*Московский Государственный Технический Университет имени Н. Э. Баумана.*

*ФАКУЛЬТЕТ “Информатики и систему управления”*

*КАФЕДРА “Программное обеспечение ЭВМ и информационные технологии”*

Отчет

По лабораторной работе №6

По курсу “Анализ Алгоритмов”

Тема “Организация конвейерных вычислений”

Студент: Бадалян Д.А.

Группа: ИУ7-51

Преподаватель: Волкова Л.Л.

Москва, 2017

**Постановка задачи**

Необходимо организовать конвейерные вычисления на произвольной вычислительной системе, сделать выводы о:

1. Конвейерных вычислениях в целом как о способе параллелизации программ
2. Особенностях их реализации

**Теория**

Конвейер – совокупность ступеней и средств передачи данных между ними, организованных таким образом, что на вход системы поступают исходные данные, затем они последовательно, в соответствии с разбиением базовой функции на подфункции, перемещаются между ступенями, подвергаясь на каждом этапе промежуточной обработке, в результате чего на выходе получается требуемый результат. Одновременно в конвейере может находиться более одного элемента входных данных.

**Выполнение**

Для выполнения был выбран язык программирование C# и пространство имён System.Threading.

Исходная задача состоит в написании некой функции F(x), которая должна отработать с N наборами входных данных и выдать N соответствующих результатов.

Функция была разбита на 3 подфункции:

F(x) = F3(F2(F1(x))))

F1(x) = x\*x

F2(x) = x + 3

F3(x) = x - 10

Было создано 3 потока, отвечающих за каждую ступень.

**Описание алгоритма:**

Как только какой-либо поток отработал с текущим набором переменных – он передает полученный результат в следующий поток (либо в результирующий буфер памяти если речь идет о последней ступени) и получает новый набор переменных от предыдущего потока (либо не от потока, если речь идет о первой ступени). Передача данных между потоками осуществляется за счет глобальных очередей.

Так как речь идет о элементарных вычислениях, коих достаточно для целей данной лабораторной работы, и достаточно примитивном взаимодействии потоков друг с другом, что и подразумевается в простейшем конвейере - схема алгоритма избыточна, словесного описания, данного выше, достаточно.

**Листинг**

static void f1()

{

int temp = 0;

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

{

temp = input[i] \* input[i];

lock (locker)

{

queue1.Enqueue(temp);

}

Console.WriteLine("1. Взял: " + input[i] + ", положил: " + (temp));

}

}

static void f2()

{

int temp;

while (work)

{

if (queue1.Count != 0)

{

lock (locker)

{

temp = queue1.Dequeue();

queue2.Enqueue(temp + 3);

}

Console.WriteLine("\t2. Взял: " + temp + ", положил: " + (temp + 3));

}

}

}

static void f3()

{

int temp2;

while (work)

{

if (queue2.Count != 0)

{

lock (locker)

{

temp2 = queue2.Dequeue();

}

output[schetchik] = temp2 - 10;

Console.WriteLine("\t\t3. Взял: " + temp2 + ", положил: " + (output[schetchik]));

schetchik++;

if (schetchik == len)

work = false;

}

}

}

**queue1, queue2, queue3** – глобальные очереди типа

**input, output –** входной/выходной массив

**schetchik –** глобальная переменная, отвечающая за индексирование выходного массива. При schetchik == len (где len длина входного, и, соответственно, выходного массива) – это будет означать конец работы потоков 2,3.

**Реализация**

Основное, что стоит отметить по поводу реализации – реализацию доступа к критической зоне, в данном случае это глобальные очереди queue1, queue2, queue3. Проблема является классической проблемой в вопросе взаимодействии потоков и имеет множество решений. В данной реализации была выбрана команда lock, которая при доступе к критической зоне блокирует доступ к ней другим процессам. Естественно это сказывается на производительности.

Т.к. цели исследования скорости работы конвейера не стояло - другие средства синхронизации опробованы не были.

**Трудоемкость**

В силу необходимости синхронизации, как уже отмечалось выше, теряется скорость работы конвейера.

Если предположить, что t – число операций, выполняемых на ступени (в среднем), то трудоемкость конвейера:

Fk = N\*t

Трудоемкость последовательной обработки данных:

Fp = 3\*N\*t

Получение точного выражения для трудоемкости конвейера не является возможным, т.к. сценарий, по которому потоки будут выполняться на CPU всегда разный. Поэтому высчитывать точное значение t на каждой ступени для получение точного значения Fp не имеет смысла.

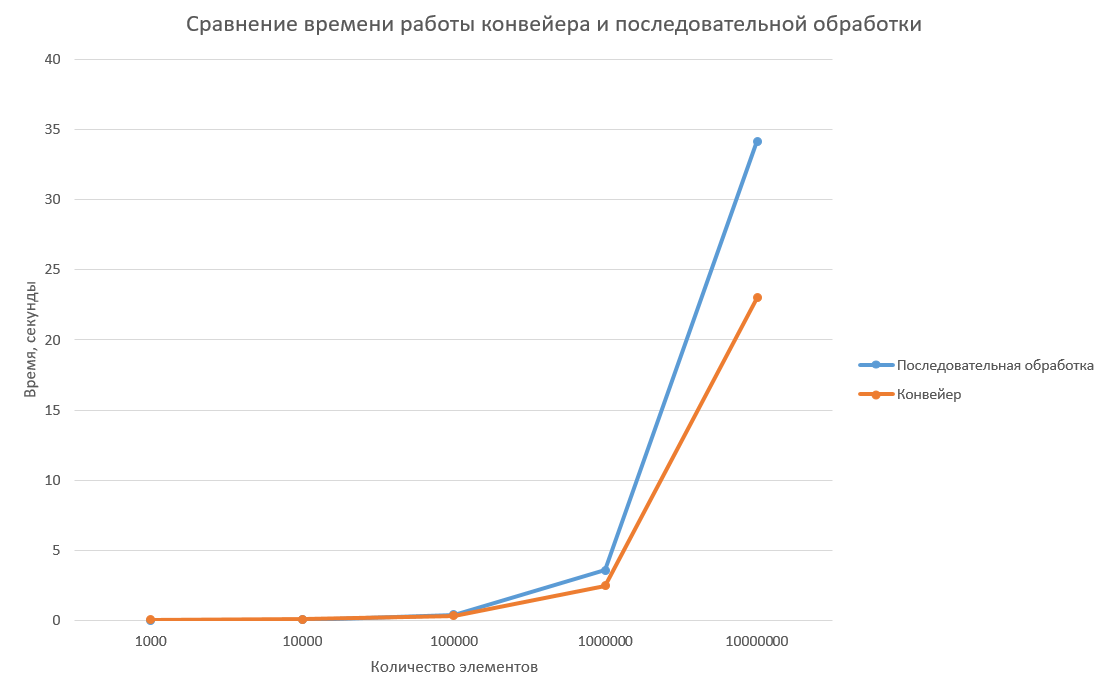
Так же стоит отметить, что количество ядер на ноутбуке, на котором выполнялась лабораторная равно 4.

И при все при этом, действительно ли потоки (в данном случае их 3) выполняются на разных ядрах и достигается ли параллельность – остается на совести методе Thread::Start(), разработчиков Visual Studio и диспетчера Windows.

**Временные эксперименты**

При функциях F1, F2, F3 последовательная обработка всегда оказывалась быстрее. (На 10.000.000 элементах это было так, дальше я смотреть не стал).

При условии приличного числа вычислений - результаты следующие (они проводятся по той же схеме, т.е. просто измены функции F1, F2, F3)



Замеры проводились с помощью объекта System.Diagnostics.Stopwatch

**Bывод**

Конвейерные вычисления – один из видов параллелизации программ, который при большом числе вычислений на ступенях - действительно их ускоряет. При том надо учитывать, что конвейерные вычисления невозможны без синхронизации, т.к. потоки, так или иначе, должны передавать друг другу данные – что и замедляет конвейер и делает его неэффективным при малом числе вычислений, проводимых на ступенях.