exemple https://nvidia.github.io/cccl/thrust/api.html.

Sur le cluster Cholesky (où Cuda est installé et les nœuds de la partition "gpu" contiennent des cartes graphiques)

Pour compiler et exécuter

- dans le répertoire `PoissonSeq`, tapez :

  ```
  python submit_run.py
  ```

- dans le répertoire `PoissonCuda`, tapez :

  ```
  python submit_run.py
  ```

`submit_run.py (re)compile avant d'exécuter`

Les affichages sont dans le fichier `output.txt`.

Si vous n'utilisez pas le cluster Cholesky, il faut utiliser une machine

- avec une carte graphique compatible (ç-à-d. une carte Nvidia), et
- où le toolkit Cuda est installé (https://developer.nvidia.com/cuda-downloads)

Pour compiler et exécuter

- dans le répertoire `PoissonSeq`, les 2 commandes sont :

```
python build.py
./install/release/PoissonSeq
```

- dans le répertoire `PoissonCuda`, les 2 commandes sont :

```
python build.py
./install/release/PoissonCuda
```

Envoyez votre code source (dans une archive compressée) par mail à marc.tajchman@cea.fr **si possible pour le 14/2/2025, sinon au plus tard le 21/2/2025**.