Université de Limoges — Faculté des Sciences & Techniques

Faculté des Sciences et Techniques de Limoges

M1 - GPGPU
Projet Nombres premiers

Maxime MARIA, Neals FOURNAISE

2021-2022

Préambule

Les nombres premiers sont, par définition, des nombres qui sont divisibles par eux-mêmes ou par le nombre 1. Par convention, on considère que le nombre 2 est le premier nombre premier.

L'une des méthodes les plus basiques pour tester la primalité d'un nombre est d'utiliser le *crible d'Eratosthène*. L'idée de l'algorithme pour déterminer si le nombre N est premier, est de tester s'il existe un nombre M tel que $2 \leq M < N$ et M divise N (c'est-à-dire que le reste de la division euclidienne est nulle).

Les nombres premiers s'utilisent en cryptographie à clé publique que ce soit pour chiffrer des données ou signer des messages. L'exemple le plus connu est l'algorithme RSA dont la sécurité repose sur le problème de la factorisation. Ainsi, décomposer efficacement des nombres en produit de facteurs de nombres premiers devient une façon d'attaquer certains systèmes ou de rechercher des vulnérabilités.

Cette décomposition nécessite la détermination d'une liste suffisante de nombres premiers et ensuite à tester le nombre de fois où un nombre premier divise le nombre cible. Par exemple :

- $2133 = 1 * 3^3 * 79^1$
- $213 = 1 * 3^1 * 71^1$;
- $7 = 1 * 7^1$;

Vous trouverez dans la figure 1 un exemple d'exécution de votre programme final, afin que vous vous inspiriez fortement de l'affichage (en particulier la ligne sur les décompositions du dernier exercice, qui sera vérifiée automatiquement) :

ATTENTION : Dans l'énoncé, on utilisera la lettre N pour représenter le nombre donné en argument de votre programme ou devant être testé.

```
==========================================
   Partie CPU sur le nombre 2133
==========================================
Test de primalite de 2133
 --> Temps du test de primalite: 0 ms
Est Premier? 0
Recherche des nombres premiers sur CPU
 --> Temps de recherche: 0.779 ms
 --> Temps de factorisation en nombre premier: 0.005 ms
 Factorisation CPU:  1 * 3^3 * 79^1
==========================================
  Partie GPU sur le nombre 2133
==========================================
 --> Temps de test de primalite (GPU): 1.39888 ms
 Est Premier? : 2133 -> 0
 --> Temps de recherche (GPU): 0.242272 ms
 --> Temps de factorisation (GPU): 0.137888 ms
 Factorisation GPU:  1 * 3^3 * 79^1
```

FIGURE 1 – Exemple d'exécution du programme final.

1 Consignes

Ainsi, vous allez devoir réaliser deux versions, une version CPU et une version GPU, de trois algorithmes.

- Test de primalité : renvoie vrai si le nombre passé en paramètre est premier, faux sinon.
- Recherche de nombres premiers : renvoie une liste de nombres premiers inférieur à une borne N donnée.
- Décomposition en produit de facteurs de nombres premiers : renvoie la liste des facteurs ainsi que leur exposant.

D'un point de vue technique, il va falloir :

- Définir le type `ULONGLONG` comme un alias d'un entier long long non-signé ou utiliser `uint64_t`.
- Définir la structure struct cell qui contiendra un nombre premier ainsi que l'exposant associé pour la décomposition.
- Réutiliser les fichiers de mesure du temps déjà vus en TP (chronoGPU et chronoCPU).
- Fournir un *makefile, un script ou une commande* qui permet de compiler votre projet !
- Proposer une architecture logicielle cohérente.
- Tester votre programme avec des entiers plus ou moins longs. Pour cela, votre programme doit accepter en ligne de commande la valeur de N.

Le projet consiste à vous approprier ces problèmes sur les nombres premiers et mettre en pratique des solutions. S'il y a des optimisations à faire, particulièrement en CUDA, elles doivent avoir du sens et être justifiées un minimum. *Important :* six algorithmes doivent être codés obligatoirement mais vous avez le droit de laisser les différentes versions écrites dans le code final. Cela permet de voir l'évolution du processus d'optimisation ou de compréhension du problème. Vous pouvez ajoutez des suffixes aux noms des fonctions pour indiquer tout cela.

2 Algorithmes séquentiels (CPU)

Pour commencer ce projet, vous devez écrire une fonction pour tester la primalité d'un nombre de manière naïve (inspirée du crible d'Eratosthène). L'algorithme consiste à vérifier pour un nombre N, si tous les nombres inférieurs ne le divisent pas.

1. Réalisez une telle fonction séquentielle pour commencer ayant pour signature
   `bool isPrimeCPU(const ULONGLONG N);`.
   Remarque : vous pouvez tester votre programme en utilisant le site `http://compoasso.free.fr/primelistweb/page/prime/liste_online_en.php`.

2. Écrivez une fonction `searchPrimesCPU(const ULONGLONG N, ...);` permettant la recherche des nombres premiers inférieurs à N.
   Remarque : l'algorithme est simple et consiste à tester la primalité de tous les nombres inférieurs à N à l'aide de la fonction précédente. Il faut savoir que l'on ne peut connaître la taille de la liste renvoyée à l'avance. Libre à vous de trouver une solution (structure de données dynamique, gestion de la mémoire manuelle, taille fixée, etc.).

Deux observations permettent d'optimiser nos deux fonctions actuelles : les potentiels diviseurs de N sont forcément compris entre 2 et $\sqrt{N}$ et il ne sert à rien de tester à diviser par les nombres non-premiers et impairs.

3. Donnez donc deux nouvelles versions de vos algorithmes en prenant en compte cela.
   `bool isPrimeCPU(const ULONGLONG N, ...);` peut désormais prendre une liste de nombres afin de limiter le nombre de diviseurs.
   `searchPrimesCPU(const ULONGLONG N, ...);` peut se servir de la nouvelle fonction de test de primalité en testant la primalité des nouveaux nombres à l'aide de la liste en cours de création.

Lorsque l'on a ces deux algorithmes, il ne reste qu'à écrire la fonction de factorisation. Si cela n'est pas déjà fait, il faut adapter les algorithmes précédents afin qu'ils fassent usage de la structure `cell` définie ci-dessus.

Le principe de la décomposition consiste à parcourir les nombres p de la liste des nombres premiers trouvés avant en testant si ce nombre p divise N. Si oui, on recommence l'algorithme avec $N = N/p$. On s'arrête quand le nombre premier à tester devient supérieur à la racine carrée de N.

4. Écrivez une fonction séquentielle réalisant la factorisation du nombre N
   `void factoCPU(ULONGLONG N, ...);`.

3 Algorithmes parallèles (GPU)

En tenant compte des remarques précédentes, vous adapterez les trois algorithmes précédents en version CUDA sur GPU.

1. Écrivez un *kernel* réalisant un test de primalité pour un nombre N : `isPrimeGPU`
   Remarques :
   - On peut associer à chaque *thread* un ou plusieurs nombres à tester par rapport à N.
   - Cette fonction renvoie un booléen, comment allez-vous gérer la réduction ?

2. Écrivez un *kernel* de recherche des nombres premiers inférieurs à N : `searchPrimesGPU`
   Remarques :
   - Le tableau contenant les nombres premiers trouvés est partagé entre tous les blocs.

3. Écrivez le *kernel* de décomposition en produit de facteurs de nombres premiers : `factoGPU`
   Remarques :
   - Vous pouvez utiliser la liste de nombres premiers récupérée par l'algorithme précédent.

4 À rendre

Ce travail est à réaliser en binôme.
Vous devez rendre un dossier nommé `GPGPU_Nom1_Nom2.zip` contenant :

1. Votre code qui doit compiler et fonctionner sur les machines de la fac (I211/I212) ;
2. Un rapport au format PDF dans lequel vous expliquerez votre code et **justifierez vos choix**. Vous discuterez notamment de la répartition des *threads* sur la grille de calcul. Vous comparerez les versions séquentielles et parallèles. Si vous avez plusieurs versions d'un même *kernel*, comparez les performances et analysez-les.

5 Attention au plagiat !

Vous le savez, vous avez même signé un engagement concernant le plagiat/la fraude à l'Université... Mais une piqûre de rappel ne fait pas de mal ! Faites très attention, la fraude peut entraîner jusqu'à l'exclusion du système universitaire français !