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 № 5

по дисциплине «Программирование параллельных процессов»

«Параллельные вычисления с использованием набора библиотек Qt 5.x»

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**

Выполнить решение задачи с помощью инструментария QtConcurrent в виде графического приложения. Реализовать возможность приостановки и возобновления вычислений. Сравнить время вычислений в параллельном и последовательном режимах.

Исходные данные генерируются с помощью датчика случайных чисел, где 100000<n<1000000, 100<An<100000. Результаты рекомендуется сравнить по времени выполнения при разном числе процессов и оформить в виде таблицы.

**Вариант №4**

Дана последовательность натуральных чисел {*a0…an–1*}. Создать приложение для поиска произведения чисел *a0\*а1\*…\*an–1*.

**Введение**

В ходе выполнения лабораторной работы на языке C++ в среде разработки Qt Creator версии 9.0.0 (Community) по описанному заданию была создана программа с графическим интерфейсом. Она была обеспечена контролем корректности вводимой информации. При некорректном вводе появляется сообщение об ошибке, и информация повторно запрашивается у пользователя. Также была добавлена возможность приостановки и возобновления вычислений. Для искусственного замедления вычислений в учебных целях в обоих режимах вычислений была использована конструкция, вводящая задержку на 1 мс. При разработке использовался комплект Qt 5.15.2 MinGW 64-bit и система автоматизации сборки проекта qmake. Далее эта программа была использована для сравнения по времени выполнения в параллельном и последовательном режимах при разном числе процессов и входных данных.

**Содержание работы**

Для начала мы создаем новый проект. Для этого переходим в Файл\Новый проект, выбираем «Приложение Qt Widgets», переходим на следующий этап нажатием кнопки «Далее». Теперь вводим название проекта (lab5) и путь к директории, в которой проект будет сохранен. Переходим на следующий этап и выбираем систему сборки qmake. Перейдя на новый этап, задаем имя класса, базовый класс, имя заголовочного файла и файла исходных текстов. Переходим на следующий этап и, если требуется инструментарий локализации Qt Linguist, указываем язык (в данном случае это не нужно). Перейдя на следующий этап, выбираем комплект Qt 5.15.2 MinGW нужной разрядности, после чего опять же переходим на следующий этап и нажимаем кнопку «Завершить».

Проект создан, теперь напишем код, реализующий программу по заданным условиям. Код программы представлен в листингах в ПРИЛОЖЕНИИ А.

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

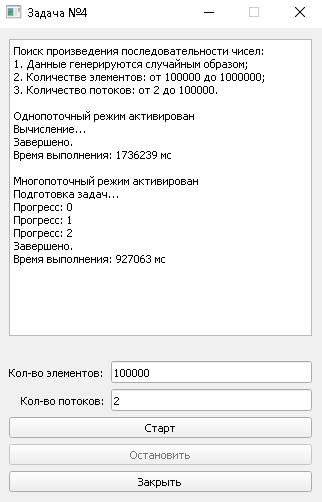


Рисунок 1 – Пример результата выполнения задачи

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 с задержкой

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Размер последовательности** | | |
| **Режим – количество потоков** | 100 000  элементов | 500 000  элементов | 1 000 000  элементов |
| Однопоточный – 1 | 1 736 239 мс | 8 508 655 мс | 16 847 137 мс |
| Многопоточный – 2 | 927 063 мс | 4 298 974 мс | 8 297 020 мс |
| Многопоточный – 10 | 845 630 мс | 4 204 554 мс | 8 290 085 мс |
| Многопоточный – 50 | 846 845 мс | 4 228 482 мс | 8 202 394 мс |

Произведем те же действия, но, отключив задержку в 1 мс для обоих режимов, имеющуюся в учебных целях для искусственного замедления вычислений. Результаты оформим в виде таблицы 2.

Таблица 2 – Результаты времени выполнения без задержки

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Размер последовательности** | | |
| **Режим – количество потоков** | 100 000  элементов | 500 000  элементов | 1 000 000  элементов |
| Однопоточный – 1 | 21 мс | 40 мс | 57 мс |
| Многопоточный – 2 | 12 мс | 16 мс | 23 мс |
| Многопоточный – 10 | 10 мс | 13 мс | 23 мс |
| Многопоточный – 50 | 11 мс | 15 мс | 20 мс |

**Результаты**

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

Можно также заметить, что слишком большое количество потоков понижают производительность вычислений, кроме тех случаев, когда выполняется работа с громоздким объемом данных. Это доказывается тем фактом, что во всех задачах в многопоточном режиме с любым объемом данных вычисления с 10 потоками было производительнее, чем с 50, кроме работы с последовательностью из 1 000 000 элементов. Таким образом, как видно из результатов, 2 потока являются наиболее оптимальным количеством для машины, на которой проводилось тестирование, так как он демонстрирует наибольший скачек производительности в сравнении с однопоточным режимом и не сильно отстает от 10 и 50 потоков.

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.

ПРИЛОЖЕНИЕ А

**Листинг 1 – Файл проекта main.cpp**

#include <QApplication>

#include <QStyleFactory>

#include "mywidget.h"

int main**(**int argc**,** char **\***argv**[])**

**{**

QApplication a**(**argc**,** argv**);**

qApp**->**setStyle**(**QStyleFactory**::**create**(**"Fusion"**));**

mywidget w**;**

w**.**setWindowTitle**(**"Задача №4"**);**

w**.**setFixedSize**(**325**,**475**);**

w**.**show**();**

**return** a**.**exec**();**

**}**

**Листинг 2 – Файл проекта mywidget.h**

#ifndef MYWIDGET\_H

#define MYWIDGET\_H

#include <QtWidgets>

struct Task **{**

int beginIndex**;**

int endIndex**;**

QList**<**qint64**>\*** listPtr**;**

**};**

class mywidget **:** public QWidget

**{**

Q\_OBJECT

public**:**

mywidget**(**QWidget **\***parent **=** 0**);**

signals**:**

public slots**:**

void clickStart**();**

void executionWithoutThread**(**int countElements**);**

void executionWithThread**(**int countElements**);**

private slots**:**

void progressValueChanged**(**int value**);**

void finished**();**

void clickClose**();**

void clickStop**();**

private**:**

QTextEdit **\***outputField**;**

QFutureWatcher**<**qint64**>** **\***watcher**;**

QElapsedTimer timer**;**

QPushButton **\***startBtn**;**

QPushButton **\***stopBtn**;**

QList**<**qint64**>** array**;**

QList**<**Task**>** tasks**;**

QFuture**<**qint64**>** future**;**

QLineEdit **\***elementsEditField**;**

QLineEdit **\***threadsEditField**;**

QLabel **\***messageLbl**;**

**};**

#endif // MYWIDGET\_H

**Листинг 3 – Файл проекта mywidget.cpp**

#include <QtConcurrent>

#include <QtWidgets>

#include "mywidget.h"

#include <cstdlib>

/\*\*

\* **@brief** perElementFunc - Функция вычисления на участве последовательности;

\* **@param** task - Задача;

\* **@return** result - Результат произведения.

\*/

qint64 perElementFunc**(**const Task task**)** **{**

qint64 result**;**

//На отрезке нити находим произведение элементов массива

**for** **(**int i**=**task**.**beginIndex**;** i**<=**task**.**endIndex**;** i**++)** **{**

result**\*=**task**.**listPtr**->**at**(**i**);**

QThread**::**msleep**(**1**);**

**}**

**return** result**;**

**}**

/\*\*

\* **@brief** reduce - Функция объединения результатов частичного произведения к общему.

\* **@param** multipl - Общее произведение.

\* **@param** semiMultipl - Частичное произведение.

\*/

void reduce**(**qint64 **&** multipl**,** const qint64 semiMultipl**)** **{**

multipl **\*=** semiMultipl**;**

**}**

/\*\*

\* **@brief** generateRandomNumber - Функция генерации случайных чисел;

\* **@param** minVal - Минимальное генерируемое число;

\* **@param** maxVal - Максимальное генерируемое число;

\* **@param** seed - Базовое число для генерации случайного;

\* **@return** - Случайное число в диапазоне.

\*/

int generateRandomNumber**(**int minVal**,** int maxVal**,** int seed**)** **{**

//В качестве базового числа используется системное время

qsrand**(**seed**);**

**return** **(**qrand**()** **%** **((**maxVal **+** 1**)** **-** minVal**)** **+** minVal**);**

**}**

/\*\*

\* **@brief** mywidget::mywidget - Конструирование виджета;

\* **@param** parent.

\*/

mywidget**::**mywidget**(**QWidget **\***parent**)** **:** QWidget**(**parent**)** **{**

//Инициализируем компоненты

outputField **=** **new** QTextEdit**();**

elementsEditField **=** **new** QLineEdit**();**

threadsEditField **=** **new** QLineEdit**();**

startBtn **=** **new** QPushButton**(**tr**(**"Старт"**));**

stopBtn **=** **new** QPushButton**(**tr**(**"Остановить"**));**

QPushButton **\***closeBtn **=** **new** QPushButton**(**tr**(**"Закрыть"**));**

messageLbl **=** **new** QLabel**();**

//Настраиваем компоненты

outputField**->**setReadOnly**(true);** //Поле вывода только для чтения

outputField**->**append**(**"Поиск произведения последовательности чисел:\n"

"1. Данные генерируются случайным образом;\n"

"2. Количестве элементов: от 100000 до 1000000;\n"

"3. Количество потоков: от 2 до 100000."**);**

messageLbl**->**setStyleSheet**(**"color: red"**);**

//Валидаторы для корректного ввода данных в диапазоне

elementsEditField**->**setValidator**(new** QIntValidator**(**100000**,** 1000000**,** **this));**

threadsEditField**->**setValidator**(new** QIntValidator**(**2**,**100000**,this));**

elementsEditField**->**setText**(**"100000"**);**

threadsEditField**->**setText**(**QString**::**number**(**QThread**::**idealThreadCount**()));**

stopBtn**->**setEnabled**(false);**

//Создаем и настраиваем разметку

QFormLayout **\***fieldsLayout **=** **new** QFormLayout**;**

fieldsLayout**->**addRow**(**tr**(**"Кол-во элементов:"**),** elementsEditField**);**

fieldsLayout**->**addRow**(**tr**(**" Кол-во потоков:"**),** threadsEditField**);**

QVBoxLayout **\***mainLayout **=** **new** QVBoxLayout**();**

mainLayout**->**addWidget**(**outputField**);**

mainLayout**->**addWidget**(**messageLbl**);**

mainLayout**->**addLayout**(**fieldsLayout**);**

mainLayout**->**addWidget**(**startBtn**);**

mainLayout**->**addWidget**(**stopBtn**);**

mainLayout**->**addWidget**(**closeBtn**);**

setLayout**(**mainLayout**);**

//Подключаем к кнопкам обработчики событий

connect**(**startBtn**,**SIGNAL**(**clicked**()),this,**SLOT**(**clickStart**()));**

connect**(**closeBtn**,**SIGNAL**(**clicked**()),this,**SLOT**(**clickClose**()));**

connect**(**stopBtn**,**SIGNAL**(**clicked**()),this,**SLOT**(**clickStop**()));**

**}**

/\*\*

\* **@brief** mywidget::clickStart - Событие нажатия на кнопку "Старт".

\*/

void mywidget**::**clickStart**()** **{**

/\*Проверяем поля с входными данными на корректный ввод

\*Если данные введены корректно, то...

\*/

**if** **(**elementsEditField**->**hasAcceptableInput**()** **&&** threadsEditField**->**hasAcceptableInput**()){**

//Настраиваем компоненты

messageLbl**->**setText**(**""**);**

elementsEditField**->**setReadOnly**(true);**

threadsEditField**->**setReadOnly**(true);**

startBtn**->**setEnabled**(false);**

//Берем количество элементов в последовательности

int countElements **=** elementsEditField**->**text**().**toInt**();**

//Заполняем массив случайными числами от 100 до 100000

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

array**.**append**(**generateRandomNumber**(**100**,**100000**,**QDateTime**::**currentMSecsSinceEpoch**()));**

**}**

//Запускаем однопоточный режим

executionWithoutThread**(**countElements**);**

//Запускаем многопоточный режим

executionWithThread**(**countElements**);**

//Если данные введены некорректно, то...

**}** **else** **{**

//Отображаем пользователю сообщение об ошибке

messageLbl**->**setText**(**"Ошибка!Входные данные выходят за границу диапазона."**);**

**}**

**}**

/\*\*

\* **@brief** mywidget::executionWithoutThread - Режим однопоточного вычисления;

\* **@param** countElements - Количество элементов в последовательности.

\*/

void mywidget**::**executionWithoutThread**(**int countElements**)** **{**

outputField**->**append**(**"\nОднопоточный режим активирован\nВычисление..."**);**

timer**.**start**();**

qint64 result **=** 1**;**

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

result**\*=**array**[**i**];**

//QThread::msleep(1);

**}**

outputField**->**append**(**tr**(**"Завершено.\nВремя выполнения: "**)+**

QString**::**number**(**timer**.**elapsed**())+**" мс"**);**

**}**

/\*\*

\* **@brief** mywidget::executionWithThread - Режим многопоточного вычисления;

\* **@param** countElements - Количество элементов в последовательности.

\*/

void mywidget**::**executionWithThread**(**int countElements**)** **{**

//Берем количество потоков и делим последовательность на участки

int countThreads **=** threadsEditField**->**text**().**toInt**();**

int perThread **=** countElements **/** countThreads**;**

timer**.**restart**();**

stopBtn**->**setEnabled**(true);**

outputField**->**append**(**"\nМногопоточный режим активирован\nПодготовка задач..."**);**

timer**.**start**();**

//Выделяем участки задачам

**for** **(**int i**=**0**;** i**<=**countThreads**-**1**;** i**++)** **{**

Task task**;**

task**.**beginIndex **=** i**\***perThread**;**

task**.**endIndex **=** **(**i**+**1**)\***perThread**-**1**;**

task**.**listPtr **=** **&**array**;**

tasks**.**append**(**task**);**

**}**

//Создаем обект-наблюдатель

watcher **=** **new** QFutureWatcher**<**qint64**>();**

//Соединяем сигналы со слотами, выполняющими их обработку

connect**(**watcher**,**SIGNAL**(**progressValueChanged**(**int**)),this,**SLOT**(**progressValueChanged**(**int**)));**

connect**(**watcher**,**SIGNAL**(**finished**()),this,**SLOT**(**finished**()));**

//Используем контейнер для результата вычислений

future **=** QtConcurrent**::**mappedReduced**(**tasks**,** perElementFunc**,** reduce**);**

watcher**->**setFuture**(**future**);**

**}**

/\*\*

\* **@brief** mywidget::progressValueChanged - Прогресс изменения состояния;

\* **@param** value - Значение состояния в многопоточном вычислении.

\*/

void mywidget**::**progressValueChanged**(**int value**)** **{**

outputField**->**append**(**tr**(**"Прогресс: "**)+**QString**::**number**(**value**));**

**}**

/\*\*

\* **@brief** mywidget::finished - Завершение многопоточного вычисления.

\*/

void mywidget**::**finished**()** **{**

outputField**->**append**(**tr**(**"Завершено.\nВремя выполнения: "**)+**

QString**::**number**(**timer**.**elapsed**())+**" мс"**);**

elementsEditField**->**setReadOnly**(false);**

threadsEditField**->**setReadOnly**(false);**

startBtn**->**setEnabled**(true);**

stopBtn**->**setEnabled**(false);**

**}**

/\*\*

\* **@brief** mywidget::clickClose - Событие нажатия на кнопку "Закрыть".

\*/

void mywidget**::**clickClose**()** **{**

watcher**->**cancel**();**

watcher**->**waitForFinished**();**

close**();**

**}**

/\*\*

\* **@brief** mywidget::clickStop - Событие нажатия на кнопку "Остановить".

\*/

void mywidget**::**clickStop**()** **{**

**if** **(**watcher**->**isFinished**())** **{** **return;** **}**

//если наблюдатель на паузе, возобновляем

**if** **(**watcher**->**isPaused**())** **{**

watcher**->**resume**();**

outputField**->**append**(**tr**(**"Возобновлено..."**));**

stopBtn**->**setText**(**tr**(**"Остановить"**));**

//иначе ставим на паузу

**}** **else** **{**

watcher**->**pause**();**

outputField**->**append**(**tr**(**"Остановлено..."**));**

stopBtn**->**setText**(**tr**(**"Возобновить"**));**

**}**

**}**

**Листинг 4 – Файл проекта lab5.pro**

QT **+=** core concurrent widgets

CONFIG **+=** c**++**11

TARGET **=** multiplication\_of\_numbers

CONFIG **+=** console

CONFIG **-=** app\_bundle

TEMPLATE **=** app

SOURCES **+=** main**.**cpp \

mywidget**.**cpp

DEFINES **+=** QT\_DEPRECATED\_WARNINGS

HEADERS **+=** \

mywidget**.**h