Projet IT390
Programmation des accélérateurs

Deux sujets sont proposés, pour des groupes de 2 personnes vous pouvez choisir l'un ou l'autre, sinon il faut faire les deux sujets. Dans les deux cas, l'évaluation se fera sur la base d'un rapport à me renvoyer à l'adresse cedric.augonnet@gmail.com avec une archive du code associé. **Ne pas hésiter à me solliciter par mail si vous avez des questions !**

Sujet A : Problème des n-corps

Les simulation à n-corps permettent de calculer, de manière approchée, le mouvement de d'objets en interaction les uns avec les autres, par exemple sous l'influence de la gravitation.

Le code nbody.c simule donc le comportement de n particules de la manière suivante :

- Les positions et les vitesses des corps sont initialisées de manière aléatoire (libre à vous de trouver une meilleure distributions si vous avez des âmes d'artistes),
- A chaque pas de temps :
    - Pour chaque corps, on calcule la force gravitationnelle exercée par l'ensemble des autres corps sur elle,
    - On met à jour la vitesse du corps en fonction des forces qui s'applique sur lui,
    - On met à jour la position du corps en fonction de sa vitesse et de sa position.

Notez que le carré de la distance qui intervient lors du calcul de la force gravitationnelle entre deux corps est ici augmenté d'une constante de petite taille, afin d'éviter une division par zéro lorsque les corps sont très proches. La masse des corps est ici supposée égale à 1.

A chaque pas de temps, le code affiche le temps de calcul depuis le dernier pas de temps, ainsi que le nombre d'interactions par secondes et une estimation très grossière du nombre d'opérations flottantes par seconde.

Par défaut, le nombre de corps est positionné à 16384, mais ce nombre peut être modifié en donnant un argument du binaire. Un second argument permet de contrôler le nombre d'itérations. Par exemple, la commande "./nbody 1024 100" permettra de simuler l'interaction de 1024 corps sur 100 pas de temps.

Les objectifs sont les suivants :

1. Portez ce code en OpenACC.
2. Portez ce code en CUDA (de manière relativement naive).
3. Analysez brièvement les performances du code avec l'outils pgprof ou nvprof afin de déterminer quels étapes sont les plus couteuses.
4. Commentez l'utilisation de la structure de données `ParticleType` sur un GPU, et adaptez le code CUDA en conséquence.

5. En vous inspirant des techniques utilisées pour le produit matriciel, améliorez le taux de réutilisation de données.
6. Bonus : Ajoutez la possibilité de suivre la position des corps (ou d'un sous ensemble) au cours du temps. Une fonction dump est donnée à titre illustratif.

Analysez l'impact sur les performances de ces différentes optimisations. Une parallélisation du code en OpenMP pourra éventuellement vous aider à valider vos algorithmes, et à comparer de manière raisonnable les performances obtenues sur CPU et sur GPU.

Sujet B : Résolution de l'équation de la chaleur en 1D

Le problème consiste ici à résoudre l'équation de la chaleur sur une grille cartésienne 1D, avec une méthode d'Euler implicite en temps, couplée à un schéma de type différences finies centrées d'ordre 2 en espace. L'objectif est d'utiliser des accélérateurs pour accélérer un solveur implicite, par exemple à l'aide de bibliothèque d'algèbre linéaire creuse.

On cherche donc la valeur de T, solution du problème suivant pour x dans [0,N-1], et en fixant les valeurs de T pour x = 0 et x = N-1.

$$\frac{\partial T}{\partial t} = c\frac{\partial^2 T}{\partial x^2}$$

En notant **A** la matrice suivante,

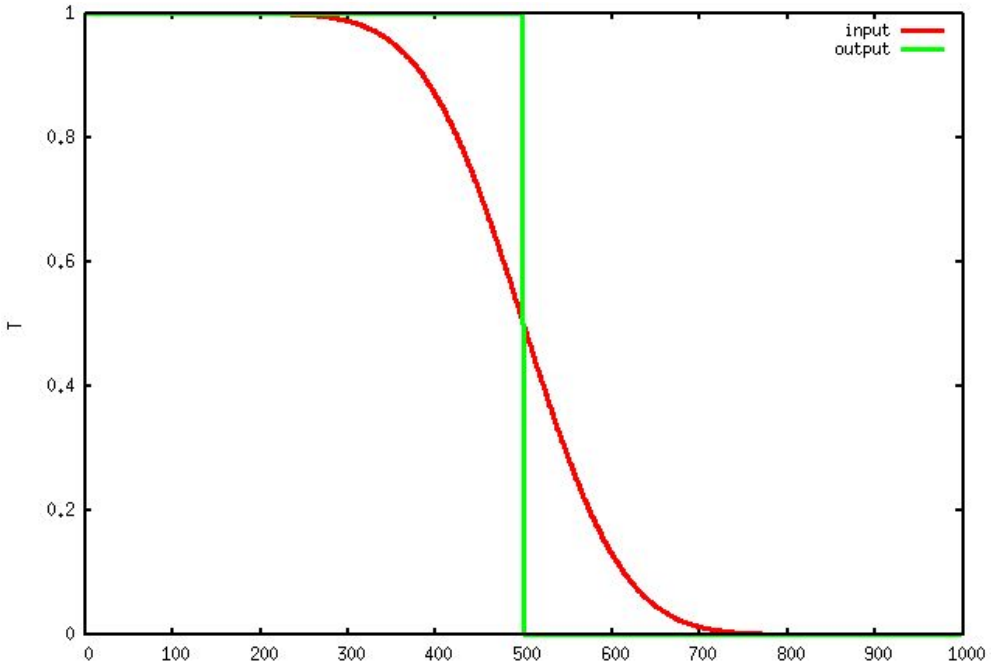
$$\begin{pmatrix} -2 & 1 & & & & \\ 1 & -2 & 1 & & & \\ & 1 & -2 & 1 & & \cdots \\ & & 1 & -2 & 1 & \\ & & \vdots & & & \ddots \end{pmatrix}$$

Le lecteur attentif aura à cœur de se convaincre que l'on peut résoudre ce problème de la manière suivante avec un algorithme de d'Euler :

Input: $\mathbf{x}^{(0)}, \mathbf{A}, c > 0, \epsilon > 0, \triangle x > 0, \triangle t > 0$
Output: $\mathbf{x}^{(0)}, \ldots, \mathbf{x}^{(N)}$ with $\mathbf{A}\,\mathbf{x}^{(N)} \approx 0$ for $N \in \mathbb{N}$
$n \leftarrow 0$;
repeat
$\mathbf{y}^{(n+1)} \leftarrow \mathbf{A}\,\mathbf{x}^{(n)}$;
$\mathbf{x}^{(n+1)} \leftarrow \mathbf{x}^{(n)} + c\frac{\triangle t}{(\triangle x)^2}\mathbf{y}^{(n+1)}$;
$n \leftarrow n + 1$;
until $||\mathbf{y}^{(n)}|| < \epsilon$;

Cet algorithme est implémenté dans le code heat.c, qui commence par créer la matrice creuse A au format creux ELLPACK, puis qui itère selon l'algorithme précédent tant que l'on a pas atteint le nombre maximum d'itérations et tant que l'on a pas convergé sur une solution stationnaire.
Le produit matrice-vecteur creux (SpMV, pour Sparse Matrix-Vector) est donné pour le format ELLPACK. On notera que la norme de y n'est pas calculée aux limites du domaine pour prendre en compte les conditions aux bords.

Le binaire obtenu avec heat.c génère un fichier texte au début et à la fin de la simulation, indiquant la solution obtenue, ce qui permet de s'assurer que de visualiser les résultats :



Cette courbe a été obtenue avec gnuplot, en prenant la sortie du code en appelant le code de la manière suivante, où le premier argument indique le nombre d'inconnues N, le second donne le nombre maximum d'itérations, et le dernier donne le paramètre epsilon qui conditionne l'arrêt du calcul sur une solution stationnaire.

```
$ ./heat 1000 10000 0.0000001
step: 10000
Err: 4.439994e-04
```

Le travail demandé ici consiste donc :

1. Commentez le choix du format creux ELLPACK pour des GPUs.
2. Portage OpenACC
    a. Accélérez le noyau SpMV avec OpenACC.
    b. Calculez la norme de y avec OpenACC.
    c. Assemblez la matrice creuse au format ELLPACK en OpenACC.
    d. Analysez les performances avec un profiler (nvprof/pgprof).
3. Portage à l'aide de bibliothèques
    a. Effectuez l'assemblage de votre matrice A au format creux de votre choix.
    b. Effectuez ces opérations à l'aide des la bibliothèques et CUBLAS (algèbre linéaire dense) et CUSPARSE (algèbre linéaire creuse). Effectuez maintenant l'assemblage de votre matrice en OpenACC. On pourra s'aider de la directive "host_data use_device" présentée ci-dessous.
    c. Analysez les performances avec un profiler (nvprof/pgprof).

Une analyse du format de matrice creuse le plus approprié sera un plus ! Remarquez à ce sujet que CUSPARSE comporte des routines de conversions de format.

Documentations :
https://docs.nvidia.com/cuda/archive/10.1/pdf/CUSPARSE_Library.pdf
https://docs.nvidia.com/cuda/archive/10.1/pdf/CUBLAS_Library.pdf

OPENACC HOST_DATA DIRECTIVE

Exposes the *device* address of particular objects to the *host* code.

```
#pragma acc data copy(x,y)
{
// x and y are host pointers
#pragma acc host_data use_device(x,y)
{
 // x and y are device pointers
}
// x and y are host pointers
}
```

X and Y are device pointers here