Практическая работа №1. Параллельная обработка с помощью библиотеки Task Parallel Library

Задания:

1. Реализовать алгоритм обработки множества графических файлов (*.bmp, *.jpg, ..). Множество файлов должно быть достаточно большим (более 20 изображений).
2. Реализовать параллельный алгоритм с помощью метода Parallel.For.
3. Реализовать возможность отмены выполнения параллельной обработки при нажатии определенной клавиши.
4. Оценить время выполнения последовательного алгоритма, параллельного алгоритма без ограничений и с ограничением степени параллелизма.
5. Для разного объема данных (например, 10 файлов и 50 файлов) получить ответы на следующие вопросы: сколько потоков участвует в обработке; сколько потоков участвует в обработке одновременно; участвует ли главный поток в обработке, сколько элементов обрабатывается каждым потоком.
6. Исследовать разные схемы разделения данных: стандартная схема, динамически сбалансированная схема, статическая схема.

Рекомендации к выполнению:

Вычислительной задачей является обработка графических файлов. Обработка состоит из нескольких этапов и может включать следующие действия: загрузка изображения из файла или Интернет-источника, изменение размера изображения, изменение разрешения, добавление надписей, изменение цветовой палитры, сохранение изображения в новом файле.

Например, с помощью объекта Graphics и методов DrawImage, DrawString можно выполнить изменение цветовой палитры и добавление надписей.

Отмена обработки. Для отмены выполнения обработки файлов необходимо использовать объект CancellationTokenSource. Отмена обработки выполняется по требованию пользователя (нажатие определенной клавиши в консольном проекте или кнопки в Windows-

проекте). Ожидание пользовательского события должно осуществляться в отдельном потоке параллельно с обработкой изображений:

```
Thread thr = new Thread( () => {
      while(true) {
            Thread.Sleep(100);
            if (Console.KeyAvailable) {
                  Console.ReadKey(true);
                  cts.Cancel();
            }
      }
});
thr.Start();
try {
      Parallel.ForEach(ImageFiles, Options, ImageWork.Processing);
}
catch (OperationCanceledException o) {
      ..
}
```

Ограничение степени параллелизма. В начале обработки данных планировщик задач выделяет несколько рабочих потоков пула. При длительной обработке эвристический алгоритм планировщика может увеличивать число участников (рабочих потоков). Такая стратегия может повышать эффективность, если параллельная обработка содержит операции ввода-вывода (работа с консолью, файловой системой, сетевые запросы). В случае незначительной доли I/O-операций от общей вычислительной нагрузки динамическое увеличение числа потоков приводит к излишней конкуренции за ядра процессора и ухудшению эффективности. Для ограничения степени параллелизма используется объект ParallelOptions, который передается в качестве аргумента в метод Parallel.For или Parallel.ForEach.

```
ParallelOptions Options = new ParallelOptions() {
            CancellationToken = cts.Token,
            MaxDegreeOfParallelism = 1
      };
Parallel.ForEach(ImageFiles, Options, ImageWork.Processing);
```

Анализ схем разделения данных. Библиотека TPL поддерживает статические и динамические схемы разделения. По умолчанию используется динамическая схема, при которой число элементов в задаче динамически увеличивается; число потоков, участвующих в обработке заранее неизвестно.

Для исследования работы стандартной схемы разделения данных, которая реализуется по умолчанию без использования объекта Partitioner, предлагается выполнить эксперименты с разным числом элементов (файлов) и зафиксировать особенности выполнения. Параметры обработки каждого элемента можно фиксировать внутри метода обработки в конкурентной коллекции.

```
void Processing(string f)
{
```

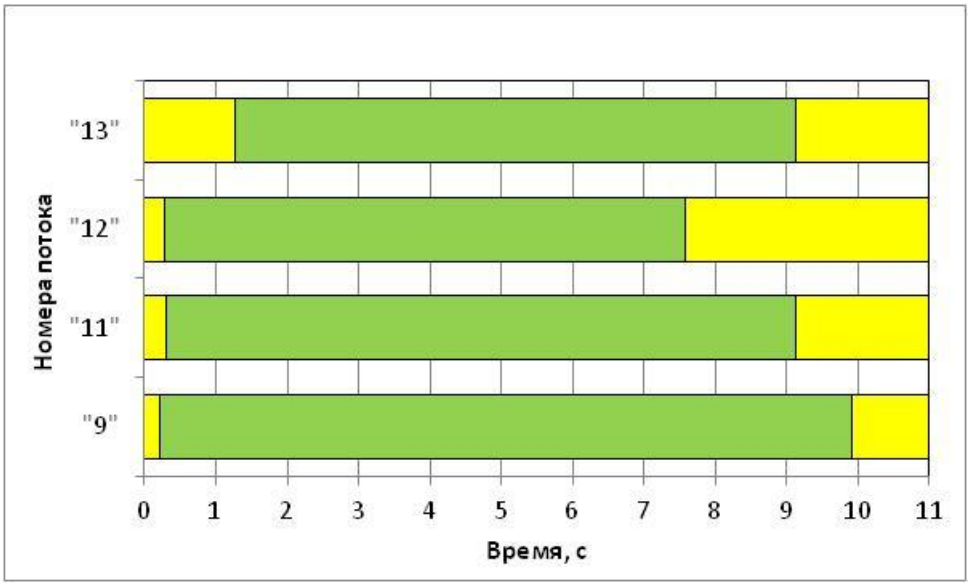
```
            double ms1 = sw.Elapsed.TotalSeconds;
            ..
            double ms2 = sw.Elapsed.TotalSeconds;
            int thr = Thread.CurrentThread.ManagedThreadId;
            int task = Task.CurrentId.Value;
            bag.Add(new TaskDetails(thr, task, idx, ms1, ms2));
        }
```

Пользовательская структура TaskDetails предназначена для хранения информации о задаче. После завершения обработки коллекция bag содержит данные о каждом элементе.

Результаты экспериментов могут быть представлены в следующей таблице:

	N = 10	N = 20	N = 50	N = 100
Участие главного потока (Да/Нет)	Да	..		
Число потоков, участвующих в обработке	4	..		
Число элементов в каждом потоке	“9”: 3 “11”: 3 “12” : 2 “13”: 2	..		
Время начало работы потоков	"9": 0.213 "11": 0.312 "12": 0.272 "13": 1.271	..		

Информация о времени участия потоков в обработке может быть представлена в графической форме. Например, следующая гистограмма, построенная в Excel, отображает время начала и время завершения обработки для каждого потока. Потоки 9, 11, 12 начали обработку практически одновременно. Поток 13 подключился к обработке позднее.



Для исследования альтернативных схем разделения данных необходимо использовать объект Partitioner.

Сбалансированная схема:

```
var PartedData = Partitioner.Create(files, true);
Parallel.ForEach(PartedData, options, Processing);
```

Статическая схема:

```
Parallel.ForEach(Partitioner.Create(0, 100), options, range =>  {
     for(int i=range.Item1; i < range.Item2; i++)
          Processing(files[i]);
});
```

Статическая схема с фиксированным размером блока

```
Parallel.ForEach(Partitioner.Create(0, 100, 25), options,
     range =>  {
          for(int i=range.Item1; i < range.Item2; i++)
               Processing(files[i]);
     });
```

Для каждой схемы для одного из экспериментов с большим числом элементов необходимо заполнить следующую таблицу:

Сбалансированная схема, N = 50

Задача, №	Число элементов	Время старта	Время завершения	Поток, №

Для сравнения разных схем декомпозиции результаты экспериментов с большим числом элементов можно представить в таблично форме:

Характеристики	Стандартная схема	Статическая схема	Статическая схема с фикс. размером	Сбалансированная схема
Время работы				
Число потоков				
Число задач				
..				

В конце работы необходимо сделать выводы об особенностях разных схем декомпозиции данных и их преимуществах в задаче обработки файлов.