ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ бюджетное ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

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

**ОТЧЕТ**

на тему:

### **Блочные вычисления. Модели времени выполнения программ. Блочные размещения массивов, дополняющие блочные вычисления**

**Выполнил:**

студент 5 курса 5 группы

Руднев Данил Олегович

Ростов-на-Дону

2019

**Содержание**

Оглавление

**1 Постановка задачи3**

**2 Решение3**

2.1 Блочное умножение матриц3

2.2 Результаты экспериментов4

2.3 Характеристики компьютера5

**3 Выводы6**

**1 Постановка задачи**

Задание 42.

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

Матрица A симметричная, хранится как нижне-треугольная. Хранится в виде одномерного массива по блочным строкам.

Матрица B нижне-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.

**2 Решение**

**2.1 Блочное умножение матриц**

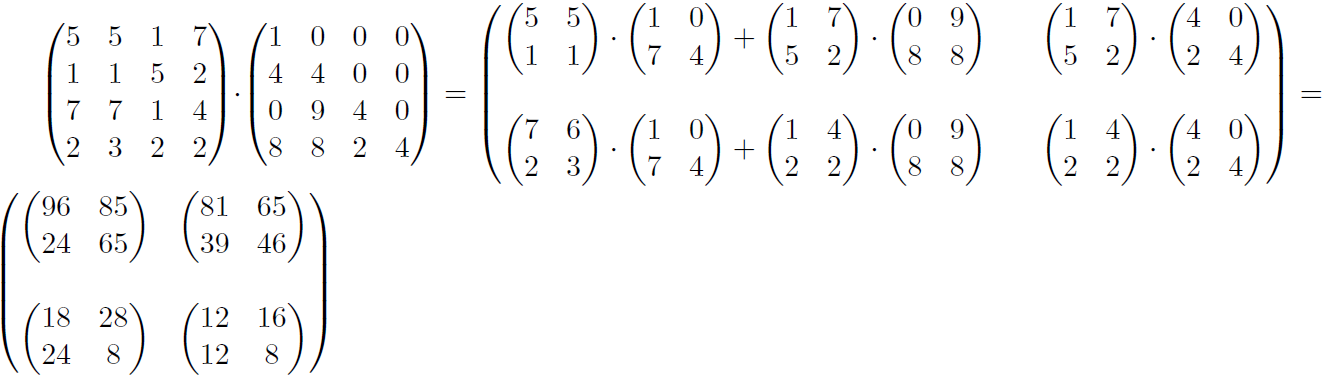


Рис. 1 Демонстрация блочного умножения матрицы 4x4 с разбиением на блоки размером 2x2

Программная реализация алгоритма умножения:

for (int k = 0; k < n; k += base)

for (int z = 0; z < n; z += base)

for (int p = z; p < n; p += base)

{

int k\_b = k + base;

int z\_b = z + base;

int p\_b = p + base;

for (int kk = k; kk < k\_b; kk++)

for (int zz = z; zz < z\_b; zz++)

for (int pp = p; pp < p\_b; pp++)

resThreaded[kk \* n + zz] += A[offsets\_A[pp \* n + kk]] \* B[offsets\_B[pp \* n + zz]];

}

Первые три цикла перемножают блоки, а последние три производят умножение внутри блоков.

В условии не уточняется, каким образом необходимо хранить результирующую матрицу C, поэтому считаем, что она должна быть записана построчно.

Распараллеливание данного блока программы организовано, как и указано в задании, двумя способами:

1. Перемножение разных пар блоков. Достигается путем указания директивы #pragma omp for перед первым циклом. Так же можно указывать и перед 2 или 3 циклами, однако, по результатам экспериментов, лучше распараллеливать именно первый цикл.
2. Параллельное вычисление произведения конкретной пары блоков. Достигается аналогичным способом, указывая директиву перед 4 циклом.

Корректность работы можно проверять сверкой сумм результирующих матриц, а так же функцией проверки совпадения элементов матриц checkArrays(). Очевидно, что матрицы должны получаться одинаковые при использовании блочного метода, а так же при использовании распараллеленного блочного метода первого и второго вариантов.

**2.2 Результаты экспериментов**

При блочном умножении матриц размером 2080х2080 были получены результаты, которые можно представить в виде графика:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| base | 2 | 4 | 6 | 8 | 12 | 16 | 20 | 24 | 36 | 48 | 72 |
| usual | 330 | 331 | 304 | 296 | 288 | 288 | 284 | 294 | 289 | 286 | 298 |
| F1 th1 | 167 | 92 | 74 | 70 | 65 | 64 | 64 | 67 | 72 | 74 | 79 |
| F1 th2 | 125 | 81 | 56 | 53 | 50 | 50 | 49 | 55 | 59 | 59 | 67 |
| F1 th4 | 125 | 95 | 61 | 58 | 57 | 57 | 59 | 69 | 72 | 79 | 82 |
| F1 th8 | 131 | 87 | 66 | 64 | 61 | 61 | 65 | 80 | 69 | 87 | 84 |
| F1 th16 | 132 | 81 | 70 | 66 | 65 | 64 | 76 | 78 | 71 | 87 | 85 |
| F2 th1 |  |  |  | 76 | 70 | 63 | 67 | 67 | 53 | 78 | 81 |
| F2 th2 |  |  |  | 94 | 59 | 50 | 51 | 48 | 32 | 70 | 64 |
| F2 th4 |  |  |  | 150 | 92 | 81 | 79 | 79 | 36 | 102 | 95 |
| F2 th8 |  |  |  | 334 | 136 | 90 | 94 | 74 | 49 | 89 | 89 |
| F2 th16 |  |  |  | 581 | 426 | 213 | 235 | 242 | 143 | 172 | 180 |

Таким образом, использование функции второго типа лучше всего работает при размере блока в 24 и показывает лучшее время выполнения, давая ускорение в 6 раз, однако, при слишком маленьком размере блока и/или большом количестве потоков происходит замедление. Это объясняется увеличением накладных расходов.

Первый вариант функции дает ускорение в 5 раз при размерах блока, больших 6 практически независимо от количества потоков.

**2.3 Характеристики компьютера**

Процессор Intel Celeron i7-3770, тактовая частота до 3.40 GHz.

Количество ядер: 4

Объем оперативной памяти: 16GB

L1 кэш данных 32кБ х 4

L1 кэш кода 32кБ х 4

L2 256кБ х 4

L3 8192кБ х 4

**3 Выводы**

Разбиение матрицы на блоки, а так же использование многопоточного вычисления существенно экономит время. Такой результат обосновывается особенностью работы с кэш-памятью: при наилучшем разбиении, количество кэш-промахов уменьшается. Как следствие, получаем более быструю работу программы.

**Дополнительная статистика приведена в файле results.txt**