Федеральное государственное бюджетное образовательное учреждение высшего образования
«Сибирский государственный университет телекоммуникаций и информатики»
(СибГУТИ)

09.03.01 Информатика и вычислительная техника
код и наименование направления подготовки

ОТЧЕТ
по учебной практике

по направлению 09.03.01 «Информатика и вычислительная техника»,
направленность (профиль) – «Электронно-вычислительные машины, комплексы, системы и сети», квалификация – бакалавр,
программа академического бакалавриата,
форма обучения – очная, год начала подготовки (по учебному плану) – 2018

Выполнил:
студент гр. ИВ-823
«11» июля 2020 г. _____ /Шиндель Э.Д./

Оценка «_____»

Руководитель практики
от университета
преподаватель кафедры ВС
«11» июля 2020 г. _____ /Ткачёва Т.А./

Новосибирск 2020

ПЛАН-ГРАФИК ПРОВЕДЕНИЯ УЧЕБНОЙ ПРАКТИКИ

Тип практики: учебная практика по получению первичных профессиональных умений и навыков, в том числе получение умений и навыков научно-исследовательской деятельности
Способ проведения практики: стационарная
Форма проведения практики: дискретно по периодам проведения практики

Тема: реализация параллельного алгоритма ранжирования PageRank.

Содержание практики

Наименование видов деятельности	Дата (начало – окончание)
Общее ознакомление со структурным подразделением СибГУТИ, вводный инструктаж по технике безопасности	03.02.20-15.02.20
Выдача задания на практику, определение конкретной индивидуальной темы, формирование плана работ	27.02.20-7.03.20
Работа с библиотечными фондами Кафедры ВС, сбор и анализ материалов по теме практики	09.03.20-28.03.20
Выполнение работ в соответствии с составленным планом - Реализация последовательного алгоритма PageRank - Распараллеливание кода - Экспериментальные исследования эффективности кода	06.04.20-06.06.20
Анализ полученных результатов и произведенной работы, составление отчета по практике	06.07.20-11.07.20

Согласовано:

Руководитель практики
от университета
преподаватель кафедры ВС _____ /Ткачёва Т.А./

ЗАДАНИЕ НА ПРАКТИКУ

Необходимо реализовать алгоритм ранжирования PageRank и распараллелить его, для дальнейшего исследования скорости выполнения программы, в зависимости от различных входных данных и количества потоков.

ВВЕДЕНИЕ

PageRank - это математическая формула, которая определяет «ценность» страницы, опираясь на количество и качество других ссылающихся на неё страниц. Цель PageRank - выяснить относительную значимость той или иной страницы в сети.

Сооснователи Google Сергей Брин и Ларри Пейдж разработали PageRank в 1997 в рамках исследовательского проекта во время учёбы в Стэнфордском университете. Это подводит нас к одному очень важному выводу: поисковые системы не всегда были такими же эффективными, как Google сегодня. Ранние поисковые системы, такие как Yahoo и Altavista, работали не очень хорошо - релевантность результатов поиска оставляла желать лучшего. PageRank стремился решить эту проблему, используя (ссылочный) граф цитирования, который сооснователи охарактеризовали как «важный ресурс, который в значительной степени остался неиспользованным в существующих поисковых системах Интернета.»

Идея основывалась на том же принципе, что и метод оценки «значимости» той или иной научной работы, т.е. опираясь на количество других источников, которые на неё ссылаются. Сергей и Ларри взяли эту концепцию за основу и применили её в Интернете, отслеживая ссылки между веб-страницами.

Эта идея сработала настолько хорошо, что стала основой для создания поисковой системы, которую теперь мы знаем как Google. И она до сих пор работает.

1. PageRank

PageRank - это числовая величина, характеризующая «важность» веб-страницы. Чем больше ссылок на страницу, тем она «важнее». Кроме того, «вес» страницы A определяется весом ссылки, передаваемой страницей B. Таким образом, PageRank - это метод вычисления веса страницы путём подсчёта важности ссылок на неё.

Предположим, что на страницу A указывают страницы T1…Tn. Параметр d - коэффициент затухания, который может быть установлен в диапазоне от 0 до 1. Обычно его устанавливают d на отметке 0.85. Также C(A) - количество исходящих ссылок со страницы A. PageRank страницы A выглядит следующим образом:

PR(A) = (1-d) + d (PR(T1)/C(T1) + … + PR(Tn)/C(Tn)).

Обратите внимание, что показатели PageRank формируют распределение вероятностей по веб-страницам, поэтому сумма всех показателей PageRank всех веб-страниц будет равна единице.

Проще говоря, при расчёте PageRank какой-либо веб-страницы учитываются три следующих фактора:

- Количество и качество входящих ссылок;
- Количество исходящих ссылок на каждой такой странице;
- PageRank каждой страницы.

Например, у страницы C есть две ссылки: одна со страницы A и одна со страницы B. Страница A сильнее страницы B и содержит меньше исходящих ссылок. Введите эту информацию в алгоритм PageRank, и вы получите PageRank страницы C.

В формуле PageRank также есть так называемый «коэффициент затухания», который имитирует вероятность того, что случайный пользователь будет продолжать переходить по ссылкам во время просмотра веб-страниц. С каждым последующим кликом он уменьшается.

Иначе, вероятность того, что вы нажмёте на ссылку на первой странице, которую посещаете, достаточно высока. Однако, вероятность того, что вы нажмёте на ссылку на следующей странице, уже немного ниже. И так далее.

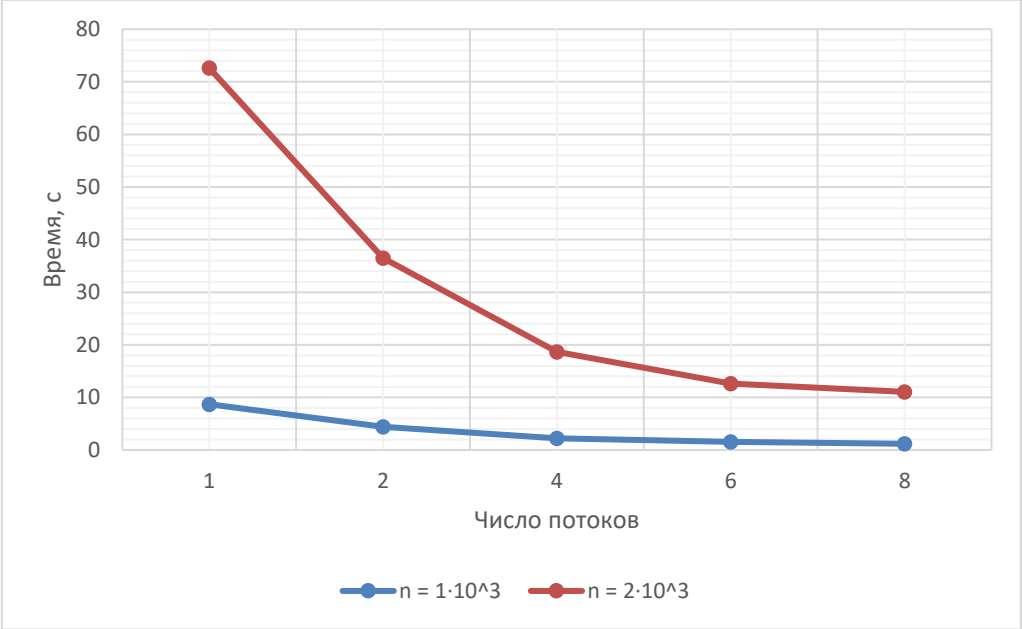
Общий «голос» умножается на «коэффициент затухания» (обычно равный 0.85) на каждой итерации пересчёта PageRank.

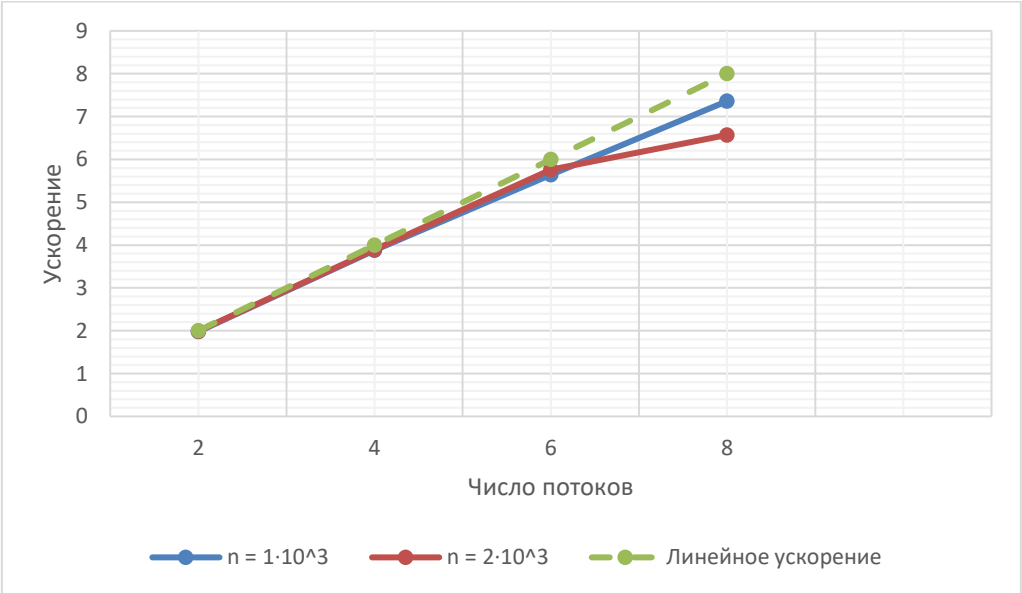
Например, если сайт «BBC» ссылается на страницу через четыре «ссылочных прокладки», то значение этой ссылки будет затухать до такой степени, что для последней страницы едва ли будет какая-то польза. Но если они будут ссылаться на ту же страницу только через две прокладки, то эта ссылка будет иметь более сильное влияние.

2. Экспериментальные исследования

В экспериментальной части было проведено два исследования: 1) зависимость времени работы алгоритма ранжирования от числа потоков при различных входных значениях; 2) зависимость коэффициента ускорения от числа потоков также при различных входных значениях.

n	Количество потоков								
	1	2		4		6		8	
	Время	Время	Ускорение	Время	Ускорение	Время	Ускорение	Время	Ускорение
$1 \cdot 10^3$	8.69	4.39	1.98	2.24	3.88	1.54	5.65	1.18	7.36
$2 \cdot 10^3$	72.61	36.49	1.99	18.66	3.89	12.6	5.76	11.05	6.57





ЗАКЛЮЧЕНИЕ

В результате учебной практики разработана и исследована параллельная версия алгоритма ранжирования PageRank. А также были проведены эксперименты, доказывающие его эффективность, по сравнению с последовательной версией программы.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Ахо А. В., Хопкрофт Д., Ульман Д. Д. Структуры данных и алгоритмы. – М.: Вильямс, 2001. – 384 с.

2. Миллер Р. Последовательные и параллельные алгоритмы: общий подход. – М.: БИНОМ, 2006. – 406 с.

3. Курносов М.Г., Берлизов Д.М. Алгоритмы и структуры обработки информации. – Новосибирск: Параллель, 2019. – 211 с.

4. The Anatomy of a Large-Scale Hypertextual Web Search Engine: [сайт] URL: http://infolab.stanford.edu/~backrub/google.html

5. ahrefsblog: [сайт] URL: https://ahrefs.com/blog/ru/google-pagerank/

ПРИЛОЖЕНИЯ

1 Исходный код main.c

```
#include <time.h>
#include "matrix.h"
#include "pagerank.h"

int main()
{
    srand(time(NULL));
    int n = 1000;

    double t = omp_get_wtime();
    page_rank(n);
    t = omp_get_wtime() - t;

    printf("Time: %.6f\n", t);

    return 0;
}
```

2 Исходный код pagerank.c

```
include "pagerank.h"

int calculate_links(matrix* m, int row)
{
    int counter = 0;
    for (int i = 0; i < m->size; ++i)
    {
        counter += m->elements[i][row];
    }
    if(counter == 0)
    {
        return m->size;
    }

    return counter;
}

double calculate_probability(matrix* m, int i, int j)
{
    double d = 0.85;
    int r = calculate_links(m, i);

    float a = (1 - d) + d * (m->elements[i][j] / r);

    return a;
}

void gen_web_matrix(matrix* m)
```

```
    {
        for (int i = 0; i < m->size; ++i)
        {
            for(int j = 0; j < m->size; ++j)
            {
                m->elements[i][j] = rand() % 2;
            }
        }
    }

    void gen_google_matrix(matrix* a, matrix* m)
    {
        int i = 0;
        int j = 0;
        int size = a->size;

        #pragma omp parallel for default(none) shared(a, m, size)
private(i, j)
        for (i = 0; i < size; ++i)
        {
            for(j = 0; j < size; ++j)
            {
                a->elements[i][j] =
                                    calculate_probability(m, i, j);
            }
        }
    }

    void matrix_solve(vector* v, matrix* m)
    {
        int size = m->size;

        vector tmp_vec;
        vector_init(&tmp_vec, size);

        for (int i = 0; i < size; ++i)
        {
            v->elements[i] = 1.0;
            tmp_vec.elements[i] = 0.0;
        }

        for(int x = 0; x < 3; ++x)
        {
            matrix_multiply(&tmp_vec, m, v);

            vector_normalize(&tmp_vec);

            for (int i = 0; i < size; ++i)
            {
                v->elements[i] = tmp_vec.elements[i];
                tmp_vec.elements[i] = 0.0;
            }
```

```
    }
    vector_free(&tmp_vec);
}

void page_rank(int size)
{
    matrix w;
    matrix_init(&w, size);
    gen_web_matrix(&w);

    matrix g;
    matrix_init(&g, size);
    gen_google_matrix(&g, &w);
    matrix_free(&w);

    vector p;
    vector_init(&p, size);
    matrix_transpose(&g);
    matrix_solve(&p, &g);
    vector_sort(&p);

    matrix_free(&g);
    vector_free(&p);
}
```

3 Исходный код pagerank.h

```
#ifndef PAGERANK_H
#define PAGERANK_H

#include "matrix.h"
#include "vector.h"
#include <omp.h>

int calculate_links(matrix* m, int row);
double calculate_probability(matrix* m, int i, int j);
void gen_web_matrix(matrix* m);
void gen_google_matrix(matrix* a, matrix* m);
void matrix_solve(vector* v, matrix* m);
void page_rank(int size);

#endif
```

4 Исходный код matrix.c

```
#include "matrix.h"

void matrix_init(matrix* m, int size)
{
    m->size = size;
    m->elements = (double**)malloc(sizeof(double*) * size);

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

```
        {
            m->elements[i] = malloc(sizeof(double) * size);
        }
    }

    void matrix_free(matrix* m)
    {
        for (int i = 0; i < m->size; ++i)
        {
            free(m->elements[i]);
        }

        free(m->elements);
        m->elements = NULL;
        m->size = 0;
        m = NULL;
    }

    void matrix_transpose(matrix* m)
    {
        double tmp = 0.0;

        for (int i = 0; i < m->size; ++i)
        {
            for (int j = i+1; j < m->size; ++j)
            {
                tmp = m->elements[i][j];
                m->elements[i][j] = m->elements[j][i];
                m->elements[j][i] = tmp;
            }
        }
    }

    void matrix_multiply(vector* tmp, matrix* m, vector* v)
    {
        int size = tmp->size;

        for (int i = 0; i < size; ++i)
        {
            for (int j = 0; j < size; ++j)
            {
                tmp->elements[i] += m->elements[i][j]
                                            * v->elements[j];
            }
        }
    }
```

5 Исходный код matrix.h

```
    #ifndef MATRIX_H
    #define MATRIX_H
```

```
#include <malloc.h>
#include <stdio.h>
#include <stdlib.h>
#include "vector.h"

typedef struct matrix
{
    double** elements;
    int size;
} matrix;

void matrix_init(matrix* m, int size);
void matrix_free(matrix* m);
void matrix_transpose(matrix* m);
void matrix_multiply(vector* tmp, matrix* m, vector* v);

#endif
```

6 Исходный код vector.c

```
#include "vector.h"

void vector_init(vector* v, int size)
{
    v->size = size;
    v->elements = (double*)malloc(sizeof(double) * v->size);
}

void vector_free(vector* v)
{
    free(v->elements);
    v->elements = NULL;
    v->size = 0;
    v = NULL;
}

void vector_normalize(vector* v)
{
    double sum = 0.0;
    for (int i = 0; i < v->size; ++i)
    {
    sum += v->elements[i];
    }
    for (int i = 0; i < v->size; ++i)
    {
    v->elements[i] /= sum;
    }
}

void vector_sort(vector* v)
{
    qsort(v->elements, v->size,
                          sizeof(double), double_compare);
```

```
}

int double_compare(const void* a, const void* b)
{
    double* arg1 = (double*) a;
    double* arg2 = (double*) b;

    if(*arg1 < *arg2) return -1;
    else if (*arg1 == *arg2) return 0;
    else return 1;
}
```

7 Исходный код vector.h

```
#ifndef VECTOR_H
#define VECTOR_H

#include <stdlib.h>
#include <stdio.h>

typedef struct vector
{
    double* elements;
    int size;
} vector;

void vector_init(vector* v, int size);
void vector_free(vector* v);
void vector_normalize(vector* v);
void vector_sort(vector* v);
int double_compare(const void* a, const void* b);

#endif
```

8 Исходный код Makefile

```
pagerank: *.c
    gcc -fopenmp -o pagerank *.c

clean:
    rm pagerank

run:
    ./pagerank
```