|  |  |  |
| --- | --- | --- |
| Лабораторная работа №1  Основы работы с потоками. | Выполнил | Доценко Е.В. |
| Группа | САПР 2.1 |
| Преподаватель | Катаев А.В. |
| Дата |  |
| Оценка |  |

Цель: Изучение работы с потоками (для С++ API операционной системы либо c++ std threads) и методов замера времени программ.

Язык: С++.

Задание:

Подсчитать в массиве количество цепочек из k (k=10) нулей.

Программа реализована с использованием динамических массивов. Для реализации многопоточности использовались c++ std threads. Программа выдает результат работы последовательной работы алгоритма и результаты параллельной работы на количестве доступных потоков. Подсчёт предполагает такой механизм, что все значения элементы массива находятся в определененном, изначально заданном диапазоне[0, 2]. Проходясь по данному массиву, программа увеличивает счетчик нулей при нахождении нуля, а при когда счётчик нулей становится равным 10, то программа увеличивает счётчик цепочек.

Далее посмотрим разные ситуации.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № | Диапазон | Кол-во элементов | Последовательная | Распараллеленная | | |
| 2 потока | 3 потока | 4 потока |
| 1 | [0,2] | 1 000 000 | 0,0037 | 0,0055 | 0,0029 | 0,0047 |
| 2 | [0,2] | 100 000 000 | 0,391 | 0,204 | 0,135 | 0,112 |
| 3 | [0,2] | 1 000 000 000 | 3,949 | 2,12 | 1,377 | 1,014 |

Для анализа были выбраны 3 разные ситуации: маленькое количество элементов(1 000 000), среднее количество элементов(100 000 000) и большое количество элементов(1 000 000 000) при диапазон [0, 2].

Для начала рассморим небольшой массив для сортировки при разном количестве потоков. Распараллеливание вычислений целесообразно лишь тогда, когда массив достаточно большой и затраты на создание потоков и подсчёты действительно малы по сравнению с последовательным вычислением. Полученные результат отчетливо показали, что последовательные вычисления при небольшом массиве данных работают быстрее, чем параллельные, но с увеличением количества данных ситуация меняется(рисунки 1-3).

Рисунок 1 – График времени вычисления на 1 000 000 элементов

Рисунок 2 – График времени вычисления на 100 000 000 элементов

Рисунок 3 – График времени вычисления на 300 000 000 элементов

Выводы: Выполнив данную работу, я поработала с потоками и написала программу по подсчёту в массиве количества цепочек из k (k=10) нулей. Алгоритм подсчёта работает быстро, поэтому для анализа использовались массивы от 1 000 000 элементов. Было выяснено, что для достижения ускорение вычислений необходимо использовать выборки, начиная с 100 000 000 элементов. С ростом количества элементов уменьшается время вычисления распараллеленной версии алгоритмы

Код программы

#include <iostream>

#include <vector>

#include <thread>

#include <cstdlib>

#include <ctime>

#include <omp.h>

// Функция для заполнения динамического массива случайными числами

void fillArrayWithRandomNumbers(int\*& arr, int size, int M) {

std::srand(std::time(0));

arr = new int[size];

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

arr[i] = std::rand() % M;

}

}

// Функция для подсчета цепочек из 10 нулей в однопоточном режиме

int countZeroChainsSingleThread(const int\* arr, int size) {

int count = 0;

int chainCount = 0;

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

if (arr[i] == 0) {

count++;

if (count == 10) {

chainCount++;

count = 0;

}

}

else {

count = 0;

}

}

return chainCount;

}

// Функция для подсчета цепочек из 10 нулей в многопоточном режиме

int countZeroChainsMultiThread(const int\* arr, int size, int numThreads) {

int chunkSize = size / numThreads;

std::vector<std::thread> threads(numThreads);

std::vector<int> chainCounts(numThreads, 0);

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

threads[t] = std::thread([&, t]() {

int start = t \* chunkSize;

int end = (t == numThreads - 1) ? size : (t + 1) \* chunkSize;

int count = 0;

int chainCount = 0;

for (int i = start; i < end; i++) {

if (arr[i] == 0) {

count++;

if (count == 10) {

chainCount++;

count = 0;

}

}

else {

count = 0;

}

}

chainCounts[t] = chainCount;

});

}

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

threads[t].join();

}

int totalChainCount = 0;

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

totalChainCount += chainCounts[t];

}

return totalChainCount;

}

int main() {

setlocale(LC\_ALL, "Russian");

const int size = 1000000; // Размер массива

int\* arr = nullptr;

int M = 2; // Верхняя граница диапазона случайных чисел

fillArrayWithRandomNumbers(arr, size, M);

double startTime, endTime;

int chains;

int\* arrCopy = new int[size];

std::copy(arr, arr + size, arrCopy); // Создание копии исходного массива

// Однопоточный режим

startTime = omp\_get\_wtime();

chains = countZeroChainsSingleThread(arrCopy, size);

endTime = omp\_get\_wtime();

std::cout << "Однопоточный режим: " << chains << " цепочек из 10 нулей." << std::endl;

std::cout << "Время выполнения в однопотоковом режиме: " << endTime - startTime << " секунд." << std::endl;

delete[] arrCopy; // Освобождаем динамически выделенную память

// Многопоточный режим с разным количеством потоков

for (int numThreads = 2; numThreads <= std::thread::hardware\_concurrency(); numThreads++) {

int\* arrCopyPar = new int[size];

std::copy(arr, arr + size, arrCopyPar); // Создание копии исходного массива

startTime = omp\_get\_wtime();

chains = countZeroChainsMultiThread(arrCopyPar, size, numThreads);

endTime = omp\_get\_wtime();

std::cout << "Многопотоковой режим: " << chains << " цепочек из 10 нулей." << std::endl;

std::cout << "Время выполнения в многопотоковом режиме на " << numThreads << " потоках : " << endTime - startTime << " секунд." << std::endl;

delete[] arrCopyPar;// Освобождаем динамически выделенную память

}

delete[] arr; // Освобождаем динамически выделенную память

return 0;

}