Parallel Programming and Multithreading in C#

Filip Ekberg

Principal Consultant & CEO

@fekberg   fekberg.com

Using the Task from the Task Parallel Library

```
Task.Run(() => {
   // This code will execute on a different context
});
```

The Task from the Task Parallel Library

Only executes on one thread

```
Task.Run(() => {
});
```

Break down a large **problem** and **compute** each piece **independently**

Task Parallel Library

```
await Task.Run(() => {
   // I’m an asynchronous operation that is awaited
});

Parallel.Invoke(
   () => { /* Parallel Thread 1 */ },
   () => { /* Parallel Thread 2 */ },
   () => { /* Parallel Thread 3 */ },
   () => { /* Parallel Thread 4 */ },
);
```

Running Work on Another Thread

```
Task.Run(() => {
    var msft  = Calculate(“MSFT”);
    var googl = Calculate(“GOOGL”);
    var ps    = Calculate(“PS”);
    var amaz  = Calculate(“AMAZ”);

    return new [] { msft, googl, ps, amaz };
});
```

Running Work on Another Thread

```
Task.Run(() => {
    var msft  = Calculate(“MSFT”);
    var googl = Calculate(“GOOGL”);
    var ps    = Calculate(“PS”);
    var amaz  = Calculate(“AMAZ”);

    return new [] { msft, googl, ps, amaz };
});
```

Goal is to run these on 4 different threads

Get **help from** the **framework** to **optimize** the **parallel operation**

Parallel Programming in .NET

Thread

Task Parallel Library

Task Parallel Library and its **Task** should be the **preferred way** to introduce **parallel programming**

```
Task.Run(() => {
});
```

No need to care about lower-level threads

Work may be scheduled on a new, or reused thread.

Parallel Programming with Task Parallel Library

```
Parallel.Invoke(
    () => {},
    () => {},
    () => {}
);
```

Parallel Programming with Task Parallel Library

```
Parallel.Invoke(
    () => {},
    () => {},
    () => {}
);

Parallel.For(0, 10, (index) => {});
```

Parallel Programming with Task Parallel Library

```
Parallel.Invoke(
    () => {},
    () => {},
    () => {}
);

Parallel.For(0, 10, (index) => {});

Parallel.ForEach(source, (element) => {});
```

Task Parallel Library provides a way to write **Parallel LINQ** (**PLINQ**)

Parallel (Extensions)

Built on-top of the Task in the Task Parallel Library

A Problem to Solve in Parallel

Parallel will **ensure** that **work** is **distributed efficiently** on **the system** that **runs** the **application**

When to Use Parallel Programming

CPU bound operations

Independent chunks of data

Use the Parallel Methods

```
Task.Run(() => { });
Task.Run(() => { });
Task.Run(() => { });
Task.Run(() => { });

Parallel.For(0, 4, (i) => {});
```

Automatically distribute the work

There's **no guarantee** that the **operations** will **run** in **parallel**

Your First Parallel Operation

The Parallel Methods Blocks the Calling Thread

```
Parallel.Invoke(...);

Parallel.For(...);

Parallel.ForEach(...);
```

Block the calling thread until all the parallel operations completed

Deadlocking with the Parallel Class

```
Parallel.For(0, 4, (index) => {

    Dispatcher.Invoke(() => {
        // Run on the UI Thread
    });

});
```

This causes a deadlock!

By default calling these **Parallel** methods will **consume as much computer power** as **possible**

Parallel Invoke

```
Parallel.Invoke(



    () => { /* Parallel Thread 1 */ },
    () => { /* Parallel Thread 2 */ },

    () => { /* Parallel Thread 3 */ },
    () => { /* Parallel Thread 4 */ }
);
```

Parallel Invoke with Max Degree of Parallelism

```
Parallel.Invoke(
    new ParallelOptions { MaxDegreeOfParallelism = 2
}

    () => { /* Parallel Thread 1 */ },
    () => { /* Parallel Thread 2 */ },

    () => { /* Parallel Thread 1 */ },
    () => { /* Parallel Thread 2 */ }
);
```

Misusing **Parallel** in **ASP.NET** can cause **bad performance** for **all users!**

Next: Using Parallel and Asynchronous Principles Together

Using Parallel and Asynchronous Principles Together

Task Parallel Library

```
await Task.Run(() => {
    // I'm an asynchronous operation that is awaited
});

Parallel.Invoke(
    () => { /* Parallel Thread 1 */ },
    () => { /* Parallel Thread 2 */ },
    () => { /* Parallel Thread 3 */ },
    () => { /* Parallel Thread 4 */ },
);
```

Don’t reinvent this, **use** the **Task Parallel Library!**

Next: Handling Exceptions

Handling Exceptions

Handling Exceptions

```
Parallel.Invoke(...);

Parallel.For(...);

Parallel.ForEach(...);
```

Automatically validates the parallel operations.

This Will Throw an Aggregate Exception

```
Parallel.Invoke(
    () => { throw new Exception(“1”); },
    () => { throw new Exception(“2”); },
    () => { throw new Exception(“3”); },
    () => { throw new Exception(“4”); },
);
```

Not yet **executed parallel operations** will **not** be **cancelled** just because one **operation fails**

Next: Processing a Collection of Data in Parallel

Processing a Collection of Data in Parallel

Normal Foreach vs Parallel.ForEach

```
foreach(var element in source)
{
    // Execute sequentially
}

Parallel.ForEach(source, (element) => {
    // Execute in parallel
});
```

Normal Foreach vs Parallel.ForEach

```
foreach(var element in source)
{
    // Execute sequentially
}

Parallel.ForEach(source, (element) => {
    // Execute in parallel
});
```

Automatically distributed work that runs in parallel

The **performance benefits** will be **more obvious** with **larger collections** to **process**

Break **won’t** automatically

stop

running operations

Example: ParallelLoopState.Break()

```
Parallel.For(0, 100, (i, state) => {

    if(i == 50)
    {
        state.Break();
    }

});
```

Scheduled iterations for indices lower than 50 will still start!

Only operations for indices over 50 won’t be scheduled to start

Normal For vs Parallel.For

```
for(int i = 0; i < 10; i++)
{
    // Execute sequentially
}

Parallel.For(0, 10, (i) => {
    // Execute in parallel
});
```

Automatically distributed work that runs in parallel

Example: Parallel.For

```
Parallel.For(0, 10, (i, state) => {


});
```

Example: Parallel.For

Inclusive

```
Parallel.For(0, 10, (i, state) => {


});
```

Exclusive

Creating Parallel Operations

```
Parallel.Invoke(...);

Parallel.For(...);

Parallel.ForEach(...);
```

Parallel.ForEach

```
Parallel.ForEach(source, (element) => {
    // Execute in parallel
});
```

Automatically distributed work that runs in parallel

The Parallel Methods Blocks the Calling Thread

```
Parallel.Invoke(...);

Parallel.For(...);

Parallel.ForEach(...);
```

Block the calling thread until all the parallel operations completed

Parallel + Asynchronous

```
await Task.Run(() => { Parallel.Invoke(...); });

await Task.Run(() => { Parallel.For(...); });

await Task.Run(() => { Parallel.ForEach(...); });
```

Summary

Implications of parallelism

Difference and similarities between parallel and asynchronous programming

Builds on-top of the Task in the Task Parallel Library

Works in any C# and .NET application

Every problem and machine won’t benefit from parallelism

Break down a problem in small pieces and solve them independently

Use thread-safe collections like ConcurrentBag<T>