izing Async/Await Performance in C#: Concurrency / Parallel Execution

The Challenge

I had two CPU-bound asynchronous methods:

1. GenerateLargeTextAsync()

2. CalculatePrimesAsync(10_000)

My goal was to run both tasks in parallel to improve performance.

First Approach: Using ConfigureAwait(false)

```
public async Task ConfigureAwaitFalseCpuBound() {
 var task1 = GenerateLargeTextAsync();
 var task2 = CalculatePrimesAsync(10_000);
 var result1 = await task1.ConfigureAwait(false);
 var result2 = await task2.ConfigureAwait(false);
 }
```

Second Approach: Using Task.WhenAll

```
public async Task AsyncCpuBound() {
 var task1 = GenerateLargeTextAsync();
 var task2 = CalculatePrimesAsync(10_000);
 await Task.WhenAll(task1, task2);
 }
```

Performance Results (In your machine results would be different but similar)

ConfigureAwaitFalseCpuBound: 14.09 ms

AsyncCpuBound: 14.38 ms

Why there is a such a difference ?

Answer:

Avoiding Synchronization Context: Using ConfigureAwait(false) prevents capturing the synchronization context, reducing overhead and slightly improving performance.

Parallel Execution: By starting both tasks before awaiting them, they run concurrently, utilizing resources more efficiently.

You can learn more about this and many other interesting topics in my upcoming book.

#dotnet #csharp #asyncprogramming #coding