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 № 3

по дисциплине «Программирование параллельных процессов»

«Распараллеливание вычислений средствами MPI»

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# (с помощью библиотеки MPI.NET) или C++ в среде Visual Studio (на усмотрение студента). 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 и элементов в массиве 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.

Примечание: В качестве первого этапа работы можно взять за основу лабораторную работу № 1 (задание A) и № 2.

**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++, с помощью инфраструктуры, реализованной библиотекой MPI, в среде разработки Visual Studio, по описанному заданию, была создана программа для вычислений в однопоточном и многопоточном режиме. Программа была обеспечена контролем корректности вводимой информации. При некорректном вводе появляется сообщение об ошибке, и информация повторно запрашивается у пользователя. Для работы приложения с использованием MPI были установлены Microsoft HPC Pack 2012, его SDK и Microsoft MPI v10.4, а среда разработки была настроена под них. Код программы представлен в листингах в ПРИЛОЖЕНИИ А.

Для запуска приложения используется утилита mpiexec.exe, где вводится команда вида (текущая директория – папка с exe файлом проекта): **mpiexec –n процессы исполнимый\_файл.exe**.В результате выполнения работы были получены, результаты, изображенные на рисунке 1.

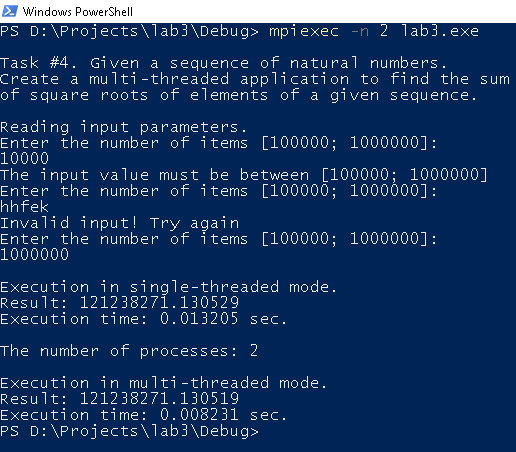


Рисунок 1 – Пример результата выполнения задачи

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** | 0.001598 | 0.006561 | 0.013205 |
| **Многопоточный** |  | | |
| 2 потока | 0.000753 | 0.004169 | 0.008231 |
| 10 потоков | 0.000157 | 0.000838 | 0.001595 |
| 50 потоков | 0.000073 | 0.000193 | 0.000320 |

**Результаты**

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

Можно также заметить, что в многопоточном режиме есть прямая зависимость от роста объема данных, то есть время выполнения вычислений уменьшается параллельно уменьшению объема данных. Также нужно отметить большую разницу между 2-мя, 10-ю и 50-ю потоками при идентичном объеме данных в пользу большого количества потоков. То есть чем больше число потоков, тем выше производительность.

Также, если сравнить производительность многопоточного режима при идентичном количестве потоков, но разном объеме данных, можно сделать вывод, что потенциал многопоточности раскрывается при росте объёма данных. Чем больше элементов в последовательности, тем выше количество элементов, обрабатываемых за 1 м/с.

В результате можно отметить, что из всех четырех технологий, рассмотренных в данной дисциплине, распараллеливание вычислений средствами MPI показало себя наиболее производительно.

ПРИЛОЖЕНИЕ А

**Листинг 1 – Файл Main.cpp**

#include "MpiexecTask.h"

int main**()**

**{**

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

// Создаем объект класса MpiexecTask и запускаем на исполнение

MpiexecTask mpiTsk**;**

mpiTsk**.**run**();**

**return** 0**;**

**}**

**Листинг 2 – Файл MpiexecTask.h**

#pragma once

#include <ctime>

#include <iostream>

#include "mpi.h"

#include "CommandLine.h"

class MpiexecTask **{**

public**:**

MpiexecTask**();**

void run**();**

private**:**

MPI\_Status status**;**

int numbThread**,** countThreads**,** countElements**;**

double time**;**

double**\*** initArrayRandomData**();**

void executionWithoutThread**(**double**\*** arrElem**);**

void executionWithRootThread**(**double**\*** arrElem**);**

void executionWithOtherThreads**();**

double calcPartElemThread**(**double **\***arrElem**,** int arrLength**);**

**};**

**Листинг 3 – Файл MpiexecTask.cpp**

#include "MpiexecTask.h"

/// <summary>Конструктор класса MpiexecTask.</summary>

MpiexecTask**::**MpiexecTask**()**

**{**

// Инициализируем библиотеку MPI

MPI\_Init**(NULL,** **NULL);**

// Сообщаем размер группы и порядковый номер вызывающей задачи

MPI\_Comm\_size**(**MPI\_COMM\_WORLD**,** **&**countThreads**);**

MPI\_Comm\_rank**(**MPI\_COMM\_WORLD**,** **&**numbThread**);**

// Досрочно выходим, если процесс всего один

**if** **(**countThreads **<** 2**)** **{**

writeError**(**"Too few processes! Press any key to exit..."**);**

MPI\_Finalize**();**

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

**}**

// Ставим точку синхронизации

MPI\_Barrier**(**MPI\_COMM\_WORLD**);**

**}**

/// <summary>Метод для запуска взаимодействия через передачу сообщений (MPI).</summary>

void MpiexecTask**::**run**()**

**{**

// Если работает 0 процесс (root процесс)

**if** **(**numbThread **==** 0**)** **{**

// Считываем входные параметры и создаем массив с случанйыми числами

CommandLine cmd**;**

countElements **=** cmd**.**readInputData**();**

double **\***arrElem **=** initArrayRandomData**();**

// Запускаем однопоточное вычисление

executionWithoutThread**(**arrElem**);**

// Запускаем многопоточное вычисление(взаимодействие) в root процессе

executionWithRootThread**(**arrElem**);**

**}**

**else** **{** // Иначе, если работают остальные процессы

// Запускаем многопоточное вычисление(взаимодействие) в остальных процессах

executionWithOtherThreads**();**

**}**

// Закрываем библиотеку MPI

MPI\_Finalize**();**

**}**

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

/// <**returns**>Массив c псевдослучайными числами.</**returns**>

double**\*** MpiexecTask**::**initArrayRandomData**()**

**{**

double **\***arrElem **=** **new** double**[**countElements**];**

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

arrElem**[**i**]** **=** rand**()** **%** **((**10000000 **+** 1**)** **-** 100**)** **+** 100**;**

**}**

**return** arrElem**;**

**}**

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

/// <**param** name="arrElem">Массив c псевдослучайными числами</**param**>

void MpiexecTask**::**executionWithoutThread**(**double**\*** arrElem**)**

**{**

time **=** MPI\_Wtime**();**

double sum **=** 0**;**

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

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

**}**

time **=** MPI\_Wtime**()** **-** time**;**

writeOutputData**(**"\nExecution in single-threaded mode."**,** sum**,** time**);**

**}**

/// <summary>Метод для запуска многопоточного вычисления в root процессе.</summary>

/// <**param** name="arrElem">Массив c псевдослучайными числами</**param**>

void MpiexecTask**::**executionWithRootThread**(**double**\*** arrElem**)**

**{**

std**::**cout **<<** "\nThe number of processes: " **<<** countThreads **<<** std**::**endl**;**

time **=** MPI\_Wtime**();**

// Помещаем кол-во элементов в другие процессы

**for** **(**int i **=** 1**;** i **<** countThreads**;** i**++)** **{**

MPI\_Send**(&**countElements**,** 1**,** MPI\_INT**,** i**,** 7**,** MPI\_COMM\_WORLD**);**

**}**

// Помещаем массив данных в другие процессы

**for** **(**int i **=** 1**;** i **<** countThreads**;** i**++)** **{**

MPI\_Send**(**arrElem**,** countElements**,** MPI\_DOUBLE**,** i**,** 3**,** MPI\_COMM\_WORLD**);**

**}**

// Вычисляем сумму для 0 процесса

double sum **=** calcPartElemThread**(**arrElem**,** countElements**);**

// Берем промеж. суммы из других процессов и суммируем

**for** **(**int i **=** 1**;** i **<** countThreads**;** i**++)** **{**

double psum **=** 0**;**

MPI\_Recv**(&**psum**,** 1**,** MPI\_DOUBLE**,** i**,** 4**,** MPI\_COMM\_WORLD**,** **&**status**);**

sum **+=** psum**;**

**}**

// Получаем и выводим результат

time **=** MPI\_Wtime**()** **-** time**;**

writeOutputData**(**"\nExecution in multi-threaded mode."**,** sum**,** time**);**

**}**

/// <summary>Метод для запуска многопоточного вычисления в остальных процессах.</summary>

void MpiexecTask**::**executionWithOtherThreads**()**

**{**

int getCountElements **=** 0**;**

// Берем ков-во элементов из 0 процесса

MPI\_Recv**(&**getCountElements**,** 1**,** MPI\_INT**,** 0**,** 7**,** MPI\_COMM\_WORLD**,** **&**status**);**

double **\***getArrElem **=** **new** double**[**getCountElements**];**

// Берем массив данных из 0 процесса

MPI\_Recv**(**getArrElem**,** getCountElements**,** MPI\_DOUBLE**,** 0**,** 3**,** MPI\_COMM\_WORLD**,** **&**status**);**

// Вычисляем промежуточные сумму и помещаем их в 0 процесс

double psum **=** calcPartElemThread**(**getArrElem**,** getCountElements**);**

MPI\_Send**(&**psum**,** 1**,** MPI\_DOUBLE**,** 0**,** 4**,** MPI\_COMM\_WORLD**);**

**}**

/// <summary>Метод для вычисления части элементов конкретным потоком.</summary>

/// <**param** name="arrElem">Массив c псевдослучайными числами</**param**>

/// <**param** name="sizeArr">Размер массива</**param**>

/// <**returns**>Промежуточная сумма корней элементов.</**returns**>

double MpiexecTask**::**calcPartElemThread**(**double**\*** arrElem**,** int sizeArr**)**

**{**

int beginPart**,** sizePart**,** endPart**;**

double psum **=** 0**;**

// Находим длину части последовательности, ее начало и конец

sizePart **=** sizeArr **/** countThreads**;**

beginPart **=** sizePart **\*** numbThread**;**

endPart **=** beginPart **+** sizePart**;**

**if** **(**numbThread **==** countThreads **-** 1**)** endPart **=** sizeArr**;**

// Вычисляем промежуточную сумму коней элементов на отрезке

**for** **(**int i **=** beginPart**;** i **<** endPart**;** i**++)** **{**

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

**}**

**return** psum**;**

**}**

**Листинг 4 – Файл CommandLine.h**

#pragma once

#include <iostream>

#include <windows.h>

#include <string>

#include <sstream>

class CommandLine

**{**

public**:**

CommandLine**();**

int readInputData**();**

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

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

private**:**

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

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

**};**

**Листинг 5 – Файл CommandLine.cpp**

#include "CommandLine.h"

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

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

**{**

std**::**cout **<<** "\nTask #4. Given a sequence of natural numbers.\n"

**<<** "Create a multi-threaded application to find the sum\n"

**<<** "of square roots of elements of a given sequence.\n"**;**

**}**

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

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

int CommandLine**::**readInputData**()**

**{**

int readResult**;**

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

std**::**cout **<<** "\nReading input parameters." **<<** std**::**endl**;**

// Пока не будет не обнаружено ошибки

**do** **{**

std**::**cout **<<** "Enter the number of items [" **<<** MIN\_COUNT\_ELEMENTS **<<** "; " **<<** MAX\_COUNT\_ELEMENTS **<<** "]:\n"**;**

// Если ввод корректный

**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 **<<** "The input value must be between ["

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

writeError**(**errorContent**.**str**());**

**}**

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

**else** **{**

writeError**(**"Invalid input! Try again"**);**

**}**

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

**return** readResult**;**

**}**

/// <summary>Дружественный метод для вывода сообщения об ошибке.</summary>

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

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

**{**

// Выводим сообщение об ошибке

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

// Перезагружаем поток вывода

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

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

**}**

/// <summary>Дружественный 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 nameMode**,** double result**,** double time**)**

**{**

std**::**cout **<<** nameMode

**<<** "\nResult: " **<<** std**::**fixed **<<** result

**<<** "\nExecution time: " **<<** time **<<** " sec.\n"**;**

**}**