AGH – UNIVERSITY OF SCIENCE AND TECHNOLOGY

Dokumentacja projektowa dla

Operacje macierzowe z wykorzystaniem technologii CUDA

Architektura I Programowanie GPU

Informatyka i Systemy Inteligentne, I rok

Adrian Smoła, Kacper Godula

Prowadzący: Jędrzej Byrski

09.06.2022

1. Opis Projektu

Celem projektu jest zaprezentowanie wybranych operacji macierzowych, wykonanych w technologii CUDA. Głównym zamierzeniem jest lepsze zrozumienie operacji zrównoleglonych używanych w działaniach na macierzach, a także znaczące przyspieszenie ich działania w stosunku do klasycznych implementacji sekwencyjnych.

2. Kompilacja

W celu uruchomienia projektu, zalecane jest uruchomienie poszczególnych projektów w Visual Studio 2022, lub po prostu uruchomić załączony plik exe.

3. Architektura projektu

a. Matrix_x_scalar – Służy do podstawowego mnożenia macierzy przez skalar używając technologii CUDA. Zawiera 3 funkcje i jednego kernela.
   i. Void main() – Jest to główna funkcja programu, w niej deklarujemy pamięć, wywołujemy inne funkcje oraz kernele.
   ii. Void printResults(float** A, float** B, int width) – Jest to funkcja użyta w celu wypisania elementów macierzy A oraz macierzy B o szerokości width
   iii. Void random_ints(float** matrix, size_t N, size_t M) - wypełnia macierz matrix pseudo losowymi liczbami, parametry N oraz M określają wymiary macierzy.
   iv. __global__ void Matrix_multiplication(float *A, float* B, float mul, int width) – Kernel używany do mnożenia Macierzy A przez wartość mul, i zapisujący jej zawartość do macierzy B. width oznacza szerokość macierzy.

b. Matrix_x_matrix – Służy do mnożenia macierzy przez macierz. Obsługuje również macierze niebędące macierzami kwadratowymi. Zawiera 3 funkcje i jeden kernel.
   i. Void main() – Jest to główna funkcja programu, w niej deklarujemy pamięć, wywołujemy inne funkcje oraz kernele.
   ii. Void printResults(float** A, float** B, float ** C, size_t N, size_t M, size_t Csize) – Jest to funkcja użyta w celu wypisania elementów macierzy A oraz macierzy B, oraz macierzy C, gdzie N i M to rozmiary macierzy wejściowych, a Csize to rozmiar macierzy wynikowej.
   iii. Void random_ints(float** matrix, size_t N, size_t M) - wypełnia macierz matrix pseudo losowymi liczbami, parametry N oraz M określają wymiary macierzy.
   iv. __global__ void MatMul(float* A, float* B, float* C, int ARows, int ACols, int BRows, int BCols, int CRows, int CCols) – Macierze A I B to macierze wejściowe, macierz C to macierz wynikowa, pozostałe parametry to ilość kolumn i rzędów w poszczególnych macierzach. Służy do wykonywania mnożenia macierzy A przez macierz B, wynik jest zapisywany do macierzy C.

c. Matrix_transpose – Służy do wykonywania transpozycji macierzy. Zawiera 3 funkcje i jeden kernel.
   i. Void main() – Jest to główna funkcja programu, w niej deklarujemy pamięć, wywołujemy inne funkcje oraz kernele.
   ii. Void printResults(float** A, float** B, size_t N, size_t M) – Jest to funkcja użyta w celu wypisania elementów macierzy A oraz macierzy B, gdzie N to rozmiar macierzy wejściowej A, a M to rozmiar macierzy wyjściowej B.

iii. Void random_ints(float** matrix, size_t N, size_t M) - wypełnia macierz matrix pseudo losowymi liczbami, parametry N oraz M określają wymiary macierzy.
iv. __global__ void Matrix_transpose(float* A, float* B, int A_rows, int A_cols) – Kernel używany do wykonywania transpozycji, Macierz A to macierz wejściowa, macierz B to macierz wyjściowa, a pozostałe dwa parametry to ilość kolumn i wierszy macierzy wejściowej.

d. Matrix_gauss – Służy do obliczenia rozwiązań zbioru równań za pomocą eliminacji gaussa. Składa się z 4 funkcji i jednego kernela. Do wykonania programu potrzebna jest przygotowana macierz w pliku tekstowym.
   i. Void main() – Jest to główna funkcja programu, w niej deklarujemy pamięć, wywołujemy inne funkcje oraz kernele.
   ii. __global__ void Kernel(float* A_, float* B_, int size) – Kernel służący do obliczania metodą eliminacji gaussa.
   iii. void copyvalue(int newchar, int* i, FILE* data, float* temp_h) – funkcja do kopiowania wartości z pliku do tabeli.
   iv. void DeviceFunc(float* temp_h, int variablesNo, float* temp1_h) – zawiera wszystkie metody związane z alokowaniem pamięci w GPU, kopiowaniem z i do pamięci GPU, oraz wywoływaniem kernela.
   v. void getvalue(float** temp_h, int* variablesNo) – funkcja służąca do odczytania danych z pliku.

e. Matrix_jordan – Służy do obliczania macierzy odwrotnej do macierzy wejściowej korzystając z metody gaussa-jordana. Funkcja ta zapisuje dwie wartości do dwóch różnych plików. W pliku input_matrix.txt możemy znaleźć macierz wejściową, wygenerowaną w kodzie, a w pliku inverse.txt – obliczoną macierz odwrotną dla macierzy wejściowej.
   i. Void main() – Jest to główna funkcja programu, w niej deklarujemy pamięć, wywołujemy inne funkcje oraz kernele.
   ii. __global__ void gaussjordan(float* A, float* I, int n, int i) – kernel wykonujący eliminacje gaussa-jordana w konkretnym wierszu
   iii. __global__ void dev(float* d_A, float* dI, int h) – Kernel odpowiedzialny za dzielenie wierszy zawierających element przestawny przez ten element przestawny.
   iv. void savetofile(float* A, string s, int n, int h) – funkcja odpowiedzialna za zapisanie macierzy A, o rozmiarze n na h, w pliku o nazwie zapisanej parametrze s.
   v. void random_floats(float* vect, int N) – funkcja odpowiedzialna za wypełnienie macierzy pseudo losowymi wartościami typu float.

4. Problemy

a. We wstępnych założeniach projektu było również wykonanie kodu obliczającego równolegle macierz dopełnień i macierzy dołączonej dla dowolnej macierzy, niestety w związku problematyką liczenia wyznaczników macierzy wyższego stopnia - czyli rzeczy niezbędnej dla obliczania macierzy dopełnień i dołączonej, potrzebna jest duża ilość rekurencyjnych wywołań, a także wygenerowanie dla macierzy wyższego stopnia milionów niewielkich tymczasowych tabel.