Projet

Accélération de calcul matriciel à l'aide de la programmation parallèle sur GPU

1. Introduction

Dans ce projet, vous allez construire étape par étape une classe C++ de matrice qui implémente des opérations matricielles classiques, ainsi qu'une classe dérivée parallèle qui permet de paralléliser la multiplication de matrices.

Les étapes 1 à 8 sont les étapes de base, valant 10 points.

Les étapes suivantes permettent d'améliorer votre code à l'aide de concepts de programmation plus poussés, ajoutant des points supplémentaires pour un total de 17 points. 3 points supplémentaires sont réservés pour des codes particulièrement bien commentés, respectant les différentes conventions de notation en C++.

Vous pouvez faire ce projet en groupe de 2 ou 3.

Les prochains cours sont là pour vous permettre d'avancer, n'hésitez pas à poser des questions.

Un seul fichier C++ est à envoyer au format :

GroupeXX.cpp

2. Règles de bonne écriture en C++

- Choisissez pour tous les noms que vous utiliserez une écriture :
    1. en Snake Case (En Serpent) : écriture en minuscule séparé par des tirets bas
    2. en Camel Case (En Chameau) : écriture en minuscule sans séparation des mots sauf la première lettre en majuscule

  mais ne les mélangez pas.

- Les noms de classe commencent toujours par une majuscule.
- Les noms de variables commencent toujours par une minuscule.
- Les variables membres d'une classe ont leur nom commençant par le préfixe **m**, que vous utilisiez l'écriture en serpent ou en chameau.
- Commentez vos codes, et particulièrement ajoutez un commentaire au-dessus de chaque déclaration de classe ou de fonction expliquant rapidement son utilité et/ou son fonctionnement.

Étape 1 : Inclure les bibliothèques nécessaires

Commencez par inclure les bibliothèques standards dont vous aurez besoin dans ce programme. Utilisez le code suivant :

```
#include <iostream >
#include <vector >
#include <thread >
#include <chrono>
#include <stdexcept >
```

Ces bibliothèques permettent d'utiliser les vecteurs, les threads, la mesure de temps, ainsi que les levées d'exception pour vérifier les conditions.

Étape 2 : Définir une classe de matrice

La classe de matrice permettra de travailler avec des matrices contenant des nombres décimaux (que vous choisirez soit float, soit double).

1. Créez une classe (à laquelle vous donnerez un nom) pour l'instant vide, qui ne contiendra que des membres **public** et **protected**.

2. Ajoutez à votre classe deux entiers protégés, définissant le nombre de lignes et le nombre de colonnes de votre matrice.

3. Ajoutez à votre classe un vecteur de vecteurs (lui aussi protégé), qui contiendra les données de votre matrice.

Étape 3 : Ajouter le constructeur

Créez un constructeur :

- Ce constructeur prend comme arguments le nombre de lignes, le nombre de colonnes de la matrice, et une valeur par défaut dont il remplira chaque élément de la matrice.

Étape 4 : Ajouter les méthodes d'accès aux éléments

1. Créez un accesseur (ou *getter*) pour le nombre de lignes.
2. Créez un accesseur (ou *getter*) pour le nombre de colonnes.
3. Créez un accesseur (ou *getter*), prenant en entrée deux indices, un pour la ligne et un pour la colonne, renvoyant la valeur de l'élément associé dans la matrice.
4. Créez un modificateur (ou *getter*), prenant en entrée deux indice ligne et colonne et un nombre d´décimal, qu'il enregistrera `a cette position dans la matrice.

Étape 5 : Surcharger l'opérateur *

Ajoutez une surcharge pour l'opérateur *

Celui-ci prendra en argument une matrice, et renverra le produit matriciel entre l'instance et la matrice en argument.

Le produit matriciel entre deux matrices A et B peut se faire simplement en suivant l'algorithme suivant :

1. Initialisez une matrice C, qui sera le résultat du produit, avec uniquement des zéros ayant pour dimension le nombre de lignes de A et le nombre de colonnes de B.
2. Faites une boucle sur les indices des lignes de la matrice A avec un itérateur **i**.
3. Imbriquez dans celle-ci une boucle sur les indices des colonnes de la matrice B avec un itérateur **j**.
4. Initialisez un résultat à 0.
5. Dans ces deux boucles imbriquées et après avoir initialisé le résultat à 0, ajoutez une troisième boucle sur les indices des colonnes de la matrice A avec un itérateur **k**.
6. Ajoutez à votre résultat le produit de l'élément (**i, k**) de la matrice A et de l'élément (**k, j**) de la matrice B.
7. Modifiez la valeur de l'élément (**i, j**) de la matrice C une fois que la boucle sur **k** est terminée.
8. Renvoyez la matrice C.

Étape 6 : Définir une classe de matrice calculant en parallèle

Il faudra ici créer une classe de matrice effectuant ce calcul en parallèle. L'objectif est ici de spécialiser l'opérateur * en utilisant des *threads*.

1 Créez une nouvelle classe de matrice héritant de la précédente.

2 Dans ses données publiques, ajoutez la ligne suivante en adaptant le nom à celui de votre classe de matrice s´séquentielle :
`using Matrix::Matrix;`

3 Dans ses données publiques, spécialisez l'opérateur *.

4 Initialisez une matrice à la bonne taille ne contenant que des 0 l'instant, qui contiendra le résultat de la multiplication.

5 Définissez une fonction *lambda*, c'est-à-dire une fonction sans signature en copiant le code ci-dessous et en l'adaptant aux types et aux noms de variables que vous avez choisis.

```
auto multiplyRow = [&](int row) {
      for (int j = 0; j < matrixB.getNbCols(); ++j) {
          float sum = 0.f;

          for (int k = 0; k < this->getNbCols(); ++k) {
            sum += this->getElement(row, k) * matrixB.getElement(k, j);
          }
          matrixC.setElement(row, j, sum);
      }
};
```

6 En vous aidant des TP précédents, créez un vecteur threads contenant des **std::thread**.

7 Créez une boucle avec un itérateur **i** allant de 0 jusqu'au nombre de lignes moins un de la matrice A.

8 Ajoutez à votre vecteur le thread suivant :
threads.emplace back(multiplyRow, i);

9 Rejoignez les **threads.**

10 Retournez votre matrice C

Étape 8 : Tester le code dans le main

Dans la fonction main(), testez les deux classes :

1. Définissez deux entiers définissant la taille de vos matrices (ligne, colonne). Pour plus de simplicité, vous pouvez utiliser des matrices carrées.
2. Créez deux matrices instanciées à l'aide de la classe séquentielle et effectuez leur multiplication.
3. Créez deux matrices instanciées équivalentes aux premières à l'aide de la classe parallèle et effectuez leur multiplication.
4. Mesurez et affichez le temps d'exécution de chaque multiplication, en vous aidant si besoin des codes faits dans le TP précèdent.
5. Augmentez ou diminuez la taille de vos matrices jusqu'à voir à partir de quel moment le calcul parallèle devient intéressant.

Étape 9 : Améliorer son code

Dans cette partie, vous n'^êtes pas obligés de réaliser les différentes étapes dans l'ordre affiché, vous pouvez choisir de privilégier celles que vous préférez.

- Ajoutez pour la matrice s´séquentielle une surcharge d'operateurs + et -.
- Plutôt que de définir vos classes de matrices en comptant des float ou des double, utilisez pour chacune un template.
- Ajoutez une méthode vérifiant si les tailles des deux matrices sont compatibles pour une multiplication. Si vous avez implémenté les operateurs + et -, ajoutez une méthode supplémentaire pour vérifier qu'elles soient compatibles pour ces opérations.
- Ajoutez de nouvelles manières de construire votre matrice.
- Ajoutez de la gestion d'erreurs : à l'aide de conditions que vous choisirez, vous pouvez lever une erreur de différentes manières :

```
throw std::invalid argument("Expliquer
l'erreur");
throw std::out of range("Expliquer l'erreur");
```