|  |  |  |
| --- | --- | --- |
| Лабораторная работа №3  Работа с OpenMP. | Выполнил | Доценко Е.В. |
| Группа | САПР 2.1 |
| Преподаватель | Катаев А.В. |
| Дата |  |
| Оценка |  |

Цель: изучение основ работы с библиотекой openmp.

Язык: С++.

Задание:

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

Программа реализована с использованием динамических массивов. Для реализации многопоточности использовалась библиотека openmp. 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] | 10 000 000 | 0,0439 | 0,0502 | 0,0941 | 0,0642 |
| 2 | [0,2] | 100 000 000 | 0,4180 | 0,3994 | 0,3989 | 0,3868 |
| 3 | [0,2] | 100 000 000 000 | 5,8840 | 6,8115 | 5,6856 | 5,5659 |

Для анализа были выбраны 3 разные ситуации: маленькое количество элементов(10 000 000), среднее количество элементов(100 000 000) и большое количество элементов(100 000 000 000) при диапазон [0, 2].

Из рисунков 1-3 можно сделать вывод, что программа работает правильно и при большом количестве элементов действительно распараллеливание помогает ускорить программу и не влияет на качество подсчётов.

Рисунок 1 – График времени вычисления на 10 000 000 элементов

Рисунок 2 – График времени вычисления на 100 000 000 элементов

Рисунок 3 – График времени вычисления на 100 000 000 000 элементов

Алгоритм использования openmp:

* #pragma omp parallel reduction(+:chainCount) создает параллельную область, где каждый поток имеет свою копию переменной localCount для локального подсчета цепочек из 10 нулей. reduction(+:chainCount) указывает на то, что значение chainCount будет суммироваться после выполнения параллельной области.
* #pragma omp for создает параллельный цикл, который разделяется между потоками. Каждый поток работает с определенной частью массива arr и подсчитывает количество цепочек из 10 нулей в своей части.
* Внутри цикла проверяется каждый элемент массива. Если элемент равен нулю, увеличивается локальный счетчик localCount. Когда localCount достигает 10, увеличивается переменная chainCount, представляющая количество цепочек из 10 нулей.
* После завершения параллельной области осуществляется суммирование значений chainCount, полученных от каждого потока, и возвращается общее количество цепочек из 10 нулей в массиве.

Выводы: Выполнив данную работу, я поработала с библиотекой openmp и написала программу по подсчёту в массиве количества цепочек из k (k=10) нулей. 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 <cstdlib>

#include <ctime>

#include <algorithm>

#include <omp.h>

#include <thread>

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 с использованием OpenMP

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

int chainCount = 0;

omp\_set\_num\_threads(threads);

#pragma omp parallel reduction(+:chainCount)

{

int localCount = 0;

#pragma omp for

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

if (arr[i] == 0) {

localCount++;

if (localCount == 10) {

chainCount++;

localCount = 0;

}

}

else {

localCount = 0;

}

}

}

return chainCount;

}

// 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;

}

int main() {

setlocale(LC\_ALL, "Russian");

const int size = 100000000; // Размер массива

int\* arr = nullptr;

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

fillArrayWithRandomNumbers(arr, size, M);

int chains;

// Однопоточный режим

double startTime = omp\_get\_wtime();

chains = countZeroChainsSingleThread(arr, size);

double 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;

// Многопоточный режим с разным количеством потоков

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

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

double startTime = omp\_get\_wtime();

chains = countZeroChainsMultiThread(arr, size, numThreads);

double 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[] arr;

return 0;

}