Государственное образовательное учреждение высшего профессионального образования «Московский Государственный Технический Университет имени Н.Э. Баумана»

ОТЧЕТ

По лабораторной работе №4

По курсу «Анализ алгоритмов»

Тема: «Параллельное умножение матриц»

Студент: Жарова Е.А.

Группа ИУ7-51

Москва, 2017

Оглавление

Постановка задачи

Необходимо изучить параллельный алгоритм Винограда для умножения матриц, реализовать его, сравнить зависимость времени работы алгоритма от числа параллельных потоков исполнения и размера матриц. Сравнить стандартный и параллельный алгоритмы.

Алгоритм Винограда

1. Описание

Алгоритм Винограда считается эффективнее за счет сокращения количества операций умножения. Результатом умножения двух матриц является скалярное произведение соответствующих строки и столбца. Такое умножение позволяет выполнить заранее часть работы.

Рассмотрим два вектора $V = (v1, v2, v3, v4)$ и $W = (w1, w2, w3, w4)$, скалярное произведение которых равно: $V \bullet W = v1w1 + v2w2 + v3w3 + v4w4$.

Данное равенство можно переписать следующим образом: $V \bullet W = (v1 + w2)(v2 + w1) + (v3 + w4)(v4 + w3) - v1v2 - v3v4 - w1w2 - w3w4$. Умножения v1v2, v3v4, w1w2, w3w4 можно рассчитать заранее.

Выражение $v1w1 + v2w2 + v3w3 + v4w4$ имеет большую трудоёмкость, чем выражение $(v1 + w2)(v2 + w1) + (v3 + w4)(v4 + w3)$.

2. Реализация

```
Matrix Matrix::vinogradMult(const Matrix &other, std::clock_t& time) const {
	if (columns != other.rows) {
		throw std::logic_error("Sizes don't match!");
	}
	Matrix result(rows, other.columns);

	unsigned* rowFactor = new unsigned[rows];
	unsigned* columnFactor = new unsigned[other.columns];

	std::clock_t start = std::clock();

	for (unsigned i = 0; i < rows; ++i) {
		rowFactor[i] = data[i][0] * data[i][1];
		for (unsigned j = 2; j < columns - 1; j += 2) {
			rowFactor[i] = rowFactor[i] + data[i][j] * data[i][j + 1];
		}
	}

```

```
21.       for (unsigned i = 0; i < other.columns; ++i) {
22.             columnFactor[i] = other.data[0][i] * other.data[1][i];
23.             for (unsigned j = 2; j < other.rows - 1; j += 2) {
24.                   columnFactor[i] = columnFactor[i] + other.data[j][i] *
    other.data[j + 1][i];
25.             }
26.       }
27.
28.       for (unsigned i = 0; i < rows; ++i) {
29.             for (unsigned j = 0; j < other.columns; ++j) {
30.                   result.data[i][j] = 0;
31.                   for (unsigned k = 0; k < columns - 1; k += 2) {
32.                         result.data[i][j] = result.data[i][j] + (data[i][k] +
    other.data[k + 1][j]) *
33.                                     (data[i][k + 1] + other.data[k][j]);
34.                   }
35.                   result.data[i][j] = result.data[i][j] - (rowFactor[i] +
    columnFactor[j]);
36.             }
37.       }
38.
39.       if (columns % 2 != 0) {
40.             unsigned columnsDiv2 = columns / 2;
41.             for (unsigned i = 0; i < rows; ++i) {
42.                   for (unsigned j = 0; j < other.columns; ++j) {
43.                         result.data[i][j] = result.data[i][j] +
    data[i][columnsDiv2] * other.data[columnsDiv2][j];
44.                   }
45.             }
46.       }
47.
48.       time = std::clock() - start;
49.
50.       delete[] columnFactor;
51.       delete[] rowFactor;
52.
53.       return result;
54. }
```

Распараллеливание алгоритма

1. Идея

Трудоёмкость алгоритма Винограда (из результатов лабораторной работы №2) имеет сложность O(nmk) для умножения матриц n×m на m×k. Следовательно, чтобы улучшить алгоритм, необходимо распараллелить ту часть алгоритма, которая содержит три вложенных цикла.

Каждый поток будет выполнять вычисление некоторых строк результирующей матрицы (вычисление результата для каждой строки происходит независимо от результата выполнения умножения для других строк). Это сделано потому что проход по строкам матрицы является более эффективным с точки зрения организации данных в памяти.

2. Реализация

```
3.  Matrix Matrix::vinogradParallel(const Matrix& other, const unsigned
    numberOfThreads, std::clock_t& time) const {
4.
5.        if (columns != other.rows) {
6.              throw std::logic_error("Sizes don't match!");
7.        }
8.        Matrix result(rows, other.columns);
9.
10.       unsigned* rowFactor = new unsigned[rows];
11.       unsigned* columnFactor = new unsigned[other.columns];
```

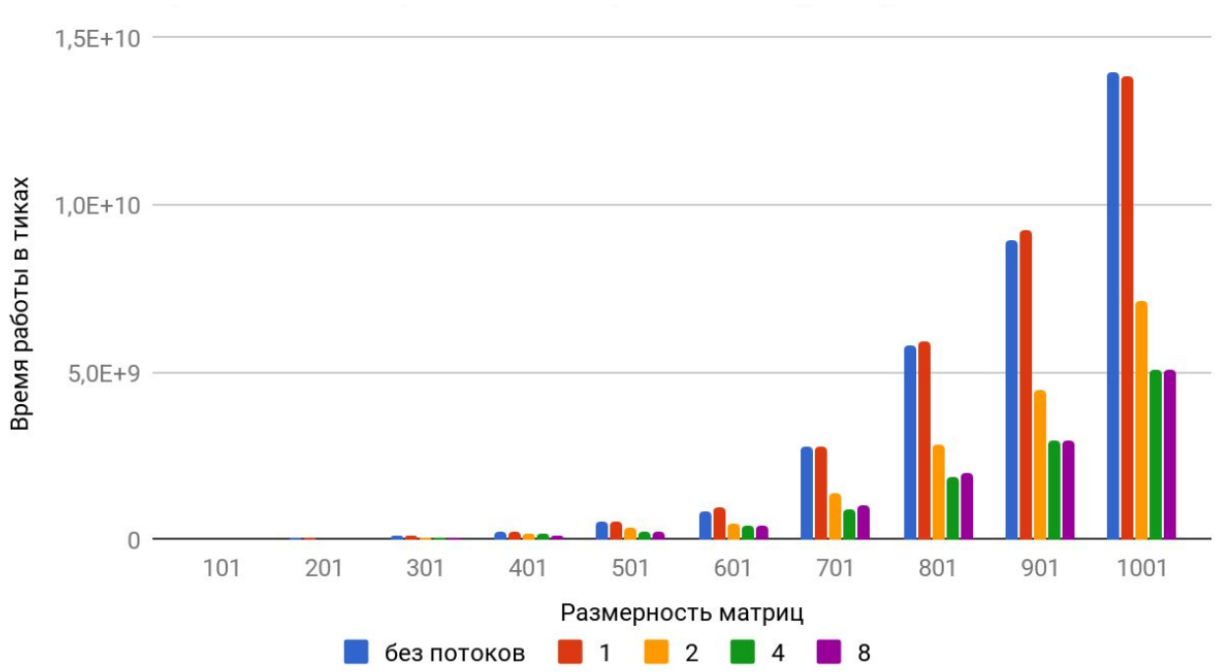
```
12.
13.         std::clock_t start = std::clock();
14.
15.         std::vector<std::thread> threadVector;
16.
17.         const double rowFactorThreadSize = rows / (double) numberOfThreads;
18.         unsigned minRow = 0;
19.         for (unsigned i = 0; i < numberOfThreads; ++i, minRow +=
    rowFactorThreadSize) {
20.                 threadVector.push_back(std::thread(&Matrix::countRowFactor, this,
    minRow, (unsigned) rowFactorThreadSize, rowFactor));
21.         }
22.         for (std::thread& thread : threadVector) {
23.                 if (thread.joinable()) {
24.                         thread.join();
25.                 }
26.         }
27.         threadVector.clear();
28.
29.         const double columnFactorThreadSize = other.columns / (double)
    numberOfThreads;
30.         unsigned minColumn = 0;
31.         for (unsigned i = 0; i < numberOfThreads; ++i, minColumn +=
    columnFactorThreadSize) {
32.                 threadVector.push_back(std::thread(&Matrix::countColumnFactor,
    &other, minColumn,
33.                                                                                 (unsigned)
    columnFactorThreadSize, columnFactor));
34.         }
35.         for (std::thread& thread : threadVector) {
36.                 if (thread.joinable()) {
37.                         thread.join();
38.                 }
39.         }
40.         threadVector.clear();
41.
42.         const double rowByThread = rows / (double) numberOfThreads;
43.         minRow = 0;
44.         for (unsigned i = 0; i < numberOfThreads; ++i, minRow += rowByThread) {
45.                 threadVector.push_back(std::thread(&Matrix::countPartialVinograd,
    this, std::ref(other), minRow, rowByThread,
46.                                                                                 rowFactor,
    columnFactor, std::ref(result)));
47.         }
48.
49.
50.
51.         for (std::thread& thread : threadVector) {
52.                 if (thread.joinable()) {
53.                         thread.join();
54.                 }
55.         }
56.
57.         if (columns % 2 != 0) {
58.                 threadVector.clear();
59.                 minRow = 0;
60.                 for (unsigned i = 0; i < numberOfThreads; ++i, minRow += rowByThread)
    {
61.                         threadVector.push_back(std::thread(&Matrix::unevenVinograd,
    this, std::ref(other), minRow, rowByThread,
62.
    std::ref(result)));
63.                 }
64.
```

```
            for (std::thread& thread : threadVector) {
                if (thread.joinable()) {
                    thread.join();
                }
            }
        }

        time = std::clock() - start;

        delete[] columnFactor;
        delete[] rowFactor;

        return result;
}
```

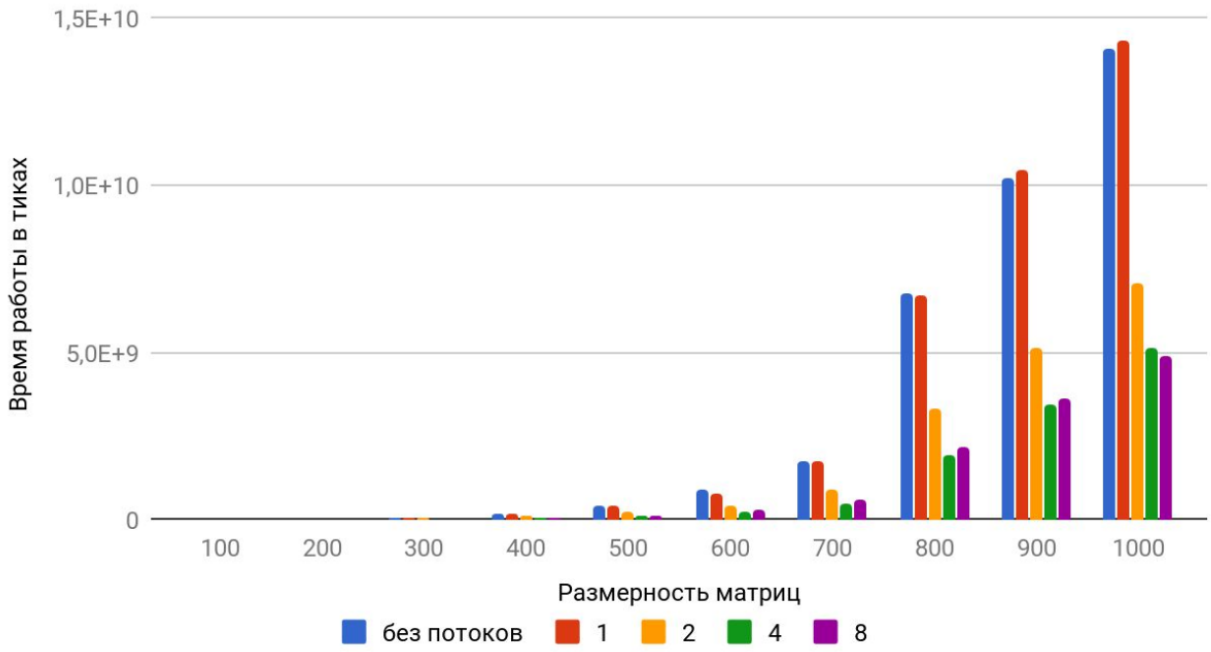
Сравнение алгоритмов

Выполнение программы производилось на компьютере, который может проводить вычисления в 4 реально параллельных потоках (для анализа быстродействия алгоритм был протестирован на 1, 2, 4 и 8 потоков). Для сравнения алгоритма Винограда и распараллеленного алгоритма Винограда было посчитано время работы для матриц размерностью 100×100, 200×200, …, 1000×1000 и 101×101, 201×201, …, 1001×1001.

Анализ быстродействия алгоритмов для матриц нечетных размерностей



Анализ быстродействия алгоритмов для матриц четных размерностей



Выводы:

- Обычный алгоритм Винограда выполняется быстрее параллельного, выполняемого на одном потоке, потому что для подготовки потока необходимы дополнительные ресурсы.
- С увеличением количества потоков (до 2, 4) время работы алгоритма сокращается
- Дальнейшее увеличение количества потоков почти не влияет на повышение быстродействия алгоритма, потому что реально вычисления на данном компьютере выполняются только в 4 потока.

Заключение

Был изучен параллельный алгоритм Винограда для умножения матриц. Он был реализован, а также было произведено сравнение зависимости времени работы алгоритма от числа параллельных потоков исполнения и размера матриц. Было выполнено сравнение стандартного и параллельного алгоритмов.