МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное
образовательное учреждение высшего образования
«Самарский национальный исследовательский университет
имени академика С.П. Королева»
(Самарский университет)

Институт информатики, математики и электроники
Факультет информатики
Кафедра технической кибернетики

ЛАБОРАТОРНАЯ РАБОТА №5

Умножение квадратных матриц с использованием библиотеки CuBLAS

по курсу
Параллельное программирование

Студент гр. 6407 _____ В.С. Гринина
Преподаватель,
к.ф.-м.н. _____ Е.С. Козлова

САМАРА 2018

ЗАДАНИЕ

С использованием библиотеки CUBLAS произвести умножение квадратных матриц размерностей N×N, сгенерированных случайно, где N =1000, 10000. Записать время выполнения умножений матриц в таблицу. Сделать выводы по полученным результатам. [1]

ВВЕДЕНИЕ

Написание своего кода, множества своих функций есть хорошо. Однако, использование различных библиотек в своем коде позволяет не только сместить фокус на какие-то более специфичные и уникальные моменты в задаче, но и сделать выполнение кода в разы быстрее, так как написанием библиотеки занимались профессионалы на протяжении долгого времени.

Одной из таких полезных библиотек является библиотека BLAS, которая представляет из себя множество функций для решения типовых задач линейной алгебры. Для технологии CUDA существует своя аналогичная библиотека с названием CuBLAS.

В данной лабораторной работе будет изучена и использована эта библиотека для задачи перемножения двух матриц.

1 Ход выполнения работы

Практически весь необходимый код для лабораторной работы есть в методических указаниях. [1] Единственное, что можно было бы добавить, так это запись матриц в поток вывода. Код, готовый для компиляции представлен в приложении.

Запуск данной программы производится так же, как и в предыдущей лабораторной работе с помощью скриптов. Необходимые скрипты для запуска также представлены в приложении.

Стоит заметить, что несмотря на достаточно большое количество кода программы, в нем в основном используются уже существующие функции библиотеки CuBLAS. Поэтому после недолгого изучения данной библиотеки разобраться в коде достаточно просто.

Для анализа работы программы засечем время ее выполнения и внесем эти данные в таблицу 1.

Таблица 1 – Время выполнения программы при различных параметрах

Размерность матрицы	1000	10000
Время выполнения	0.096328	3.243154

Результаты выполнения программы трудно сравнивать между собой. Однако, полагаясь на опыт, можно заметить, что перемножение матриц небольших размерностей происходит мгновенно. Что касается перемножения больших матриц, то данный результат также является замечательным, так как запуск последовательной программы перемножения матриц такой размерности сложно себе позволить где-либо.

ЗАКЛЮЧЕНИЕ

В ходе проведения данной лабораторной работы был дописан и изучен уже написанный код перемножения матриц с использованием библиотеки CuBLAS. Была изучена сама библиотека CuBLAS.

Для убеждения в том, что данная библиотека работает качественно, была запущена программа с различными параметрами и засечено время ее выполнения. Данная программа показала прекрасные результаты. Разумеется, это неспроста, ведь над библиотекой CuBLAS продолжительно время усердно работали специалисты, которые, наверняка делали упор на многие аспекты, например, непосредственно метод перемножения матриц, рациональная работа с памятью.

На основе результатов данной финальной в изучаемом курсе лабораторной работы можно заявить, что параллельное программирование и использование качественных библиотек позволяют повысить работоспособность программы во много раз, что и требовалось доказать.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Козлова Е.С. Методические указания к лабораторной работе №1 по курсу «Параллельное программирование» Методические указания к лабораторным работам [Текст]/ Сост. Козлова Е.С. – Самара, 2017. 12 с.

[2] Параллельные вычисления [Электронный ресурс] // Википедия: свободная энцикл. – Электрон. дан. – [Б. м.], 2017. – URL: https://en.wikipedia.org/wiki/Concurrent_computing (дата обращения: 10.09.2018).

[3] Описание библиотеки CuBLAS [Электронный ресурс] // Cuda Toolkit Documentation : – Электрон. дан. – [Б. м.], 2015. – URL: https://docs.nvidia.com/cuda/cublas/index.html (дата обращения: 28.11.2018)

[4] CUDA [Электронный ресурс] // Википедия: свободная энцикл. – Электрон. дан. – [Б. м.], 2018. – URL: https://en.wikipedia.org/wiki/CUDA (дата обращения: 15.11.2018).

[5] Страуструп, Б. Язык программирования C++ [Текст]/ / Б. Страуструп. – М:Бином, 2011 –1136 с

ПРИЛОЖЕНИЕ А
КОД ПРОГРАММЫ

```
#include <cstdlib>
#include <curand.h>
#include <cublas_v2.h>
#include <iostream>

//GPU_fill_rand() - Функция случайной генерации матрицы
//gpu_blas_mmul() - Функция умножения матриц
//print_matrix() - Функция вывода матрицы
void print_matrix(float *a, int n, int m) {
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < m; j++) {
      printf("%f ", a[i * m + j]);
    }
    printf("\n");
  }
}


// Fill the array A(nr_rows_A, nr_cols_A) with random numbers
on GPU
void GPU_fill_rand(float *A, int nr_rows_A, int nr_cols_A) {
  // Create a pseudo-random number generator
  curandGenerator_t prng;
  curandCreateGenerator(&prng, CURAND_RNG_PSEUDO_DEFAULT);
  // Set the seed for the random number generator using the
system clock
  curandSetPseudoRandomGeneratorSeed(prng, (unsigned long
long)clock());
  // Fill the array with random numbers on the device
  curandGenerateUniform(prng, A, nr_rows_A * nr_cols_A);
}

// Multiply the arrays A and B on GPU and save the result in C
// C(m,n) = A(m,k) * B(k,n)
void gpu_blas_mmul(const float *A, const float *B, float *C,
const int m, const int k, const int n) {
  int lda=m,ldb=k,ldc=m;
  const float alf = 1;
  const float bet = 0;
  const float *alpha = &alf;
  const float *beta = &bet;
  // Create a handle for CUBLAS
  cublasHandle_t handle;
  cublasCreate(&handle);
  // Do the actual multiplication
```

```
  cublasSgemm(handle, CUBLAS_OP_N, CUBLAS_OP_N, m, n, k,
alpha, A, lda, B, ldb, beta, C, ldc);
  // Destroy the handle
  cublasDestroy(handle);
}

int main() {
  // Allocate 3 arrays on CPU
  int nr_rows_A, nr_cols_A, nr_rows_B, nr_cols_B, nr_rows_C,
nr_cols_C;

  // for simplicity we are going to use square arrays
  nr_rows_A = nr_cols_A = nr_rows_B = nr_cols_B = nr_rows_C =
nr_cols_C = 1000;
  float *h_A = (float *)malloc(nr_rows_A * nr_cols_A *
sizeof(float));
  float *h_B = (float *)malloc(nr_rows_B * nr_cols_B *
sizeof(float));
  float *h_C = (float *)malloc(nr_rows_C * nr_cols_C *
sizeof(float));
  // Allocate 3 arrays on GPU
  float *d_A, *d_B, *d_C;
  cudaMalloc(&d_A,nr_rows_A * nr_cols_A * sizeof(float));
  cudaMalloc(&d_B,nr_rows_B * nr_cols_B * sizeof(float));
  cudaMalloc(&d_C,nr_rows_C * nr_cols_C * sizeof(float));
  // Fill the arrays A and B on GPU with random numbers
  GPU_fill_rand(d_A, nr_rows_A, nr_cols_A);
  GPU_fill_rand(d_B, nr_rows_B, nr_cols_B);
  // Optionally we can copy the data back on CPU and print the
arrays
  cudaMemcpy(h_A,d_A,nr_rows_A * nr_cols_A *
sizeof(float),cudaMemcpyDeviceToHost);
  cudaMemcpy(h_B,d_B,nr_rows_B * nr_cols_B *
sizeof(float),cudaMemcpyDeviceToHost);

  // Создание обработчиков событий
  cudaEvent_t start, stop;
  float gpuTime = 0.0f;
  cudaEventCreate(&start);
  cudaEventCreate(&stop);

  // Установка точки старта
  cudaEventRecord(start, 0);

  // Multiply A and B on GPU
  gpu_blas_mmul(d_A, d_B, d_C, nr_rows_A, nr_cols_A,
nr_cols_B);
  // Copy (and print) the result on host memory
```

```
  cudaGetLastError();
  // Синхронизация устройств
  cudaDeviceSynchronize();
  // Установка точки окончания
  cudaEventRecord(stop, 0);

  // Расчет времени
  cudaEventElapsedTime(&gpuTime, start, stop);
  printf("N = %d, time spent executing %s: %.9f seconds\n",
nr_rows_A, "kernel", gpuTime/1000);

  cudaMemcpy(h_C,d_C,nr_rows_C * nr_cols_C *
sizeof(float),cudaMemcpyDeviceToHost);
  // std::cout << "A =" << std::endl;
  // print_matrix(h_A, nr_rows_A, nr_cols_A);
  // std::cout << "B =" << std::endl;
  // print_matrix(h_B, nr_rows_B, nr_cols_B);
  // std::cout << "C =" << std::endl;
  // print_matrix(h_C, nr_rows_C, nr_cols_C);

  //Free GPU memory
  cudaFree(d_A);
  cudaFree(d_B);
  cudaFree(d_C);
  // Free CPU memory
  free(h_A);
  free(h_B);
  free(h_C);
  return 0;
}
```

ПРИЛОЖЕНИЕ Б
КОД СКРИПТОВ ЗАПУСКА

Основной скрипт

```
#!/bin/bash

set -euxo pipefail

nvcc -g -G -O0 -lcublas -I/home/COMMON/cuda-
6.5/samples/common/inc/ main.cu -o main

qsub \
 -V \
 -l "nodes=1:ppn=1:gpu" \
 -l walltime=00:01:00 \
 -N "svxf_mpi_lab4_gpu" \
 -j oe \
 -A tk \
 -o ./output \
 qsub_script.sh
```

Скрипт непосредственно запуска

```
#!/bin/bash

cd $PBS_O_WORKDIR

export
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/home/COMMON/cuda6.5/lib64

./main
```