Министерство образования Российской Федерации
Московский Государственный Технический Университет
им. Н.Э. Баумана

Отчёт по лабораторной работе № 1
По курсу: "Анализ алгоритмов"
Тема:"Умножение матриц"

Студент: Орехова Е.О. ИУ7-51

Преподаватель: Волкова Л.Л.

4 марта 2018 г.

Содержание

1 Постановка задачи

В ходе выполнения лабораторной работы необходимо реализовать умножение матриц обычным алгоритмом и алгоритмом Винограда. Улучшить алгоритм Винограда. Сравнить эти алгоритмы.

2 Умножение матриц

Пусть даны две матрицы, A и B, размерности $a \times n$ и $n \times b$ соответственно, тогда результатом их умножения будет матрица C, размерности $a \times b$, в которой

$$C_{i,j} = \sum_{k=1}^{n} A_{i,k} * Bk, j \tag{1}$$

3 Алгоритм Винограда

Если посмотреть на результат умножения двух матриц, то видно, что каждый элемент в нем представляет собой скалярное произведение соответствующих строки и столбца исходных матриц. Можно заметить также, что такое умножение допускает предварительную обработку, позволяющую часть работы выполнить заранее. Рассмотрим два вектора $V = (v_1, v_2, v_3, v_4)$ и $W = (w_1, w_2, w_3, w_4)$. Их скалярное произведение равно

$$V * W = v_1 w_1 + v_2 w_2 + v_3 w_3 + v_4 w_4 \tag{2}$$

Это равенство можно переписать в виде:

$$V*W = (v_1+w_2)(v_2+w_1)+(v_3+w_4)(v_4+w_3)-v_1v_2-v_3v_4-w_1w_2-w_3w_4 \tag{3}$$

Из этого следует, что произведение матриц можно выполнить эффективнее, произведя некоторые вычисления заранее.

4 Реализация

Листинг 1: Стандартный алгоритм умножения матриц

```
public static void Simple_Multiplication(ref int[,]C,
   int[,] A, int[,] B, int size)
{
        for (int i = 0; i < size; i++)
```

```
                for (int j = 0; j < size; j++)
                {
                        C[i, j] = 0;
                        for (int k = 0; k < size; k++)
                                C[i, j] = C[i, j] +
                                   A[i,k] * B[k,j];
                }
}
```

Листинг 2: Алгоритм Винограда

```
public static void Vinograd(ref int[,] C,int[,] A,
   int[,] B, int size, int [] Rows, int [] Column)
{

        for (int i = 0; i < size; i++)
        {
                Rows[i] = A[i, 0] * A[i, 1];
                for (int j = 1; j < size/2; j++)
                        Rows[i] = Rows[i]+ A[i, 2 * j] *
                           A[i, 2 * j + 1];
        }


        for (int i = 0; i < size; i++)
        {
                Column[i] = B[0, i] * B[1, i];
                for (int j = 1; j < size/2; j++)
                        Column[i] = Column[i]+ B[2 * j,
                           i] * B[2 * j + 1, i];
        }


        for(int i = 0; i< size; i++)
                for (int j = 0; j<size;j++)
                {
                        C[i, j] = -Rows[i] - Column[j];
                        for (int k = 0; k < size/2; k++)
                                C[i, j] = C[i,
                                   j]+(A[i,2*k+1]+B[2*k,j])
                                   *
                                   (A[i,2*k]+B[2*k+1,j]);
                }
        if (size % 2 == 1)
```

```
        {
                for (int i = 0; i < size; i++)
                        for (int j = 0; j < size; j++)
                                C[i, j] = C[i, j]+
                                   A[i,size-1] *
                                   B[size-1,j];
        }
}
```

Листинг 3: Улучшенный алгоритм Винограда

```
public static void Vinograd_Better(ref int[,] C, int[,]
   A, int[,] B, int size, int[] Rows, int[] Column)
{
        int d = size / 2;
        int new_size = size - 1;

        for (int i = 0; i < size; i++)
        {
                Rows[i] = A[i, 0] * A[i, 1];
                for (int j = 2; j < new_size; j+=2)
                        Rows[i] += A[i, j] * A[i, j + 1];
        }

        for (int i = 0; i < size; i++)
        {
                Column[i] = B[0, i] * B[1, i];
                for (int j = 2; j < new_size; j+=2)
                        Column[i] += B[j, i] * B[j + 1,
                           i];
        }

        if (size % 2 == 1)
        {
                for (int i = 0; i < size; i++)
                        for (int j = 0; j < size; j++)
                        {
                                C[i, j] = -Rows[i] -
                                   Column[j] + A[i, size
                                   - 1] * B[size - 1, j];
                                for (int k = 0; k < d;
                                   k++)
                                        C[i, j] += (A[i,
                                           2 * k + 1] +
```

```
                                                B[2 * k, j])
                                                * (A[i, 2 *
                                                k] + B[2 * k
                                                + 1, j]);
                                }
        }
        else
        {
                for (int i = 0; i < size; i++)
                        for (int j = 0; j < size; j++)
                        {
                                C[i, j] = -Rows[i] -
                                   Column[j];
                                for (int k = 0; k < d;
                                   k++)
                                        C[i, j] += (A[i,
                                           2 * k + 1] +
                                           B[2 * k, j])
                                           * (A[i, 2 *
                                           k] + B[2 * k
                                           + 1, j]);
                                }
        }
}
```

5 Теоретическая оценка

Операции, имеющие сложность 1: +, -, * ,/, <, >, <=, >=, =, ==, +=, -=, /=, *=, []

Размер исходных матриц $N \times N$.

5.1 Стандартный алгоритм:

$$f = 2 + N(2 + 2 + N(2 + 3 + 2 + N(2 + 11))) = 2 + 4N + N^2(7 + 13N) =$$
$$= 2 + 4N + 7N^2 + 13N^3$$

5.2 Алгоритм Винограда:

в лучшем случае:

$$f_{best} = 2(2 + N(2 + 6 + 3 + \frac{N}{2}(2 + 12))) + 2 + N(2 + 2 + N(2 + 7 + 3 + N(2 + 22))) + 2 = \\ = 24N^3 + 26N^2 + 26N + 8$$

в худшем случае:

$$f_{worst} = f_{best} + 2 + N(2 + 2 + N(2 + 13)) = 24N^3 + 41N^2 + 30N + 10$$

5.3 Улучшенный алгоритм Винограда:

Эффективность алгоритма улучшена следующими операциями:

1. Для сокращения операций в цикле в начале алгоритма вычисляется $d = \frac{N}{2}$.
2. Операции вида $x = x + k$ заменены на $x+ = k$.
3. Проверка на четность выполняется вначале цикла.

Общая часть:

$$f = 2 + 2 + 2(2 + N(2 + 7 + 2 + \frac{N}{2}(2 + 8))) + 2 = 10 + 22N + 10N^2$$

в худшем случае:

$$f_{worst} = f + 2 + N(2 + 2 + N(2 + 15 + 2 + \frac{N}{2}(2 + 20))) = \\ = f + 2 + 4N + 19N^2 + 11N^3 = 12 + 26N + 29N^2 + 11N^3$$

в лучшем случае:

$$f_{best} = f + 2 + N(2 + 2 + N(2 + 7 + 2 + \frac{N}{2}(2 + 20))) = f + 2 + 4N + 11N^2 + 11N^3 = \\ = 12 + 26N + 22N^2 + 11N^3$$

6 Эксперимент

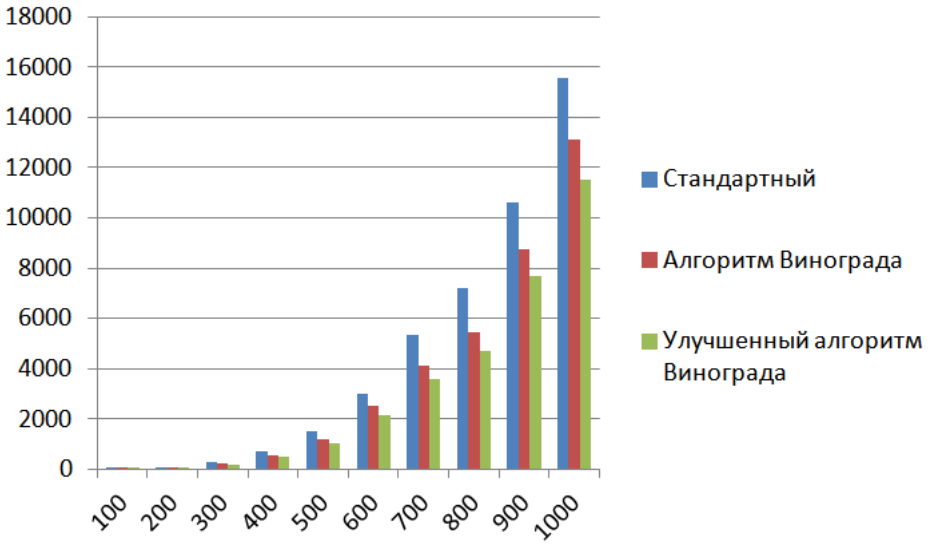

Рис. 1: Время умножения матриц в мс.

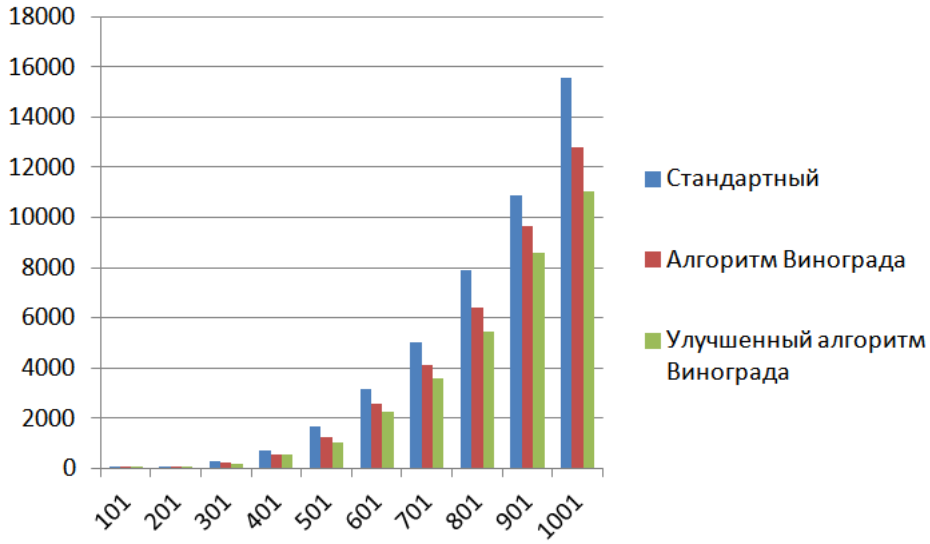

Рис. 2: Время умножения матриц в мс.

Из диаграмм видно, что алгоритм Винограда выполняется быстрее, несмотря на то, что имеет большую сложность. Это можно объяснить тем, что

реальная сложность умножения относительно сложения на вычислительной машине отличается от теоретической. Поэтому алгоритм Винограда работает быстрее.

7 Заключение

В ходе выполнения лабораторной работы были изучены и реализованы различные алгоритмы умножения матриц. Получены временные характеристики.