ФЕДЕРАЛЬНОЕ АГЕНТСТВО ЖЕЛЕЗНОДОРОЖНОГО ТРАНСПОРТА

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ИРКУТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ПУТЕЙ СООБЩЕНИЯ» (ФГБОУ ВО ИрГУПС)

Факультет: «Управления на транспорте и информационные технологии»

Кафедра: «Информационные системы и защита информации»

Лабораторная работа № 2

по дисциплине «Программирование параллельных процессов»

«Распараллеливание вычислений средствами OpenMP»

ЛР.430500.09.04.04.186.ПЗ

Выполнил: Проверил:

студент гр. ПИм 1-22-1 преподаватель

Емельянова К.А Черкашин Е.А.

« » 2022 г. « » 2022г.

Иркутск 2022

**Задание**

Лабораторная работа выполняется на языке C++ в среде Visual Studio. Во всех заданиях следует обеспечить контроль вводимой информации. При некорректном вводе – повторно запрашивать информацию у пользователя.

Выполнить вычисления в однопоточном и многопоточном режиме и сравнить времена выполнения. Количество потоков задаётся пользователем. Количество исходных данных не кратно в общем случае количеству потоков. Исходные данные для задания генерируются с помощью генератора псевдослучайных чисел, где 100000<n<1000000, 100<An<10000000. Результаты сравниваются по времени выполнения при разном числе процессов и объёме данных и оформляются в виде таблицы. В отчёте приводятся снимки экрана, программный код, таблицы тестов и замеров времени выполнения, формулируется вывод. Количество потоков по-умолчанию при необходимости получать средствами OpenMP.

Примечание: В качестве первого этапа работы рекомендуется переписать лабораторную работу № 1 (задание A) на язык C++, а затем применить к нему средства распараллеливания.

**Вариант №4**

Дана последовательность натуральных чисел {a0…an–1}. Создать многопоточное приложение для поиска суммы корней квадратных из ai.

**Содержание работы**

В ходе выполнения лабораторной работы на языке C++ в среде разработки Visual Studio по описанному заданию была создана программа для вычислений в однопоточном и многопоточном режиме. Для многопоточного режима создано два варианта реализации – с использованием критической секции и редуктивной переменной. Программа была обеспечена контролем корректности вводимой информации. При некорректном вводе появляется сообщение об ошибке, и информация повторно запрашивается у пользователя. Код программы представлен в листингах в ПРИЛОЖЕНИИ А.

В результате выполнения работы были получены, результаты, изображенные на рисунке 1.

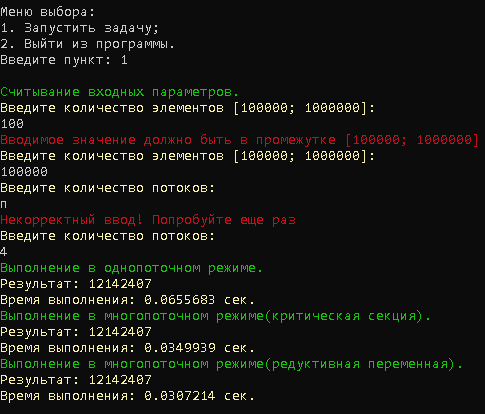


Рисунок 1 – Пример результата выполнения задачи

Далее выполним вычисления в однопоточном и многопоточном режиме и сравним время выполнения при разном числе процессов и объёме данных. Результаты оформим в виде таблицы 1.

Таблица 1 – Результаты времени выполнения (в секундах)

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Размер последовательности** | | |
| **Режим** | 100 000  элементов | 500 000  элементов | 1 000 000  элементов |
| **Однопоточный** | 0.114183 | 0.366934 | 0.704406 |
| **Многопоточный (критическая секция)** |  | | |
| 2 потока | 0.037565 | 0.145228 | 0.285821 |
| 10 потоков | 0.065447 | 0.158899 | 0.290506 |
| 50 потоков | 0.106483 | 0.143194 | 0.299613 |
| **Многопоточный (редуктивная переменная)** |  | | |
| 2 потока | 0.032206 | 0.154515 | 0.316837 |
| 10 потоков | 0.0312307 | 0.158419 | 0.289096 |
| 50 потоков | 0.0316949 | 0.149427 | 0.344674 |

**Результаты**

В результате выполнения лабораторной работы сделаем выводы, что в данной технологии многопоточный режим в обеих реализациях имеет огромное преимущество над однопоточным режимом, так как во всех тестах он намного производительнее.

Также нужно отметить, что в многопоточном режиме с критической секцией при малом объеме данных оптимальным является 2-х поточный режим, а хуже всего себя показывает 50-ти поточный. Однако при росте объема данных разница между 2-мя, 10-ю и 50-ю потоками нивелируется и становится минимальной. Таким образом, в данной реализации многопоточности слишком большое количество потоков понижает производительность вычислений, кроме тех случаев, когда выполняется работа с громоздким объемом данных.

Можно также заметить, что в многопоточном режиме с редуктивной переменной есть прямая зависимость от роста объема данных, то есть время выполнения вычислений уменьшается параллельно уменьшению объема данных. А разница между 2-мя, 10-ю и 50-ю потоками при идентичном объеме данных всегда минимальна.

В завершении, сравнивая между собой две реализации многопоточного режима, можно отметить, что при малом объеме данных и одинаковом числе потоков многопоточный режим с редуктивной переменной производительнее режима с критической секцией, однако при росте объема данных разница между ними нивелируется и становится минимальной.

ПРИЛОЖЕНИЕ А

**Листинг 1 – Файл Main.cpp**

#include "CommandLine.h"

int main**()**

**{**

setlocale**(**LC\_ALL**,** "Russian"**);**

// Запускаем консольное приложение

CommandLine cmd**;**

cmd**.**startWork**();**

**return** 0**;**

**}**

**Листинг 2 – Файл CommandLine.h**

#pragma once

#include <iostream>

#include <windows.h>

class CommandLine

**{**

public**:**

CommandLine**();**

void startWork**();**

friend void errorOutput**(**std**::**string content**);**

friend void setColorCmd**(**int numColor**);**

**};**

**Листинг 3 – Файл CommandLine.cpp**

#include "CommandLine.h"

#include "WorkTask.h"

/// <summary>Конструктор класса CommandLine.</summary>

CommandLine**::**CommandLine**()**

**{**

setColorCmd**(**3**);**

std**::**cout **<<** "Вариант №4. Дана последовательность натуральных чисел {a0…an–1}.\n"

**<<** "Создать многопоточное приложение для поиска суммы квадратных\n"

**<<** "корней из элементов данной последовательности.\n"**;**

**}**

/// <summary>Метод для запуска работы с командной строкой.</summary>

void CommandLine**::**startWork**()**

**{**

int choise**;**

bool createFlag**;**

// Пока программа не завершит работу, будет появляться меню выбора

**do** **{**

createFlag **=** **false;**

setColorCmd**(**7**);**

std**::**cout **<<** "\nМеню выбора:\n1. Запустить задачу;\n2. Выйти из программы.\nВведите пункт: "**;**

std**::**cin **>>** choise**;**

**switch** **(**choise**)**

**{**

**case** 1**:**

createFlag **=** **true;**

**break;**

**case** 2**:**

exit**(**0**);**

**break;**

**default:**

errorOutput**(**"Неизвестная команда! Попробуйте еще раз"**);**

**}**

**if** **(**createFlag**)** **{**

// Создаем объект класса WorkTask для работы с задачей

WorkTask wTsk**;**

**}**

**}** **while** **(true);**

**}**

/// <summary>Метод для вывода сообщения об ошибке.</summary>

/// <**param** name="content">Содержание сообщения</**param**>

void errorOutput**(**std**::**string content**)**

**{**

// Выводим красное сообщение об ошибке

setColorCmd**(**4**);**

std**::**cout **<<** content **<<** std**::**endl**;**

setColorCmd**(**7**);**

// Перезагружаем поток вывода

std**::**cin**.**clear**();**

std**::**cin**.**ignore**(**256**,** '\n'**);**

**}**

/// <summary>Метод для смены цвета текста в командной строке.</summary>

/// <**param** name="numColor">Номер цвета</**param**>

void setColorCmd**(**int numColor**)**

**{**

HANDLE handle **=** GetStdHandle**(**STD\_OUTPUT\_HANDLE**);**

SetConsoleTextAttribute**(**handle**,** numColor**);**

**}**

**Листинг 4 – Файл WorkTask.h**

#pragma once

#include <iostream>

#include <vector>

#include <string>

#include <sstream>

class WorkTask **{**

public**:**

WorkTask**();**

private**:**

const int MIN\_COUNT\_ELEMENTS **=** 100000**;**

const int MAX\_COUNT\_ELEMENTS **=** 1000000**;**

int countElements**;**

int countThreads**;**

std**::**vector **<**int**>** arrElem**;**

void readInputData**();**

int readElementsFromConsole**();**

int readThreadsFromConsole**();**

friend void writeOutputData**(**std**::**string name**,** int result**,** double time**);**

void initArrayRandomData**();**

**};**

**Листинг 5 – Файл WorkTask.cpp**

#include "WorkTask.h"

#include "CalculateTask.h"

#include "CommandLine.h"

/// <summary>Конструктор класса WorkTask.</summary>

WorkTask**::**WorkTask**()**

**{**

readInputData**();** // Читаем входные данные

initArrayRandomData**();** // Инициализируем последовательность чисел

// Создаем объект класса CalculateTask для вычислений

CalculateTask cTsk**(**arrElem**,** countThreads**);**

**}**

/// <summary>Метод для чтения входных данных.</summary>

void WorkTask**::**readInputData**()**

**{**

setColorCmd**(**10**);**

std**::**cout **<<** "\nСчитывание входных параметров." **<<** std**::**endl**;**

countElements **=** readElementsFromConsole**();** // Вводим количество элементов

countThreads **=** readThreadsFromConsole**();** // Вводим количество потоков

**}**

/// <summary>Метод для ввода с консоли количества элементов в последовательности.</summary>

/// <**returns**>Число элементов в последовательности.</**returns**>

int WorkTask**::**readElementsFromConsole**()**

**{**

int readResult**;**

bool errorFlag **=** **true;**

// Пока не будет не обнаружено ошибки

**do** **{**

setColorCmd**(**14**);**

std**::**cout **<<** "Введите количество элементов [" **<<** MIN\_COUNT\_ELEMENTS **<<** "; " **<<** MAX\_COUNT\_ELEMENTS **<<** "]:\n"**;**

setColorCmd**(**7**);**

// Если ввод корректный и число положительное

**if** **(**std**::**cin **>>** readResult **&&** readResult **>=** 0**)** **{**

// Если число соответствует заданному диапазону

**if** **(**readResult **>=** MIN\_COUNT\_ELEMENTS **&&** readResult **<=** MAX\_COUNT\_ELEMENTS**)** **{**

errorFlag **=** **false;**

**}** // Иначе сообщение об ошибке диапазона

**else** **{**

std**::**stringstream errorContent**;**

errorContent **<<** "Вводимое значение должно быть в промежутке ["

**<<** MIN\_COUNT\_ELEMENTS **<<** "; "**<<** MAX\_COUNT\_ELEMENTS **<<** "]"**;**

errorOutput**(**errorContent**.**str**());**

**}**

**}** // Иначе сообщение о некорректном вводе

**else** **{**

errorOutput**(**"Некорректный ввод! Попробуйте еще раз"**);**

**}**

**}** **while** **(**errorFlag**);**

**return** readResult**;**

**}**

/// <summary>Метод для ввода с консоли количества потоков для вычислений.</summary>

/// <**returns**>Число потоков.</**returns**>

int WorkTask**::**readThreadsFromConsole**()**

**{**

int readResult**;**

bool errorFlag **=** **true;**

// Пока не будет не обнаружено ошибки

**do** **{**

setColorCmd**(**14**);**

std**::**cout **<<** "Введите количество потоков:\n"**;**

setColorCmd**(**7**);**

// Если ввод корректный и число положительное

**if** **(**std**::**cin **>>** readResult **&&** readResult **>** 0**)** **{**

errorFlag **=** **false;**

**}** // Иначе сообщение о некорректном вводе

**else** **{**

errorOutput**(**"Некорректный ввод! Попробуйте еще раз"**);**

**}**

**}** **while** **(**errorFlag**);**

**return** readResult**;**

**}**

/// <summary>Метод для инициализации последовательности псевдослучайными числами.</summary>

void WorkTask**::**initArrayRandomData**()**

**{**

//srand(time(0)); для действительных случайных чисел

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

arrElem**.**push\_back**(**rand**()** **%** **((**10000000 **+** 1**)** **-** 100**)** **+** 100**);**

**}**

**}**

/// <summary>Метод для вывода результатов вычислений и времени выполнения.</summary>

/// <**param** name="name">Наименование режима вычисления</**param**>

/// <**param** name="result">Результат</**param**>

/// <**param** name="time">Время выполнения</**param**>

void writeOutputData**(**std**::**string name**,** int result**,** double time**)**

**{**

setColorCmd**(**10**);**

std**::**cout **<<** name**;**

setColorCmd**(**14**);**

std**::**cout **<<** "\nРезультат: " **<<** result **<<** "\nВремя выполнения: " **<<** time **<<** " сек.\n"**;**

**}**

**Листинг 6 – Файл CalculateTask.h**

#pragma once

#include <iostream>

#include <ctime>

#include <stdio.h>

#include <omp.h>

#include <conio.h>

#include <vector>

class CalculateTask **{**

public**:**

CalculateTask**(**std**::**vector **<**int**>** arr**,** int countTh**);**

private**:**

int countThreads**;** // Количество потоков

std**::**vector **<**int**>** arrElem**;** //Посдеовательность чисел

void executionWithoutThread**();**

void executionWithThreadCS**();**

void executionWithThreadRV**();**

**};**

**Листинг 7 – Файл CalculateTask,cpp**

#include "CalculateTask.h"

#include "WorkTask.h"

/// <summary>Конструктор класса CalculateTask.</summary>

/// <**param** name="arr">Последовательность чисел</**param**>

/// <**param** name="countTh">Количество потоков</**param**>

CalculateTask**::**CalculateTask**(**std**::**vector **<**int**>** arr**,** int countTh**):** arrElem**(**arr**),** countThreads**(**countTh**)**

**{**

// Запускаем режимы вычислений

executionWithoutThread**();**

executionWithThreadCS**();**

executionWithThreadRV**();**

**}**

/// <summary>Метод для однопоточного вычисления.</summary>

void CalculateTask**::**executionWithoutThread**()**

**{**

double result **=** 0**;**

double time **=** omp\_get\_wtime**();** // старт таймера

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

result **+=** sqrt**(**arrElem**[**i**]);**

**}**

time **=** omp\_get\_wtime**()** **-** time**;** // Получаем время

// Выводим результ и время выполнения

writeOutputData**(**"Выполнение в однопоточном режиме."**,** result**,** time**);**

**}**

/// <summary>Метод для многопоточного вычисления с использованием критической секции.</summary>

void CalculateTask**::**executionWithThreadCS**()**

**{**

double result **=** 0**;**

double pResult **=** 0**;**

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

omp\_set\_dynamic**(false);**

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

#pragma omp parallel firstprivate(pResult) shared(result)

**{**

pResult **=** 0**;**

#pragma omp for

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

pResult **+=** sqrt**(**arrElem**[**i**]);**

**}**

#pragma omp critical

**{**

result **+=** pResult**;**

**}**

**}**

time **=** omp\_get\_wtime**()** **-** time**;**

writeOutputData**(**"Выполнение в многопоточном режиме(критическая секция)."**,** result**,** time**);**

**}**

/// <summary>Метод для многопоточного вычисления с использованием редуктивной переменной.</summary>

void CalculateTask**::**executionWithThreadRV**()**

**{**

double result **=** 0**;**

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

#pragma omp parallel for reduction (+:result)

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

result **+=** sqrt**(**arrElem**[**i**]);**

**}**

time **=** omp\_get\_wtime**()** **-** time**;**

writeOutputData**(**"Выполнение в многопоточном режиме(редуктивная переменная)."**,** result**,** time**);**

**}**