МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ   
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ   
«ВЯТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

ФАКУЛЬТЕТ КОМПЬЮТЕРНЫХ И ФИЗИКО-МАТЕМАТИЧЕСКИХ НАУК

КАФЕДРА ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

Отчёт по лабораторной работе №3  
по дисциплине «Параллельные вычисления»

**Многопоточность в C#**

Выполнил: студент группы ФИб-4301-51-00     / К.О. Дёмин /

Проверил: ст.преподаватель каф. ПМИ     / А.В. Торбеева /

Киров 2019

Оглавление

[Цель работы 3](#_Toc21881090)

[Задание 3](#_Toc21881091)

[Листинг 4](#_Toc21881092)

[Полученные результаты 11](#_Toc21881093)

[Выводы по лабораторной работе 12](#_Toc21881094)

# Цель работы

Рассмотреть различные способы хранения матриц, исследовать их умножение.

# Задание

1. Разработать программу, выводящую числа от 1 до 100 в двух потоках, в первом – четные, во втором – нечетные.
2. Разработать программу, генерирующую *N* потоков, каждый из которых выводит свой номер в бесконечном цикле. Количество потоков *N* задает пользователь с клавиатуры.
3. Разработать программу, генерирующую *N* потоков в пуле потоков CLR. Каждый поток раз в 10 миллисекунд выводит свой идентификатор и номер шага.
4. Разработать программу, включающую в себя последовательный и параллельный алгоритм вычисления произведения квадратной матрицы на вектор. Каждый алгоритм реализуется в отдельном методе. Матрица и вектор генерируются случайным образом по заданной размерности *N*. Параллельный алгоритм должен учитывать доступное ему количество процессоров. Программа запрашивает у пользователя размерность *N*, после чего выводит время вычислений для последовательного и параллельного алгоритмов. Сами матрицы выводить не нужно.

Провести тестирование программ на матрицах размерности *N* = 500, 1000 и 2000. На каждом примере запустить не менее 3 раз. В таблицу занести среднее время выполнения на одном примере в секундах.

1. Написать параллельную программу, вычисляющую приближенное значение числа ln 2 как сумму при заданном *N*.

Замерить время работы программы для *N* = 107, 108 и 109 на 1, 2, 4 и 8 потоках. На каждом примере запустить не менее 10 раз. В таблицу занести среднее время выполнения на одном примере в секундах.

# Листинг

Задание 1

using System;

using System.Threading;

namespace task\_1

{

    class Program

    {

        static int number = 1;

        static object thisLock = new object();

        static void Main(string[] args)

        {

            Thread first = new Thread(() =>

            {

                while(true)

                {

                    //  блокируем

                    lock (thisLock)

                    {

                        //  если нечетное, то пропускаем, иначе выводим

                        if (number % 2 == 1) continue;

                        Console.WriteLine(number);

                        if (++number == 100) break;  //  выходим из цикла при 100

                    }

                }

            });

            Thread second = new Thread(() =>

            {

                while (true)

                {

                    lock (thisLock)

                    {

                        //  если четное, то пропускаем, иначе выводим

                        if (number % 2 == 0) continue;

                        Console.WriteLine(number);

                        if (++number == 100) break;  //  выходим из цикла при 100

                    }

                }

            });

            first.Start();

            second.Start();

            Console.ReadKey();

        }

    }

}

Задание 2

using System;

using System.Threading;

namespace task\_2

{

    class Program

    {

        static void Main(string[] args)

        {

            Console.WriteLine("Введите число потоков:");

            int n = int.Parse(Console.ReadLine());

            Thread[] threads = new Thread[n];

            //  создаём потоки и выводим информацию в консоль

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

            {

                int index = i;

                threads[i] = new Thread(() => Print(index));

                threads[i].Start();

            }

            Console.ReadKey();

        }

        static void Print(int id)

        {

            //  бесконечно выводим информацию

            while (true)

            {

                Console.WriteLine($"Thread : {id}");

            }

        }

    }

}

Задание 3

using System;

using System.Threading;

namespace task\_3

{

    class Program

    {

        static void Main(string[] args)

        {

            Console.WriteLine("Введите N:");

            int n = int.Parse(Console.ReadLine());

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

                ThreadPool.QueueUserWorkItem(JobForAThread);

            Console.ReadKey();

        }

        static void JobForAThread(object value)

        {

            long step = 0;

            while(true)

            {

                //  вывод идентификатора и номер шага

                Console.WriteLine($"ID: {Thread.CurrentThread.ManagedThreadId}, step: {++step}");

                Thread.Sleep(10);

            }

        }

    }

}

Задание 4

using System;

using System.Diagnostics;

namespace task\_4

{

    public class Program

    {

        const int Iterations = 3;

        static void Main(string[] args)

        {

            Console.WriteLine("Введите n");

            int n = int.Parse(Console.ReadLine());

            Stopwatch stopwatch = new Stopwatch();  //  объект для замерения времени

            TimeSpan[] timeSpans = new TimeSpan[2];

            Matrix matrix = new Matrix(n);  //  матрица

            Vector vector = new Vector(n);  //  вектор

            Func<Vector, Vector>[] actions = new Func<Vector, Vector>[]

            {

                matrix.Mul,

                matrix.ParallelMul

            };  //  операции

            for (int iter = 0; iter < Iterations; ++iter)

            {

                matrix.Generate();  //  генерируем матрицу

                vector.Generate();  //  генерируем вектор

                //Console.WriteLine(matrix);

                //Console.WriteLine(vector);

                for (int j = 0; j < actions.Length; ++j)

                {

                    stopwatch.Restart();  //  перезапускаем замер времени

                    var result = actions[j](vector);

                    stopwatch.Stop();  //  заканчиваем замерение

                    //Console.WriteLine(result);

                    timeSpans[j] += stopwatch.Elapsed;  //  добавляем время

                }

            }

            for(int i = 0; i < actions.Length; ++i)

                Console.WriteLine($"{actions[i].Method.Name}: {timeSpans[i].TotalSeconds / Iterations:f3} сек.");  //  выводим информацию

            Console.ReadKey();

        }

    }

}

using System;

using System.Linq;

using System.Text;

using System.Threading;

namespace task\_4

{

    public class Matrix

    {

        /// <summary>

        /// массив матрицы

        /// </summary>

        int[] arr;

        /// <summary>

        /// лок

        /// </summary>

        object thisLock = new object();

        /// <summary>

        /// Размер матрицы

        /// </summary>

        public int Size { get; }

        /// <summary>

        /// Обращение по индексу

        /// </summary>

        /// <param name="y"></param>

        /// <param name="x"></param>

        /// <returns></returns>

        public int this[int y, int x]

        {

            get

            {

                return arr[y \* Size + x];

            }

            set

            {

                arr[y \* Size + x] = value;

            }

        }

        /// <summary>

        /// Конструктор

        /// </summary>

        /// <param name="size"></param>

        public Matrix(int size)

        {

            Size = size;

            arr = new int[Size \* Size];

        }

        /// <summary>

        /// Построение строки

        /// </summary>

        /// <returns></returns>

        public override string ToString()

        {

            StringBuilder stringBuilder = new StringBuilder();

            for (int y = 0; y < Size; ++y)

            {

                for (int x = 0; x < Size; ++x)

                    stringBuilder.Append(this[y, x] + " ");

                stringBuilder.Append("\n");

            }

            return stringBuilder.ToString();

        }

        /// <summary>

        /// Генератор

        /// </summary>

        public void Generate()

        {

            Random rand = new Random();

            arr = arr.Select(i => i = rand.Next(10)).ToArray();

        }

        /// <summary>

        /// Умножение на вектор

        /// </summary>

        /// <param name="vector"></param>

        /// <returns></returns>

        public Vector Mul(Vector vector)

        {

            Vector vector1 = new Vector(vector.Size);

            for (int y = 0; y < Size; ++y)

                for (int x = 0; x < Size; ++x)

                    vector1[y] += this[y, x] \* vector[x];

            return vector1;

        }

        /// <summary>

        /// Параллельное умножение на вектор

        /// </summary>

        /// <param name="vector"></param>

        /// <returns></returns>

        public Vector ParallelMul(Vector vector)

        {

            int count = Environment.ProcessorCount;

            if (Size < count)

                count = Size;

            //  если размер матрицы < числа потоков, то уменьшаем количество потоков

            int rowsPerThread = (int)Math.Ceiling(Size / (decimal)count);  //  количество строк на поток

            Thread[] threads = new Thread[count];  //  объекты потоков

            Vector result = new Vector(Size);  //  результат

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

            {

                int index = i;

                threads[i] = new Thread(() => Calc(ref result, vector, index, rowsPerThread));

                threads[i].Start();

            }

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

                threads[i].Join();

            return result;

        }

        void Calc(ref Vector result, Vector vector, int index, int rowsPerThread)

        {

            Vector myVector = new Vector(Size);  //  временный вектор для потока

            int start, end;  //  строки начала и конца вычисления

            start = index \* rowsPerThread;

            end = (index + 1) \* rowsPerThread;

            //  если конец больше размера матрицы, то уменьшаем его

            if (end > Size)

                end = Size;

            //  умножаем строки на вектор и сохраняем во временный вектор

            for (int y = start; y < end; ++y)

            {

                for (int x = 0; x < Size; ++x)

                {

                    myVector[y] += this[y, x] \* vector[x];

                }

            }

            //  прибавляем к главной переменной

            lock (thisLock)

            {

                result += myVector;

            }

        }

    }

}

using System;

using System.Linq;

namespace task\_4

{

    public class Vector

    {

        private int[] data;  //  массив значений

        public int Size { get; }  //  размер вектора

        /// <summary>

        /// Конструктор

        /// </summary>

        /// <param name="size"></param>

        public Vector(int size)

        {

            Size = size;

            data = new int[size];

        }

        /// <summary>

        /// Построение строки

        /// </summary>

        /// <returns></returns>

        public override string ToString()

        {

            return string.Join('\t', data);

        }

        /// <summary>

        /// Обращение по индексу к вектору

        /// </summary>

        /// <param name="x"></param>

        /// <returns></returns>

        public int this[int x]

        {

            get

            {

                return data[x];

            }

            set

            {

                data[x] = value;

            }

        }

        /// <summary>

        /// Генератор

        /// </summary>

        public void Generate()

        {

            Random rand = new Random();

            data = data.Select(i => i = rand.Next(10)).ToArray();

        }

        /// <summary>

        /// Операция сложения

        /// </summary>

        /// <param name="first"></param>

        /// <param name="second"></param>

        /// <returns></returns>

        public static Vector operator+(Vector first, Vector second)

        {

            Vector vector = new Vector(first.Size);

            for (int i = 0; i < first.Size; ++i)

                vector[i] = first[i] + second[i];

            return vector;

        }

    }

}

Задание 5

using System;

using System.Diagnostics;

using System.Threading;

namespace task\_5

{

    class Program

    {

        static object thisLock = new object();

        static int Iterations = 10;

        static void Main(string[] args)

        {

            Console.WriteLine("Введите n");

            int n = int.Parse(Console.ReadLine());

            Console.WriteLine("Введите число потоков");

            int threads = int.Parse(Console.ReadLine());

            if (n < threads)

                threads = n;

            TimeSpan timeSpan = new TimeSpan();

            Stopwatch stopwatch = new Stopwatch();  //  объект для замерения времени

            for (int iter = 0; iter < Iterations; ++iter)

            {

                stopwatch.Restart();  //  перезапускаем замер времени

                var val = Calc(threads, n);  //  считаем значение

                stopwatch.Stop();  //  выводим

                Console.WriteLine(val);

                timeSpan += stopwatch.Elapsed;  //  добавляем время

            }

            Console.WriteLine($"Время: {timeSpan.TotalSeconds / Iterations:f3} сек.");  //  выводим информацию

            Console.ReadKey();

        }

        static double Calc(int threadCount, int n)

        {

            Thread[] threads = new Thread[threadCount];  //  потоки

            int intervalPerThread = (int)Math.Ceiling(n / (double)threadCount);  //  интервал для одного потока

            double result = 0;  //  результат

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

            {

                int index = i;

                threads[i] = new Thread(() =>

                {

                    double res = 0;  //  результат на поток

                    int start, end;

                    start = index \* intervalPerThread;  //  начало интервала

                    end = start + intervalPerThread;  //  конец интервала

                    if (end > n)

                        end = n;  //  если конец интервала > n, то сокращаем его

                    for (int k = start + 1; k <= end; ++k)  //  считаем для (start; end]

                    {

                        res += Math.Pow(-1, k - 1) / k;

                    }

                    //  прибавляем к общему результату

                    lock (thisLock)

                    {

                        result += res;

                    }

                });

                threads[i].Start();

            }

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

                threads[i].Join();

            return result;

        }

    }

}

# Полученные результаты

Тестирование проводилось на четырёхъядерном и восьми поточном процессоре Intel Core i7 4770 с частотой 3.7 ГГц, 16 гб ОЗУ в двухканальном режиме с частотой 1600 МГц.

Умножение квадратной матрицы на вектор. Программа автоматически получала число потоков процессора (8). Время в таблице указано в секундах.

|  |  |  |
| --- | --- | --- |
| Размерность | Алгоритм | |
| последовательный | параллельный |
| 500 | 0,001 | 0,002 |
| 1000 | 0,002 | 0,002 |
| 2000 | 0,007 | 0,004 |

Приближенное значение числа ln2 вычислялось на этой же системе. Время в таблице указано в секундах.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Размерность | Число потоков | | | |
| 1 | 2 | 4 | 8 |
| 107 | 0,085 | 0,041 | 0,030 | 0,022 |
| 108 | 0,815 | 0,410 | 0,254 | 0,190 |
| 109 | 8,126 | 4,067 | 2,507 | 1,843 |

# Выводы по лабораторной работе

Распараллеливание даёт прирост на больших объемах данных. При этом, необходимо помнить про работу с памятью и синхронизацию, так как это напрямую влияет на время работы и результат. Если потоки каждый раз будут вытеснять данные из кэша процессора или обращаться к разным участкам памяти, то последовательный алгоритм окажется быстрее, так как один поток будет совершать больше операций с данными без простаивания.

Результаты перемножения матрицы на вектор с небольшим объёмом данных показывает, что прироста от использования максимального числа потоков может и не быть, так как тратится время на запуск потоков и их синхронизацию. Кроме того, при перемножении матрицы на вектор потоки используют общие данные в виде массивов, что увеличивает время работы. При вычислении логарифма увеличение числа потоков уменьшает общее время работы, так как потоки работают с собственными данными и тратят мало времени на синхронизацию.