Using Parallel LINQ (PLINQ)

Filip Ekberg
Principal Consultant & CEO

@fekberg fekberg.com

Parallelize your **LINQ** to **speed** up the **execution**

Parallel LINQ

“Parallel implementation of the Language-Integrated Query (**LINQ**) pattern”

https://learn.microsoft.com/en-us/dotnet/standard/parallel-programming/introduction-to-plinq

Parallel LINQ

Works with Any LINQ Flavor

Method Syntax

```
var result = source
     .Select(Compute)
     .Sum();
```

Query Syntax

```
var result =
     (from i in source
     select Compute(i))
    .Sum();
```

Parallel LINQ

May result in a much faster execution

PLINQ will perform an internal analysis on the query to determine if it's suitable for parallelization

Construct a Parallel Query from **IEnumerable<T>**

```
IEnumerable<T> source = ...
```

Construct a Parallel Query from **IEnumerable<T>**

```
IEnumerable<T> source = ...

ParallelQuery<T> query = source.AsParallel();
```

Example: Sequential Query

```
var source = new [] { 1, 2, 3, 4 };
var query = source
            .Select(Compute);
var result = query.Sum(); // Execute query
```

Example: Parallel Query

```
var source = new [] { 1, 2, 3, 4 };

var query = source
            .AsParallel()
            .Select(Compute);

var result = query.Sum(); // Execute query
```

Example: Parallel Query

```
var source = new [] { 1, 2, 3, 4 };

var query = source
            .AsParallel()
            .Select(Compute);

var result = query.Sum();
```

Use all available resources to process the query

Configure a Parallel Query

```
var query = source
    .AsParallel()




    .Select(Compute);
```

Configure a Parallel Query

```
var query = source
    .AsParallel()
    .WithCancellation(token)



    .Select(Compute);
```

Configure a Parallel Query

```
var query = source
    .AsParallel()
    .WithCancellation(token)
    .WithDegreeOfParallelism(2)


    .Select(Compute);
```

Configure a Parallel Query

```
var query = source
    .AsParallel()
    .WithCancellation(token)
    .WithDegreeOfParallelism(2)
    .WithExecutionMode(ParallelExecutionMode.ForceParallelism)
    .WithMergeOptions(ParallelMergeOptions.Default)
    .Select(Compute);
```

Don’t overuse **AsParallel()** as it adds **overhead**

Parallel to Sequential

```
ParallelQuery<T> pquery = source.AsParallel();

IEnumerable<T> squery =  query.AsSequential();
```

Example: Parallel + Sequential

```
var query = source
            .AsParallel()
            .Select(Compute)
            .AsSequential()
            .Select(...);
```

Example: Parallel + Sequential

```
var query = source
            .AsParallel()
            .Select(Compute)
            .AsSequential()
            .Select(...);
```

This runs in Parallel

This does not run in Parallel

All your **LINQ** operation **can't** be **parallelized**

Will It Run in Sequentially or in Parallel?

Faster to run sequentially?

Then it will not run in parallel.

Unsafe to run in parallel?

Then it will not run in parallel.

Forcing Parallelism

```
var query = source
    .AsParallel()
    .WithExecutionMode(ParallelExecutionMode.ForceParallelism)
    .Select(Compute);
```

Only force parallelism if **you** are **absolutely certain** it **will** run **faster**

AsParallel()

Don’t assume queries will automatically run faster

Performance improvement noticeable on large collections

Considering locking **best practices** is **important** for **PLINQ** as well

Creating a Parallel Language Integrated Query

It's not always as easy as adding AsParallel()

Sequential When Queries Contain

- ✓ Select, indexed Where, indexed SelectMany, or ElementAt clause after an ordering or filtering operator that has removed or rearranged original indices
- ✓ Take, TakeWhile, Skip, SkipWhile operator and where indices in the source sequence are not in the original order
- ✓ Zip or SequenceEquals, unless one of the data sources has an originally ordered index and the other data source is indexable
- ✓ Concat, unless it is applied to indexable data sources
- ✓ Reverse, unless applied to an indexable data source

Source: learn.microsoft.com/en-us/dotnet/standard/parallel-programming/understanding-speedup-in-plinq

Ordered Parallel Query

```
var source = new [] { 1, 2, 3, 4 };

var query = source
              .AsParallel()
              .AsOrdered() // Persist the order
              .Select(Compute);
```

Parallel Operations with the Task Parallel Library

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

Parallel.For(0, 100, (i) => {});
Parallel.ForEach(elements, (e) => {});
Parallel.Invoke(() => {});

elements.AsParallel()
        .ForAll((e) => {});
```