Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

Кафедра компьютерных систем и программных технологий

Отчет по лабораторной работе №1

По дисциплине «Параллельные вычисления»

«Разработка программ с использованием pthread и OpenMP в языке C++»

Работу выполнили студенты группы №13541/4 Родина В.В.

Работу принял преподаватель Стручков И. В.

Санкт-Петербург

2017

Цель работы

Научиться создавать программы с использованием многопоточных технологий. Познакомиться с работой библиотек pthread и OpenMP для языка C++. Проанализировать прирост производительности при использовании многопоточных библиотек.

Постановка задачи

Задача: Определить вероятность появления 3-грамм в тексте на русском языке.
Решить следующую задачу тремя способами:

1. Однопоточной программой.
2. Программой с использованием библиотеки pthread.
3. Программой с использованием библиотеки OpenMP.

Обзор задачи

Используемое оборудование:
Процессор Intel(R) Core(TM) i5-4210U CPU @ 1.70GHz, 2.39 GHz, ядер: 4, логических процессоров: 4 (*hyper-threading*)
N-грамма — последовательность из n элементов.
Вероятность появления n-граммы в тексте подсчитывается формулой:

$$F=n/L,$$

где n – число появлений целевой n-граммы, а L – число всех возможных n-грамм.

Однопоточная задача

Написанная однопоточная программа представлена в листинге 1.
Листинг 1. Однопоточная программа

```
#include <iostream>
#include <vector>
#include <string>
#include <fstream>
#include <sstream>
#include <cctype>
#include <ctime>
#include <unordered_map>

using namespace std;

string formatStr(string& str)
{
	string newStr = "";
	for (char& ch : str)
	{
		if (!iswalpha((unsigned char)ch))
			continue;
		ch = tolower(ch);
		newStr.push_back(ch);
	}
	return newStr;
}

vector<string>* formatFile(vector<string>* text)
{
	vector<string>* newVec = new vector<string>;
	for (string& str : *text)
	{
		str = formatStr(str);
```

```
                        if (strcmp(str.c_str(), "") != 0)
                                newVec->push_back(str);
        }
        delete text;
        return newVec;
}

vector<string>* readFileByWords(string fileName)
{
        vector <string>* vec = new vector < string > ;
        ifstream file(fileName);
        if (!file.is_open())
                return nullptr;
        string word;
        while (file >> word)
        {
                vec->push_back(word);
        }
        return vec;
}

unordered_map<string, int>* computeFrequencyNew(int n, vector <string>* text)
{
        if (!n || !text->size()) //Если хотя бы один вектор пуст
                return nullptr;
        string word;
        //int n = nGramm->size(); //n-gramm
        if (text->size() < n)
                return 0;
        unordered_map<string, int>* resMap = new unordered_map< string, int >;
        for (int i = 0; i < text->size()-n; i++)
        {
                string str = "";
                for (int j = 0; j < n; j++)
                {
                        str = str + " " + text->at(i + j);
                }
                unordered_map<string, int>::iterator it;
                it = resMap->find(str);
                if (it == resMap->end())
                        resMap->insert(pair<string, int>(str, 1));
                else
                        it->second++;
        }
        return resMap;
}

unordered_map<string, int>::iterator findeMaxValue(unordered_map<string, int>* resMap)
{
        int currentMax = 0;
        unordered_map<string, int>::iterator iter;
        for (auto it = resMap->begin(); it != resMap->end(); ++it)
        {
                if (it->second > currentMax)
                {
                        currentMax = it->second;
                        iter = it;
                }
        }
        return iter;
}

int main(int argc, char *argv[])
{
        setlocale(LC_ALL, "rus");
        cout << "Program was started" << endl;
        string fileName("C:\\Users\\lera\\Documents\\Visual Studio 2013\\Projects\\parallel\\ParallelSimple\\ParallelSimple\\Пикник на
обочине.txt"fileName(argv[0]);//("text.txt"); ("текст.txt");//
```

```
	if (argc >= 2)
	{
		fileName = argv[1];//("text.txt"); ("текст.txt");//
	}

	vector<string> *vecText = readFileByWords(fileName);
	if (vecText == nullptr)
	{
		cout << "File not found\n" << endl;
		delete vecText;
		return 1;
	}

	vecText = formatFile(vecText);
	unsigned int timeAfterReading = clock();
	unordered_map<string, int>* resMap = computeFrequencyNew(3, vecText);
	unordered_map<string, int>::iterator itMax = findeMaxValue(resMap);
	int n = itMax->second; //Числитель - число появлений конкретной nGramm-ы
	int L = resMap->size(); //Знаменатель - число всех возможных nGramm
	float frequency = ((float)n / (float)L) * 100;
	cout.setf(std::ios::fixed);
	cout << "nGramm: " << itMax->first << endl;
	cout << "Number of occurrences:" << n << endl;
	cout << "Number of all nGramm:" << L << endl;
	cout << "Frequency:" << frequency << "%" << endl;
	cout.precision(2); //2 - число символов после точки

	delete vecText;
	delete resMap;
	unsigned int timeEnd = clock();
	unsigned int workTime = (timeEnd - timeAfterReading);// / 1000; //в секундах
	cout << "runtime = " << timeEnd  << endl; // время работы программы
	cout << "runtime without reading = " << workTime << endl; // время работы программы
	system("pause");
	ofstream fout("workTime.txt", ios_base::app); // открываем файл для добавления информации к концу файла

	if (!fout.is_open()) // если файл не открыт
		cout << "Файл не может быть открыт!\n"; // сообщить об этом

	fout << workTime << endl;
	fout.close();
	return 0;
}
```

Обзор функций:

int main – главная функция.

Входные параметры:

string filename – имя файла исходного текста, в котором будет производиться поиск n-грамм.

Возвращаемое значение: 0 – если программа выполнена успешно и 1 при ошибке.

vector<string>* readFileByWords – функция позволяет считать файл и поместить его в вектор слов.

Входные параметры:

string fileName – имя входного файла

Возвращаемое значение:

Возвращает указатель на вектор, состоящий из слов исходного файла fileName.

vector<string>* formatFile – функция удаляет все лишние символы из вектора слов и оставляет только буквы.

Входные параметры:
vector<string>* text – указатель на вектор со словами.
Возвращаемое значение:
Возвращает вектор со словами без служебных символов.

string formatStr – функция удаляет все лишние символы из слова и оставляет только буквы.
Входные параметры:
string& str– ссылка на редактируемое слово.
Возвращаемое значение:
Возвращает отредактированное слово.

int computeFrequency – функция посчитывает количество появлений n-граммы в тексте.
Входные параметры:
vector<string>* nGramm – указатель на мектор слов n-граммы.
vector <string>* text – указатель на вектор слов текста.
Возвращаемое значение:
Возвращает 0 при выявлении ошибок и количество найденных n-грамм в случае успеха.

Выполнение программы:
Для тестирования программы использовалась книга братьев Стругацких «Пикник на обочине».

```
Program was started
nGramm:  на самом деле
Number of occurrences:10
Number of all nGramm:44740
Frequency:0.022351%
runtime = 1831
runtime without reading = 1589
Для продолжения нажмите любую клавишу . . .
```

Из вывода программы видно самую часто встречающуюся n-грамму, число нахождений n-граммы, общее число n-грамм, вероятность появления n-граммы, время выполнения программы и время работы алгоритма без учета времени на считывание файла.

Многопоточная программы с использованием библиотеки pthread

Код многопоточной программы с использованием библиотеки pthread представлен в листинге 2.
Листинг 2. Многопоточная программы с использованием библиотеки pthread.

```
#include <iostream>
#include <vector>
#include <string>
#include <fstream>
#include <sstream>
#include <thread>
#include <ctime>
#include <cctype>
#include <mutex>
#include <concurrent_unordered_map.h>
```

```
using namespace concurrency;
using namespace std;

mutex mapMutex;
string formatStr(string& str)
{
	string newStr = "";
	for (char& ch : str)
	{
		if (!iswalpha((unsigned char)ch))
			continue;
		ch = tolower(ch);
		newStr.push_back(ch);
	}
	return newStr;
}

vector<string>* formatFile(vector<string>* text)
{
	vector<string>* newVec = new vector<string>;
	for (string& str : *text)
	{
		str = formatStr(str);
		if (strcmp(str.c_str(), "") != 0)
			newVec->push_back(str);
	}
	delete text;
	return newVec;
}

vector<string>* readFileByWords(string fileName)
{
	vector <string>* vec = new vector < string >;
	ifstream file(fileName);
	if (!file.is_open())
		return nullptr;
	string word;
	while (file >> word)
	{
		vec->push_back(word);
	}
	return vec;
}

vector<string>* readStringByWord(string str)
{
	vector<string>* vec = new vector < string >;
	istringstream iss(str);
	string word;
	while (iss >> word)
	{
		vec->push_back(word);
	}

	return formatFile(vec);
}

void computeFrequencyNew(int n, vector <string>* text, int start, int end,
concurrent_unordered_map<string, int>* resMap)
{
	if (!n || !text->size()) //Если хотя бы один вектор пуст
		return;
	string word;
	if (text->size() < n)
		return;
	int endOfEnd;
	if (end == text->size() - 1)
		endOfEnd = end - n+1;
	else
		endOfEnd = end;
	for (int i = start; i < endOfEnd; i++)
	{
		string str = "";
		for (int j = 0; j < n; j++)
		{
			str = str + " " + text->at(i + j);
```

```
                }

                concurrent_unordered_map<string, int>::iterator it;
                //mapMutex.lock();
                it = resMap->find(str);
                if (it == resMap->end())
                        resMap->insert(pair<string, int>(str, 1));
                else
                        it->second++;
                //mapMutex.unlock();
        }
        return;
}

concurrent_unordered_map<string, int>::iterator findeMaxValue(concurrent_unordered_map<string, int>*
resMap)
{
        int currentMax = 0;
        concurrent_unordered_map<string, int>::iterator iter;
        for (auto it = resMap->begin(); it != resMap->end(); ++it)
        {
                if (it->second > currentMax)
                {
                        currentMax = it->second;
                        iter = it;
                }
        }
        return iter;
}

int main(int argc, char *argv[])
{
        setlocale(LC_ALL, "rus");
        std::cout << "Program was started" << endl;

        string fileName("C:\\Users\\lera\\Documents\\Visual Studio
2013\\Projects\\parallel\\ParallelSimple\\ParallelSimple\\Пикник на обочине.txt"text.txt");
//("текст.txt");//fileName(argv[0]);//
        int numberOfProcesses = 4;

        if (argc >= 3)
        {
                fileName = argv[1];//("text.txt"); ("текст.txt");//
                numberOfProcesses = atoi(argv[2]);
        }
        float frequency = 0;
        //int n = 0;

        vector<string> *vecText = readFileByWords(fileName);
        if (vecText == nullptr)
        {
                std::cout << "error in reading data" << endl;
                return 1;
        }

        vecText = formatFile(vecText);
        unsigned int timeAfterReading = clock();
        int grammSize = 3;
        int textSize = vecText->size();
        //Делим на процессы
        vector<thread*> vecThreads;
        concurrent_unordered_map<string, int>* resMap = new concurrent_unordered_map < string, int >;

        int step = textSize / numberOfProcesses; //Пример: 8-длина текста, 2- число проц. 8/2=4
        int start = 0;                                          //от 0
        int end = step - 1;                                     //до 3 и от 4 до 7

        textSize = vecText->size();

        step = textSize / numberOfProcesses; //Пример: 8-длина текста, 2- число проц. 8/2=4
        start = 0;                                              //от 0
        end = step - 1;                                 //до 3 и от 4 до 7

        for (int i = 1; i <= numberOfProcesses; ++i)
        {
                auto th = new std::thread(computeFrequencyNew, grammSize, vecText, start, end, resMap);
                vecThreads.push_back(th);
```

```
                start = end - grammSize + 1;

                //последний процесс захватывает все слова до конце
                if (i == numberOfProcesses)
                        end = textSize - 1;
                else
                        end = end + step;
        }

        for (auto &th : vecThreads)
        {
                th->join();
                delete th;
        }

        //}

        concurrent_unordered_map<string, int>::iterator itMax = findeMaxValue(resMap);

        int n = itMax->second; //Числитель - число появлений конкретной nGramm-ы

        int L = resMap->size(); //Знаменатель - число всех возможных nGramm


        frequency = ((float)n / (float)L) * 100;

        std::cout.setf(std::ios::fixed);
        std::cout << "nGramm: " << itMax->first << endl;
        std::cout << "Number of occurrences:" << n << endl;
        std::cout << "Number of all nGramm:" << L << endl;

        std::cout << "Frequency:" << frequency << endl;

        delete vecText;
        delete resMap;

        unsigned int timeEnd = clock();
        unsigned int workTime = (timeEnd - timeAfterReading);// / 1000; //в секундах
        cout << "runtime = " << timeEnd << endl; // время работы программы
        cout << "runtime without reading = " << workTime << endl; // время работы программы
        system("pause");

        ofstream fout("workTime.txt", ios_base::app); // открываем файл для добавления информации к
концу файла

        if (!fout.is_open()) // если файл не открыт
                cout << "Файл не может быть открыт!\n"; // сообщить об этом

        fout << workTime << endl;
        fout.close();

        return 0;
}
```

Программа по умолчанию использует 4 потока.

Для реализации многопоточности используется класс thread.

Класс thread предоставляет один поток выполнения. Потоки позволяют нескольким фрагментам кода работать асинхронно и одновременно.

Точка распараллеливания:

- Функция computeFrequency запускается в n потоках. В каждом потоке обрабатывается свой промежуток слов в исходном векторе.

Функции программы подобны функциям однопоточного приложения и выполняют те же задачи.

Функция computeFrequency изменена для того, чтобы ее можно было использовать при распараллеливании.

void **computeFrequencyNew**(vector <string>* text, int start, int end, concurrent_unordered_map<string, int>* resMap)

Добавленные входные параметры:
int start – индекс слова в векторе исходного текста, с которого необходимо начинать подсчет n-грамм.
int end - – индекс слова в векторе исходного текста, до которого необходимо вести подсчет n-грамм.
concurrent_unordered_map<string, int>* resMap - параметр, передаваемый для адресации результата выполнения функции. Необходим, т.к. класс thread использует функции, возвращающие void.
concurrent_unordered_map – контейнер предполагающий использование в многопоточных приложениях без добавления дополнительной синхронизации.

В программу добавлено принятия третьего аргумента – число потоков.
Для тестирования программы использовалась книга братьев Стругацких «Пикник на обочине».

```
Program was started
nGramm:  до сих пор
Number of occurrences:10
Number of all nGramm:44740
Frequency:0.022351
runtime = 852
runtime without reading = 602
Для продолжения нажмите любую клавишу . . .
```

Как видно из результата, выполнение программы по времени значительно превосходит однопоточное приложение.
1-602/1589= 0.62
Таким образом, многопоточная программа быстрее однопоточной на 62%.
Так же можно заметить, что найденная 3-грамма отличается от результата однопоточного приложения. Как оказалось, в тексте две самые популярные 3-граммы.

Многопоточная программа с использованием библиотеки OpenMP

Код многопоточной программы с использованием библиотеки OpenMP представлен в листинге 3.
Листинг 3. Многопоточная программы с использованием библиотеки OpenMP.

```
#include <iostream>
#include <vector>
#include <string>
#include <fstream>
#include <sstream>
#include <omp.h>
#include <ctime>
#include <Windows.h>
#include <map>
#include <concurrent_unordered_map.h>

using namespace concurrency;
using namespace std;

int numThread = 4;
using namespace std;

string formatStr(string& str)
```

```
{
	string newStr = "";
	for (char& ch : str)
	{
		if (!iswalpha((unsigned char)ch))
			continue;
		ch = tolower(ch);
		newStr.push_back(ch);
	}
	return newStr;
}

vector<string>* formatFile(vector<string>* text)
{
	vector<string>* newVec = new vector<string>;
	for (string& str : *text)
	{
		str = formatStr(str);
		if (strcmp(str.c_str(), "") != 0)
			newVec->push_back(str);
	}
	delete text;
	return newVec;
}

vector<string>* readFileByWords(string fileName)
{
	vector <string>* vec = new vector < string >;
	ifstream file(fileName);
	if (!file.is_open())
		return nullptr;
	string word;
	while (file >> word)
	{
		vec->push_back(word);
	}
	return vec;
}

concurrent_unordered_map<string, int>* computeFrequencyNew(int n, vector <string>* text)
{
	if (!n || !text->size()) //Если хотя бы один вектор пуст
		return nullptr;
	string word;
	if (text->size() < n)
		return 0;
	concurrent_unordered_map<string, int>* resMap = new concurrent_unordered_map< string, int >;
	omp_set_num_threads(numThread);
#pragma omp parallel for
	for (int i = 0; i < text->size() - n; i++)
	{
		string str = "";
		for (int j = 0; j < n; j++)
		{
			str = str + " " + text->at(i + j);
		}
		concurrent_unordered_map<string, int>::iterator it;
		it = resMap->find(str);
		if (it == resMap->end())
			resMap->insert(pair<string, int>(str, 1));
		else
			it->second++;
	}
	return resMap;
}

concurrent_unordered_map<string, int>::iterator findeMaxValue(concurrent_unordered_map<string, int>*
resMap)
{
	int currentMax = 0;
	concurrent_unordered_map<string, int>::iterator iter;
	for (auto it = resMap->begin(); it != resMap->end(); ++it)
	{
		if (it->second > currentMax)
		{
			currentMax = it->second;
			iter = it;
```

```
                }
        }
        return iter;
}

int main(int argc, char *argv[])
{
        setlocale(LC_ALL, "rus");
        cout << "Program was started" << endl;

        string fileName("D:\\University\\Master2\\Parallel\\C\\ParallelOpenMP\\ParallelOpenMP\\Пикник
на обочине.txt");//fileName(argv[0]);//("text.txt"); ("текст.txt");//("text.txt"); //
        if (argc >= 3)
        {
                fileName = argv[1];//("text.txt"); ("текст.txt");//
                numThread = atoi(argv[2]);
        }

        vector<string> *vecText = readFileByWords(fileName);
        if (vecText == nullptr)
        {
                cout << "File not found\n" << endl;
                delete vecText;
                return 1;
        }
        vecText = formatFile(vecText);
        unsigned int timeAfterReading = clock();
        concurrent_unordered_map<string, int>* resMap = computeFrequencyNew(3, vecText);
        concurrent_unordered_map<string, int>::iterator itMax = findeMaxValue(resMap);
        int n = itMax->second; //Числитель - число появлений конкретной nGramm-ы
        int L = resMap->size(); //Знаменатель - число всех возможных nGramm
        float frequency = ((float)n / (float)L) * 100;
        cout.setf(std::ios::fixed);
        cout << "nGramm: " << itMax->first << endl;
        cout << "Number of occurrences:" << n << endl;
        cout << "Number of all nGramm:" << L << endl;
        cout << "Frequency:" << frequency << "%" << endl;
        cout.precision(2); //2 - число символов после точки
        delete vecText;
        delete resMap;
        unsigned int timeEnd = clock();
        unsigned int workTime = (timeEnd - timeAfterReading);// / 1000; //в секундах
        cout << "runtime = " << timeEnd << endl; // время работы программы
        cout << "runtime without reading = " << workTime << endl; // время работы программы
        system("pause");
        ofstream fout("workTime.txt", ios_base::app); // открываем файл для добавления информации к
концу файла
        if (!fout.is_open()) // если файл не открыт
                cout << "Файл не может быть открыт!\n"; // сообщить об этом

        fout << workTime << endl;
        fout.close();
        return 0;
}
```

Программа построена на основе однопоточной программы и имеет малейшие изменения по сравнению с ней. В отличие от многопоточной программы с использованием pthread, данная программа минимально изменяет код исходной программы. В некоторых местах лишь добавлены директивы компилятора #pragma. Также в настройках проекта добавлена поддержка OpenMP.

Точка распараллеливания такая же как и в программе с библиотекой pthread.

Для тестирования программы использовалась книга братьев Стругацких «Пикник на обочине».

```
Program was started
nGramm:  до сих пор
Number of occurrences:10
Number of all nGramm:44740
```

```
Frequency:0.022351%
runtime = 943
runtime without reading = 725
Для продолжения нажмите любую клавишу . . .
```

Как и в случае с предыдущей программой разница в скорости выполнения сильно заметна.
1 - 725/1589= 0.54
Прирост производительности 54%.

Многократный запуск программ и подсчет вероятностных характеристик

Для подсчета вероятностных характеристик был создан скрипт, запускающий программу 100 раз и подсчитывающий временя ее исполнения. На основе полученных данных подсчитывается математическое ожидание, дисперсия и доверительных интервал.
Скрипт представлен в листинге 3
Листинг 3. Скрипт многократного запуска программы.

```
# -*- coding: cp1251 -*-

import sys
import subprocess
from math import sqrt

# arguments
args = list(sys.argv )

programm = "D:\University\Master2\Parallel\C\ParallelTread\Release\ParallelTread.exe"
inputFile = "D:\University\Master2\Parallel\C\ParallelTread\ParallelTread\Пикник на обочине.txt"
numRepeats = 100

if len(args) >= 5:
    programm = args [1]
    inputFile = args [2]
    nGramm = args[3]
    numRepeats = int ( args [4])
    #sys.exit (" Program parameters: programm  path | input file | nGramm | num repeats")


# program
PIPE = subprocess.PIPE

for threads in [1, 2, 4, 8]:#[4]:#
    timeList = []
    for num in range(numRepeats):
        p = subprocess.Popen([programm, inputFile, str(threads)], stdout=PIPE)
        for line in p.stdout:
            if 'runtime without reading = ' in line :
                timeList.append(int(line.split()[-1]))

    m=sum(timeList)/numRepeats
    disp = 0.00
    for val in timeList :
        disp = disp + (val - m) ** 2

    if numRepeats == 1:
        disp = disp / numRepeats
    else :
        disp = disp / ( numRepeats - 1)

    sigma = sqrt(disp)
```

```
t=1.984
interHigh = m + t*(sigma/(sqrt(numRepeats)))
interLow = m - t*(sigma/(sqrt(numRepeats)))
print("{} threads : average = {}, dispersion = {}, interval = [{}, {}]".format(threads , m , disp, interHigh, interLow))
```

На основе выводов данного скрипта была построена сводная таблица результатов для всех программ:

Табл.1. Сводная таблица результатов для 100 запусков каждой программы.

	Число потоков	Мат. Ожид.	Дисперси	Дов. Интер. 0.95%
Simple	1	55	22.54	[54.05- 55.94]
pThread	1	59	12.21	[58.30- 59.69]
	2	42	12.98	[41.28 - 42.71]
	4	40	25.20	[39.00 - 40.99]
	8	**39**	18.40	[38.14 - 39.85]
OpenMP	1	55	15.81	[54.21- 55.78]
	2	**38**	10.28	[37.36- 38.63]
	4	44	62.23	[42.43- 45.56]
	8	48	95.04	[46.06- 49.93]

Из таблицы 1 видно, что многопоточные приложения выигрывают по скорости выполнения у однопоточного. Так же видно, что программа с библиотекой pThread выигрывает у программы с использованием библиотеки OpenMP. Увеличение числа потоков до 8 не дало значительного прироста, т.к. процессор используемой системы имеет 4 логических потока.

Как не странно оптимальное количество потоков для программы с использованием библиотеки OpenMP оказалось равно 2. Однако для 4-ех потоков видно возрастание дисперсии, что говорит о том, что в данной конфигурации приложение срабатывает не всегда одинаково.

Данные результаты так же, зависят от загруженности системы в конкретный момент времени, что может серьезно влиять на скорость выполнения программ.

Выводы

В ходе работы создано три программы на языке C++ для решения задачи поиска n-грамм в тексте. Предложенные решения позволяют решать задачу с любым количеством слов в n-грамме (т.е., менять n). В ходе работы изучены и использованы библиотеки многопоточного программирования pThread и OpenMP. Реализованные на их основе решения могут легко изменять количество исполняющих потоков. Полученные решения протестированы на большом входном файле. Для одного набора входных данных проведен анализ вероятностных характеристик (табл.1). В результате экспериментов установлено, что наиболее эффективность использования библиотеки pThread эквивалента эффективности использования библиотеки OpenMP. Данный результат является логичным, т.к., в отличие от библиотеки MPI, которая предполагает распараллеливание между несколькими машинами, библиотека OpenMP рассчитана на выполнение программы на одном компьютере.

Использование библиотеки OpenMP удобно, т.к. код и логика однопоточного приложения почти никак не изменяются. Непосредственно в данном случае для распараллеливания было добавлено 3-6 строк кода и изменена настройка компиляции.