Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

 «Пермский национальный исследовательский политехнический университет»

Электротехнический факультет

Кафедра «Информационные технологии и автоматизированные системы»

Направление 09.03.04 – «Программная инженерия»

**ОТЧЕТ**

по лабораторной работе №2

на тему: «Перемножение матриц»

Выполнил: студент группы РИС-20-1б

Шумилов Л.С.

Проверил: доцент кафедры ИТАС

Щапов В.А.

Пермь, 2024

**Цель работы** – сравнить различные способы обхода матриц в процессе их умножения.

**Задание**:

1. Реализовать все 6 способов умножения матриц;
2. Сравнить время выполнения каждого из них, найти самый быстрый.

**Ход работы**

Начнём с общеизвестного «Наивного» умножения матриц, схема которой представлена на Рисунке 1.



Рисунок . Наивное умножение матриц

В коде, данный способ представлен в Листинге 1.

Листинг 1. «Наивное» умножение матриц

static void DoNaiveMultiplication(int[,] a, int[,] b, int[,] target)

{

for (var i = 0; i < Length; i += 1)

{

for (var j = 0; j < Length; j += 1)

{

for (var k = 0; k < Length; k += 1)

{

target[i, j] += a[i, k] \* b[k, j];

}

}

}

}

Стоит обратить внимание на перебираемые параметры i, j, k. Эти параметры можно менять местами. Так получаем 2! = 6 вариантов перемножения матриц.

На Листинге 2. Представлены остальные 5 вариантов перемножения матриц, где i, j, k перебираются в разном порядке.

Листинг 2. Пять вариантов перемножения матриц

static void DoSecondMultiplication(int[,] a, int[,] b, int[,] target)

{

for (var i = 0; i < Length; i += 1)

{

for (var k = 0; k < Length; k += 1)

{

for (var j = 0; j < Length; j += 1)

{

target[i, j] += a[i, k] \* b[k, j];

}

}

}

}

static void DoThirdMultiplication(int[,] a, int[,] b, int[,] target)

{

for (var j = 0; j < Length; j += 1)

{

for (var i = 0; i < Length; i += 1)

{

for (var k = 0; k < Length; k += 1)

{

target[i, j] += a[i, k] \* b[k, j];

}

}

}

}

static void DoFourthMultiplication(int[,] a, int[,] b, int[,] target)

{

for (var j = 0; j < Length; j += 1)

{

for (var k = 0; k < Length; k += 1)

{

for (var i = 0; i < Length; i += 1)

{

target[i, j] += a[i, k] \* b[k, j];

}

}

}

}

static void DoFifthMultiplication(int[,] a, int[,] b, int[,] target)

{

for (var k = 0; k < Length; k += 1)

{

for (var i = 0; i < Length; i += 1)

{

for (var j = 0; j < Length; j += 1)

{

target[i, j] += a[i, k] \* b[k, j];

}

}

}

}

static void DoSixthMultiplication(int[,] a, int[,] b, int[,] target)

{

for (var k = 0; k < Length; k += 1)

{

for (var j = 0; j < Length; j += 1)

{

for (var i = 0; i < Length; i += 1)

{

target[i, j] += a[i, k] \* b[k, j];

}

}

}

}

На Рисунке 2. представлен результат запуска всех способов перемножения матриц с замером времени на каждый способ.

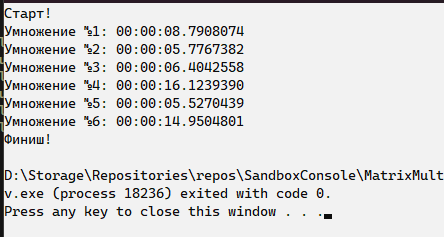


Рисунок . Замер времени перемножения всех 6 способов

Четвёртый и шестой способы показали наихудшее время в виду неэффективного порядка обхода матрицы, при одном столбце перебирается множество строк.

Обход сначала по строкам внутри столбца быстрее в виду наличии кэша в центральном процессоре, который при обращении к одной ячейке в памяти, одновременно обращается к нескольким следующим в конкретном участке памяти и записывает их в кэш. Так, обращение к следующей ячейка памяти многократно ускоряется.

Второй и пятый способ оказались быстрее первого в виду особенности выполнения кода в C# - JIT-компиляции. При таком способе выполнения, при первом выполнении код компилируется «на лету». Второе и последующее выполнение осуществляется уже по ранее скомпилированному коду.