ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт математики, механики и компьютерных наук им. И.И. Воровича

Кафедра алгебры и дискретной математики

ОТЧЕТ

НА ТЕМУ:

БЛОЧНЫЕ ВЫЧИСЛЕНИЯ. МОДЕЛИ ВРЕМЕНИ ВЫПОЛНЕНИЯ ПРОГРАММ. БЛОЧНЫЕ РАЗМЕЩЕНИЯ МАССИВОВ, ДОПОЛНЯЮЩИЕ БЛОЧНЫЕ ВЫЧИСЛЕНИЯ

Выполнила:
студентка 4 курса 1 группы
Кирпилянская Елена Петровна

Ростов-на-Дону
2018

Содержание

Постановка задачи 3
Алгоритм решения 3
Результаты работы программы 4
Характеристики компьютера 5
Выводы 5
Приложение 1 7

Постановка задачи

Задание 10.

Написать программу блочного умножения двух матриц C = A*B. Матрица A верхне-треугольная. Хранится в виде одномерного массива по блочным строкам. Матрица B нижне-треугольная. Хранится в виде одномерного массива по блочным столбцам. Распараллелить блочную программу умножения двух матриц C = A*B с использованием технологии OpenMP двумя способами

- Перемножение каждых двух блоков выполнить параллельно
- В разных вычислительных ядрах одновременно перемножать разные пары блоков.

Определить оптимальные размеры блоков в обоих случаях. Провести численные эксперименты и построить таблицу сравнений времени выполнения различных программных реализаций решения задачи. Определить лучшие реализации. Проверить корректность (правильность) программ.

Алгоритм решения

$$\begin{pmatrix} 5 & 5 & 1 & 7 \\ 1 & 1 & 5 & 2 \\ 7 & 7 & 1 & 4 \\ 2 & 3 & 2 & 2 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & 0 \\ 4 & 4 & 0 & 0 \\ 0 & 9 & 4 & 0 \\ 8 & 8 & 2 & 4 \end{pmatrix} = \begin{pmatrix} \begin{pmatrix} 5 & 5 \\ 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 \\ 7 & 4 \end{pmatrix} + \begin{pmatrix} 1 & 7 \\ 5 & 2 \end{pmatrix} \cdot \begin{pmatrix} 0 & 9 \\ 8 & 8 \end{pmatrix} & \begin{pmatrix} 1 & 7 \\ 5 & 2 \end{pmatrix} \cdot \begin{pmatrix} 4 & 0 \\ 2 & 4 \end{pmatrix} \\ \begin{pmatrix} 7 & 6 \\ 2 & 3 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 \\ 7 & 4 \end{pmatrix} + \begin{pmatrix} 1 & 4 \\ 2 & 2 \end{pmatrix} \cdot \begin{pmatrix} 0 & 9 \\ 8 & 8 \end{pmatrix} & \begin{pmatrix} 1 & 4 \\ 2 & 2 \end{pmatrix} \cdot \begin{pmatrix} 4 & 0 \\ 2 & 4 \end{pmatrix} \end{pmatrix} =$$

$$\begin{pmatrix} \begin{pmatrix} 96 & 85 \\ 24 & 65 \end{pmatrix} & \begin{pmatrix} 81 & 65 \\ 39 & 46 \end{pmatrix} \\ \begin{pmatrix} 18 & 28 \\ 24 & 8 \end{pmatrix} & \begin{pmatrix} 12 & 16 \\ 12 & 8 \end{pmatrix} \end{pmatrix}$$

Рис. 1 Демонстрация блочного умножения матрицы 4x4 с разбиением на блоки размером 2x2

Алгоритм умножения будет выглядеть следующим образом:

1. Организуем стандартный алгоритм умножения для блоков:

```
for (i = 0; i<S; ++i)
            for (j = 0; j<S; ++j)
                  for (k = j; k<S; ++k)
```

S=N/k,где N – количество строк/столбцов исходной матрицы, sz_b – количество строк/столбцов в блоке.

2. Найдем индекс начала необходимого блока, записанного в векторе.

Так, для вектора, полученного из верхне-треугольной матрицы A путем

построчной записи блоков, получено выражение: `int *a = A + (i * (S + 1) - (i + 1)*i / 2 + (k - i)) * sz_b*sz_b.` Для матрицы B: `int *b = B + (j * (S + 1) - (j + 1)*j / 2 + (k - j)) * sz_b*sz_b.`

3. Организуем стандартный алгоритм умножения для элементов внутри блока.

   Внутри всех блоков расположение элементов построчное, поэтому для перехода на следующую строку требуется умножение индекса строки на количество элементов. Изменение индексов элементов будет происходить одинаково во всех блоках независимо от представления матрицы в векторе.

   В условии не уточняется, каким образом мы будем хранить результирующую матрицу C, поэтому будем считать, что она должна быть записана в виде вектора, блоки которой расположены построчно. Тогда вычисление размера блока будет выглядеть следующим образом: `int *c = C + i * N*sz_b + j * sz_b*sz_b.`

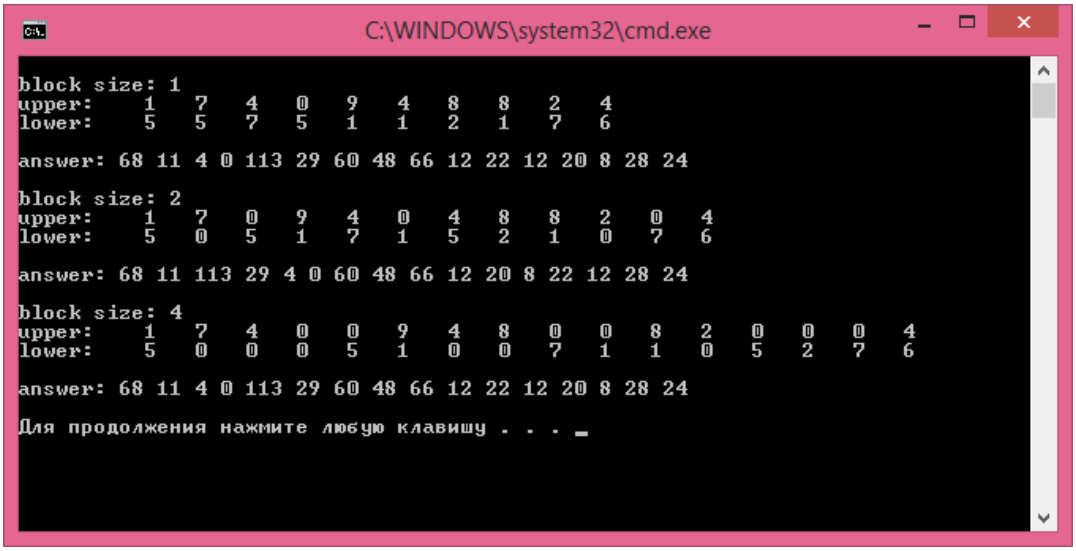
Проверка правильности работы программы

$$A := Matrix(4, 4, [1, 7, 4, 0, 0, 9, 4, 8, 0, 0, 8, 2, 0, 0, 0, 4]); B := Matrix(4, 4, [5, 0, 0, 0, 5, 1, 0, 0, 7, 1, 1, 0, 5, 2, 7, 6]);$$
$$evalm(A\&\cdot B);$$

$$A := \begin{bmatrix} 1 & 7 & 4 & 0 \\ 0 & 9 & 4 & 8 \\ 0 & 0 & 8 & 2 \\ 0 & 0 & 0 & 4 \end{bmatrix}$$

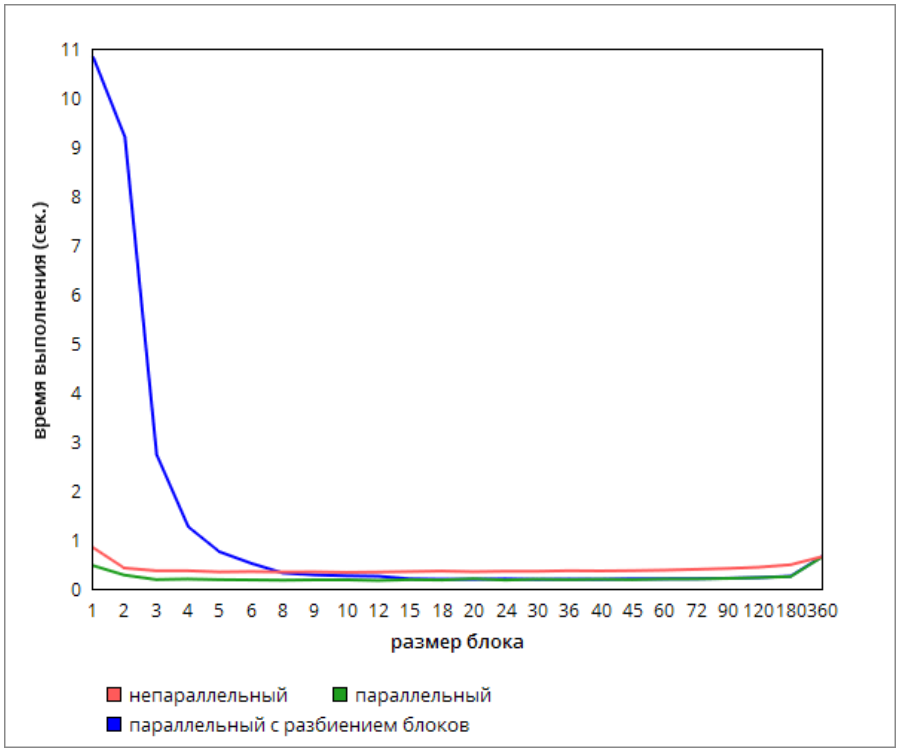
$$B := \begin{bmatrix} 5 & 0 & 0 & 0 \\ 5 & 1 & 0 & 0 \\ 7 & 1 & 1 & 0 \\ 5 & 2 & 7 & 6 \end{bmatrix}$$

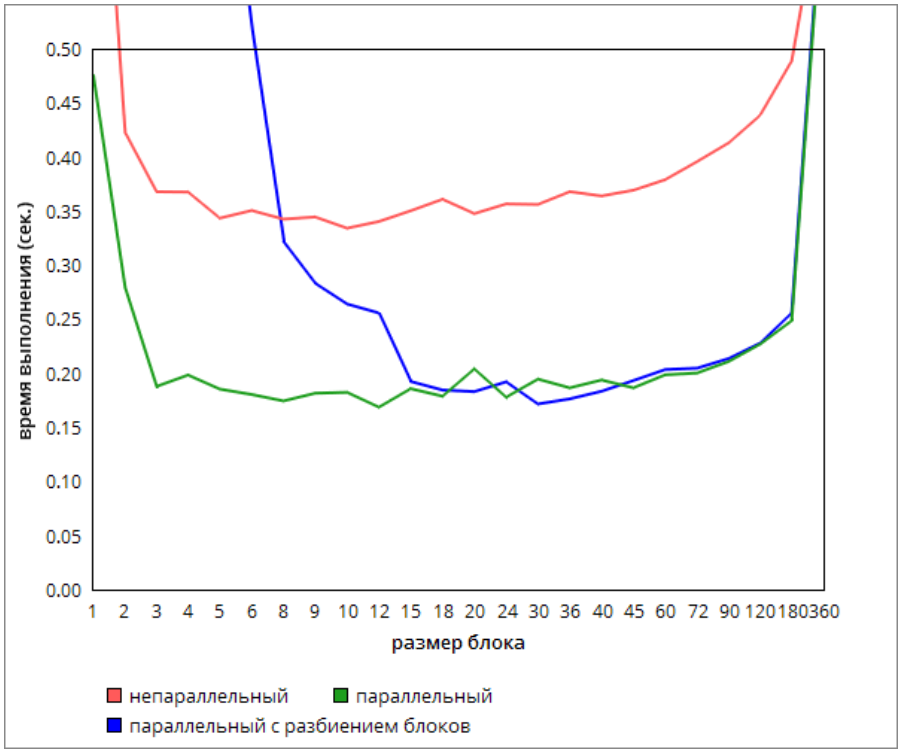
$$\begin{bmatrix} 68 & 11 & 4 & 0 \\ 113 & 29 & 60 & 48 \\ 66 & 12 & 22 & 12 \\ 20 & 8 & 28 & 24 \end{bmatrix}$$



Результаты работы программы

При блочном умножении матриц размером 360x360 получили результаты, которые можно представить в виде графика:





Характеристики компьютера

Процессор Intel Celeron N2840, тактовая частота до 2,58 GHz.

Количество ядер: 2

Количество потоков: 2

Базовая тактовая частота процессора: 2.16 GHz

Частота сигналов: 2.58 GHz

Кэш-память:

L2: 1 MB 16-way

L1 Data: 2x24 KB 6-way

L1 Inst.: 2x32 KB 8-way

Оперативная память:

Тип: DDR3

Объем оперативной памяти: 4GB

Частота процессора: 666.4 MHz

Соотношение системной шины к памяти (FSB:DRAM): 1:8

Выводы

Неоптимальные варианты - разбиения матрицы на блоки размером 1x1 и 360x360. Такой результат обосновывается особенностью работы с кэш-памятью: при наилучшем разбиении, количество кэш-промахов уменьшается. Как следствие, получаем более быструю работу программы.

Приложение 1

Зависимость времени выполнения операции блочного умножения от размера блока

Размер блока	Не параллельный	Параллельный	Параллельный с резервированием памяти для блоков
1	0.846971	0.476048	10.844859
2	0.423209	0.279921	9.214135
3	0.367781	0.187684	2.746657
4	0.367511	0.198319	1.272940
5	0.343350	0.185299	0.754207
6	0.350426	0.180331	0.518665
8	0.342452	0.174369	0.321701
9	0.344398	0.181515	0.283104
10	0.334063	0.182304	0.263995
12	0.340254	0.168604	0.255653
15	0.350170	0.185700	0.192370
18	0.360816	0.178744	0.184448
20	0.347543	0.204000	0.182986
24	0.356491	0.177828	0.192004
30	0.356042	0.194536	0.171557
36	0.367835	0.186478	0.176263
40	0.363873	0.193704	0.183316
45	0.369165	0.186488	0.193217
60	0.378884	0.198531	0.203388
72	0.395597	0.200146	0.204663
90	0.413019	0.210897	0.213589
120	0.438782	0.227146	0.228074
180	0.489373	0.249011	0.256329
360	0.655403	0.650470	0.660069