МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ   
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ   
«ВЯТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

ФАКУЛЬТЕТ КОМПЬЮТЕРНЫХ И ФИЗИКО-МАТЕМАТИЧЕСКИХ НАУК

КАФЕДРА ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

Отчёт по лабораторной работе №2  
по дисциплине «Параллельные вычисления»

**Исследование особенностей реализации классического алгоритма умножения матриц**

Выполнил: студент группы ФИб-4301-51-00     / К.О. Дёмин /

Проверил: ст.преподаватель каф. ПМИ     / А.В. Торбеева /

Киров 2019

Оглавление

[Цель работы 3](#_Toc20734266)

[Задание 3](#_Toc20734267)

[Листинг 4](#_Toc20734268)

[Полученные результаты 8](#_Toc20734269)

[Выводы по лабораторной работе 9](#_Toc20734270)

# Цель работы

Рассмотреть различные способы хранения матриц, исследовать их умножение.

# Задание

1. Создать две целочисленные матрицы размера *N* × *N* с использованием двумерных динамических массивов. Реализовать классический алгоритм умножения матриц со всеми возможными перестановками порядка циклов.
2. Провести тестирование программ на матрицах размерности *N* = 500, 1000 и 2000. На каждом примере запустить не менее 3 раз. В таблицу занести среднее время выполнения на одном примере в секундах.

Таблица 1 – Время выполнения классического алгоритма умножения матриц

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Размерность | Порядок циклов | | | | | |
| ijk | ikj | jik | jki | kij | kji |
| 500 |  |  |  |  |  |  |
| 1000 |  |  |  |  |  |  |
| 2000 |  |  |  |  |  |  |

1. Выполнить задания 1-2 с использованием представления матриц в виде одномерных динамических массивов.

# Листинг

using System;

using System.Diagnostics;

namespace lab\_2

{

    internal class Program

    {

        /// <summary>

        /// Количество итераций

        /// </summary>

        private const int Iterations = 3;

        private static void Main(string[] args)

        {

            //  вводим N

            Console.WriteLine("Введите N");

            int n = int.Parse(Console.ReadLine());

            //  тестируем

            Test<TwoDimensionalMatrix>(n);

            Test<SingleDimensionalMatrix>(n);

            Console.ReadKey();

        }

        /// <summary>

        /// Тесты

        /// </summary>

        /// <typeparam name="T"></typeparam>

        /// <param name="n"></param>

        private static void Test<T>(int n) where T : Matrix

        {

            var type = typeof(T);  //  получаем тип

            Console.WriteLine(type.Name);  //  выводим тип

            var first = (T)Activator.CreateInstance(type, n);  //  создаём объект первой матрицы

            first.Generate();  //  генерируем

            //Console.WriteLine(first.ToString());

            var second = (T)Activator.CreateInstance(type, n);  //  создаём второй объект

            second.Generate(227);  //  генерируем с сидом

            //Console.WriteLine(second.ToString());

            Stopwatch stopwatch = new Stopwatch();  //  объект для замерения времени

            TimeSpan result;  //  время работы

            for (int i = 0; i < first.Muls.Length; ++i)

            {

                result = new TimeSpan();

                for (int iter = 0; iter < Iterations; ++iter)

                {

                    stopwatch.Restart();  //  перезапускаем замер времени

                    first.Muls[i](second);  //  перемножаем

                    stopwatch.Stop();  //  заканчиваем замерение

                    result += stopwatch.Elapsed;  //  добавляем время

                }

                Console.WriteLine($"{first.Muls[i].Method.Name}: {result.TotalSeconds / Iterations:f3} сек.");  //  выводим информацию

            }

        }

    }

}

using System;

using System.Text;

namespace lab\_2

{

    internal abstract class Matrix

    {

        /// <summary>

        /// Делегат

        /// </summary>

        /// <param name="second"></param>

        /// <returns></returns>

        public delegate Matrix Mul(Matrix second);

        /// <summary>

        /// Методы умножения матриц

        /// </summary>

        public Mul[] Muls;

        /// <summary>

        /// Размеры матриц

        /// </summary>

        public int Size { get; }

        public Matrix MulIJK(Matrix second)

        {

            Matrix result = (Matrix)Activator.CreateInstance(this.GetType(), this.Size);

            for (int i = 0; i < result.Size; ++i)

                for (int j = 0; j < result.Size; ++j)

                    for (int k = 0; k < result.Size; ++k)

                        result[i, j] += this[i, k] \* second[k, j];

            return result;

        }

        public Matrix MulIKJ(Matrix second)

        {

            Matrix result = (Matrix)Activator.CreateInstance(this.GetType(), this.Size);

            for (int i = 0; i < result.Size; ++i)

                for (int k = 0; k < result.Size; ++k)

                    for (int j = 0; j < result.Size; ++j)

                        result[i, j] += this[i, k] \* second[k, j];

            return result;

        }

        public Matrix MulJIK(Matrix second)

        {

            Matrix result = (Matrix)Activator.CreateInstance(this.GetType(), this.Size);

            for (int j = 0; j < result.Size; ++j)

                for (int i = 0; i < result.Size; ++i)

                    for (int k = 0; k < result.Size; ++k)

                        result[i, j] += this[i, k] \* second[k, j];

            return result;

        }

        public Matrix MulJKI(Matrix second)

        {

            Matrix result = (Matrix)Activator.CreateInstance(this.GetType(), this.Size);

            for (int j = 0; j < result.Size; ++j)

                for (int k = 0; k < result.Size; ++k)

                    for (int i = 0; i < result.Size; ++i)

                        result[i, j] += this[i, k] \* second[k, j];

            return result;

        }

        public Matrix MulKIJ(Matrix second)

        {

            Matrix result = (Matrix)Activator.CreateInstance(this.GetType(), this.Size);

            for (int k = 0; k < result.Size; ++k)

                for (int i = 0; i < result.Size; ++i)

                    for (int j = 0; j < result.Size; ++j)

                        result[i, j] += this[i, k] \* second[k, j];

            return result;

        }

        public Matrix MulKJI(Matrix second)

        {

            Matrix result = (Matrix)Activator.CreateInstance(this.GetType(), this.Size);

            for (int k = 0; k < result.Size; ++k)

                for (int j = 0; j < result.Size; ++j)

                    for (int i = 0; i < result.Size; ++i)

                        result[i, j] += this[i, k] \* second[k, j];

            return result;

        }

        /// <summary>

        /// Перегрузка [,]

        /// </summary>

        /// <param name="y"></param>

        /// <param name="x"></param>

        /// <returns></returns>

        public abstract int this[int y, int x] { get; set; }

        /// <summary>

        /// Конструктор

        /// </summary>

        /// <param name="size"></param>

        protected Matrix(int size)

        {

            Size = size;

            Muls = new Mul[]

            {

                MulIJK,

                MulIKJ,

                MulJIK,

                MulJKI,

                MulKIJ,

                MulKJI

            };

        }

        /// <summary>

        /// Построение строки

        /// </summary>

        /// <returns></returns>

        public override string ToString()

        {

            StringBuilder stringBuilder = new StringBuilder();

            for (int y = 0; y < Size; ++y)

            {

                for (int x = 0; x < Size; ++x)

                    stringBuilder.Append(this[y, x] + " ");

                stringBuilder.Append("\n");

            }

            return stringBuilder.ToString();

        }

        /// <summary>

        /// Генерирование матрицы

        /// </summary>

        /// <param name="seed"></param>

        public void Generate(int seed = 0)

        {

            Random rand = new Random(seed);

            for (int y = 0; y < Size; ++y)

            {

                for (int x = 0; x < Size; ++x)

                {

                    this[y, x] = rand.Next(10);

                }

            }

        }

    }

}

namespace lab\_2

{

    /// <summary>

    /// массив

    /// </summary>

    internal class SingleDimensionalMatrix : Matrix

    {

        private int[] arr;  //  массив

        /// <summary>

        /// конструктор

        /// </summary>

        /// <param name="size"></param>

        public SingleDimensionalMatrix(int size) : base(size)

        {

            //  создаём массив

            arr = new int[size \* size];

        }

        /// <summary>

        ///  перегрузка оператора this[y,x]

        /// </summary>

        /// <param name="y"></param>

        /// <param name="x"></param>

        /// <returns></returns>

        public override int this[int y, int x]

        {

            get

            {

                return arr[y \* Size + x];

            }

            set

            {

                arr[y \* Size + x] = value;

            }

        }

    }

}

namespace lab\_2

{

    /// <summary>

    /// матрица двумерный массив

    /// </summary>

    internal class TwoDimensionalMatrix : Matrix

    {

        private int[,] arr;  //  массив

        /// <summary>

        /// Конструктор

        /// </summary>

        /// <param name="size"></param>

        public TwoDimensionalMatrix(int size) : base(size)

        {

            arr = new int[Size, Size];

        }

        /// <summary>

        /// Оператор this

        /// </summary>

        /// <param name="y"></param>

        /// <param name="x"></param>

        /// <returns></returns>

        public override int this[int y, int x]

        {

            get

            {

                return arr[y, x];

            }

            set

            {

                arr[y, x] = value;

            }

        }

    }

}

# Полученные результаты

Результаты, полученные при двумерном динамическом массиве:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Размерность | Порядок циклов | | | | | |
| ijk | ikj | jik | jki | kij | kji |
| 500 | 1,959 | 1,607 | 1,831 | 1,808 | 1,642 | 1,788 |
| 1000 | 15,104 | 11,144 | 13,121 | 23,740 | 11,375 | 23,731 |
| 2000 | 222,859 | 111,131 | 174,167 | 260,244 | 106,771 | 285,023 |

Результаты, полученные при одномерном динамическом массиве:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Размерность | Порядок циклов | | | | | |
| ijk | ikj | jik | jki | kij | kji |
| 500 | 1,421 | 1,262 | 1,510 | 1,584 | 1,273 | 1,684 |
| 1000 | 12,619 | 8,923 | 10,689 | 18,579 | 8,726 | 18,274 |
| 2000 | 211,689 | 95,172 | 171,503 | 241,335 | 92,538 | 235,669 |

Как показало тестирование, наибольшая производительность достигается при перемножении с порядком циклов: k, i, j и одномерным динамическим массивом.

# Выводы по лабораторной работе

Таким образом, правильная работа с памятью позволяет достичь большей производительности. Так как блоки реже перемещаются из озу в кэш, процессор совершает больше операций и не простаивает. Кроме того, одномерный массив хранится последовательно, что позволяет последовательно брать данные, не собирая их из разных областей памяти.