МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

ВЯТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНСТИТУТ МАТЕМАТИКИ И ИНФОРМАЦИОННЫХ СИСТЕМ

ФАКУЛЬТЕТ КОМПЬЮТЕРНЫХ И ФИЗИКО-МАТЕМАТИЧЕСКИХ НАУК

КАФЕДРА ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

Отчет по лабораторной №1

по дисциплине «Системное программирование»

**Исследование особенностей реализации**

**классического алгоритма умножения матриц**

|  |  |  |  |
| --- | --- | --- | --- |
| Выполнил: | студент гр. ФИБ-3301-51-01 |  | / В. Р. Кочкин / |
| Проверил: | к.ф.-м.н. доцент каф. ПМиИ |  | / В. А. Бызов / |

Киров 2022

Цель работы

Рассмотреть различные способы хранения матриц, исследовать их умножение.

Задания

**Задание 1**

Создать две целочисленные матрицы размера N × N с использованием

двумерных динамических массивов. Реализовать классический алгоритм

умножения матриц.

Провести тестирование программы на матрицах размерности N = 512, 1024 и

2048 без оптимизации (с ключом –O0 или –Od, в конфигурации Debug в MS Visual

Studio) и с ключом оптимизации –O2 (конфигурация Release в MS Visual Studio).

На каждом примере запустить не менее 3 раз. В таблицу занести среднее время

выполнения на одном примере в секундах. Сделать выводы.

**Полученные результаты**

*Таблица 1 – Время выполнения алгоритма умножения матриц*

*с двойными указателями, с*

|  |  |  |
| --- | --- | --- |
| Размер матрицы | Ключ оптимизации | |
| -О0 | -О2 |
| 512 | 0.671117 | 0.33663 |
| 1024 | 7.26515 | 3.01879 |
| 2048 | 81.4434 | 73.5588 |

Из таблицы видно, что с ключом оптимизации -O2(Release) среднее время выполнения программы меньше, чем с ключом -O0(Debug). Далее при выполнении заданий будет использоваться конфигурация Release.

Листинг программы приведен в [приложении А задание 1](#_Задание_1.).

**Задание 2**

Создать две целочисленные матрицы размера N × N с использованием

одномерных динамических массивов. Реализовать классический алгоритм

умножения матриц.

Провести тестирование программы на матрицах размерности N = 512, 1024 и

2048 с ключом оптимизации –O2 (конфигурация Release в MS Visual Studio).

На каждом примере запустить не менее 3 раз. В таблицу занести среднее время

выполнения на одном примере в секундах. Сравнить полученные результаты.

Сделать выводы.

*Таблица 2 – Время выполнения алгоритма умножения матриц*

*с одинарными и двойными указателями, с*

|  |  |  |
| --- | --- | --- |
| Размер матрицы | Способ хранения матрицы в памяти | |
| двойные указатели | одинарные указатели |
| 512 | 0.33663 | 0.393739 |
| 1024 | 3.01879 | 11.6257 |
| 2048 | 73.5588 | 105.096 |

Из таблицы видно, что время выполнения умножения матрицы, хранящихся в двумерных динамических массивах, меньше, чем умножения матриц, хранящихся в одномерных динамических массивах.

Листинг программы приведен в [приложении А задание 2](#_Задание_2.).

**Задание 3**

Для матриц, хранящихся в двумерных динамических массивах, реализовать

классический алгоритм умножения со всеми возможными перестановками порядка

циклов.

Провести тестирование программ на матрицах размерности N = 512, 1024 и

2048. На каждом примере запустить не менее 3 раз. В таблицу занести среднее

время выполнения на одном примере в секундах. Сравнить результаты. Сделать

выводы.

*Таблица 3 – Время выполнения классического алгоритма умножения матриц с двойными указателями, с*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Размерность | Порядок циклов | | | | | |
| ijk | ikj | jik | jki | kij | kji |
| 512 | 0.33663 | 0.224657 | 0.335284 | 0.399877 | 0.238298 | 0.492492 |
| 1024 | 3.01879 | 1.69388 | 2.84366 | 19.6073 | 7.51839 | 17.8677 |
| 2048 | 73.5588 | 19.8535 | 67.8934 | 155.924 | 15.7463 | 170.597 |

Из таблицы видно, что две самые удачные перестановки порядка циклов – ikj и kij, самые неудачные – kji и jik.

**Задание 4**

Выполнить задание 3 с использованием представления матриц в виде

одномерных динамических массивов.

*Таблица 4 – Время выполнения классического алгоритма умножения матриц с одинарными указателями, с*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Размерность | Порядок циклов | | | | | |
| ijk | ikj | jik | jki | kij | kji |
| 512 | 0.393739 | 0.113033 | 0.383332 | 0.584201 | 0.126207 | 0.586687 |
| 1024 | 11.6257 | 0.799954 | 14.9245 | 14.3784 | 1.06703 | 14.6665 |
| 2048 | 105.096 | 6.70179 | 107.1 | 233.203 | 8.57851 | 230.174 |

Из таблицы видно, что две самые удачные перестановки порядка циклов – ikj и kij, самые неудачные – kji и jik.

Вывод

В ходе лабораторной работы я рассмотрел различные способы хранения матриц (одинарные и двойные указатели) и исследовал их умножение. Были произведены замеры времени выполнения программ и на их основании сделаны выводы. Все тесты проводились на процессоре AMD Ryzen 5 3500U 4/8 2.1 ГГц - 3.7 ГГц.

# **Приложения**

## **Приложение А. Листинги программ**

### Задание 1.

#include <iostream>  
#include <cstdlib>  
#include <chrono>  
#include <ctime>  
  
int\*\* create\_matrix(int);  
void fill\_matrix(int\*\*, int, int, int);  
void delete\_matrix(int\*\*, int);  
void print\_matrix(int\*\*, int);  
int\*\* multiply\_matrix(int\*\*, int\*\*, int);  
double calculate\_multiply\_time(int\*\*, int\*\*, int);  
  
const int SIZE = 512;  
const int LEFT\_BORDER = 0;  
const int RIGHT\_BORDER = 10 + 1;  
  
int main()  
{  
 srand(time(nullptr));  
  
 int\*\* matrix\_1 = create\_matrix(SIZE);  
 int\*\* matrix\_2 = create\_matrix(SIZE);  
 fill\_matrix(matrix\_1, SIZE, LEFT\_BORDER, RIGHT\_BORDER);  
 fill\_matrix(matrix\_2, SIZE, LEFT\_BORDER, RIGHT\_BORDER);  
  
 */\*print\_matrix(matrix\_1, SIZE);  
 std::cout << std::endl;  
 print\_matrix(matrix\_2, SIZE);  
 std::cout << std::endl;\*/* double time = calculate\_multiply\_time(matrix\_1, matrix\_2, SIZE);  
 std::cout << "Time to multiply: " << time << "s" << std::endl;  
  
 delete\_matrix(matrix\_1, SIZE);  
 delete\_matrix(matrix\_2, SIZE);  
}  
  
int\*\* create\_matrix(int size)  
{  
 int\*\* matrix = new int\* [size];  
 for (int i = 0; i < size; i++) {  
 matrix[i] = new int[size];  
 }  
  
 for (int i = 0; i < size; i++) {  
 for (int j = 0; j < size; j++) {  
 matrix[i][j] = 0;  
 }  
 }  
  
 return matrix;  
}  
  
void fill\_matrix(int\*\* matrix, int size, int left\_border, int right\_border)  
{  
 for (int i = 0; i < size; i++) {  
 for (int j = 0; j < size; j++) {  
 matrix[i][j] = rand() % right\_border - left\_border;  
 }  
 }  
}  
  
void delete\_matrix(int\*\* matrix, int size)  
{  
 for (int i = 0; i < size; i++) {  
 delete[] matrix[i];  
 }  
}  
  
void print\_matrix(int\*\* matrix, int size)  
{  
 for (int i = 0; i < size; i++) {  
 for (int j = 0; j < size; j++) {  
 std::cout << matrix[i][j] << " ";  
 }  
 std::cout << std::endl;  
 }  
}  
  
int\*\* multiply\_matrix(int\*\* matrix\_1, int\*\* matrix\_2, int size)  
{  
 int\*\* result = create\_matrix(size);  
 for (int i = 0; i < size; i++) {  
 for (int j = 0; j < size; j++) {  
 for (int k = 0; k < size; k++) {  
 result[i][j] += matrix\_1[i][k] \* matrix\_2[k][j];  
 }  
 }  
 }  
  
 return result;  
}  
  
double calculate\_multiply\_time(int\*\* matrix\_1, int\*\* matrix\_2, int size)  
{  
 std::chrono::time\_point<std::chrono::high\_resolution\_clock> start, end;  
 start = std::chrono::high\_resolution\_clock::now();  
  
 int\*\* result = multiply\_matrix(matrix\_1, matrix\_2, size);  
  
 end = std::chrono::high\_resolution\_clock::now();  
 std::chrono::duration<double> diff = end - start;  
  
 */\*print\_matrix(result, size);  
 std::cout << std::endl;\*/* return diff.count();  
}

### Задание 2.

#include <iostream>  
#include <cstdlib>  
#include <chrono>  
#include <ctime>  
  
int\* create\_matrix(int);  
void fill\_matrix(int\*, int, int, int);  
void delete\_matrix(const int\*, int);  
void print\_matrix(int\*\*, int);  
int\* multiply\_matrix(const int\*, const int\*, int);  
double calculate\_multiply\_time(int\*, int\*, int);  
  
const int SIZE = 512;  
const int LEFT\_BORDER = 0;  
const int RIGHT\_BORDER = 10 + 1;  
  
int main()  
{  
 srand(time(nullptr));  
  
 int\* matrix\_1 = create\_matrix(SIZE);  
 int\* matrix\_2 = create\_matrix(SIZE);  
 fill\_matrix(matrix\_1, SIZE, LEFT\_BORDER, RIGHT\_BORDER);  
 fill\_matrix(matrix\_2, SIZE, LEFT\_BORDER, RIGHT\_BORDER);  
  
 */\*print\_matrix(matrix\_1, SIZE);  
 std::cout << std::endl;  
 print\_matrix(matrix\_2, SIZE);  
 std::cout << std::endl;\*/* double time = calculate\_multiply\_time(matrix\_1, matrix\_2, SIZE);  
 std::cout << "Time to multiply: " << time << "s" << std::endl;  
  
 delete\_matrix(matrix\_1, SIZE);  
 delete\_matrix(matrix\_2, SIZE);  
}  
  
int\* create\_matrix(int size)  
{  
 int\* matrix = new int [size \* size];  
  
 for (int i = 0; i < size \* size; i++) {  
 matrix[i] = 0;  
 }  
  
 return matrix;  
}  
  
void fill\_matrix(int\* matrix, int size, int left\_border, int right\_border)  
{  
 for (int i = 0; i < size \* size; i++) {  
 matrix[i] = rand() % right\_border - left\_border;  
 }  
}  
  
void delete\_matrix(const int\* matrix, int size)  
{  
 delete[] matrix;  
}  
  
void print\_matrix(int\* matrix, int size)  
{  
 for (int i = 0; i < size \* size; i++) {  
 if (i != 0 && i % size == 0) {  
 std::cout << std::endl;  
 }  
 std::cout << matrix[i] << " ";  
 }  
}  
  
int\* multiply\_matrix(const int\* matrix\_1, const int\* matrix\_2, int size)  
{  
 int\* result = create\_matrix(size);  
 for (int i = 0; i < size; i++) {  
 for (int j = 0; j < size; j++) {  
 for (int k = 0; k < size; k++) {  
 result[size \* i + j] += matrix\_1[size \* i + k] \* matrix\_2[size \* k + j];  
 }  
 }  
 }  
  
 return result;  
}  
  
double calculate\_multiply\_time(int\* matrix\_1, int\* matrix\_2, int size)  
{  
 std::chrono::time\_point<std::chrono::high\_resolution\_clock> start, end;  
 start = std::chrono::high\_resolution\_clock::now();  
  
 int\* result = multiply\_matrix(matrix\_1, matrix\_2, size);  
  
 end = std::chrono::high\_resolution\_clock::now();  
 std::chrono::duration<double> diff = end - start;  
  
 */\*print\_matrix(result, size);  
 std::cout << std::endl;\*/* return diff.count();  
}