Национальный исследовательский университет
"Высшая школа экономики"
Факультет компьютерных наук
Программная инженерия

Домашнее задание 4

Вариант 1

Программа, вычисляющая векторное произведение двух квадратных матриц (OpenMP)

Агроскин Александр Викторович

БПИ 199

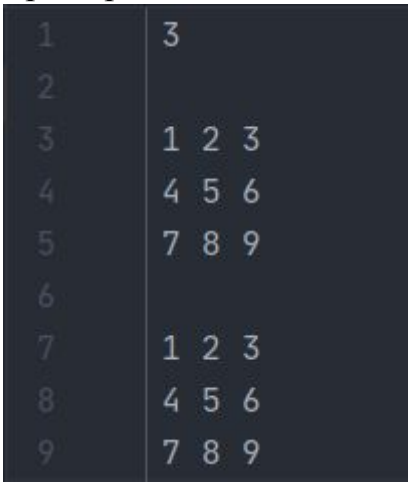
Постановка задания

Вычислить векторное произведение квадратных матриц A и B.
Входные данные: произвольные квадратные матрицы A и B одинаковой размерности. Размер матриц задается входным параметром. Количество потоков является входным параметром, при этом размерность матриц может быть не кратна количеству потоков.

Ограничение на входные и выходные данные

Входные данные должны быть заданы в качестве текстового файла. Первая строка файла содержит положительный размер n выходных матриц. На следующих 2n строках файла расположены матрицы для перемножения, каждая строка содержит n чисел, разделенных пробелами.

Пример входных данных:



Программа через консоль запрашивает у пользователя количество потоков, которое она может использовать. После получения результата программа выводит время, затраченное на вычисления, в консоль и выводит матрицу результата в файл вывода.

Файлы ввода и вывода задаются как аргументы командной строки, где первый аргумент - путь от исполняемого файла к файлу ввода, второй - к файлу вывода.

Пример аргументов командной строки

Алгоритм решения

Программа использует модель многопоточности ***итеративный параллелизм***. При запуске программа требует у пользователя количество доступных ей потоков. С помощью функций `omp_set_dynamic(0)` и `omp_set_num_threads(thread_num)` программа выключает динамическое распределение потоков (для того, чтобы гарантировалось использование всех заданных) и задает их количество (**thread_num** было введено пользователем до этого).

После этого с помощью директивы **#pragma omp parallel for** программа параллельно обрабатывает каждую строку матрицы-результата, вызывая функцию CalculateRow. Все переменные, которые используются внутри параллельного цикла, объявлены как **shared**, для предоставления каждому потоку доступа к ним.

Функция CalculateRow:

```
void CalculateRow(const matrix& matrix_a, const matrix& matrix_b, std::vector<int>* row,
                  int row_num, int dim) {
    for (int i = 0; i < dim; ++i) {
        int sum = 0;
        for (int j = 0; j < dim; ++j) {
            sum += matrix_a[row_num][j] * matrix_b[j][i];
        }
        row->push_back(sum);
    }
}
```

Параллельный фрагмент кода:

```
    auto begin = std::chrono::steady_clock::now();
#pragma omp parallel for shared(dim) shared(matrix_a) shared(matrix_b) \
    shared(matrix_res) default(none)
    for (int i = 0; i < dim; ++i) {
        CalculateRow(matrix_a, matrix_b, &(matrix_res[i]), i, dim);
    }
    auto end = std::chrono::steady_clock::now();
```

Тестирование программы

В репозитории лежат три примера входных данных: файлы input_3.txt, input_20.txt и input_1000.txt. Файлы содержат матрицы размерности 3, 20 и 1000 соответственно.

Проверим, что программа работает на простых входных данных.

Содержание input_3.txt:

```
3

1 2 3
4 5 6
7 8 9

1 2 3
4 5 6
7 8 9
```

При запуске вводим количество потоков и получаем время вычислений:

```
/home/avagr/CLionProjects/CrossProductOpenMP/cmake-build-debug/CrossProductOpenMP ../input_3.txt ../output.txt
Please enter the desired number of threads: 2
Done in 49 microseconds
Process finished with exit code 0
```

В выходном файле находится матрица – результат умножения:

```
30      36      42
66      81      96
102     126     150
```

Результат верный.

Тестировать производительность уместней всего на больших матрицах. Ниже приведено время работы программы в микросекундах для разного входного количества потоков. Тестирование проводилось на ноутбуке с процессором intel i5-10210U (4 cores 8 threads). Тестировалось произведение двух матриц 1000 x 1000, для сравнения приведены результаты из предыдущего отчета с использованием **std::thread**.

Количество потоков	**Время в микросекундах (std::thread)**	**Время в микросекундах (OpenMP)**
1	12835682	11991865
2	10569878	12334292
3	5356627	5234825
4	4355367	3400940
5	2875024	2677403
6	3078790	3318183
7	2679534	2600438
8	2707397	2667179
12	2730003	2650863
16	2868519	2546468

Приложение 1

Список используемых источников

1. https://en.wikipedia.org/wiki/Loop-level_parallelism
2. https://www.cs.umd.edu/users/meesh/cmsc411/website/projects/unroll/main.htm
3. https://bisqwit.iki.fi/story/howto/openmp/

Приложение 2

Код программы

```
#include <iostream>
#include <fstream>
#include <vector>
#include <chrono>
#include <iomanip>
#include <omp.h>

typedef std::vector<std::vector<int>> matrix;

void ReadMatrix(std::ifstream* input, matrix* matrix_b, int dim) {
    for (int i = 0; i < dim; ++i) {
        (*matrix_b)[i].resize(dim);
        for (int j = 0; j < dim; ++j) {
            (*input) >> (*matrix_b)[i][j];
        }
    }
}

void CalculateRow(const matrix& matrix_a, const matrix& matrix_b,
std::vector<int>* row,
                  int row_num, int dim) {
    for (int i = 0; i < dim; ++i) {
        int sum = 0;
        for (int j = 0; j < dim; ++j) {
            sum += matrix_a[row_num][j] * matrix_b[j][i];
        }
        row->push_back(sum);
    }
}

int main(int argc, char* argv[]) {
    if (argc != 3) {
        std::cout << "Please provide the following arguments <input_path>
<output_path>\n";
        return 0;
    }
    std::ifstream input(argv[1]);
    int dim = 0;
    input >> dim;
    matrix matrix_a(dim);
    matrix matrix_b(dim);
    matrix matrix_res(dim);
    ReadMatrix(&input, &matrix_a, dim);
    ReadMatrix(&input, &matrix_b, dim);
    input.close();
    int thread_num;
    std::cout << "Please enter the desired number of threads: ";
    std::cin >> thread_num;
    thread_num--;
    omp_set_dynamic(0);
    omp_set_num_threads(thread_num);
    auto begin = std::chrono::steady_clock::now();
#pragma omp parallel for shared(dim) shared(matrix_a) shared(matrix_b) \
    shared(matrix_res) default(none)
    for (int i = 0; i < dim; ++i) {
        CalculateRow(matrix_a, matrix_b, &(matrix_res[i]), i, dim);
    }
    auto end = std::chrono::steady_clock::now();
```

```
    std::ofstream output(argv[2]);
    for (int i = 0; i < dim; ++i) {
        for (int j = 0; j < dim; ++j) {
            output << std::left << std::setw(6) << matrix_res[i][j] << " ";
        }
        output << "\n";
    }
    output.close();
    std::cout << "Done in "
              << std::chrono::duration_cast<std::chrono::microseconds>(end -
begin).count()
              << " microseconds";
    return 0;
}
```