МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ

Ордена Трудового Красного Знамени федеральное государственное бюджетное образовательное учреждение высшего образования

**«Московский технический университет связи и информатики»**

Кафедра «Информационные технологии»

Лабораторная работа №8

**«Разработка параллельных программ с применением технологии OpenMP»**

по дисциплине

Информационные технологии и программирование

Выполнил: студент гр. БЭИ2202

Кулешов А. С.

Вариант 16

Проверил: Халабия Р.Ф.

Москва, 2024 г

1. Цели и задачи

получить навыки распараллеливания существующих последовательных программ на языке C++. Оценить производительность параллельной программы при использовании разного количества ядер процессора.

1. Постановка задачи

Провести эксперимент по анализу ускорения вычислений на многоядерном процессоре. Работу провести в следующей последовательности.

1. Составить и отладить программу, решающую задачу, на языке С++ в обычном, последовательном режиме вычислений. Доказать, что программа выполняется верно.
2. Продумать декомпозицию задачи (функциональную или по данным).
3. Продумать алгоритм распараллеливания и коммуникации.
4. Найти функции библиотеки параллельного программирования, наиболее подходящие к выбранному алгоритму распараллеливания задачи.
5. Отладить программу, работающую в параллельном режиме.
6. Добавить в программы функции оценки времени выполнения из библиотеки OpenMP.
7. Увеличить объем вычислений в задачах (количество сигналов, размер массивов, временную сложность вычисления функций). Функцию можно задать как сумму других функций, например разложить синус рядом Тейлора, где n = 1000.
8. Провести несколько экспериментов по оценке времени выполнения задачи с различным количеством потоков.
9. Проанализировать и оценить результаты численных экспериментов.
10. Сделать выводы о влиянии на результат различных факторов (архитектуры вычислительной системы, эффективности программы, выбранных алгоритмов, сложности вычислений, времени коммуникаций).

Вариант 16. Состовить программу методом паралельного программирования вычисления матрицы C как функцию от заданных матриц A и B, заполненнных случайными числами: C = (B-A)+A^2.

1. Листинг программы

#include <iostream>

#include <vector>

#include <random>

#include <omp.h>

**using** **namespace** std;

**void** **initialize\_matrix**(std::vector<std::vector<**double**>>& matrix, **int** n, **int** m) {

std::random\_device rd;

std::mt19937 gen(rd());

std::uniform\_real\_distribution<> dis(**0.0**, **1.0**);

**for** (**int** i = **0**; i < n; ++i) {

**for** (**int** j = **0**; j < m; ++j) {

matrix[i][j] = dis(gen);

}

}

}

**void** **print\_matrix**(**const** std::vector<std::vector<**double**>>& matrix) {

**for** (**const** **auto**& row : matrix) {

**for** (**const** **auto**& elem : row) {

std::cout << elem << " ";

}

std::cout << "**\n**";

}

}

std::vector<std::vector<**double**>> multiply\_matrices(**const** std::vector<std::vector<**double**>>& A, **const** std::vector<std::vector<**double**>>& B) {

**int** n = A.size();

**int** m = A[**0**].size();

**int** p = B[**0**].size();

std::vector<std::vector<**double**>> C(n, std::vector<**double**>(p, **0.0**));

#pragma omp parallel **for**

**for** (**int** i = **0**; i < n; ++i) {

**for** (**int** j = **0**; j < p; ++j) {

**for** (**int** k = **0**; k < m; ++k) {

C[i][j] += A[i][k] \* B[k][j];

}

}

}

**return** C;

}

/// C = (B - A) + A^2

**void** compute\_C(**const** std::vector<std::vector<**double**>>& A, **const** std::vector<std::vector<**double**>>& B, std::vector<std::vector<**double**>>& C) {

**int** n = A.size();

**int** m = A[**0**].size();

std::vector<std::vector<**double**>> A\_squared = multiply\_matrices(A, A);

#pragma omp parallel **for** collapse(**2**)

**for** (**int** i = **0**; i < n; ++i) {

**for** (**int** j = **0**; j < m; ++j) {

C[i][j] = (B[i][j] - A[i][j]) + A\_squared[i][j];

}

}

}

/// -fopenmp

**int** main() {

ios::sync\_with\_stdio(**0**); cin.tie(**0**);

**int** n = **300**;

**int** m = **300**;

omp\_set\_num\_threads(**1**);

std::vector<std::vector<**double**>> A(n, std::vector<**double**>(m));

std::vector<std::vector<**double**>> B(n, std::vector<**double**>(m));

std::vector<std::vector<**double**>> C(n, std::vector<**double**>(m, **0.0**));

initialize\_matrix(A, n, m);

initialize\_matrix(B, n, m);

std::cout << "Matrix A:" << std::endl;

print\_matrix(A);

std::cout << "Matrix B:" << std::endl;

print\_matrix(B);

**double** startTime = omp\_get\_wtime();

compute\_C(A, B, C);

**double** endTime = omp\_get\_wtime();

std::cout << "Matrix C = (B - A) + A^2:" << std::endl;

print\_matrix(C);

std::cout << std::endl << "Elapsed time: " << endTime - startTime << " seconds" << "**\n**";

**return** **0**;

}

1. Псевдокод

Шаг 1. Инициализируем матрицы A и B случайными числами.

Шаг 2. Создаем пустую матрицу C с нулевыми значениями.

Шаг 3. Определяем функцию для умножения двух матриц

Шаг 3.1 Создаем результирующую матрицу A\_squared с нулевыми значениями.

Шаг 3.2 Для каждой строки i в матрице A, а также для каждого столбца j в матрице A, а также k в строке i и столбце j: A\_squared[i][j] += A[i][k] \* A[k][j]

Шаг 4. Начинаем отчёт времени исполнения программы.

Шаг 5. Вычисляем матрицу A\_squared умножением матрицы A на себя.

Шаг 6. Для каждой строки I и каждого столбца j: C[i][j] = (B[i][j] - A[i][j]) + A\_squared[i][j].

Шаг 7. Заканчиваем отчёт времени исполнения программы.

Шаг 8. Выводим результат работы программы и время исполнения

1. Контрольный тест. Результаты программы

Тесты на двух машинах:

1. AMD Ryzen 5 5600G 3.9GHz 6 ядер, 12 потоков, 32GB DDR4 RAM
2. INTEL(R) XEON(R) GOLD 6548Y+ 2.5–4.1GHz 64 ядра, 128 потоков, 128GB DDR5 RAM ???

Результаты запуска приведены в таблице 1.

Таблица 1 — Результаты запуска программы

|  |  |  |  |
| --- | --- | --- | --- |
| Машина | Исходные данные | В одном потоке | Параллельно |
| 1 | N=M=300 | 0,286 секунд | 0.05 секунд |
| N=M=1000 | 11.4 секунд | 2.12 секунд |
| 2 | N=M=300 | ? секунд | ? секунд |
| N=M1000 | ? секунд | ? секунд |

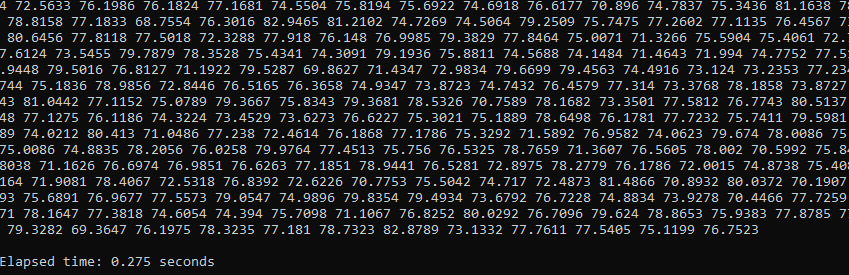


Рисунок 1 – результат работы программы

1. Выводы по работе

В ходе выполнения задания была разработана программа на языке C++, демонстрирующая разницу между запуском операций в одном потоке и параллельно в нескольких.