Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

**Высшая школа искусственного интеллекта**

**Лабораторная работа 4**

**Синхронизация доступа к динамическим спискам**

по дисциплине «Параллельное программирование»

Выполнил

студент гр.3530203/80102 Л.Д. Челищева

<*подпись*>

Руководитель К.А. Туральчук

<*подпись*>

«\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2021 г.

Санкт-Петербург

2021

# Описание условий тестирования

В ходе данной работы выполняется вариант задания 7.

В качестве файлов-источников выступают 20 отрывков из книги Гюго «Отверженные» весом от 10 до 22 Кб каждый (в некоторых случаях куски текста были перенесены из одного файла в другой для более равномерного распределения), всего 9187 уникальных слов на русском и французском. Файлы расположены в папке files, расположенной в одной папке с программой. Файлы можно без ошибок в программе заменить на любые другие, желательно числом не меньше потоков.

В методах ниже рассматривается подсчет частот появления слов в тексте. Результат работы каждого отдельного метода можно увидеть в переменных SerialResult, LocalResult, GlobalResult и DecResult.

Метод проверки результатов представляет собой сравнения результирующего словаря для параллельных методов со словарем из последовательного метода (словарь параллельного метода передается через входной аргумент). Проверка происходит через вычисление абсолютной разницы между частотами для одного слова. Если у параллельного слово не найдено или разница больше заданного отклонения Е – счетчик ошибок увеличивается.

Перевод из абсолютных частот в относительные в результирующем словаре всегда выполняется последовательно.

В задаче 3 (декомпозиция по задачам) всегда рассматривается 2 читателя.

# Результаты тестирования

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Число потоков | Последо-вательный, Время, мс | Декомпозиция по файлам с локальными буферами | | Декомпозиция по файлам с глобальным буфером | | Декомпозиция по задачам | |
| Время, мс | Отклонений | Время, мс | Отклонений | Время, мс | Отклонений |
| 4 | 531,7726 | 72,8331 | 0 | 75,7181 | 0 | 124,8644 | 0 |
| 8 | 86,4327 | 0 | 91,9868 | 0 | 813,9585 | 0 |
| 3 | 59,8576 | 0 | 86,8703 | 0 | 85,0218 | 0 |
| 5 | 89,2208 | 0 | 80,0835 | 0 | 162,3769 | 0 |

# Полный код программы

using System;

using System.Collections.Generic;

using System.Linq;

using System.IO;

using System.Text.RegularExpressions;

using System.Threading;

using System.Collections.Concurrent;

using System.Diagnostics;

namespace lab4

{

class Program

{

static int repeat = 10; //Число повторов

static int threads = 8; //Всего потоков

static string[] files = Directory.GetFiles("files"); //Директория с файлами

//Последовательный

static Dictionary<string, double> SerialResult;

//Параллельный - Локальные буферы

static Dictionary<string, int>[] localwordcount;

static Thread[] LocalThreads;

static Dictionary<string, double> LocalResult;

static double lwords;

//Параллельный - Глобальный буфер

static ConcurrentDictionary<string, int> globakwordcount;

static Thread[] GlobalThreads;

static Dictionary<string, double> GlobalResult;

static double gwords;

//Параллельный - Декомпозиция по задачам

static ConcurrentBag<string> dectextslist;

static int rthreads = 2; // Число читателей

static Thread[] Readers;

static int sthreads = threads - rthreads; //Число статистиков

static Thread[] Statisticians;

static ConcurrentDictionary<string, int> decwordcount;

static Dictionary<string, double> DecResult;

static double dwords;

static bool stop;

static void Main(string[] args)

{

int V = ('L' + 'C') % 8;

//Вариант 7

Console.WriteLine("Вариант {0}", V);

Stopwatch stopWatch = new Stopwatch();

double stime = 0;

double ltime = 0;

double gtime = 0;

double dtime = 0;

double ldiff = 0;

double gdiff = 0;

double ddiff = 0;

for (int rep = 0; rep < repeat+1; rep++)

{

//Последовательный

stopWatch.Start();

SerialResult = Serial();

stopWatch.Stop();

if (rep > 0)

stime += stopWatch.Elapsed.TotalMilliseconds;

stopWatch.Reset();

//Параллельный - Локальные буферы

stopWatch.Start();

localwordcount = new Dictionary<string, int>[threads];

LocalThreads = new Thread[threads];

for (int i = 0; i < threads; i++)

LocalThreads[i] = new Thread(ParallelLocal);

for (int i = 0; i < threads; i++)

LocalThreads[i].Start(i);

for (int i = 0; i < threads; i++)

LocalThreads[i].Join();

LocalResult = new Dictionary<string, double>();

lwords = 0;

foreach (Dictionary<string, int> lwc in localwordcount)

lwords += lwc.Values.Sum();

foreach (Dictionary<string, int> lwc in localwordcount)

foreach (string k in lwc.Keys)

{

if (LocalResult.ContainsKey(k))

LocalResult[k] += lwc[k] / lwords;

else

LocalResult.Add(k, lwc[k] / lwords);

}

stopWatch.Stop();

if(rep>0)

ltime += stopWatch.Elapsed.TotalMilliseconds;

stopWatch.Reset();

if (rep > 0)

ldiff += CompareToSerial(LocalResult);

//Параллельный - Глобальный буфер

stopWatch.Start();

globakwordcount = new ConcurrentDictionary<string, int>();

GlobalThreads = new Thread[threads];

for (int i = 0; i < threads; i++)

GlobalThreads[i] = new Thread(ParallelGlobal);

for (int i = 0; i < threads; i++)

GlobalThreads[i].Start(i);

for (int i = 0; i < threads; i++)

GlobalThreads[i].Join();

GlobalResult = new Dictionary<string, double>();

gwords = globakwordcount.Values.Sum();

foreach (string k in globakwordcount.Keys)

{

if (GlobalResult.ContainsKey(k))

GlobalResult[k] += globakwordcount[k] / gwords;

else

GlobalResult.Add(k, globakwordcount[k] / gwords);

}

stopWatch.Stop();

if (rep > 0)

gtime += stopWatch.Elapsed.TotalMilliseconds;

stopWatch.Reset();

if (rep > 0)

gdiff += CompareToSerial(GlobalResult);

//Параллельный - Декомпозиция по задачам

stopWatch.Start();

decwordcount = new ConcurrentDictionary<string, int>();

dectextslist = new ConcurrentBag<string>();

Readers = new Thread[rthreads];

Statisticians = new Thread[sthreads];

stop = false;

for (int i = 0; i < rthreads; i++)

Readers[i] = new Thread(ParallelReaders);

for (int i = 0; i < sthreads; i++)

Statisticians[i] = new Thread(ParallelStatisticians);

for (int i = 0; i < rthreads; i++)

Readers[i].Start(i);

for (int i = 0; i < sthreads; i++)

Statisticians[i].Start();

for (int i = 0; i < rthreads; i++)

Readers[i].Join();

stop = true;

for (int i = 0; i < sthreads; i++)

Statisticians[i].Join();

DecResult = new Dictionary<string, double>();

dwords = decwordcount.Values.Sum();

foreach (string k in decwordcount.Keys)

{

if (DecResult.ContainsKey(k))

DecResult[k] += decwordcount[k] / dwords;

else

DecResult.Add(k, decwordcount[k] / dwords);

}

stopWatch.Stop();

if (rep > 0)

dtime += stopWatch.Elapsed.TotalMilliseconds;

stopWatch.Reset();

if (rep > 0)

ddiff += CompareToSerial(DecResult);

}

Console.WriteLine("На {0} тестов, {1} потоков:", repeat, threads);

Console.WriteLine("Среднее время работы последовательного алгоритма: {0}", stime);

Console.WriteLine("Среднее время работы декомпозиции по файлам с локальными буферами: {0}, отклонений от последовательного: {1}", ltime / repeat,ldiff / repeat);

Console.WriteLine("Среднее время работы декомпозиции по файлам с глобальным буфером: {0}, отклонений от последовательного: {1}", gtime / repeat, gdiff / repeat);

Console.WriteLine("Среднее время работы декомпозиции по задачам ({2} потока-читателя): {0}, отклонений от последовательного: {1}", dtime / repeat, ddiff / repeat, rthreads);

Console.ReadKey();

}

static Dictionary<string, double> Serial()

{

string file;

string[] words;

Dictionary<string, int> wordcount = new Dictionary<string, int>();

Dictionary<string, double> wordfreq = new Dictionary<string, double>();

foreach (string f in files)

{

file = File.ReadAllText(f);

file = ClearText(file);

words = file.Split(' ');

foreach (string w in words)

{

if (wordcount.ContainsKey(w))

wordcount[w] += 1;

else

wordcount.Add(w, 1);

}

}

double allword = wordcount.Values.Sum();

foreach (string wck in wordcount.Keys)

{

wordfreq.Add(wck, (double)wordcount[wck]/allword);

}

return wordfreq;

}

//Параллельный - Локальные буферы

static void ParallelLocal(object o)

{

int num = (int)o;

int start = num \* files.Length / threads;

int fin = (num + 1) \* files.Length / threads > files.Length? files.Length: (num + 1) \* files.Length / threads;

string file;

string[] words;

localwordcount[num] = new Dictionary<string, int>();

for (int i = start; i<fin; i++)

{

file = ClearText(File.ReadAllText(files[i]));

words = file.Split(' ');

foreach (string w in words)

{

if (localwordcount[num].ContainsKey(w))

localwordcount[num][w] += 1;

else

localwordcount[num].Add(w, 1);

}

}

}

//Параллельный - Глобальный буфер

static void ParallelGlobal(object o)

{

int num = (int)o;

int start = num \* files.Length / threads;

int fin = (num + 1) \* files.Length / threads > files.Length ? files.Length : (num + 1) \* files.Length / threads;

string file;

string[] words;

for (int i = start; i < fin; i++)

{

file = ClearText(File.ReadAllText(files[i]));

words = file.Split(' ');

foreach (string w in words)

globakwordcount.AddOrUpdate(w, 1, (itemKey, itemValue) => itemValue + 1);

}

}

//Параллельный - Декомпозиция по задачам

//Читатели

static void ParallelReaders(object o)

{

int num = (int)o;

int start = num \* files.Length / rthreads;

int fin = (num + 1) \* files.Length / rthreads > files.Length ? files.Length : (num + 1) \* files.Length / rthreads;

string file;

for (int i = start; i < fin; i++)

{

file =File.ReadAllText(files[i]);

dectextslist.Add(file);

}

}

static void ParallelStatisticians()

{

string text;

string[] words;

while (!stop||!dectextslist.IsEmpty)

{

if(dectextslist.TryTake(out text))

{

if (!string.IsNullOrEmpty(text))

{

words = ClearText(text).Split(' ');

foreach(string w in words)

decwordcount.AddOrUpdate(w, 1, (itemKey, itemValue) => itemValue + 1);

text = string.Empty;

}

}

}

}

static string ClearText(string text)

{

string result = Regex.Replace(text.ToLower(), "[^а-яa-z ]", "");

while(result.Contains(" ")) result = result.Replace(" ", " ");

return result;

}

static int CompareToSerial(Dictionary<string, double> dict)

{

double e = 0.00001;

int result = 0;

foreach(string k in SerialResult.Keys)

{

try

{

if (Math.Abs(dict[k] - SerialResult[k]) > e)

result++;

}

catch

{

result++;

}

}

return result;

}

}

}

# Выводы

В ходе лабораторной работы были рассмотрены способы распараллеливания обработки текстовых файлов с целью получения статистики, а также варианты синхронизации доступа к динамическим коллекциям (в частности использование потокбезопасных коллекций ConcurrentBag. и ConcurrentDictionary).

На основе проведенного тестирования можно отметить, что методы с локальными буферами и одним глобальным буфером в большинстве ситуаций показали себя примерно одинаково хорошо, хотя более эффективным все же оказался метод с локальными буферами, не требующий синхронизации доступа к коллекциям.

Метод декомпозиции по задачам, в связи с крайней неравномерностью распределения задач между потоками (потоки-читатели только считывали свои файлы целиком и затем простаивали) показал себя наименее эффективным, но все же в 3 случаях из 4 (кроме случая, когда потоков было достаточно много, 8) более эффективным чем последовательное выполнение. Однако в ситуации когда существовало 2 потока-читателя и 1 поток-писатель метод даже показал себя эффективнее чем декомпозиция по файлам с глобальным буфером.