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:
студентка 4 курса 1 группы
Тикиджи-Хамбуоьян Артем Рубенович

Ростов-на-Дону
2018

Содержание

Постановка задачи

Задание 23.

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 строкам. 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: inp.in_j * inp.bl_cnt - (inp.in_j - 1)*inp.in_j/2 + inp.in_i - inp.in_j. Для матрицы B: inp.in_i * inp.bl_cnt - (inp.in_i - 1)*inp.in_i/2 + inp.in_j - inp.in_i.
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: (i * inp.bl_cnt + j)*inp.bl_size*inp.bl_size.

Проверка правильности работы программы

Проверка корректности работы блочного перемножения производится непосредственно в программе, производится стандартное перемножение входных матриц (представленных в виде двумерных массивов), и проходит сравнение результатов. После чего печатается результат сравнения (если результаты совпали выводится 1, иначе 0).

Результаты работы программы

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

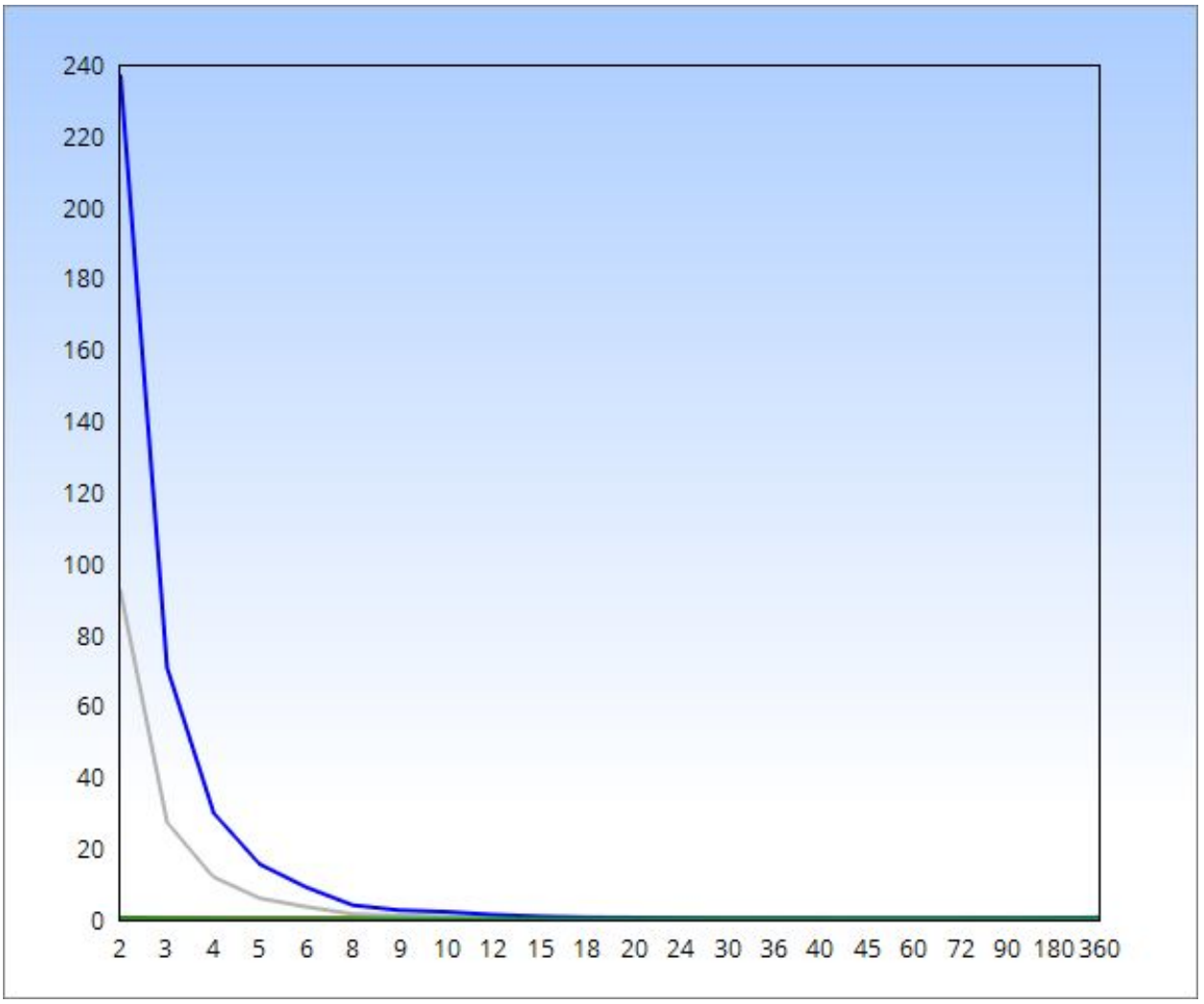

Рис. 2 График времени перемножения матриц (ось X - размер блока, ось Y - время перемножения в секундах)

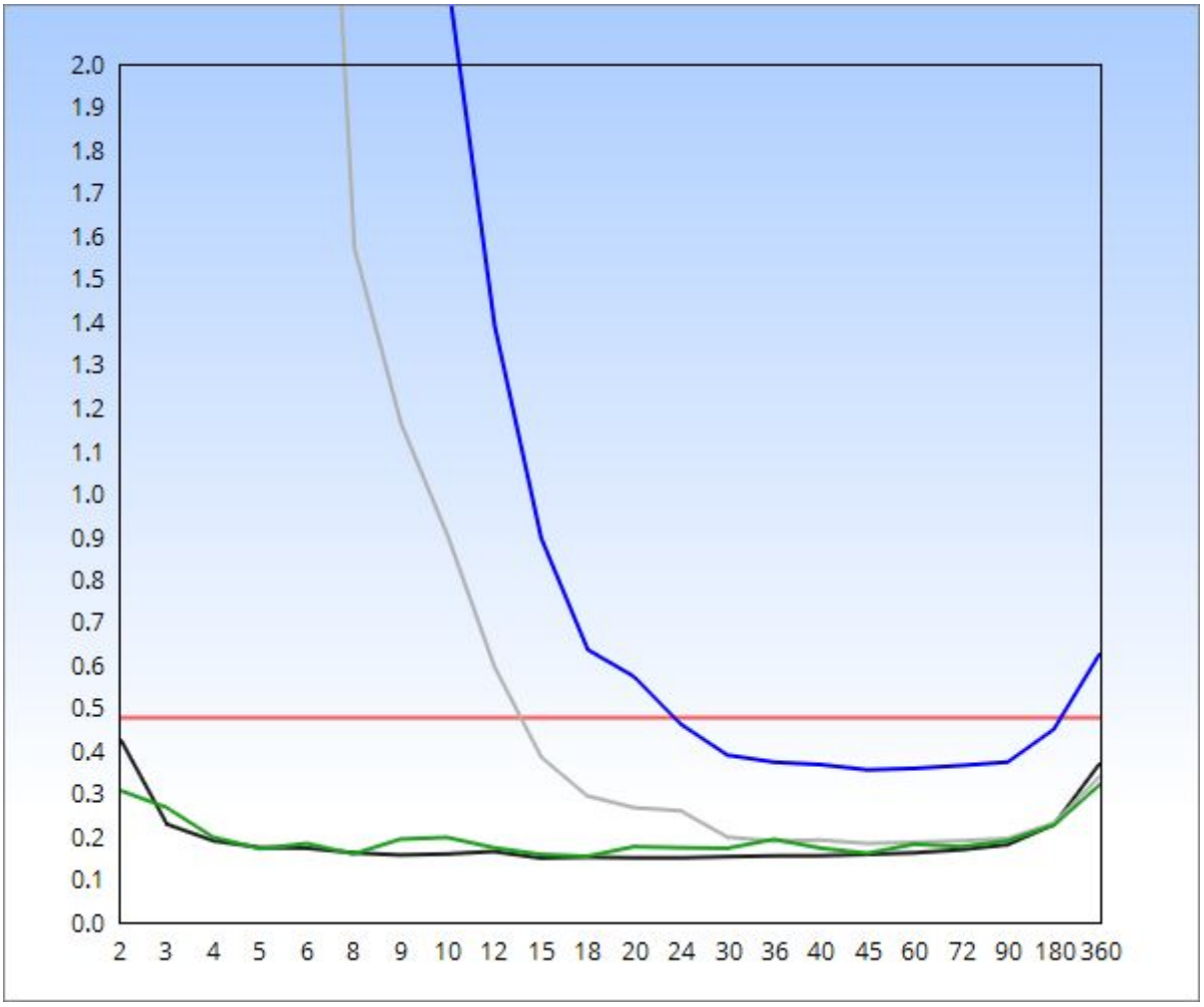

Рис. 3 График времени перемножения матриц (ось X - размер блока, ось Y - время перемножения в секундах)

На рисунках 2 и 3, изображены графики времени перемножения.

- Красный - перемножение матриц стандартным способом
- Зеленый - блочное перемножение матриц без распараллеливания
- Черный - блочное перемножение матриц распараллеленное перемножение блоков на 2 потока
- Серый - блочное перемножение матриц распараллеленное перемножение внутри блока на 2 потока
- Синий - блочное перемножение матриц распараллеленное перемножение внутри блока на 4 потока

Характеристики компьютера

Процессор Intel Intel Core i5, тактовая частота до 2,3 GHz.
Количество процессоров: 1
Количество ядер: 2
Количество потоков в одном ядре: 2
Базовая тактовая частота процессора: 2.16 GHz
Кэш-память:
L2 (per Core): 256 KB
L3 : 3 MB
Оперативная память:
Тип: DDR3
Объем оперативной памяти: 16GB
Частота процессора: 1600 MHz

Выводы

Неоптимальные варианты - разбиения матрицы на блоки размером 2x2 и 360x360. Такой результат обосновывается особенностью работы с кэш-памятью: при наилучшем разбиении, количество кэш-промахов уменьшается. Как следствие, получаем более быструю работу программы.
Неоптимальность некоторых методов распараллеливания обусловлены:

1. Особенностями ОС. (при большой нагрузке на процессор, ОС запускает свой для вытеснения нагружающих процессов)
2. Распараллеливание вычислений внутри блоков не имеет смысла при малых размерах блока.
3. По невыясненной причине на выполнение программы предоставлялось лишь одно ядро (данный вывод был сделан на основании мониторинга загруженности процессора, к сожалению сринов не сохранилось), как следствие - производительность при использовании 2x потоков было выше чем при использовании 4x потоков.