Ejecución en Paralelo

El propósito de la TPL(Task Parallel Library, biblioteca de procesamiento paralelo basado en tareas) es aumentar la productividad de los desarrolladores simplificando el proceso de agregar paralelismo y simultaneidad a las aplicaciones.

La TPL escala el grado de simultaneidad de manera dinámica para usar con mayor eficacia todos los procesadores disponibles.

Además, la TPL se encarga de la división del trabajo, la programación de los subprocesos en ThreadPool, la compatibilidad con la cancelación, la administración de los estados y otros detalles de bajo nivel.

Al utilizar la TPL, el usuario puede optimizar el rendimiento del código mientras se centra en el trabajo para el que el programa está diseñado

Parallel.Invoke()

Esta parte esta mas vista y no es tan novedosa porque se ha realizado con `Task` y `Thread`, consiste en ejecutar una seria de funciones o métodos simultáneamente en hilos independientes.

```
Parallel.Invoke(
    () => Metodo1(),
    () => Metodo2(2)
);
```

Solamente con esa instrucción, el, automáticamente lee los métodos y los ejecuta de manera simultanea, asique simplifica la creación de `Task` o `Threads`

Parallel.For()

Este método consiste en, mediante `Task`, crear un hilo de ejecución por cada iteración que tenga que realizar el for, osea que si tienes que recorrer una lista de 100 elementos, creara 100 hilos los cuales se repartirán entre los diferentes núcleos del procesador y se añadirán a la cola para ser ejecutados.

```
Parallel.For(0, 100, (i, state) => {
    Console.WriteLine($"LAMBDA  --  {i}, {state.IsStopped}");
});
```

- El primer parámetro del método se envía el numero por el que se empieza
- El segundo parámetro se envía el numero final de la iteración

- En el tercer parámetro se envían 1 o 2 parámetros
    - `int`: que contendrá el número por el que va la iteración
    - `ParallelLoopState`: un objeto que se encargara de gestionar los estados de los hilos, pudiendo parar la ejecución, etc.

Semejanzas de bucle `for` tradicional(secuencial) con el Parallel:

```
//tradicional
for(int x = 0; x < 100; x++){
    // Código
}
```

- `int x = 0;` Este sería el primer parámetro del bucle Paralelo
- `x < 100;` Este seria el segundo parámetro, pero solo habría que mandar el 100
- `x++` No es necesario
- `{ //código }` Lo que va en este apartado, iría en la lambda del otro, osea, el tercer parámetro

Parallel.ForEach()

El bucle ForEach funciona igual que el For, pero lee los objetos que implementan la interfaz `IEnumerable`, como `List<T>`, `Dictionary<T1, T2>`, etc.

Funciona de la misma manera, creando un hilo por cada iteración y se escribe de la siguiente forma

```
Parallel.ForEach(listas, (linea, state) => {
    Console.WriteLine($"ForEach LAMBDA  --  {linea}");
});
```

- El primer parámetro se envía el objeto que queremos leer, un `List<string>` por ejemplo
- El segundo parámetro va la lambda que puede recibir dos parámetros
    - `obj` contendrá un objeto del tipo de la lista y solo 1 elemento de dicha lista, es lo mismo que si a un array le hacemos un `objetoArray[x]` con un `for` normal
    - `ParallelLoopState` un objeto que se encargara de gestionar los estados de los hilos, pudiendo parar la ejecución, etc.