САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Архитектура ЭВМ

Отчет

по домашней работе №5

«OpenMP»

Выполнил(а): Миленин Иван Александрович

студ. гр. М3135

Санкт-Петербург

2020

Цель работы: знакомство со стандартом распараллеливания команд метода OpenMP.

Инструментарий и требования к работе: рекомендуется использовать C, C++. Возможно использовать Python и Java.

Теоретическая часть

OpenMP – это библиотека для параллельного программирования вычислительных систем для таких языков, как C, C++ и Fortran. Самая первая версия данной библиотеки была выпущена в 1997 году для языка Fortran. Далее в 1998 году вышли версии для C и C++.

Любая программа состоит из наборов областей двух типов:

1) Последовательные области, где при выполнении программы создаётся только один поток, являющийся единственным на протяжении выполнения всей программы.
2) Области распараллеливания, в которых порождается ряд параллельных потоков (см. Рисунок №1). Данные потоки могут выполняться как на одном, так и на нескольких процессорах, за что отвечают алгоритмы операционных систем.

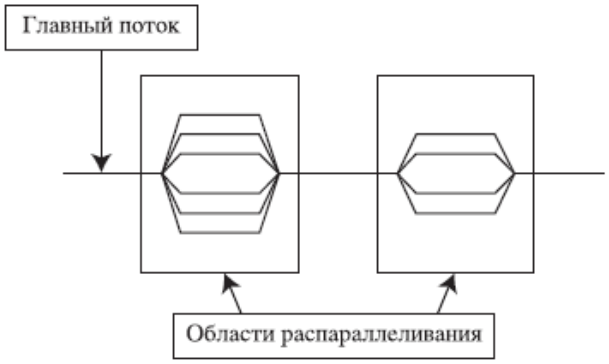


Рисунок №1 – распределение главного потока на области распараллеливания

Теперь необходимо перечислить ключевые элементы OpenMP:

1) Конструкции для создания потоков (директива parallel). Она создает параллельный регион для следующего за ней структурированного блока. Каждый из созданных потоков выполнит одинаковый код, содержащийся в блоке, но не одинаковый набор команд. В разных потоках могут выполняться различные ветви или обрабатываться различные данные.
2) Конструкции распределения работы между потоками (директивы DO/for и section). Данная директива занимается ускорением циклов. Если мы применим директиву parallel на цикл, то мы не ускорим программу, а сделаем один цикл на каждом потоке. Однако при использовании директивы for при выполнении цикла в параллельном регионе итерации цикла должны быть распределены между потоками группы.
3) Конструкции для управления работой с данными (выражения shared и private для определения класса памяти переменных). По умолчанию выражения считаются shared. Частные данные принадлежат потоку и могут быть модифицированы только им. Общие данные доступны всем потокам.
4) Конструкции для синхронизации потоков (директивы critical, atomic и barrier). Вычисления в последовательном блоке, как правило, могут быть продолжены, если завершены все процессы в параллельном структурном блоке и их результаты корректно переданы в последовательный блок. Именно для обеспечения такой корректной передачи данных и необходима процедура синхронизации параллельных потоков.

Директива crtical отвечает за условное отключение работы параллельности на выделенном участие, что делает параллельный код однопоточным. Директива atomic отвечает за корректность работы присваивания оператора и предотвращение прерывания доступа, чтения и записи данных со стороны других потоков. Директива barrier устанавливает режим ожидания окончания выполнения действий каждого потока.

Также стоит отметить 4 переменные, благодаря которым возможно гибкое управление режимами многопоточной программы:

1) OMP_DYNAMIC - разрешает или запрещает динамическое изменение числа нитей.
2) OMP_NUM_THREADS - устанавливает число потоков при исполнении параллельных областей приложения.
3) OMP_NESTED - разрешает или запрещает вложенный параллелизм.
4) OMP_SCHEDULE - определяет способ распределения итераций в цикле.

Практическая часть

Был выбран вариант №7.

Для нахождения определителя матрицы был использован метод Гаусса. Для его использования необходимо привести матрицу к треугольному виду и найти произведение диагонали матрицы. Далее будут рассмотрены обычный и параллельный алгоритм триангуляции матрицы.

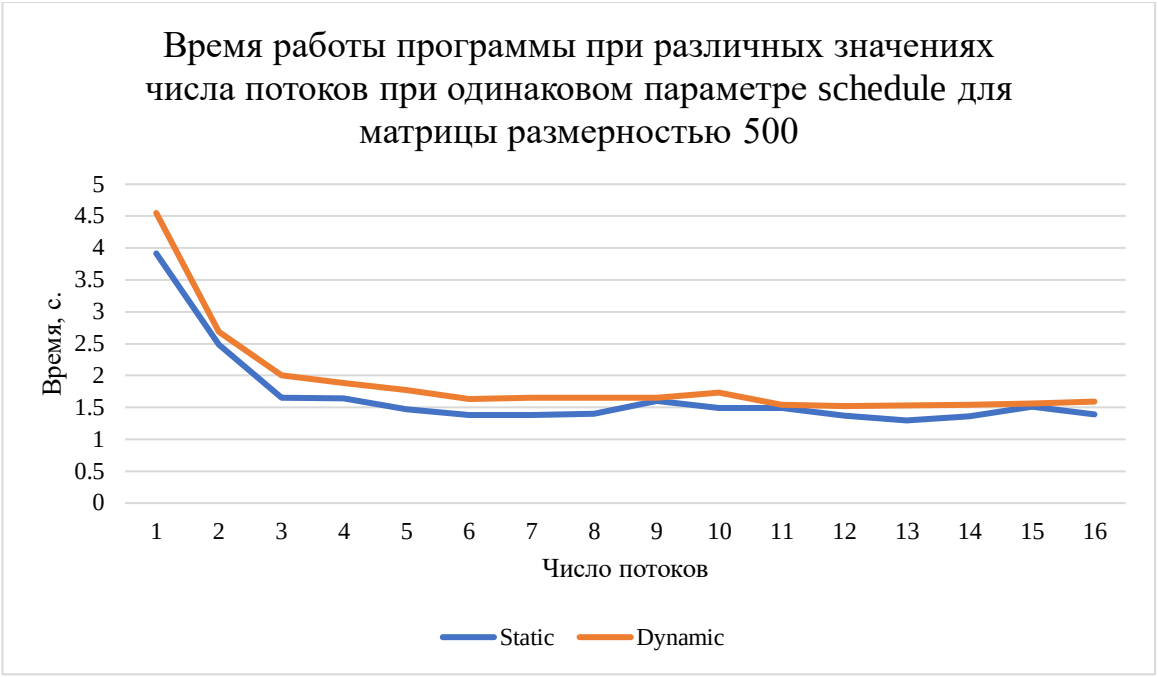
Для получения матрицы треугольного вида без использования распараллеливания необходимо обратить i-тый столбец в 0, для чего необходим поиск строки с максимальным элементом и её перестановки с i-

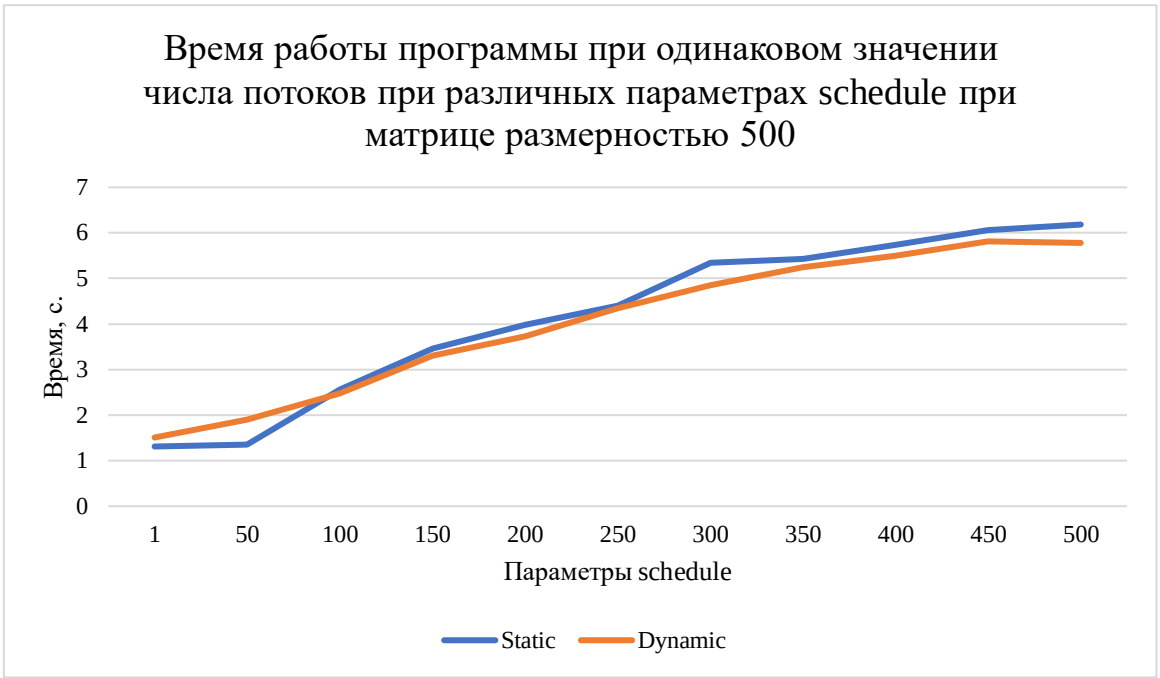
той строки. Далее произвести обращение в 0. Также стоит отметить то, что ведется счетчик количества перестановок, так как при нечетном числе перестановок определитель будет отрицательным.

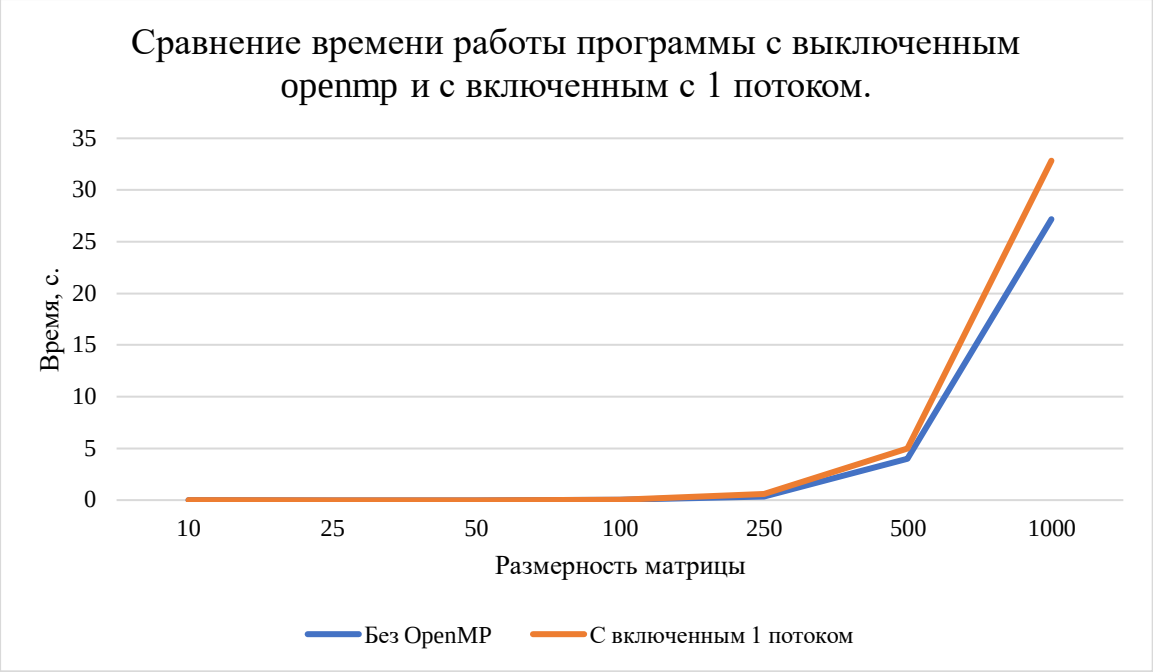
Теперь рассмотрим распараллеленный алгоритм триангуляции. Для его ускорения необходимо было ускорить обнуление с помощью эквивалентных преобразований элементов i-того столбца матрицы и добавление к j-той строке матрицы элементов i-той строки, домноженных на константу. Также понятно, что поиск максимального элемента в столбце не принесет сильного ускорения, поэтому было принято решение сделать данный поиск однопоточным.

Также в программе присутствует обработка ошибки ввода неверного формата файла. Ниже представлены графики, на которых представлено:

1) Время работы программы при различных значениях числа потоков при одинаковом параметре schedule для матрицы размерностью 500
2) Время работы программы при одинаковом значении числа потоков при различных параметрах schedule при матрице размерностью 500
3) Сравнение времени работы программы с выключенным OpenMP и с включенным OpenMP с 1 потоком.


Время работы программы при различных значениях числа потоков при одинаковом параметре schedule для матрицы размерностью 500
Время, с.
5
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
Число потоков
Static
Dynamic


Время работы программы при одинаковом значении числа потоков при различных параметрах schedule при матрице размерностью 500
Время, с.
7
6
5
4
3
2
1
0
1 50 100 150 200 250 300 350 400 450 500
Параметры schedule
Static
Dynamic



Листинг

< DeterminationFinding.cpp>

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

using namespace std;

float parallel_gauss_determinant(vector<vector<float>>& mat, int size);
int paralel_col_max(const vector<vector<float>>& mat, int numOfCol, int size);
int paralel_triangulation(vector<vector<float>>& mat, int size);

float parallel_gauss_determinant(vector<vector<float>>& mat, int size) {
    unsigned int swapCount = paralel_triangulation(mat, size);
    float det = 1;
    if (swapCount % 2 == 1) det = -1;
#pragma omp parallel for schedule(static)
    for (int i = 0; i < size; i++) {
        det *= mat[i][i];
    }
    return det;
}

int paralel_col_max(const vector<vector<float>>& mat, int numOfCol, int size) {
    float max = abs(mat[numOfCol][numOfCol]);
    int maxPos = numOfCol;
#pragma omp parallel for schedule(static)
    for (int i = numOfCol + 1; i < size; i++) {
        float element = abs(mat[i][numOfCol]);
        if (element > max) {
            max = element;
            maxPos = i;
        }
    }
```

```
    return maxPos;
}

int paralel_triangulation(vector<vector<float>>& mat, int size) {
    unsigned int swapCount = 0;
    for (int i = 0; i < size - 1; i++) {
        unsigned int imax = paralel_col_max(mat, i, size);
        if (i != imax) {
            swap(mat[i], mat[imax]);
            swapCount++;
        }
#pragma omp parallel for schedule(static)
        for (int j = i + 1; j < size; j++) {
            float mul = -mat[j][i] / mat[i][i];
#pragma omp parallel for schedule(static)
            for (int k = i; k < size; k++) {
                mat[j][k] += mat[i][k] * mul;
            }
        }
    }
    return swapCount;
}

int main(int argc, char** fileName) {
    int treads = atoi(fileName[1]);
    if (treads != 0) {
        omp_set_num_threads(treads);
    }
    ifstream in(fileName[2]);
    string temp = fileName[2];
    int lenght = temp.length();
    try {
        if (!in) {
            throw 0;
        }
        else if (lenght < 4 || fileName[2][lenght - 4] != '.' ||
            fileName[2][lenght - 1] != 't' || fileName[2][lenght - 2] != 'x' ||
            fileName[2][lenght - 3] != 't') {
            throw 0;
        }
        int n;
        in >> n;
        vector<vector<float>> matrix(n);
        for (int i = 0; i < n; i++) {
            matrix[i].resize(n);
            for (int j = 0; j < n; j++) {
                //matrix[i][j] = rand();
                in >> matrix[i][j];
            }
        }
        in.close();
        double start;
        double end;
        start = omp_get_wtime();
        printf("Determinant: %g\n", parallel_gauss_determinant(matrix, n));
        end = omp_get_wtime();
        matrix.clear();
        printf("\nTime (%i thread(s)): %f ms\n", treads, end - start);
    }
```

```
catch (int i) {
    printf("Unknown or missing extension. Please use the .txt format");
    in.close();
}
return 0;
}
```