VPS 5	Parallel and Distributed Software Systems	WT 20/21, Exercise 5

Deadline: 15.02.2022 00:00

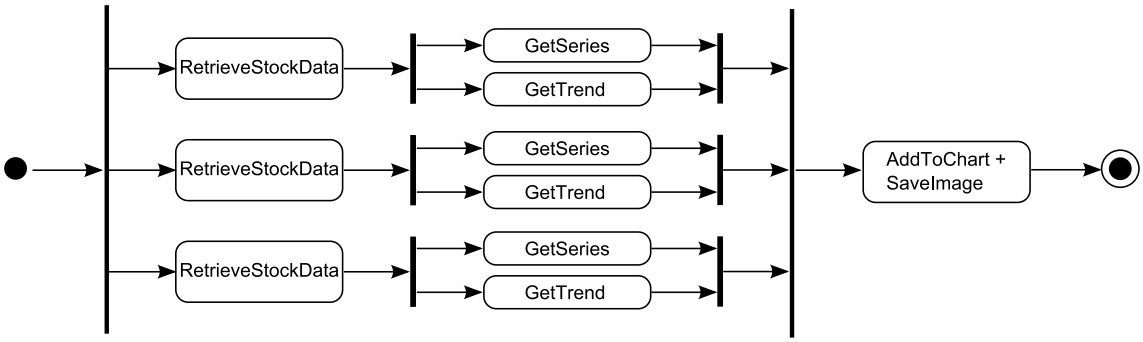
Name Maximilian Mitter

Points _____ **Effort in hours** 4h

1. Stock Data Visualization (6 + 6 Points)

Web requests can often be executed asynchronously to improve the responsiveness of an application and to reduce the total response time of multiple requests. On Moodle we provide a sequential implementation of a stock data visualization program that downloads price histories of three stocks from *quandl.com* and shows them in a line chart.

a) Implement an asynchronous version of the stock data visualization program using the .NET Task Parallel Library. Your version should execute tasks as shown in the following activity diagram. Use continuations to create chains of several tasks.



b) Implement a second version of the stock data visualization program which uses the keywords *async* and *await*. Make sure the tasks are again modeled as shown in the activity diagram.

2. Parallel Wator (8 + 4 Points)

Implement a parallel version of the Wator simulation from the first exercise using the .NET Task Parallel Library. You can parallelize either your already optimized version or the original version provided on Moodle.

a) Think about how you can separate data and work of the Wator simulation to allow parallel processing and consider appropriate synchronization. Describe and implement your decomposition and parallelization concept and explain your approach and its advantages and maybe also disadvantages.

b) Test your parallel Wator simulation, compare its runtime with your optimized version and the original version from Exercise 1 and calculate its speedup.

Exercise 5

Setup

- RAM: `16GB`
- CPU: `4x2.3GHz`
- OS: `Windows 10 Home`
- IDE: `Visual Studio 2022`

Stock Data Visualization

Sequential Implementation

The code-snippet below shows a parallel implementation of the data retrieval. Using the `.ContinueWith()` function one can define a sequence with which the tasks should be executed.

```
private void TaskImplementation() {
    var tasks = names.Select( // for each stock in list (start of async/parallel)
        name => RetrieveStockDataAsync(name) // retrieve data from service
            .ContinueWith((taskObj) => taskObj.Result.GetValues())
            .ContinueWith((t) => {
                var values = t.Result;
                var toSeriesTask = new[] {
                    GetSeriesAsync(values, name), GetTrendAsync(values, name)
                };

                Task.WaitAll(toSeriesTask);
                return toSeriesTask.Select(x => x.Result);
            })).ToArray();

    Task.WhenAll(tasks).ContinueWith((t) => {
        DisplayData(t.Result.SelectMany(x => x).ToList());
        SaveImage("chart");
    });
}
```

Async Implementation

Since async/await are just syntactic sugar making it easier for the developer, the functionality itself doesn't change, but the code is much shorter and better to read.

```
private async Task AsyncImplementation() {
    var tasks = names.Select(async (name) => {
        var stockData = await RetrieveStockDataAsync(name);
        var values = await stockData.GetValuesAsync();

        var seriesTask = GetSeriesAsync(values, name);
```

```
            var trendTask = GetTrendAsync(values, name);

            return new[] {
            await seriesTask, await trendTask
            };
        });

        var results = await Task.WhenAll(tasks);
        var seriesList = results.SelectMany(x => x).ToList();

        DisplayData(seriesList);
        SaveImage("chart async");
    }
```