Федеральное агентство связи

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Сибирский государственный университет

телекоммуникаций и информатики»

**Расчетно-графическое задание**

по дисциплине «Теория программирование графических процессов»

на тему

**Сравнительный анализ производительности программ, реализующих алгоритмы линейной алгебры с использованием библиотек Thrust, cuBLAS «сырого» CUDA C кода»**

Выполнил: студент группы МГ-111

Пляскина А.Ю

Проверил: профессор кафедры ПМиК

Малков Е. А.

Новосибирск 2022

**СОДЕРЖАНИЕ**

[Введение 3](#_Toc41915607)

[1. Теоретические сведения 4](#_Toc41915608)

[1.1. Архитектура CUDA 4](#_Toc41915609)

[1.2. Библиотека Thrust 4](#_Toc41915610)

[1.3. Библиотека cuBLAS 5](#_Toc41915611)

[2. Экспериментальные исследования 6](#_Toc41915612)

[2.1. Организация экспериментов 6](#_Toc41915613)

[2.2. Результаты экспериментов 6](#_Toc41915614)

[Заключение 7](#_Toc41915615)

[Приложения 8](#_Toc41915616)

# ВВЕДЕНИЕ

CUDA – это архитектура параллельных вычислений от NVIDIA, позволяющая существенно увеличить вычислительную производительность благодаря использованию GPU (графических процессоров).

На сегодняшний день продажи CUDA процессоров достигли миллионов, а разработчики программного обеспечения, ученые и исследователи широко используют CUDA в различных областях, включая обработку видео и изображений, вычислительную биологию и химию, моделирование динамики жидкостей, восстановление изображений, полученных путем компьютерной томографии, сейсмический анализ, трассировку лучей и многое другое.

 Направление вычислений эволюционирует от «централизованной обработки данных» на центральном процессоре до «совместной обработки» на CPU и GPU. Для реализации новой вычислительной парадигмы компания NVIDIA изобрела архитектуру параллельных вычислений CUDA, на данный момент представленную в графических процессорах [GeForce](https://www.nvidia.com.ua/object/geforce_family_ru.html), [ION](https://www.nvidia.com.ua/object/sff_ion_ru.html) и [Tesla](https://www.nvidia.com.ua/object/tesla-high-performance-computing-ru.html) и обеспечивающую необходимую базу разработчикам ПО.

Целью расчетно-графической работы является ознакомление с архитектурой CUDA и проведение сравнительного анализа производительности программ, реализующих алгоритмы линейной алгебры с использованием библиотек Thrust, cuBLAS и «сырого» CUDA C кода.

# Теоретические сведения

В ходе выполнения данной работы исследовался алгоритм умножения матрицы на вектор. Умножение матрицы на вектор – это частный случай матричного умножения, где вектор имеет размер 1×n. Умножение матрицы на вектор производится по правилу «строка на столбец».

## Архитектура CUDA

Вычислительный узел с CPU и *main memory* называется *host*, GPU называется *device*. Программа, выполняемая на host'е содержит код – ядро, которое загружается на *device* в виде многочисленных копий. Все копии загруженного кода – нити (*threads*), объединяются в блоки (*blocks*) по 512-1024 нити в каждом. Все блоки объединяются в сеть (*grid*) с максимальным количеством блоков равным 65536. Все нити имеют совместный доступ на запись/чтение к памяти большого объема – *global memory*, на чтение к кэшируемым *constant memory* и *texture memory*. Нити одного блока имеют доступ к быстрой памяти небольшого объема – *shared memory*.

## Библиотека Thrust

Одной из библиотек для CUDA является библиотека *thrust*. Основным отличием данной библиотеки от других является то, что *thrust* – это библиотека, основанная на использовании шаблонов языка *С*++. Все классы и функции в этой библиотеки – шаблонные, все, что вам нужно для работы с этой библиотекой – это подключить соответствующие заголовочные файлы.

Простейшим алгоритмом, поддерживаемым *thrust*, является *thrust::transform*, выполняющий некоторую операцию над входными значениями и записывающим результат выполнения операции по заданному итератору. Сама операция задается при помощи функтора, при этом файл *thrust/functional.h* содержит ряд уже готовых к использованию функторов. Простейшим случаем является использование унарного функтора – есть один входной вектор, есть функтор, принимающий один аргумент, и есть выходной вектор.

## Библиотека cuBLAS

Библиотека *cuBLAS* – это GPU-ускоренная версия библиотеки стандартных базовых подпрограмм линейной алгебры (*BLAS*. Используя API-интерфейс *cuBLAS*, можно ускорить работу своих приложений путем развертывания интенсивных вычислений на одном графическом процессоре или масштабирования и эффективного выполнения работы в конфигурациях с несколькими GPU.

К особенностям библиотеки *cuBLAS* можно отнести хранение информации по столбцам, линейную индексацию массивов, совместимость с *FORTRAN*’ом.

Библиотека *cuBLAS* обеспечивает эффективное выполнение линейных алгебраических преобразований над матрицами и векторами. Её применение позволяет значительно ускорить выполнение многих арифметических операций и дает выигрыш в производительности даже по сравнению с настройкой вручную или автоматической оптимизацией этих операций.

# Экспериментальные исследования

## Организация экспериментов

Экспериментальные замеры времени проводились на кластере СибГУТИ.

## Результаты экспериментов

Таблица 2.1 – Зависимость времени *t* выполнения программы от размерности *n* матрицы и способа реализации программы.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Способ реализации | Размерность матрицы, *n* | | | | | |
| 128 \* 100 | 256 \* 128 | 512 \* 256 | 1024 \* 512 | 2048 \* 1024 | 4096 \* 2048 |
| *CUDA C* | 0,12046 | 0,12247 | 0,12608 | 0,12096 | 0,11672 | 0,12646 |
| *Thrust* | 0,50319 | 1,06239 | 3,93160 | 110,19271 | 440,65123 | 1765,79259 |
| *cuBLAS* | 0,22339 | 0,22468 | 0,22247 | 0,21017 | 0,20767 | 0,22219 |

Рисунок 2.1 – Зависимость времени *t* выполнения программы от размерности *n* матрицы и способа реализации программы

# ЗАКЛЮЧЕНИЕ

В результате выполнения работы был реализован алгоритм умножения матрицы на вектор с помощью библиотек *thrust* и *cuBLAS*. В ходе эксперимента было обнаружено, что «сырой» CUDA код работал быстрее своих конкурентов. Библиотека *thrust* на деле оказалась очень медленной и неэффективной.

# ПРИЛОЖЕНИЯ

1. Исходные коды разработанной программы

rgz.cu

#include <cuda.h>

#include <stdlib.h>

#include <stdio.h>

#include <iostream>

#include <cuda\_runtime.h>

#include "cublas\_v2.h"

#include <thrust/device\_vector.h>

#include <thrust/transform.h>

#include <thrust/functional.h>

#include <time.h>

void fillMatrix(float\* a, int rows, int cols)

{

for (int j = 0; j < cols; j++) {

for (int i = 0; i < rows; i++) {

a[j \* rows + i] = rand() % 100 - 10;

}

}

}

void showMatrix(float\* a, int rows, int cols)

{

printf("\nMatrix:\n");

for(int i = 0; i < rows; i++) {

for(int j = 0; j < cols; j++) {

fprintf(stdout, "%g\t", a[j \* rows + i]);

}

fprintf(stdout, "\n");

}

fprintf(stdout, "\n");

}

void fillVector(float\* x, int cols)

{

for (int i = 0; i < cols; i++) {

x[i] = rand() % 100 - 10;

}

}

void showVector(float\* x, int cols)

{

printf("Vector:\n");

for (int i = 0; i < cols; i++) {

fprintf(stdout, "%g\t", x[i]);

}

fprintf(stdout, "\n");

}

void showResultVector(float\* y, int rows)

{

printf("\nResult vector:\n");

for (int i = 0; i < rows; i++) {

fprintf(stdout, "%g\t", y[i]);

}

fprintf(stdout, "\n");

}

\_\_global\_\_ void mult(float\* a, float\* b, float\* c, int rows, int cols)

{

int i = blockIdx.x \* blockDim.x + threadIdx.x;

int j;

c[i] = 0;

for (j = 0; j < cols; j++) {

c[i] += a[j \* rows + i] \* b[j];

}

}

void rawCuda(float \*a, float \*x, float \*y, int rows, int cols)

{

float \*adevice, \*xdevice, \*ydevice;

cudaMalloc((void\*\*)&adevice, rows \* cols \* sizeof(\*a));

cudaMalloc((void\*\*)&xdevice, cols \* sizeof(\*x));

cudaMalloc((void\*\*)&ydevice, rows \* sizeof(\*y));

cudaMemcpy(adevice, a, rows \* cols \* sizeof(float), cudaMemcpyHostToDevice);

cudaMemcpy(xdevice, x, cols \* sizeof(float), cudaMemcpyHostToDevice);

dim3 threads(rows);

dim3 grid(1);

mult << <grid, threads >> > (adevice, xdevice, ydevice, rows, cols);

cudaDeviceSynchronize();

cudaMemcpy(y, ydevice, rows \* sizeof(float), cudaMemcpyDeviceToHost);

cudaFree(adevice);

cudaFree(xdevice);

cudaFree(ydevice);

}

void thrustCu(float \*a, float \*x, float \*y, int rows, int cols)

{

thrust::device\_vector<float> a\_device(a, a + (rows \* cols));

thrust::device\_vector<float> b\_device(cols);

thrust::device\_vector<float> x\_device(x, x + cols);

thrust::device\_vector<float> y\_device(cols);

float res = 0.0f;

for (int i = 0; i < rows; i++) {

for (int j = 0; j < cols; j++)

b\_device[j] = a\_device[j \* rows + i];

thrust::transform(

b\_device.begin(),

b\_device.end(),

x\_device.begin(),

y\_device.begin(),

thrust::multiplies<float>());

for (int j = 0; j < cols; j++)

res = thrust::reduce(y\_device.begin(), y\_device.end());

y[i] = res;

res = 0.0f;

}

}

void cuBlas(float \*a, float \*x, float \*y, int rows, int cols)

{

float \*adevice, \*xdevice, \*ydevice;

cudaMalloc((void\*\*)&adevice, rows \* cols \* sizeof(\*a));

cudaMalloc((void\*\*)&xdevice, cols \* sizeof(\*x));

cudaMalloc((void\*\*)&ydevice, rows \* sizeof(\*y));

cublasHandle\_t handle;

cublasCreate(&handle);

cublasSetMatrix(rows, cols, sizeof(\*a), a, rows, adevice, rows);

cublasSetVector(cols, sizeof(\*x), x, 1, xdevice, 1);

cublasSetVector(rows, sizeof(\*y), y, 1, ydevice, 1);

float al = 1.0f;

float bet = 1.0f;

cublasSgemv(handle, CUBLAS\_OP\_N, rows, cols, &al, adevice, rows, xdevice, 1, &bet, ydevice, 1);

cublasGetVector(rows, sizeof(\*y), ydevice, 1, y, 1);

cudaFree(adevice);

cudaFree(xdevice);

cudaFree(ydevice);

cublasDestroy(handle);

}

int main (int argc, char \* argv [])

{

if(argc < 3) {

fprintf(stderr, "Invalid arguments number!\n<source><rows><cols>\n: ");

return -1;

}

int choice;

int rows = atoi(argv[1]);

int cols = atoi(argv[2]);

clock\_t start, end;

float \*a = (float\*)malloc(rows \* cols \* sizeof(float));

float\* x = (float\*)malloc(cols \* sizeof(float));

float\* y = (float\*)malloc(rows \* sizeof(float));

printf("\n1)CUDA C\n2)Thrust\n3)cuBLAS\n\n");

scanf("%d", &choice);

fillMatrix(a, rows, cols);

showMatrix(a, rows, cols);

fillVector(x, cols);

showVector(x, cols);

if (choice == 1) {

start = clock();

rawCuda(a, x, y, rows, cols);

end = clock();

}

else if (choice == 2) {

start = clock();

thrustCu(a, x, y, rows, cols);

end = clock();

}

else if (choice == 3) {

start = clock();

cuBlas(a, x, y, rows, cols);

end = clock();

}

else {

printf("\nEnter a value from 1 to 3\n");

return -1;

}

showResultVector(y, rows);

double timer = (double)(end - start) / CLOCKS\_PER\_SEC;

fprintf(stdout, "\nTotalTime = %f\n\n", timer);

free(a);

free(x);

free(y);

}