Programmation GPU

(2023-2024)

Projet

Filtrage par motif approché

adrien.roussel@cea.fr
mickael.boichot@cea.fr

Les objectifs de ce projet sont :

— porter une application sur GPU en utilisant CUDA
— mettre en oeuvre les optimisations GPU vues en cours
— réaliser une étude de performance en comparant l'application portée sur GPU avec une implémentation OpenMP parallèle

Ce projet est une adaptation du projet du cours CSC5001 de Télécom SudParis.

I Description

Le filtrage par motif[1] est une classe importante d'algorithme en Big Data et HPC. Par exemple en bioinformatique, de tels algorithmes sont utilisés pour détecter la présence d'une séquence de nucléotides donnée dans une chaîne d'ADN. Un autre domaine d'application est la fouille de données. Un exemple de recherche de motif est la détection exacte de chaîne de caractère comme celle implementée dans l'outil GNU grep.

L'application cible de ce projet introduit la notion d'approximation dans le filtrage par motif. Le filtrage par motif est *approchée* dans le sens où on élargit la recherche du motif à un ensemble de motifs contenant quelques différences avec ce dernier. La notion de *distance* est utilisee pour décider si un motif est suffisamment proche du motif recherché.

Le code repose sur le calcul de la distance de Levenshtein et plus précisement sur l'implémentation en C disponible ici.

```c
#define MIN3(a, b, c) ((a) < (b) ? \
    ((a) < (c) ? (a) : (c)) : ((b) < (c) ? (b) : (c)))

int levenshtein(char *s1, char *s2) {
    unsigned int s1len, s2len, x, y, lastdiag, olddiag;

    s1len = strlen(s1);
    s2len = strlen(s2);

    unsigned int column[s1len+1];

    for (y = 1; y <= s1len; y++) {
        column[y] = y;
    }

    for (x = 1; x <= s2len; x++) {
        column[0] = x;

        for (y = 1, lastdiag = x-1; y <= s1len; y++) {
            olddiag = column[y];
            column[y] = MIN3(
```

1. "pattern matching" en Anglais, "appariement de formes" en Québecois

```
22                  column[y] + 1,
23                  column[y-1] + 1,
24                  lastdiag + (s1[y-1] == s2[x-1] ? 0 : 1));
25              lastdiag = olddiag;
26          }
27      }
28      return(column[s1len]);
29  }
```

Nous utiliserons la distance de Levenshtein dans notre problème de la manière suivante. Nous considérerons que la première chaîne de caractère **s1** est le motif à rechercher et la chaîne de caractère **s2** la base de données de séquences ADN. La fonction précédente calcule la distance entre les chaînes **s1** et **s2** complètes. Dans notre cas, il est seulement nécessaire de calculer la distance entre la chaîne **s1** et les sous-chaînes de **s2** de la taille du motif. Ainsi, en itérant l'algorithme sur toutes les positions possibles de correspondance avec le motif (i.e. chaque caractère de la base de données), on peut compter le nombre total de correspondances du motif en fonction de la distance maximale cible.

En plus d'adapter l'algorithme, nous avons effectué les changements suivants :

— permettre la détection de plusieurs motifs
— permettre à l'utilisateur de chosir la distance maximale souhaitée pour détecter une correspondance
— allouer le tableau *column* en dehors de la fonction *levenshtein*

II Organisation du code source

Le code source pour ce projet est disponible ici.

Cette archive contient :

— *src/* : répertoire contenant le code source *apm.c*
— *obj/* : répertoire qui contiendra les fichiers générés lors de la compilation
— *dna/* : répertoire contentant des petites séquences d'ADN à des fins de test
— un makefile *Makefile*

Pour compiler l'application, utilisez la commande *make* pour générer le binaire *apm*. Une aide est affichée en lançant l'exécutable sans argument.

```
$ ./apm
Usage: ./apm approximation_factor dna_database pattern1 pattern2 ...
```

Le premier argument *approximation_factor* est un entier représentant la distance maximale autorisée entre le motif et une potentielle correspondance. Le second argument *dna_database* est un fichier texte contenant une base de données de séquence ADN. Le reste des arguments sont les motifs (sensibles à la casse) à rechercher dans la base de données.

Pour tester l'application, vous pouvez utiliser les deux fichiers situés dans le répertoire *dna*. Pour évaluer les performances de l'application sur des base de données plus conéquentes, vous pouvez télécharger des séquençage de génome ici. Par exemple, les chromosomes humains sont disponible ici. Utilisez le format FASTA compatible avec l'application.

III Remarques

Le parallélisme exploité dans le code peut se trouver dans la recherche d'un seul motif (en découpant la chaîne en plusieurs morceaux) et/ou dans la recherche

simutanée de plusieurs motifs.

IV Attentes

Le projet s'effectue en **binôme**.

Nous attendons **trois implémentations** de l'algorithme *apm* :

1. une implémentation OpenMP CPU parallèle
2. une implémentation CUDA GPU

L'implémentation (1) donnera un point de comparaison supplémentaire (et plus "juste") avec la version GPU. Néanmoins elle ne constitue pas le coeur du projet qui se concentre sur la programmation GPU. Nous vous conseillons donc d'y passer un temps restreint.

Pour chaque version nous attendons

— une vérification de la correction de l'implémentation (i.e. vérifier à l'aide de tests que la version parallèle fournit bien le même résultat que la version séquentielle)

— et une comparaison de performance avec les autres versions et la version séquentielle

Important : Ne modifiez pas la sortie standard du programme. Elle sera utilisée par nos scripts de test pour valider la correction de votre implémentation.

V Soutenance

La soutenance aura lieu le **09/01/2024**.

Une archive de votre code source doit nous être envoyé par mail avant le **22/12/2024** à minuit.

Lors de la soutenance, nous souhaitons que les points suivants soient abordés :

— description et motivation des stratégies de parallélisation employées

— choix d'implémentation et difficultés rencontrées pour les modèles de programmation CUDA et OpenMP

— étude des performance des implémentations de l'application *apm* et analyse des résultats obtenus

— pistes d'améliorations possibles des implémentations proposées

La soutenance sera composée de **10 minutes de présentation** suivies de **10 minutes de questions**.

Remarque importante

Nous évaluerons en premier lieu la démarche scientifique. Ce qui signifie que nous attachons une plus grande importance à la présentation de votre démarche et de vos résultats qu'à vos résultats eux mêmes.