Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

**Высшая школа искусственного интеллекта**

**Лабораторная работа 2**

**Поиск простых чисел**

по дисциплине «Параллельное программирование»

Выполнил

студент гр.3530203/80102 Л.Д. Челищева

<*подпись*>

Руководитель К.А. Туральчук

<*подпись*>

«\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2021 г.

Санкт-Петербург

2021

# Общие условия

Система содержит 4 физических и 4 же логических ядра.

n – число элементов

m – число потоков

bitmap – карта непростых элементов

simple – список простых элементов

В таблицах – время в мс.

# Последовательный алгоритм

Для числа n:

|  |  |  |
| --- | --- | --- |
| 10 | 1000 | 100000 |
| 0,51922 | 1,42783 | 14,96345 |

# Параллельный алгоритм №1: декомпозиция по данным

|  |  |  |  |
| --- | --- | --- | --- |
| m n | 10 | 1000 | 100000 |
| 2 | 31,15472 | 22,39718 | 40,58419 |
| 4 | 58,53644 | 63,43213 | 59,67482 |
| 8 | 115,00706 | 86,74757 | 125,77571 |

# Параллельный алгоритм №2: декомпозиция набора простых чисел

|  |  |  |  |
| --- | --- | --- | --- |
| m n | 10 | 1000 | 100000 |
| 2 | 33,24408 | 31,45353 | 156,65394 |
| 4 | 51,23845 | 56,81977 | 139,53423 |
| 8 | 115,17304 | 122,33724 | 233,62393 |

# Параллельный алгоритм №3: применение пула потоков

|  |  |  |
| --- | --- | --- |
| 10 | 1000 | 100000 |
| 1,11693 | 2,22862 | 36,08664 |

# Параллельный алгоритм №4: последовательный перебор простых чисел

Применяется критическая секция.

|  |  |  |  |
| --- | --- | --- | --- |
| m n | 10 | 1000 | 100000 |
| 2 | 26,78489 | 26,34492 | 85,85137 |
| 4 | 46,37177 | 48,84147 | 88,85747 |
| 8 | 139,05758 | 142,47078 | 163,6824 |

# Полный код программы

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.Linq;

using System.Threading;

namespace lab2

{

class Program

{

static int n = 1000; //Элементы

static int m = 8; //Потоки

static int sqrtn = (int)Math.Floor(Math.Sqrt(n));

static bool[] bitmap = new bool[n+1]; //не простые, индекс с 1

static List<int> simple = new List<int>();

static int startsimp;

static Thread[] threads = new Thread[m];

static int step;

static int count = 0;

static int index = 1;

static object locksect = new object();//Крит.секц

static void Main(string[] args)

{

Stopwatch stopWatch = new Stopwatch();

double sum=0;

for (int t = 0; t < 10; t++)

{

stopWatch.Start();

for (int i = 2; i <= sqrtn; i++)

Check(i, sqrtn);

AddSimp(1,sqrtn);

simple.Sort();

startsimp = simple.Count;

/\*========================Последовательное=========================\*/

//Run0();

/\*========================№1=========================\*/

/\*

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

{

threads[i] = new Thread(Run1);

}

step = (n - sqrtn) / m;

for (int i = 0; i < m - 1; i++)

{

threads[i].Start(new int[] { i \* step + sqrtn + 1, (i + 1) \* step + sqrtn });

}

threads[m - 1].Start(new int[] { (m - 1) \* step + sqrtn + 1, n });

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

{

threads[i].Join();

}

\*/

/\*=======================№2==========================\*/

/\*

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

{

threads[i] = new Thread(Run2);

}

step = startsimp / m;

for (int i = 0; i < m - 1; i++)

{

threads[i].Start(new int[] { i \* step + 1, (i + 1) \* step + 1 });

}

threads[m - 1].Start(new int[] { (m - 1) \* step + 1, startsimp });

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

{

threads[i].Join();

}

AddSimp(sqrtn + 1, n);

\*/

/\*=======================№3==========================\*/

/\*

for (int i = 1; i < simple.Count; i++)

{

ThreadPool.QueueUserWorkItem(Run3, simple.ElementAt(i));

}

while (count < simple.Count - 1) { }

AddSimp(sqrtn + 1, n);

\*/

/\*=======================№4==========================\*/

/\*

for (int i = 0; i < m; i++) threads[i] = new Thread(Run4);

for (int i = 0; i < m; i++) threads[i].Start();

for (int i = 0; i < m; i++) threads[i].Join();

AddSimp(sqrtn + 1, n);

\*/

stopWatch.Stop();

sum+=stopWatch.Elapsed.TotalMilliseconds;

stopWatch.Reset();

bitmap = new bool[n + 1];

simple.Clear();

startsimp = 0;

step = 0;

count = 0;

index = 1;

locksect = new object();

}

sum = sum / 10;

Console.WriteLine("Среднее время на 10 замеров, {1} элементов, {2} потоков: {0} мс", sum, n, m);

Console.ReadKey();

}

public static void Check(int num, int fin)

{

for (int i = num + 1; i <= fin; i++)

{

if (!bitmap[i] & (i % num) == 0)

{

bitmap[i] = true;

}

}

}

public static void AddSimp(int start, int fin)

{

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

if (!bitmap[i]) simple.Add(i);

}

public static void Run0() //Последовательное

{

bool flag;

for (int j = sqrtn + 1; j <= n; j++)

{

flag = false;

for (int i = 1; i < startsimp; i++)

{

if (j % simple.ElementAt(i) == 0)

{

bitmap[j] = true;

flag = true;

break;

}

}

if (!flag)

{

simple.Add(j);

}

}

}

public static void Run1(object o) //декомпозиция по данным

{

int[] data = (int[])o;

int start = data[0];

int fin = data[1];

bool flag;

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

{

flag = false;

for (int i = 1; i < startsimp; i++)

{

if (j % simple.ElementAt(i) == 0)

{

bitmap[j] = true;

flag = true;

break;

}

}

if (!flag)

{

simple.Add(j);

}

}

}

public static void Run2(object o) //декомпозиция набора простых чисел

{

int[] data = (int[])o;

int start = data[0];

int fin = data[1];

for (int j = sqrtn+1; j <= n; j++)

{

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

{

if (j % simple.ElementAt(i) == 0)

{

bitmap[j] = true;

break;

}

}

}

}

static void Run3(object o) //применение пула потоков

{

int num = (int)o;

for (int i = sqrtn + 1; i <= n; i++)

{

if (i % num == 0) bitmap[i] = true;

}

lock (locksect)

{

count++;

}

}

static void Run4() //последовательный перебор простых чисел

{

while (true)

{

int num;

lock (locksect)

{

if (index >= simple.Count) return;

num = simple.ElementAt(index);

index++;

}

for (int i = sqrtn + 1; i <= n; i++)

{

if (i % num == 0) bitmap[i] = true;

}

}

}

}

}

# Выводы

В ходе лабораторной работы были реализованы последовательный и параллельные алгоритмы поиска простых чисел; выполнен анализ быстродействия алгоритмов при разном объеме данных, разном числе потоков.

Наибольшую эффективность на исследуемых значениях показали пул потоков и последовательное выполнение.

Для 4-ядерного процессора наилучшие значения на 4 потоках при достаточно большом количестве элементов показал алгоритм декомпозиции набора простых чисел. В остальных случаях 2 потока показали наибольшую эффективность.