Лабораторна робота #5

Виконав: Кузьменко Владислав(КН-22)

Тема: Технологія OpenMP
Мета: Познайомитись з базовими директивами OpenMP та навчитись їх застосовувати для розпаралелювання послідовних програм

Теоретичні відомості.

OpenMP дозволяє легко і швидко створювати багатопоточні програми на алгоритмічних мовах Fortran і C/C++. При цьому директиви OpenMP аналогічні директивам препроцесора для мови C/C++ і є аналогом коментарів у алгоритмічній мові Fortran. Це дозволяє в будь-який момент розробки паралельної реалізації програмного продукту при необхідності повернутися до послідовного варіанту програми. OpenMP – це стандартна модель для паралельного програмування в середовищі зі спільною пам’яттю. У даній моделі усі процеси спільно використовують загальний адресний простір, до якого вони асинхронно звертаються із запитами на читання і запис. У таких моделях для управління доступом до загальної пам’яті використовуються різні механізми синхронізації типу семафорів та блокування процесів. Перевага цієї моделі з точки зору програмування полягає в тому, що поняття монопольного володіння даними відсутнє, отже, не потрібно явно задавати обмін даними між потоками, що їх задають, та потоками, що їх використовують. Ця модель, з одного боку, спрощує розробку програми, але, з іншого боку, ускладнює розуміння і управління локальністю даних, написання детермінованих програм.

Git

C++: https://github.com/x1nett/parprog_lab5cpp.git

Лістинг коду

ConsoleApplication1 (Global Scope)

```cpp
#include <iostream>
#include <omp.h>
#include <climits>

using namespace std;

const int ROWS = 10000;
const int COLS = 1000;

int matrix[ROWS][COLS];

void init_matrix() {
    for (int i = 0; i < ROWS; ++i)
        for (int j = 0; j < COLS; ++j)
            matrix[i][j] = 1 + (i + j) % 100;

    for (int j = 0; j < COLS; ++j)
        matrix[ROWS / 2][j] = -100;
}

long long total_sum(int num_threads) {
    long long sum = 0;
    double t1 = omp_get_wtime();

#pragma omp parallel for reduction(+:sum) num_threads(num_threads)
    for (int i = 0; i < ROWS; ++i)
        for (int j = 0; j < COLS; ++j)
            sum += matrix[i][j];

    double t2 = omp_get_wtime();
    cout << "Total sum computed in " << t2 - t1 << " seconds with " << num_threads << " threads.\n";

    return sum;
}
```

```cpp
void row_with_min_sum(int num_threads, int& min_row_index, long long& min_row_sum) {
    min_row_sum = LLONG_MAX;
    min_row_index = -1;

    double t1 = omp_get_wtime();

#pragma omp parallel num_threads(num_threads)
    {
        int local_min_index = -1;
        long long local_min_sum = LLONG_MAX;

#pragma omp for
        for (int i = 0; i < ROWS; ++i) {
            long long row_sum = 0;
            for (int j = 0; j < COLS; ++j)
                row_sum += matrix[i][j];

            if (row_sum < local_min_sum) {
                local_min_sum = row_sum;
                local_min_index = i;
            }
        }

#pragma omp critical
        {
            if (local_min_sum < min_row_sum) {
                min_row_sum = local_min_sum;
                min_row_index = local_min_index;
            }
        }
    }

    double t2 = omp_get_wtime();
    cout << "Row with min sum computed in " << t2 - t1 << " seconds with " << num_threads << " threads.\n";
}
```

```cpp
int main() {
    init_matrix();
    omp_set_nested(1);

    int num_threads = 8;
    long long total;
    int min_index;
    long long min_value;

    double t_start = omp_get_wtime();

#pragma omp parallel sections
    {
#pragma omp section
        {
            total = total_sum(num_threads);
        }

#pragma omp section
        {
            row_with_min_sum(num_threads, min_index, min_value);
        }
    }

    double t_end = omp_get_wtime();

    cout << "=== Results ===\n";
    cout << "Total matrix sum = " << total << endl;
    cout << "Min row index = " << min_index << ", sum = " << min_value << endl;
    cout << "Overall execution time: " << t_end - t_start << " seconds\n";

    return 0;
}
```

результат виконання програми: