Noter til "Concurrency - Parallel Tasks"

2. Task Basics

- En task er en isoleret, logisk enhed af arbejde.
- Tasks er ikke tråde, men sekventielle operationer.
- Brug `System.Threading.Tasks`.

3. Start af Tasks

- Brug `Parallel.Invoke`, `Task.Run` eller `Task.Factory.StartNew` til at starte tasks.
- Eksempel:

```
csharp
Kopier kode
public void DoAll()
{
    Task t1 = Task.Run((Action) DoLeft);
    Task t2 = Task.Run((Action) DoRight);
    Task.WaitAll(t1, t2);
}
```

4. Vent på Task Færdiggørelse

- Brug `Task.Wait`, `Task.WaitAll` eller `Task.WaitAny` for at vente på task færdiggørelse.
- Eksempel:

```
csharp
Kopier kode
public void DoAllUsingWait()
{
    Task t1 = Task.Run((Action)DoLeft);
    Task t2 = Task.Run((Action)DoRight);

    Task.Wait(t1);
    Task.Wait(t2);
}
```

5. Default Task Scheduler

- TPL bruger en task scheduler til at planlægge tasks.
- Worker tråde styres af `.NET ThreadPool`.
- Task Scheduler kan inline ventende tasks for at forbedre ydeevnen.

6. Passing Data til Tasks

- Brug closures eller state objects til at sende data til tasks.
- Eksempel med closures:

```
csharp
Kopier kode
public void DoWork()
{
    int data1 = 42;
    string data2 = "The Answer";
    Task.Run(() =>
    {
        Console.WriteLine(data2 + ": " + data1);
        });
    }
```

- Eksempel med state objects:

```
csharp
Kopier kode
class Work
{
    public int Data1;
    public string Data2;
    public void Run()
    {
        Console.WriteLine(Data1 + ": " + Data2);
    }
}

public static void Main()
{
    Work w = new Work { Data1 = 42, Data2 = "The Answer" };
    Task.Run(w.Run);
}
```

7. Task-Based Asynchronous Pattern (TAP)

- Brug `async` og `await` til at skrive asynkron kode.
- Eksempel:

```
csharp
Kopier kode
private async void btnGetHtml_Click(object sender, RoutedEventArgs e)
{
    tbxLength.Text = "Fetching...";
    string url = tbxUrl.Text;
    HttpClient client = new HttpClient();
    string text = await client.GetStringAsync(url);
    tbxLength.Text = text.Length.ToString();
}
```

Noter til "Concurrency - Dependencies, Futures"

1. Introduktion til Dependencies og Futures

- Forelæser: Henrik Bitsch Kirk
- Institution: Aarhus Universitet

2. Dependencies

- Dependencies kan være en udfordring for parallelisme.
- Brug af continuations til at håndtere dependencies:

```
csharp
Kopier kode
var f = Task.Factory;
var build1 = f.StartNew(() => Build("project1"));
var build2 = f.StartNew(() => Build("project2"));
var build3 = f.StartNew(() => Build("project3"));

var build4 = f.ContinueWhenAll(new[] { build1 }, x => Build("project4"));
var build5 = f.ContinueWhenAll(new[] { build1, build2, build3 }, x =>
Build("project5"));
var build6 = f.ContinueWhenAll(new[] { build3, build4 }, x =>
Build("project6"));
var build7 = f.ContinueWhenAll(new[] { build5, build6 }, x =>
Build("project7"));
var build8 = f.ContinueWhenAll(new[] { build5 }, x => Build("project8"));

Task.WaitAll(build1, build2, build3, build4, build5, build6, build7,
build8);
```

3. Futures

- Futures bruges til at få resultater fra tasks, når de bliver tilgængelige.
- Eksempel:

```
csharp
Kopier kode
static void Main()
{
    var a = "A";
    Task<string> futureB = Task.Run(() => F1(a));
    var c = F2(a);
    var d = F3(c);
    var f = F4(futureB.Result, d);
    Console.WriteLine(f);
}
```

4. Dependencies i Iterationer

- Iterationer kan have beregninger, hvor iteration i+1 afhænger af beregninger i iteration i.
- Parallelisering af beregninger inden for hver iteration er mulig.
- Eksempel på varmefordeling:

```
csharp
Kopier kode
static double[,] ParallelSimulation(int plateSize, int timeSteps)
{
    var prevIter = new double[plateSize, plateSize];
    var currIter = new double[plateSize, plateSize];

    for (var y = 0; y < plateSize; y++) prevIter[y, 0] = 255.0f;

    for (int step = 0; step < timeSteps; step++)
    {
        Parallel.For(1, plateSize - 1, y =>
        {
            for (int x = 1; x < plateSize - 1; x++)
            {
                currIter[y, x] = ((prevIter[y, x - 1] + prevIter[y, x + 1]
+ prevIter[y - 1, x] + prevIter[y + 1, x]) * 0.25f);
            }
        });
        Swap(ref prevIter, ref currIter);
    }
    return prevIter;
}
```