TASK PARALLEL LIBRARY

MODULE 2

September, 2015

Module goals

- Introduction The Task Parallel Library
- Why Tasks?
- Parallel Extensions

Task Parallel Library

- **Task** - Independent unit of work
- **Task Parallelism** - Process of running tasks

Task Parallel Library

Benefits:

- More efficient and more scalable use of system resources
- More programmatic control than is possible with a thread or work item

Task Parallel Library

- Task
- Task<TResult>
- TaskFactory
- Parallel

Task<TResult> Class

Method/Property	Description
`Task<TResult>(Func<TResult>)`	Initializes a newTask<TResult>with the specified function
`static Factory`	Provides access to factory methods for creating and configuring Task<TResult>instances
`Id`	Gets an ID for this Task instance
`Result`	Gets the result value of this Task<TResult>
`Status`	Gets the TaskStatus of this task
`ContinueWith(Action<Task<TResult>>)`	Creates a continuation that executes asynchronously when the target task completes
`Start()`	Starts the Task, scheduling it for execution to the current TaskScheduler
`Wait()`	Waits for the Task to complete execution

Tasks vs Threads

- Huge cost of thread creation
- Context switching
- ThreadPool
  - without overhead of Thread creation
  - without un-necessary context switching (if not required)

Tasks vs Threads

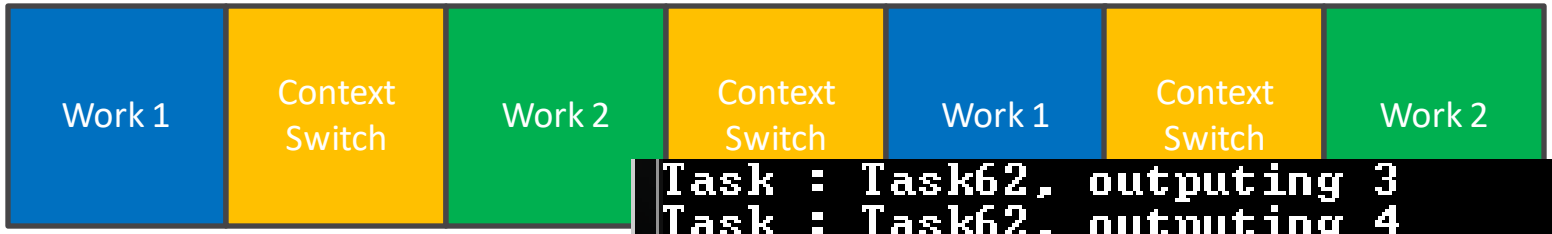
```
// How to create and run Thread
// Create thread
var thread = new Thread(() =>
                        {
                            // My custom task here...
                        });

// Run thread
thread.Start();
```

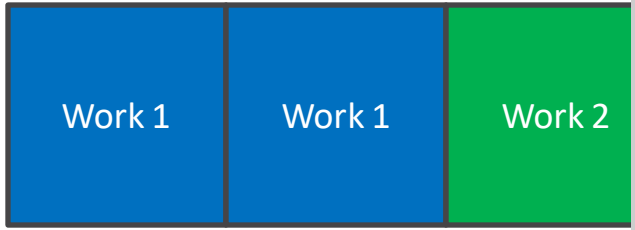
```
// How to create and run Task
Task.Factory.StartNew(() =>
                        {
                            // My custom task here...
                        });
```

Tasks vs Threads

Threads



Tasks




```
Task : Task62, outputing 3
Task : Task62, outputing 4
Task : Task62, outputing 5
Task : Task56, outputing 7
Task : Task56, outputing 8
Task : Task56, outputing 9
Task : Task62, outputing 6
Task : Task62, outputing 7
Task : Task62, outputing 8
Task : Task62, outputing 9
Traditional Thread Approach   : 331ms
Task Parallel Library Approach: 54ms
```

Tasks vs ThreadPools

```
// How to run parallel task via ThreadPool
ThreadPool.QueueUserWorkItem(obj => DoSomeWork());
```

```
// How to wait parallel task that's created via ThreadPool
var resetEvent = new ManualResetEvent(false);
ThreadPool.QueueUserWorkItem(obj =>
                                {
                                    DoSomeWork();
                                    resetEvent.Set();
                                });
resetEvent.WaitOne();
```

Tasks vs ThreadPools

```
// How to create and run Task
var task = new Task(DoSomeWork);
task.Start();
```

```
// How to wait a Task(s)
// Create a Task(s)
var task1 = Task.Factory.StartNew(DoSomeWork);
var task2 = Task.Factory.StartNew(DoSomeWork);
var task3 = Task.Factory.StartNew(DoSomeWork);

// Wait for all Tasks that we need
Task.WaitAll(task1, task2, task3);
```

```
// How to run Task right after some work is done
Task.Factory.StartNew(DoSomeWork)
    .ContinueWith(DoAnotherWork);
```

```
// How to build a Task chain
Task.Factory.StartNew(FetchImagesFromFlicker)
    .ContinueWith((Func<Task, List<string>>)SearchImagesFromMyLastBirthday)
    .ContinueWith(PublishThisOnFacebook)
    .ContinueWith(NotifyMyFriends)
    .Wait(AllMyFriendsConfirmReceipt);
```

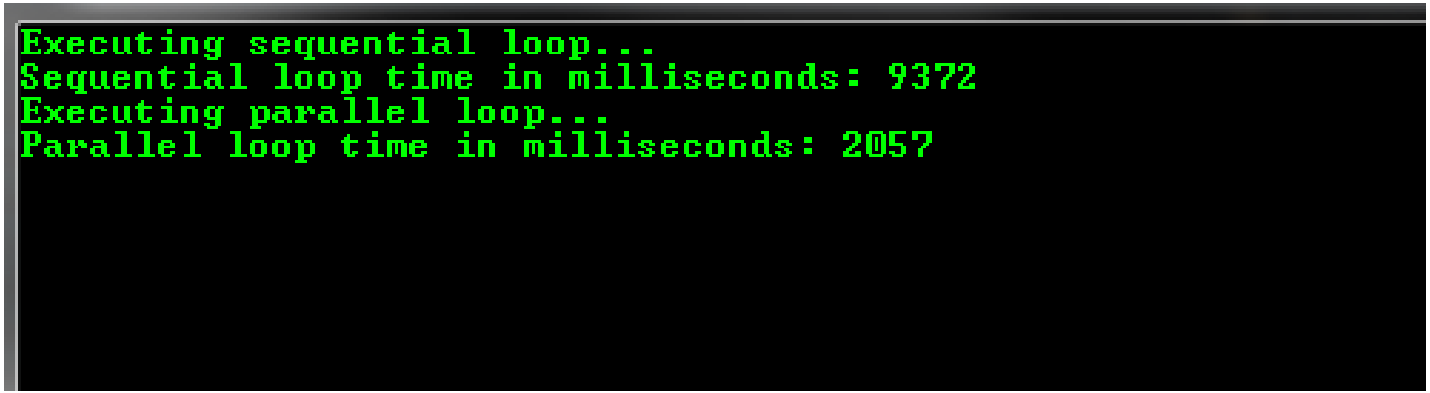
Parallel Class

Method	Description
`For(Int32, Int32, Action<Int32>)`	Executes a **for** loop in which iterations may run in parallel
`ForEach<TSource>(IEnumerable<TSource>, Action<TSource>)`	Executes a **foreach** operation on an IEnumerable in which iterations may run in parallel
`Invoke(Action[])`	Executes each of the provided actions, possibly in parallel

Parallel Extensions

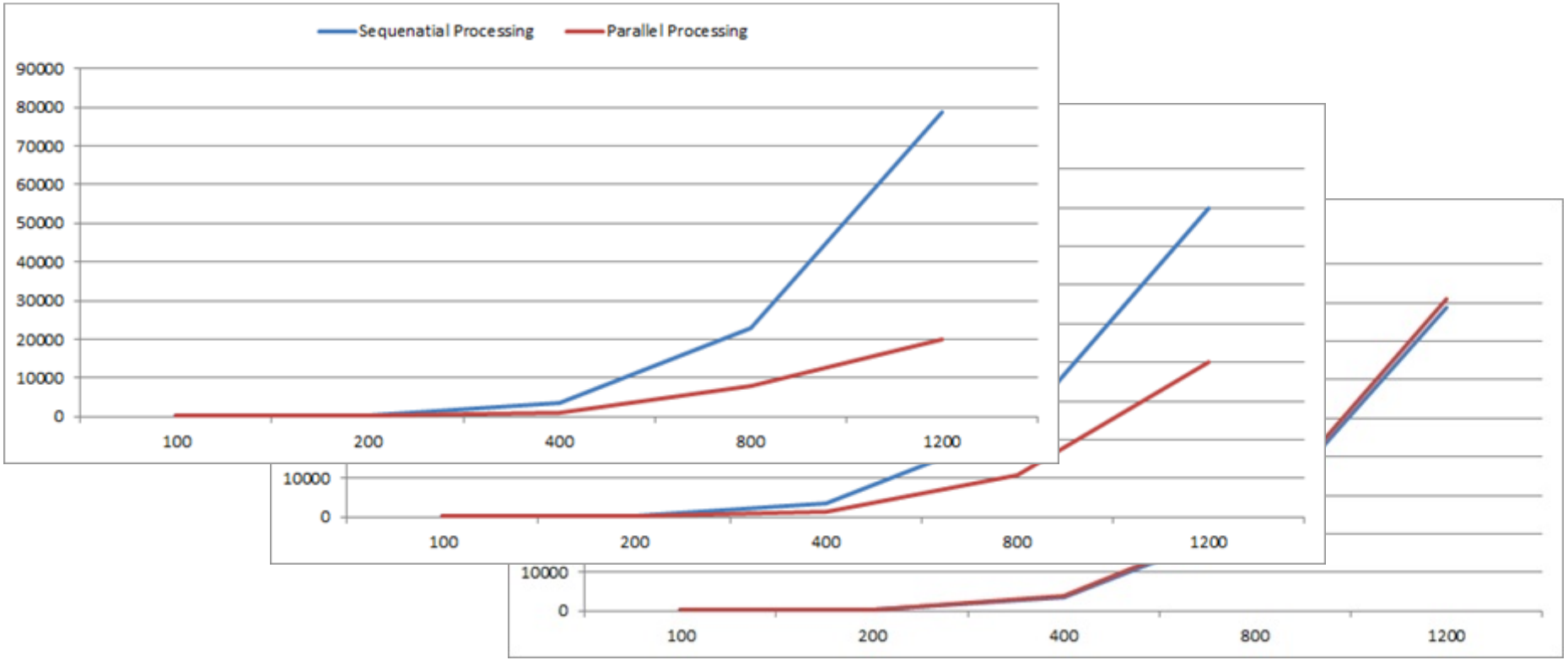
```
// Process all items in collection via foreach
foreach (var item in sourceCollection)
{
    Process(item);
}
```

```
// Process all items in collection via Parallel.ForEach
Parallel.ForEach(sourceCollection, Process);
```



Parallel Extensions

On a Quad Core mashine



THANK YOU!