Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Кафедра «Высшая школа интеллектуальных систем и суперкомпьютерных технологий»

Отчет по лабораторной работе №2

«Поиск простых чисел»

по дисциплине «Параллельное программирование»

Выполнил В.В. Сухомлинов
студент гр.3530203/70101
Руководитель К.А. Туральчук

« » октябрь 2020 г.

Санкт-Петербург
2020

Последовательный алгоритм «Решето Эратосфена»

Алгоритм заключается в последовательном переборе уже известных простых чисел, начиная с двойки, и проверке разложимости всех чисел диапазона (m, n] на найденное простое число m.

Код

```
static int count = 10;
static int sqrtn = (int)Math.Floor(Math.Sqrt(count));

static double totalTime = 0.0;

static int[] baseArray = new int[count];
static bool[] checkArray = new bool[baseArray.Length];

static void Main(string[] args)
{
            Thread.CurrentThread.CurrentCulture = new
System.Globalization.CultureInfo("de-DE"); // чтобы разделителем дробных чисел
была запятая

            Stopwatch watch = new Stopwatch();

            for (int k = 0; k < 6; k++)
            {
                for (int i = 0; i < count; i++)
                {
                    baseArray[i] = i + 1;
                    checkArray[i] = false;
                }

                watch.Start();

                for (int i = 1; i < sqrtn; i++)
                {
                    if (!checkArray[i]) findSimpleFor(i + 1);
                }

                watch.Stop();

                totalTime += k == 0 ? 0 : watch.Elapsed.TotalMilliseconds;

                watch.Reset();
            }

            Console.WriteLine(totalTime / 5);

            for (int i = 1; i < count; i++)
            {
                if (!checkArray[i]) Console.Write("{0} ", baseArray[i]);
            }
```

```
        }

        static void findSimpleFor(int num)
        {
            for(int i = num; i < count; i++)
            {
                if (!checkArray[i] && baseArray[i] % num == 0) checkArray[i] =
true;
            }
        }
```

N	10	100	10 000	100 000	1 000 000
t	0,00022	0,00194	1,84322	27,57958	653,76264

Модифицированный последовательный алгоритм поиска

В модифицированном алгоритме выделяются два этапа:

1-ый этап: поиск простых чисел в интервале от $2...\sqrt{n}$ с помощью классического метода решета Эратосфена (базовые простые числа).

2-ой этап: поиск простых чисел в интервале от простые числа, в ыявленные на первом этапе.

Код

```
       static int count = 1000000;
       static int sqrtn = (int)Math.Floor(Math.Sqrt(count));
       static double totalTime = 0.0;
       static int[] baseArray = new int[count];
       static List<int> baseSimpleList = new List<int>(); // массив найденных
простых чисел
       static bool[] checkArray = new bool[baseArray.Length];

       static void Main(string[] args)
       {
              Thread.CurrentThread.CurrentCulture = new
System.Globalization.CultureInfo("de-DE"); // чтобы разделителем дробных чисел
была запятая
              Stopwatch watch = new Stopwatch();
              for (int k = 0; k < 6; k++)
              {
              for (int i = 0; i < count; i++)
              {
                     baseArray[i] = i + 1;
                     checkArray[i] = false;
              }
              baseSimpleList.Clear();
              watch.Start();
              for (int i = 1; i < sqrtn; i++)
              {
```

```
                if (!checkArray[i])
                {
                        findSimpleFor(i + 1, i + 1, sqrtn);
                        baseSimpleList.Add(i + 1);
                }
        }
        foreach (int num in baseSimpleList)
        {
                findSimpleFor(num, sqrtn, count);
        }
        watch.Stop();
        totalTime += k == 0 ? 0 : watch.Elapsed.TotalMilliseconds;
        watch.Reset();
        }
        Console.WriteLine(totalTime / 5);
        for (int i = 1; i < count; i++)
        {
        if (!checkArray[i]) Console.Write("{0} ", baseArray[i]);
        }
    }
    static void findSimpleFor(int num, int from, int to) // этап поиска простых
чисел
    {
        for (int i = from; i < to; i++)
        {
        if (!checkArray[i] && baseArray[i] % num == 0) checkArray[i] = true;
        }
    }
```

N	10	100	10 000	100 000	1 000 000
t	0,00026	0,003	0,9817	26,06542	589,04608

Параллельный алгоритм 1: декомпозиция по данным

Идея распараллеливания заключается в разбиении диапазона n...$\sqrt{n}$ на равные части. Каждый поток обрабатывает свою часть чисел, проверяя на разложимость по каждому базовому простому числу.

Код

```
    static int count = 1000000;
    static int threadCount = 10;
    static int sqrtn = (int)Math.Truncate(Math.Sqrt(count));
    static int partCount = (int)Math.Round((double)(count - sqrtn) /
threadCount); // длина промежутка для обработки потоком
    static double totalTime = 0.0;
    static int[] baseArray = new int[count];
    static List<int> baseSimpleList = new List<int>(); // найденные базовые
простые числа
    static bool[] checkArray = new bool[baseArray.Length];
    static Thread[] threads = new Thread[threadCount];
    static void Main(string[] args)
    {
```

```
            Thread.CurrentThread.CurrentCulture = new
System.Globalization.CultureInfo("de-DE"); // чтобы разделителем дробных чисел
была запятая
            Stopwatch watch = new Stopwatch();
            for (int k = 0; k < 6; k++)
            {
            for (int i = 0; i < count; i++)
            {
                  baseArray[i] = i + 1;
                  checkArray[i] = false;
            }
            baseSimpleList.Clear();
            watch.Start();
            // ЭТАП 1
            for (int i = 1; i < sqrtn; i++)
            {
                  if (!checkArray[i])
                  {
                        findSimpleFor(i + 1, i + 1, sqrtn);
                        baseSimpleList.Add(i + 1);
                  }
            }
            // ЭТАП 2
            for (int i = 0; i < threadCount; i++) threads[i] = new
Thread(findSimpleForObject);
            for (int i = 0; i < threadCount; i++)
            {
                  int start = sqrtn + i * partCount;
                  int temp = sqrtn + (i + 1) * partCount;
                  int end = temp > count ? count : temp;
                  threads[i].Start(new int[] { start, end });
            }
            for (int i = 0; i < threadCount; i++) threads[i].Join();
            watch.Stop();
            totalTime += k == 0 ? 0 : watch.Elapsed.TotalMilliseconds;
            watch.Reset();
            }
            Console.WriteLine(totalTime / 5);
            for (int i = 1; i < count; i++)
            {
            // if (!checkArray[i]) Console.Write("{0} ", baseArray[i]);
            }
      }
      static void findSimpleForObject(object p)
      {
            int[] param = (int[])p;
            foreach (int num in baseSimpleList)
            findSimpleFor(num, param[0], param[1]);
      }
      static void findSimpleFor(int num, int from, int to)
      {
            for (int i = from; i < to; i++)
            {
            if (!checkArray[i] && baseArray[i] % num == 0) checkArray[i] = true;
            }
      }
```

N	10	100	10 000	100 000	1 000 000
2	16,94802	17,93058	25,27894	46,67454	329,51368
3	15,58066	25,01822	21,98588	29,46686	254,03894
4	25,23368	30,87648	25,81678	35,1844	246,6146
5	37,33512	45,01866	55,12474	56,95194	204,23542
10	50,53042	52,61918	59,293	69,33964	236,8812

Параллельный алгоритм 2: декомпозиция набора простых чисел

В этом алгоритме разделяются базовые простые числа. Каждый поток работает с ограниченным набором простых чисел и проверяет весь оставшийся диапазон.

Код

```
        static int count = 1000000;
        static int threadCount = 10;
        static int sqrtn = (int)Math.Truncate(Math.Sqrt(count));
        static double totalTime = 0.0;
        static int[] baseArray = new int[count];
        static List<int> baseSimpleList = new List<int>();
        static bool[] checkArray = new bool[baseArray.Length];
        static Thread[] threads = new Thread[threadCount];
        static void Main(string[] args)
        {
                Thread.CurrentThread.CurrentCulture = new
System.Globalization.CultureInfo("de-DE"); // чтобы разделителем дробных чисел
была запятая
                Stopwatch watch = new Stopwatch();
                for (int k = 0; k < 6; k++)
                {
                for (int i = 0; i < count; i++)
                {
                        baseArray[i] = i + 1;
                        checkArray[i] = false;
                }
                baseSimpleList.Clear();
                watch.Start();
                // ЭТАП 1
                for (int i = 1; i < sqrtn; i++)
                {
                        if (!checkArray[i])
                        {
                                findSimpleFor(i + 1, i + 1, sqrtn);
                                baseSimpleList.Add(i + 1);
                        }
                }
                // ЭТАП 2
                for (int i = 0; i < threadCount; i++) threads[i] = new
Thread(findSimpleForObject);
```

```
            int loopCount = baseSimpleList.Count < threadCount ?
baseSimpleList.Count : threadCount;
            int ratio = baseSimpleList.Count / loopCount;
            for (int i = 0; i < threadCount; i++)
            {
                  if (i < baseSimpleList.Count)
                  {
                        threads[i].Start(new int[] { i * ratio, (i + 1) * ratio
});
                  }
            }
            for (int i = 0; i < loopCount; i++) threads[i].Join();
            watch.Stop();
            totalTime += k == 0 ? 0 : watch.Elapsed.TotalMilliseconds;
            watch.Reset();
            }
            Console.WriteLine(totalTime / 5);
            for (int i = 1; i < count; i++)
            {
            //if (!checkArray[i]) Console.Write("{0} ", baseArray[i]);
            }
      }
      static void findSimpleForObject(object p)
      {
            int[] param = (int[])p;
            for (int i = param[0]; i < param[1]; i++)
            findSimpleFor(baseSimpleList[i], sqrtn, count);
      }
      static void findSimpleFor(int num, int from, int to)
      {
            for (int i = from; i < to; i++)
            {
            if (!checkArray[i] && baseArray[i] % num == 0) checkArray[i] = true;
            }
      }
```

N	10	100	10 000	100 000	1 000 000
2	15,76934	18,83344	19,8453	28,8578	323,76608
3	17,2868	18,50844	33,33936	38,16882	248,82142
4	16,67432	21,40374	33,2006	43,11822	217,58496
5	19,63088	37,07324	45,71906	46,60152	220,28554
10	18,40742	46,1241	58,2816	104,03748	233,9842

Параллельный алгоритм 3: применение пула потоков

Применение пула потоков позволяет автоматизировать обработку независимых рабочих элементов. В качестве рабочих элементов предлагается использовать проверку всех чисел оставшегося диапазона на разложимость по одному базовому простому числу.

Код

```
	static int count = 10000;
	static int sqrtn = (int)Math.Truncate(Math.Sqrt(count));
	static int finishCount = 0; // счетчик завершенных задач
	static double totalTime = 0.0;
	static int[] baseArray = new int[count];
	static List<int> baseSimpleList = new List<int>();
	static bool[] checkArray = new bool[baseArray.Length];
	static object sync = new object();
	static void Main(string[] args)
	{
		Thread.CurrentThread.CurrentCulture = new
System.Globalization.CultureInfo("de-DE"); // чтобы разделителем дробных чисел
была запятая
		Stopwatch watch = new Stopwatch();
		for (int k = 0; k < 6; k++)
		{
		for (int i = 0; i < count; i++)
		{
			baseArray[i] = i + 1;
			checkArray[i] = false;
		}
		baseSimpleList.Clear();
		watch.Start();
		// ЭТАП 1
		for (int i = 1; i < sqrtn; i++)
		{
			if (!checkArray[i])
			{
				findSimpleFor(i + 1, i + 1, sqrtn);
				baseSimpleList.Add(i + 1);
			}
		}
		// ЭТАП 2
		finishCount = 0;
		for (int i = 0; i < baseSimpleList.Count; i++)
		{
			ThreadPool.QueueUserWorkItem(findSimpleForObject, new object[]
{ baseSimpleList[i] });
		}
		while (true)
		{
			lock (sync)
			{
				if (finishCount >= baseSimpleList.Count) break;
			}

			// ожидаем
		}
		watch.Stop();
		totalTime += k == 0 ? 0 : watch.Elapsed.TotalMilliseconds;
		watch.Reset();
		}
		Console.WriteLine(totalTime / 5);

		for (int i = 1; i < count; i++)
		{
		if (!checkArray[i]) Console.Write("{0} ", baseArray[i]);
```

```
            }
        }
        static void findSimpleForObject(object p)
        {
            int param = (int)((object[])p)[0];
            findSimpleFor(param, sqrtn, count);
            Interlocked.Increment(ref finishCount);
        }
        static void findSimpleFor(int num, int from, int to)
        {
            for (int i = from; i < to; i++)
            {
            if (!checkArray[i] && baseArray[i] % num == 0) checkArray[i] = true;
            }
        }
```

N	10	100	10 000	100 000	1 000 000
t	0,07204	0,0933	0,44916	7,48846	138,20022

Параллельный алгоритм 4: последовательный перебор простых чисел

Идея алгоритма заключается в последовательном переборе базовых простых чисел разными потоками.

Код

```
        static int count = 1000000;
        static int threadCount = 2;
        static int sqrtn = (int)Math.Truncate(Math.Sqrt(count));
        static int partCount = (int)Math.Round((double)(count - sqrtn) /
threadCount);

        static object sync = new object();

        static int currentIndex = 0;

        static double totalTime = 0.0;

        static int[] baseArray = new int[count];
        static List<int> baseSimpleList = new List<int>();
        static bool[] checkArray = new bool[baseArray.Length];

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

        static void Main(string[] args)
        {
            Thread.CurrentThread.CurrentCulture = new
System.Globalization.CultureInfo("de-DE"); // чтобы разделителем дробных чисел
была запятая

            Stopwatch watch = new Stopwatch();
```

```
            for (int k = 0; k < 6; k++)
            {
                for (int i = 0; i < count; i++)
                {
                    baseArray[i] = i + 1;
                    checkArray[i] = false;
                }

                baseSimpleList.Clear();
                currentIndex = 0;

                watch.Start();

                // ЭТАП 1

                for (int i = 1; i < sqrtn; i++)
                {
                    if (!checkArray[i])
                    {
                        findSimpleFor(i + 1, i + 1, sqrtn);
                        baseSimpleList.Add(i + 1);
                    }
                }

                // ЭТАП 2

                for (int i = 0; i < threadCount; i++) threads[i] = new
Thread(findSimpleForObject);

                for (int i = 0; i < threadCount; i++)
                {
                    threads[i].Start();
                }

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

                watch.Stop();

                totalTime += k == 0 ? 0 : watch.Elapsed.TotalMilliseconds;

                watch.Reset();
            }

            Console.WriteLine(totalTime / 5);
            /*
            for (int i = 1; i < count; i++)
            {
                if (!checkArray[i]) Console.Write("{0} ", baseArray[i]);
            }*/
        }

        static void findSimpleForObject()
        {
            int currentSimple;
```

```
        while (true)
        {
            lock (sync)
            {
                if (currentIndex >= baseSimpleList.Count) break;
                currentSimple = baseSimpleList[currentIndex];
                currentIndex++;
            }

            findSimpleFor(currentSimple, sqrtn, count);
        }
    }

    static void findSimpleFor(int num, int from, int to)
    {
        for (int i = from; i < to; i++)
        {
            if (!checkArray[i] && baseArray[i] % num == 0)
            {
                checkArray[i] = true;
            }

        }
    }
```

N	10	100	10 000	100 000	1 000 000
2	51,1441	43,5637	50,47906	75,16214	344,94208
3	64,6497	75,61694	68,49764	97,77238	290,36154
4	78,2153	70,85518	76,06924	96,469	243,27346
5	71,91008	80,23318	102,54326	121,1497	247,79584
10	164,11712	171,05056	157,03432	190,12536	305,73098

Вывод

Последовательные версии алгоритма поиска простых чисел работают быстрее параллельных на несколько порядков при малых количествах данных - <10 000. При их увеличении, >1 000 000, каждая параллельная реализация на 4 потоках работает в два раза быстрее, чем последовательная. При этом наибольший выигрыш скорости заметен при использовании пулла потоков.