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**

**Выполнил:**

Студент 4 курса 1 группы

Троян Никита Эдуардович

Ростов-на-Дону

2018

Содержание

Постановка задачи .............................................................................................................................. 3

Алгоритм решения .............................................................................................................................. 3

Результаты работы программы .......................................................................................................... 7

График зависимости времени от размера блока.................................................................................8

Характеристики компьютера ............................................................................................................. 9

Выводы ..................................................................................................................................................9

Приложение 1 .......................................................................................................................................10

**Постановка задачи**

**Задание 24**

Написать программу блочного умножения двух матриц C = A\*B.

Матрица A симметричная, хранится как нижне-треугольная. Хранится в

виде одномерного массива по блочным столбцам.

Матрица B верхне-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**

Подготовительная работа:

Создаем файлы fileА и fileВ, в которые записываем нижнее-треугольную и верхнее-треугольную матрицы:

G:\Super\1.png

Считываем файлы в массивы arr\_a и arr\_b, заметим, что в массиве также хранятся нули:

G:\Super\2.png

Достраиваем к матрице А симметричные элементы. Делаем это из необходимости, чтоб блоки, находящиеся на главной диагонали являлись симметричными матрицами. Иначе при переводе массива в вектор, содержащий только нижнее-треугольную матрицу, блоки на главной диагонали будут содержать нули, там, где должны быть симметричные элементы.

G:\Super\3.png

Основной алгоритм:

Далее, по заданию, матрицы хранятся в виде одномерного массива по блочным столбцам.

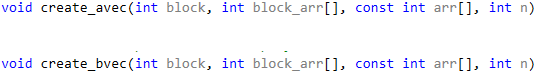
A= VectorA=(,,)

Где - блок матрицы А .

Так как =, нет необходимости хранить блок . И обращаться к нему будем, как к .

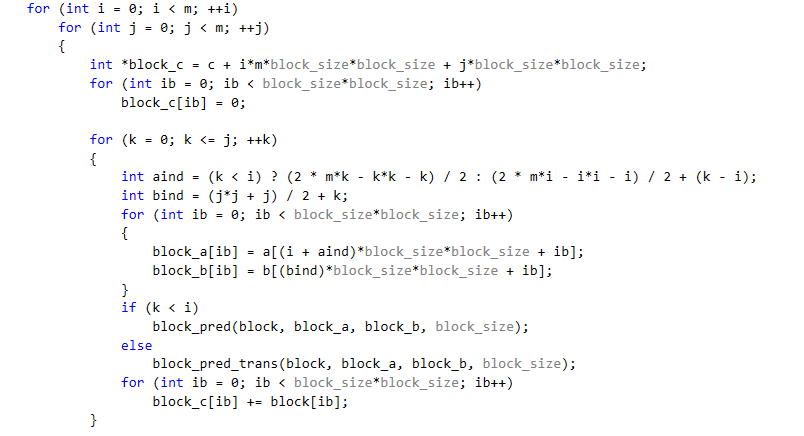
B= VectorA=(,,)

Где - блок матрицы А . Далее нули не будут храниться в векторе В.



Алгоритм умножения будет выглядеть следующим образом:

1. Организуем стандартный алгоритм умножения для блоков:



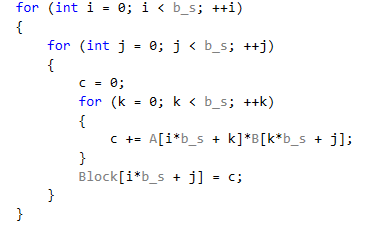
* Где (k<i) - условие, необходимое для обращения к симметричным блока матрицы А, которые не хранятся в строке.
* Индекс начала необходимого блока, записанного в векторе.

Так, для вектора, полученного из симметричной матрицы A (в нижне-треугольном виде) и В (верхнее-треугольная) путем построчной записи блоков, найдены индексы для обращения к любому блоку.

G:\evyj4.png

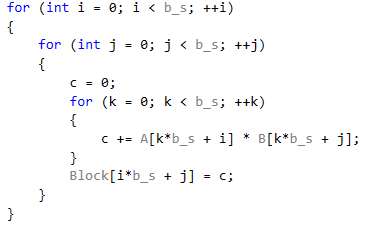
* Стандартный алгоритм умножения для элементов внутри блока.

Функция block\_pred



* Алгоритм умножения для элементов внутри транспонированного блока и обычного (необходимо, когда обращаемся к блоку из матрицы А , который находится выше главной диагонали и не содержится в векторе)

Функция block\_pred\_trans



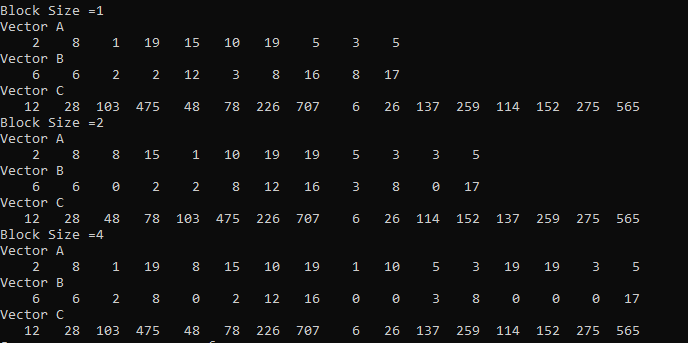
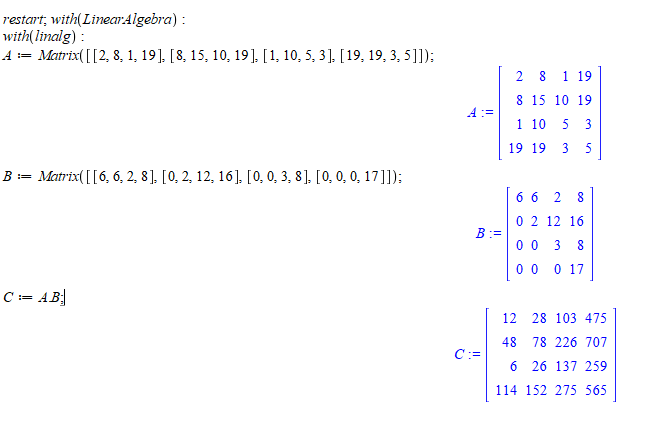
* В условии не уточняется, каким образом мы будем хранить результирующую матрицу C, поэтому будем считать, что она должна быть записана в виде вектора, блоки которой расположены построчно. Тогда вычисление размера блока будет выглядеть следующим образом:

G:\evyj3.png

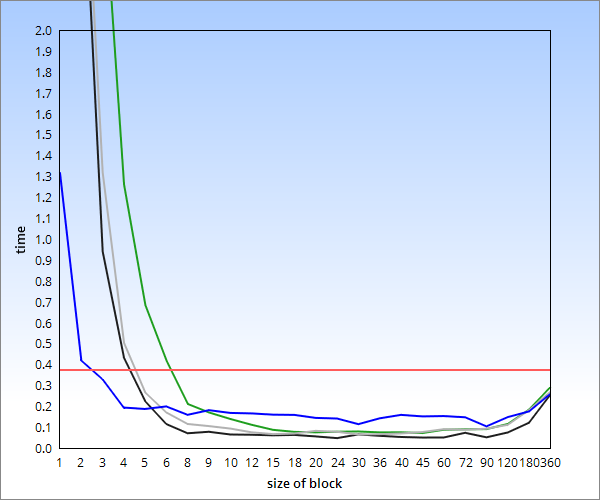
* Для распараллеливания используем 8 потоков

G:\Super\9.png

**Проверка правильности работы программы**



**График зависимости времени от размера блока**

****

- не блочное умножение

- блочное умножение

- блочное умножение, параллельно по блокам

- блочное умножение, параллельно по блокам, с выделением памяти для блоков

- блочное умножение, параллельно по строкам

**Характеристики компьютера**

Процессор Intel i5 5200U, тактовая частота до 2,58 GHz.

Количество ядер: 4

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: 8GB

Частота процессора: 666.4 MHz

**Выводы**

Неоптимальные варианты - разбиения матрицы на блоки размером 1x1 и 360x360. Такой результат обосновывается особенностью работы с кэш-памятью: при оптимальных разбиениях (10x10….60x60), 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**  **По блокам** | **Параллельный с резервированием памяти для блоков** | **Параллельный по строкам** |
| 1 | 1,319057 | 23,108770 | 66,622759 | 23,218720 |
| 2 | 0,420175 | 3,227345 | 8,773317 | 3,027620 |
| 3 | 0,330273 | 1,319397 | 2,764126 | 0,941998 |
| 4 | 0,193653 | 0,505135 | 1,263440 | 0,433566 |
| 5 | 0,186714 | 0,266875 | 0,686151 | 0,224992 |
| 6 | 0,198858 | 0,170325 | 0,418432 | 0,114631 |
| 8 | 0,158758 | 0,114937 | 0,210995 | 0,070503 |
| 9 | 0,180966 | 0,104786 | 0,170097 | 0,077651 |
| 10 | 0,167843 | 0,092998 | 0,138960 | 0,064581 |
| 12 | 0,165469 | 0,075014 | 0,111069 | 0,063438 |
| 15 | 0,159547 | 0,066443 | 0,086958 | 0,060651 |
| 18 | 0,158360 | 0,069215 | 0,077184 | 0,062539 |
| 20 | 0,144019 | 0,082057 | 0,074927 | 0,055424 |
| 24 | 0,141023 | 0,077651 | 0,078030 | 0,047279 |
| 30 | 0,114108 | 0,064662 | 0,079145 | 0,064926 |
| 36 | 0,142043 | 0,066175 | 0,074685 | 0,058284 |
| 40 | 0,158491 | 0,069549 | 0,075118 | 0,052675 |
| 45 | 0,151209 | 0,076341 | 0,071591 | 0,050024 |
| 60 | 0,152815 | 0,089890 | 0,087256 | 0,050350 |
| 72 | 0,146988 | 0,087424 | 0,089332 | 0,072670 |
| 90 | 0,103519 | 0,092288 | 0,090348 | 0,051118 |
| 120 | 0,147908 | 0,110976 | 0,115127 | 0,074352 |
| 180 | 0,175250 | 0,182201 | 0,184090 | 0,121132 |
| 360 | 0,259809 | 0,269623 | 0,292061 | 0,255492 |