Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра "Вычислительной техники"

**ОТЧЁТ**

По лабораторной №1

по дисциплине "Распределенные вычисления"

на тему " Разработка параллельной программы, обеспечивающей контроль использования вычислительных ресурсов"

Вариант № 7

Выполнили студенты группы 20ВВП1:

Тумасов В.В

Ермолаев А.Д

Приняли:

Федюнин Р. Н.

Елфимов А.В

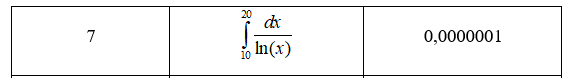
Пенза 2024

Разработка параллельной программы, обеспечивающей контроль использования вычислительных ресурсов

**Цель работы:** изучение подхода к обеспечению программного контроля использования вычислительных ресурсов системы и разработка с его использованием программы параллельной обработки данных, обеспечивающей использование вычислительных ресурсов на заданном уровне.

Лабораторное задание:

Разработать консольную программу с использованием языка С/С++ и набора функций Windows API, реализующую параллельный расчет в соответствии с номером варианта и заданием из Приложения 1. Параллелизм должен быть реализован на уровне потоков. Уровень загрузки вычислительных ресурсов (общий уровень загрузки процессоров (ядер процессора)) не должен превышать значения заданного в процентах на этапе выполнения программы с консоли.



Ход выполнения лабораторной работы:

1)Определение функции:

double Function(double x) {

return 1.0 / log(x);

}

Здесь определена подынтегральная функция ​, которая будет использоваться в вычислениях.

2)Определение функции интегрирования:

double CalculateIntegral(double start, double end, double step) {

double sum = 0.0;

for (double x = start; x < end; x += step) {

sum += Function(x) \* step;

}

return sum;

}

Функция CalculateIntegral вычисляет интеграл методом прямоугольников. Она принимает в качестве аргументов начальный и конечный пределы интегрирования (start и end) и шаг интегрирования (step). В цикле она проходит по диапазону от start до end с шагом step, вычисляя значение функции в каждой точке и умножая на шаг. Сумма этих значений дает приближенное значение определенного интеграла.

3)Подготовка к параллельным вычислениям:

for (int i = 0; i < numThreads; ++i) {

double threadStart = start + i \* ((end - start) / numThreads);

double threadEnd = start + (i + 1) \* ((end - start) / numThreads);

threads.emplace\_back(CalculateIntegralInThread, threadStart, threadEnd, step, &results[i]);

}

Здесь определяются поддиапазоны для каждого потока на основе общего диапазона интегрирования. Создаются потоки, каждый из которых вызывает функцию CalculateIntegralInThread для своего поддиапазона.

4)Вычисления в каждом потоке:

void CalculateIntegralInThread(double start, double end, double step, double\* result) {

\*result = CalculateIntegral(start, end, step);

}

В каждом потоке вызывается функция CalculateIntegral для своего поддиапазона, и результат сохраняется в соответствующий элемент вектора results.

5)Ожидание завершения потоков:

for (auto& t : threads) {

t.join();

}

Основной поток ждет завершения всех созданных потоков.

6)Суммирование результатов:

for (const auto& result : results) {

totalResult += result;

}

7)Вывод результата:

cout << "Integral result: " << totalResult << endl;

Итоговый результат выводится на экран.

Листинг программы:

#include <iostream>

#include <vector>

#include <thread>

#include <cmath>

using namespace std;

double Function(double x) {

return 1.0 / log(x);

}

double CalculateIntegral(double start, double end, double step) {

double sum = 0.0;

for (double x = start; x < end; x += step) {

sum += Function(x) \* step;

}

return sum;

}

void CalculateIntegralInThread(double start, double end, double step, double\* result) {

\*result = CalculateIntegral(start, end, step);

}

int main() {

double start = 10.0;

double end = 20.0;

double step = 0.0000001;

int numThreads = thread::hardware\_concurrency();

vector<thread> threads;

vector<double> results(numThreads);

double totalResult = 0.0;

for (int i = 0; i < numThreads; ++i) {

double threadStart = start + i \* ((end - start) / numThreads);

double threadEnd = start + (i + 1) \* ((end - start) / numThreads);

threads.emplace\_back(CalculateIntegralInThread, threadStart, threadEnd, step, &results[i]);

}

for (auto& t : threads) {

t.join();

}

for (const auto& result : results) {

totalResult += result;

}

cout << "Integral result: " << totalResult << endl;

return 0;

}

Результат выполнения программы:



Рисунок 1-Результат работы программы

Вывод:в процессе выполнения лабораторной работы был изучен метод создания и использования потоков для параллельного выполнения задач,на примере вычисления определенного интеграла.