|  |  |  |
| --- | --- | --- |
| Лабораторная работа №2  Исследование методов синхронизации, эксперименты с ошибками синхронизации. | Выполнил | Доценко Е.В. |
| Группа | САПР 2.1 |
| Преподаватель | Катаев А.В. |
| Дата |  |
| Оценка |  |

Цель: Изучение методов синхронизации многопоточных программ и агрегации данных при многопоточных вычислениях.

Язык: С++.

Задание:

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

В данной работе были реализованы 3 метода синхронизации потоков: атомарные операции, мьютекс и индивидуальные переменные для вычислений в каждом потоке. Также был реализован вариант, при котором происходит неверный подсчёт из-за ошибок синхронизации потоков (гонка).

Все функции подсчёта в своей основе имеют следующий алгоритм: 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.

Как можно наблюдать на рисунках 1-3, при использований методов синхронизации количество подсчитанных цепочек совпадает с эталонным(последовательным) подсчётом, что говорит о том, что функции работают корректно. Также можно заметить, что, если не использовать методы синхронизации, то программа хоть и начинает работать быстрее с увеличением размера массива, однако результат на выходе не соответвует действительности.

На графиках 1-3, можно наблюдать, что, действительно, при большом количестве данных затраты на создание потоков, использование методов синхранизации и паралелльное вычисления существенно малы по сравнению с затратами на последовательное вычисление.

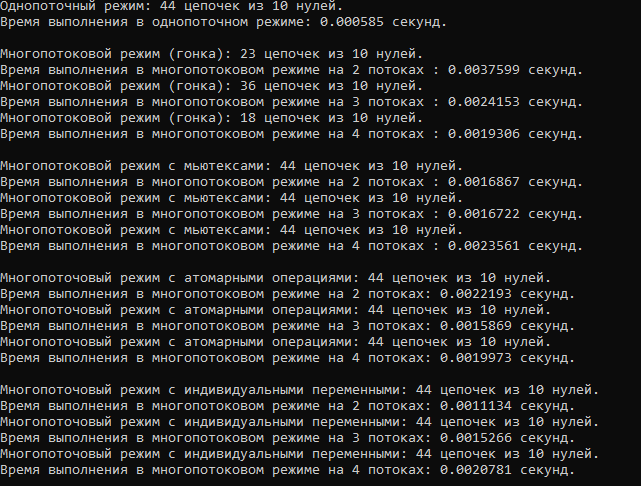


Рисунок 1 - Результат выполнения программы на 100 000 элементах массива

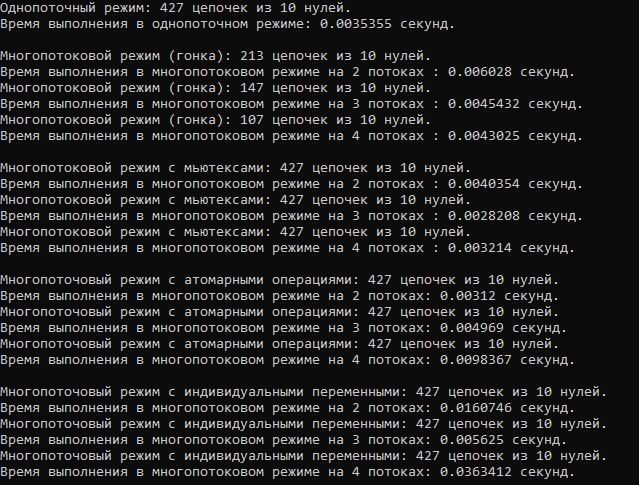


Рисунок 2 - Результат выполнения программы на 1 000 000 элементах массива

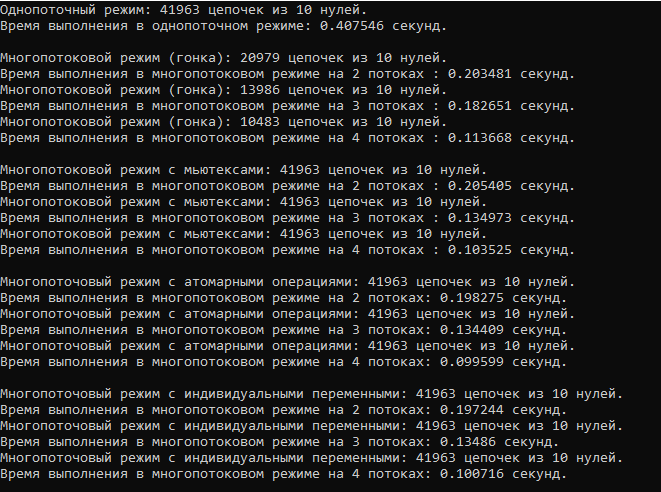


Рисунок 3 - Результат выполнения программы на 100 000 000 элементах массива

График 1 –Время обработки 100 000 000 элементов с помощью мьютексов

График 2 – Время обработки 100 000 000 элементов с помощью атомарных операций

График 3 – Время обработки 100 000 000 элементов с помощью индивидуальных переменных

Выводы: Выполнив данную работу, я изучила методы синхронизации многопоточных программ и агрегации данных при многопоточных вычислениях и написала программу по подсчёту в массиве количества цепочек из k (k=10) нулей. Для синхронизации использовались такие методы как атомарные операции, мьютекс и индивидуальные переменные. На 100 000 000 элементах лучший результат показали атомарные операции.

Код программы

#include <iostream>

#include <vector>

#include <thread>

#include <cstdlib>

#include <ctime>

#include <omp.h>

#include <mutex>

std::mutex mtx; // Объявляем мьютекс для синхронизации доступа к общим ресурсам

std::atomic<int> totalChainCountAtomic(0); // Объявляем атомарную переменную для синхронизации

// 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 (с гонкой)

int countZeroChainsMultiThreadWithMutexIncorrect(const int\* arr, int size, int numThreads, int& totalChainCount) {

int chunkSize = size / numThreads;

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

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;

int test = totalChainCount;

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

if (arr[i] == 0) {

count++;

if (count == 10) {

chainCount++;

count = 0;

}

}

else {

count = 0;

}

}

totalChainCount = chainCount + test;

});

}

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

threads[t].join();

}

return totalChainCount;

}

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

int countZeroChainsMultiThreadWithMutex(const int\* arr, int size, int numThreads, int& totalChainCount) {

int chunkSize = size / numThreads;

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

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;

}

}

// Захватываем мьютекс перед обновлением общей переменной

mtx.lock();

totalChainCount += chainCount;

mtx.unlock();

});

}

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

threads[t].join();

}

return totalChainCount;

}

// Функция для подсчета цепочек из 10 нулей в многопоточном режиме с атомарными операциями

void countZeroChainsMultiThreadWithAtomic(const int\* arr, int size, int numThreads) {

int chunkSize = size / numThreads;

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

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;

}

}

// Используем атомарную операцию для обновления общей переменной

totalChainCountAtomic.fetch\_add(chainCount, std::memory\_order\_relaxed);

});

}

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

threads[t].join();

}

}

// Функция для подсчета цепочек из 10 нулей в многопоточном режиме с индивидуальными переменными

int countZeroChainsMultiThreadWithIndividualVars(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 = 100000000000; // Размер массива

int\* arr = nullptr;

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

fillArrayWithRandomNumbers(arr, size, M);

double startTime, endTime;

int chains;

startTime = omp\_get\_wtime();

chains = countZeroChainsSingleThread(arr, size);

endTime = omp\_get\_wtime();

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

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

std::cout << "" << std::endl;

// Многопоточный режим с мьютексами некорректный

int totalChainCount = 0;

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

startTime = omp\_get\_wtime();

totalChainCount = 0; // Обнуляем общую переменную

chains = countZeroChainsMultiThreadWithMutexIncorrect(arr, size, numThreads, totalChainCount);

endTime = omp\_get\_wtime();

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

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

}

std::cout << "" << std::endl;

// Многопоточный режим с мьютексами

totalChainCount = 0;

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

startTime = omp\_get\_wtime();

totalChainCount = 0; // Обнуляем общую переменную

chains = countZeroChainsMultiThreadWithMutex(arr, size, numThreads, totalChainCount);

endTime = omp\_get\_wtime();

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

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

}

std::cout << "" << std::endl;

// Многопоточный режим с атомарными операциями

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

startTime = omp\_get\_wtime();

totalChainCountAtomic = 0; // Обнуляем атомарную переменную

countZeroChainsMultiThreadWithAtomic(arr, size, numThreads);

endTime = omp\_get\_wtime();

chains = totalChainCountAtomic.load(std::memory\_order\_relaxed); // Получаем значение атомарной переменной

std::cout << "Многопоточовый режим с атомарными операциями: " << chains << " цепочек из 10 нулей." << std::endl;

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

}

std::cout <<""<< std::endl;

// Многопоточный режим с индивидуальными переменными

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

startTime = omp\_get\_wtime();

chains = countZeroChainsMultiThreadWithIndividualVars(arr, size, numThreads);

endTime = omp\_get\_wtime();

std::cout << "Многопоточовый режим с индивидуальными переменными: " << chains << " цепочек из 10 нулей." << std::endl;

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

}

delete[] arr; // Освобождаем динамически выделенную память

return 0;

}