Projet HPCA : le plus grand rectangle

Version du 28 octobre 2016

Ce projet aborde un problème d'algorithmique simple et est directement inspiré d'un sujet [1] de TP rédigé par Sophie Tison pour l'UE ACT (Algorithmes et ComplexiTé) du master informatique de l'Université de Lille 1 (qui abordait le problème d'un point de vue algorithmique et séquentiel). Ce problème simple, qu'on rencontre aussi dans des concours de programmation (séquentielle), permet en effet d'aborder différents algorithmes, plus ou moins efficaces en séquentiel. Or ces différents algorithmes présentent un « potentiel » de parallélisation très variable. Le but de ce projet est donc d'implémenter des versions parallèles de ces algorithmes, sur GPU puis sur CPU, afin de les comparer, de déterminer les « points de croisement » au niveau performance des différents couples (algorithme, architecture) et de déterminer in fine quel est le couple (algorithme, architecture) le plus performant.

1 Présentation du problème

On considère la région rectangulaire du plan déterminée par le rectangle $(0,0)(0,h)(l,h)(l,0)$, h et l étant deux entiers positifs, ainsi que n points à coordonnées entières à l'intérieur de cette région. On souhaite dessiner un rectangle dont la base est sur l'axe des x, dont l'intérieur ne contienne aucun des n points et qui soit de surface maximale.

Par exemple si $h = 20, l = 25$, et qu'il y a 5 points $(2,5), (5,17), (11,4), (16,6), (20,1)$, voici quelques exemples de rectangles qu'on peut dessiner (voir figure 1) : $(5,0)(5,20)(11,20)(11,0)$ de surface 120, $(0,0), (0,1), (25,1), (25,0)$ de surface 25, $(0,0), (0,4), (20,4), (20,0)$ de surface 80, $(20,0), (20,20), (25,20)(25,0)$ de surface 100 ... La surface maximale qu'on peut obtenir est 153 avec le rectangle $(2,0)(2,17), (11,17)(11,0)$.

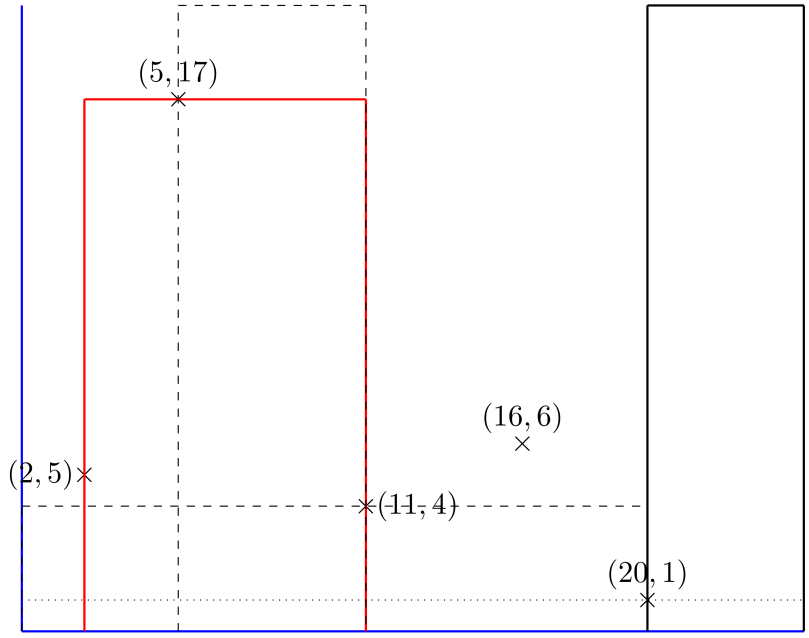

FIGURE 1 – Exemple de problème.

Donc les données en entrée du problème sont :
— l, h, deux entiers strictement positifs
— n, un entier positif
— les coordonnées entières des n points $(x_i, y_i), 0 < x_i < l, 0 < y_i < h$.

1. Voir : `http://www.fil.univ-lille1.fr/portail/index.php?dipl=MInfo&sem=S7&ue=ACT&label=Semainier`

La sortie attendue est la surface maximale d'un rectangle vérifiant les contraintes.

Pour vérifier que vos codes sont corrects, de « petits » exemples avec solutions sont disponibles dans le répertoire `Verif` (sur les machines `ppti-gpu-*`) sous :
`/users/Enseignants/fortin/Public/HPCA_oct2016/Projet/JeuxDeDonnees`

Le répertoire `Perf` contient des jeux de données générés aléatoirement, ainsi que le code C effectuant cette génération. Vous pouvez utiliser ces jeux de données pour vos tests de performance, et si nécessaire générer des jeux de données supplémentaires. Pour vous aider dans la conception et l'écriture des algorithmes, dans tous ces exemples les points sont triés par x croissants, et le premier point correspond à $x = 0$, et le dernier à $x = l$.

Les mesures de temps pour GPU ne devront pas prendre en compte l'initialisation du GPU, ni les allocations/libérations mémoire sur le GPU et sur l'hôte.

2 Partie 1 : algorithmes naïfs

Le but de cette première partie est de déployer deux premiers algorithmes sur un GPU en CUDA.

Le premier algorithme (naïf) est présenté en algorithme 1 et offre une complexité théorique en $O(n^3)$. Il se base notamment sur la remarque suivante : un rectangle de surface maximale respectant les contraintes a nécessairement deux sommets de la forme $(x_i, 0)$ et $(x_j, 0)$ avec $0 \leq i < j \leq n - 1$.

Algorithme 1 Algorithme N°1 en $O(n^3)$.

1: **Pour tout** couple (i,j) avec $i < j$ **faire**
2: Déterminer le point d'ordonnée minimale y_{min} parmi les points d'indice i+1..j-1 (y_{min} vaut h si $i + 1 == j$)
3: Calculer à l'aide d'y_{min} la surface $S_{i,j}$ du plus grand rectangle ayant son côté gauche en $x = i$ et son côté droit en $x = j$, et vérifiant les contraintes
4: **Fin pour**
5: Renvoyer $Max(S_{i,j})$

Pour éviter les nombreux calculs redondants présents dans l'algorithme 1, on peut calculer la valeur y_{min} pour l'intervalle (i,j) ainsi : $y_{min(i,j)} = Min(y_{min(i,j-1)}, y_j)$. Ainsi en commençant par le cas $i + 1 == j$ (où $y_{min(i,j)}$ vaut h), on calcule les différents $y_{min(i,j)}$ et $S_{i,j}$ pour j croissant, ce qui permet d'obtenir une complexité théorique en $O(n^2)$ (algorithme N°2).

Implémentez tout d'abord une première version fonctionnelle de ces deux algorithmes en C sur CPU, puis déployez-les sur GPU en CUDA. Vous pourrez manipuler les coordonnées via des entiers non signés sur 32 bits (`unsigned int` en CUDA), mais les calculs et le résultat devront utiliser des entiers non signés sur 64 bits (`unsigned longlong` en CUDA).

Afin de vérifier que votre implémentation GPU est correcte (pour les jeux de données dans le répertoire `Perf`), on vérifiera son résultat en la comparant à l'exécution sur le CPU (sauf coût de calcul prohibitif pour le CPU). On présentera notamment le gain du GPU par rapport au CPU. Afin de comparer honnêtement le GPU par rapport CPU, on pourra aussi accélérer le code CPU via OpenMP.

Vous pourrez ensuite optimiser votre code GPU en présentant plusieurs versions (si possible incrémentales, sauf si ce n'est pas pertinent). Pour chaque version, vous indiquerez la meilleure performance obtenue après avoir déterminé la taille optimale des blocs de threads sur GPU. Si cela vous aide, vous pourrez aussi prendre certaines hypothèses sur le nombre de points à traiter (et éventuellement essayer de les lever par la suite).

3 Partie 2 : algorithmes avancés

Pour la deuxième partie, on s'intéresse à des algorithmes plus efficaces pour résoudre ce problème.

3.1 Diviser pour régner

Il est possible de construire un algorithme plus efficace (en termes de complexité théorique) à l'aide du paradigme « Diviser pour régner » . Pour cela, on commence par chercher l'indice m du point d'ordonnée $y_{min(0,n-1)}$, et on résoud récursivement le problème sur les deux sous-parties $[0, m]$ et $[m, n-1]$. La solution sur $[0, n-1]$ est alors le maximum entre les solutions des deux sous-parties et la surface du rectangle de hauteur $y_{min(0,n-1)}$ et de longueur l (seul sous-rectangle pouvant être « à cheval » sur les deux sous-parties tout en respectant les contraintes). La complexité théorique $T(n)$ d'un tel algorithme s'exprime en $T(n) = 2T(n/2) + O(n)$ (dans le meilleur des cas, à savoir en considérant que les deux sous-parties sont de tailles égales, c'est-à-dire que le point d'ordonnée minimale est au milieu de la liste des points), ce qui d'après le *master theorem* [2] implique une complexité en $O(n \log n)$ (toujours dans le meilleur des cas).

Comment paralléliser ce type de calcul sur CPU ?
Quelles accélérations parallèles obtenez-vous sur CPU ?

Comment déployer ce type d'algorithme sur GPU ?
Pour cette dernière question, on pourra envisager un déploiement purement GPU ou un déploiement hybride CPU+GPU.

3.2 MPI

On pourra ensuite s'intéresser à une parallélisation MPI afin d'exploiter en parallèle les 2 noeuds `ppti-gpu-*`.

Après avoir choisi l'algorithme le plus pertinent en MPI, vous pourrez le paralléliser sur les CPU des 2 noeuds ou sur les GPU des 2 noeuds.

Dans le cas d'une parallélisation sur les CPU des 2 noeuds, on pourra comparer une parallélisation MPI+OpenMP et une parallélisation MPI pur.

3.3 Défi pour ceux qui veulent aller plus loin ...

Il est même possible d'obtenir un algorithme linéaire pour ce problème. On suppose pour cela qu'il n'y a pas deux points à la même abscisse (un pré-traitement linéaire des données permet d'obtenir cette propriété si nécessaire) et que le premier point a pour coordonnées (0,0).

Il faut alors parcourir l'ensemble des points par abscisse croissante via un point courant (x,y), et maintenir à jour une pile contenant les points (triés par abscisse et par ordonnée croissantes) d'ordonnée inférieure ou égale à celle du point précédent le point courant. Ceci permet de ne considérer que les rectangles ayant leur côté droit en x : le rectangle entre le point courant et le précédent, ainsi que tous les rectangles formés entre le point courant (côté droit), un point de la pile d'ordonnée strictement supérieure à y (haut du rectangle) et le point précédent dans la pile (côté gauche). Et on met ensuite à jour la pile pour le point suivant.

Vous pourrez comparer les performances de cet algorithme séquentiel à celles de vos implémentations parallèles des algorithmes précédents, voire tenter de paralléliser cet algorithme (sur CPU, et si toutefois cela est possible ...).

4 Travail à remettre

Pour chacune des deux parties, vous devrez remettre le code source, sous la forme d'une archive `tar` compressée et nommée suivant le modèle `projet_HPCA_nom1_nom2.tar.gz`. L'archive ne doit contenir aucun exécutable, et les différentes versions demandées devront être localisées dans des répertoires différents. Chaque répertoire devra contenir un fichier `Makefile` : la commande `make` devra permettre de lancer la compilation, et la commande `make exec` devra lancer une exécution parallèle représentative avec des paramètres appropriés. Un fichier `Makefile` situé à la racine de votre projet devra permettre (avec la commande `make`) de lancer la compilation de chaque version.

A la fin du projet, vous devrez remettre un rapport au format `pdf` (de 5 à 10 pages, nommé suivant le modèle `rapport_HPCA_nom1_nom2.pdf`) présentant vos algorithmes, vos choix d'implémentation (sans code source), vos résultats et vos conclusions pour les deux parties. L'analyse du comportement de vos programmes sera particulièrement

2. Voir par exemple : `https://fr.wikipedia.org/wiki/Master_theorem`

appréciée. Les machines n'étant pas strictement identiques, on précisera dans le rapport la (ou les) machine(s) utilisée(s) pour les tests de performance. Vous ne serez bien sur pas pénaliser si vous utilisez un CPU ou un GPU moins puissant que les autres étudiants.

5 Quelques précisions importantes

— Le projet est à réaliser par binôme (aucun trinôme ne sera accepté).
— Le code de la première partie, accompagné des slides de votre soutenance (au format `pdf`, et donc sans animations, dans un fichier nommé suivant le modèle `projet_HPCA_slides_nom1_nom2.pdf` ; prévoir une soutenance de 10 minutes, suivie de 5 minutes de questions), est à remettre au plus tard le mercredi 30 / 11 / 2016 à 22h00 (heure locale). Les soutenances de présentation de la première partie auront lieu lors de la séance de TDTP du jeudi 01 / 12 / 2016.
Le code de la seconde partie et le rapport final sont à remettre au plus tard le dimanche 22 / 01 / 2017 à 22h00 (heure locale).
— Les remises se feront par courriel à `pierre.fortin@lip6.fr`
— En cas d'imprévu ou de problème technique commun, n'hésitez pas à nous contacter pour que nous puissions vous proposer une solution ou une alternative.

