Fachhochschule Nordwestschweiz
Hochschule für Technik

PARALLEL PROGRAMMING

Martin Kropp, Yves Senn
University of Applied Sciences Northwestern Switzerland

Fachhochschule Nordwestschweiz
Hochschule für Technik

Learning Targets

- You
  - can explain the purpose and functioning of the Parallel, Task and PLINQ concepts
  - can explain the differences and challenges of data and task parallelism strategies
  - can apply the parallel concepts for programming

Agenda

- Intro
- Threading
- Parallel
- Tasks

Fachhochschule Nordwestschweiz
Hochschule für Technik

Concepts

- Multithreading

  Use of multiple threads

- Concurrency

  Order in which multiple tasks execute is not determined

- Parallelism

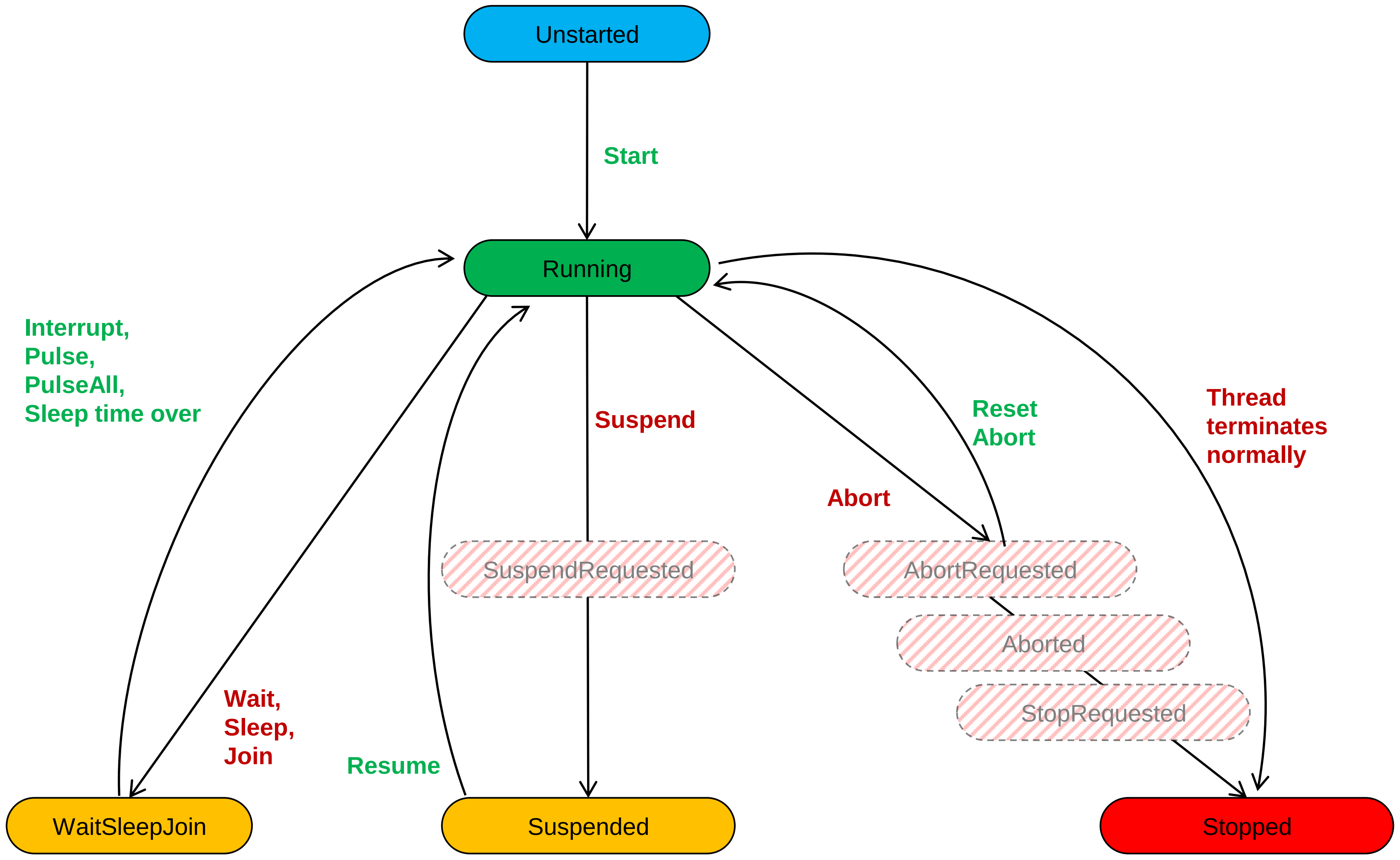
  Simultaneous execution (e.g. on multiple cores)

Multi-Threading example

```
class Printer {
    char ch; int sleepTime;

    public Printer(char c, int t) { ch = c; sleepTime = t; }

    public void Print() {
        for (var i = 0; i < 100; i++) {
            Console.Write(ch);
            Thread.Sleep(sleepTime);
        }
    }
}

class Test {
    static void Main() {
        var a = new Printer('.', 60);
        var b = new Printer('*', 70);
        new Thread(() => a.Print()).Start();
        new Thread(() => b.Print()).Start();
    }
}
```

Fachhochschule Nordwestschweiz
Hochschule für Technik

Thread states



Background threads

Two types of threads:

- *Foreground thread*
  Program will not terminate as long as at least one foreground thread is running

- *Background thread*
  Background threads do not prevent the program from terminating

```
var bgThread = new Thread(…);
bgThread.IsBackground = true;
bgThread.Start();
```

Passing data to a Thread

Be aware of closure

```
static void Main()
{
   var msg = "Hello";
   var thread = new Thread(() => Print(msg + " from t!"));
   thread.Start();
}

static void Print(string message)
{
   Console.WriteLine(message);
}
```

Thread pooling

- Thread creation requires quite some resources
  - ~1’000’000 clock cycles
  - About 1MB of memory *)
  - Requires kernel interaction

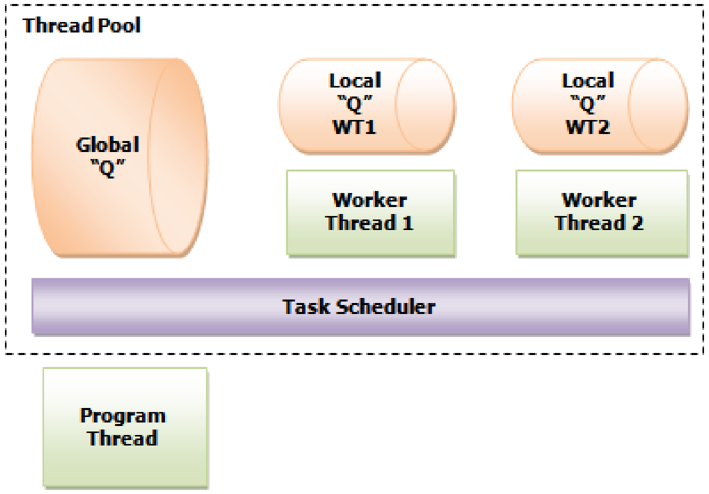
→ Recycle used threads

```
void SomeFunction(object o)
{
   //...
}

ThreadPool.QueueUserWorkItem(SomeFunction);
```

ThreadPool

`ThreadPool` offers automatic thread management and recycling:

- Number of threads is limited, additional requests are queued
- Used for short-running tasks
- Don’t change thread priority or thread state
- Background threads only
- Control over the thread only inside the method given (abort, etc…)



.NET threading API

- Number of Cores:
  ```
  Environment.ProcessorCount
  ```
- Starting a new thread:
  ```
  var t = new Thread(() => { /* ... */ });
  t.Start();
  ```
- Wait for another thread to finish:
  ```
  t.Join();
  ```
- Use a thread from the ThreadPool:
  ```
  ThreadPool.QueueUserWorkItem((o) => { /* … */ });
  ```

.NET threading API

- Mutual exclusion:
```
lock(someObject)
{
  //...
}
```

- Using semaphores:
```
var sem = new Semaphore(0, 3);
sem.WaitOne();
sem.Release();
```

- Using barriers:
```
var b = new Barrier(7);
b.SignalAndWait();
```

Using multithreading

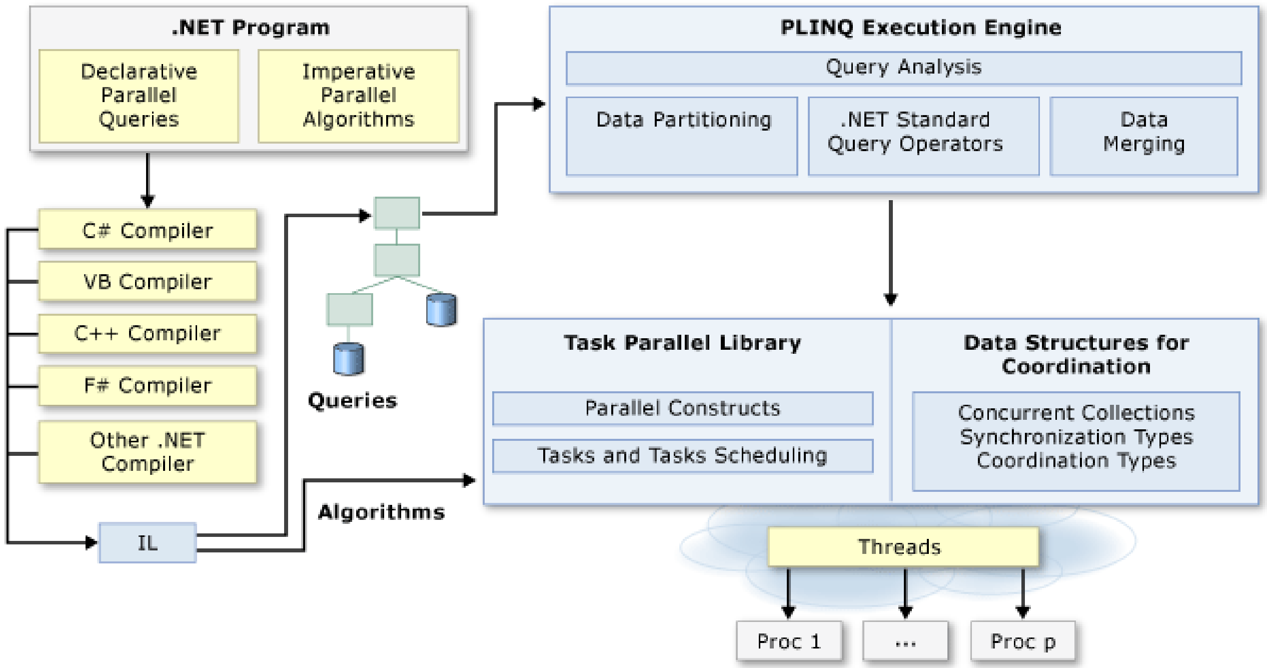
To make use of multiple cores, you have to

1. Split your algorithm into multiple parts
2. Execute parts in parallel via multithreading
3. Collate results from the multiple threads

Worksheet – Part 1

Parallel Extensions (PFX)

- Task Parallel Library (TPL) offers abstractions and reuse-approach over threads
- PLINQ allows to parallelize LINQ statements, based on the TPL



What PFX offers

- Provides high-level abstractions for parallel programming with `Parallel`, `Task`, PLINQ

- Takes care of
  - Partitioning
  - Parallel execution
  - Collation of results

Partitioning Strategies

Data parallelism

- The simultaneous execution of the **same function across split data a data set**
- Example: Processing 1000 elements; two cores work on 500 elements each
- Supported by TPL class **`Parallel`**

Task parallelism

- The simultaneous execution of **multiple and different functions** across the same or different data sets.
- Example: Sharpen and resize 1000 pictures; first task sharpens pictures, second task resizes pictures
- Supported by TPL class **`Task`**

Parallel.For

```
// sequential execution
for (var i = 0; i < 10; i++)
{
    Console.WriteLine(i);
}


// parallel execution
Parallel.For(0, 10, i =>
{
    Console.WriteLine(i);
});
```

Caveat: Execution order is unspecified.

Fachhochschule Nordwestschweiz
Hochschule für Technik

Parallel.ForEach

```
string[] capitals = { "London", "Paris", "..." };


// sequential execution
foreach (var city in capitals)
{
    Console.WriteLine(city);
}

// parallel execution
Parallel.ForEach(capitals, city =>
{
    Console.WriteLine(city);
});
```

Caveat: Execution order is unspecified.

Parallel.Invoke

```
Parallel.Invoke(  () => Function1(),
                  () => Function2(),
                  ...
                );
```

- Executes action delegates, possibly in parallel
- Invoke returns when all actions are finished

Caveat: Execution order is unspecified.

Fachhochschule Nordwestschweiz
Hochschule für Technik

Task.Run

```
Task<double>[] tasks = {
    Task.Run(() => DoComputation1()),
    Task.Run(() => DoComputation2())
};

var results = new double[tasks.Length];
for (var i = 0; i < tasks.Length; i++)
    results[i] = await tasks[i];
```

Caveat: Execution order is unspecified.

Worksheet – Part 2

PFX components

Component	Partitions work	Collates results
PLINQ	Yes	Yes
`Parallel`	Yes	No
`Task`	No	No

Note: These techniques are only relevant in CPU-bound scenarios

Using PLINQ

```
var parallelQuery = Enumerable.Range(3, 30)
    .Where(n => SomePredicate(n))
    .Sum(n => n * n);
```

```
var parallelQuery = Enumerable.Range(3, 30).AsParallel()
    .Where(n => SomePredicate(n))
    .Sum(n => n * n);
```

AsParallel vs ParallelEnumerable

- Types of parallelization:
  - Chunk Partitioning: chunk by chunk is processed

```
var parallelQuery = Enumerable.Range(3, 30).AsParallel()
    .Where(n => SomePredicate(n))
    .Sum(n => n * n);
```

  - Range Partitioning: range of work preassigned

```
var parallelQuery = ParallelEnumerable.Range(3, 30)
    .Where(n => SomePredicate(n))
    .Sum(n => n * n);
```

https://blogs.msdn.microsoft.com/pfxteam/2007/12/02/chunk-partitioning-vs-range-partitioning-in-plinq/

Worksheet – Part 3

PLINQ considerations

- Results are not in the same order as the input
  - Ordering can be forced with `.AsOrdered()`
  - Lift ordering requirement with `.AsUnordered()`

- Limit the number of threads with `.WithDegreeOfParallelism(4)`

- PLINQ only parallelizes work, if it suspects benefits.
  You can force parallelization with
  `.WithExecutionMode(ParallelExecutionMode.ForceParallelism)`

PLINQ considerations

Beware of code with side-effects:

```
//The following query multiplies each element by its
//position. It should output squares.
var i = 0;
var query = Enumerable.Range(0,999)
                      .AsParallel()
                      .Select(n => n * i++);
```

Closures

NEVER use functions with side-effects in (P)LINQ!

Side-effect-free functions

A function has “side-effects” when it modifies something outside the function

Side-effect-free examples: ✓

```
x => x * x
x => list[x]
```

Functions with side-effects:

```
x => list.Add(x)
x => File.Create(x+".txt")
```

…there's much more

- Thread-safe, scalable collections
  - `IProducerConsumerCollection<T>`
    - `ConcurrentQueue<T>`
    - `ConcurrentStack<T>`
    - `ConcurrentBag<T>`
  - `ConcurrentDictionary<TKey,TValue>`
- Phases and work exchange
  - `Barrier`
  - `BlockingCollection<T>`
  - `CountdownEvent`
- Partitioning
  - `{Orderable}Partitioner<T>`
    - `Partitioner.Create`
- Initialization
  - `Lazy<T>`
    - `LazyInitializer.EnsureInitialized<T>`
  - `ThreadLocal<T>`
- Locks
  - `ManualResetEventSlim`
  - `SemaphoreSlim`
  - `SpinLock`
  - `SpinWait`
- Cancellation
  - `CancellationToken{Source}`
- Exception handling
  - `AggregateException`

Tips

1. Use PLINQ
2. Use side-effect-free functions only
3. Avoid excessive locking
4. Prefer coarse-grained parallelization

Basic Resources

- Overview & Intro
  - MS Parallel Programming Home
    https://docs.microsoft.com/en-us/dotnet/standard/parallel-programming/
  - Threading in C# Part 5: Parallel Programming (Joseph Albahari):
    www.albahari.com/threading/part5.aspx
- Potential Pitfalls
  - http://msdn.microsoft.com/en-us/library/dd997392.aspx

Advanced Topics

- The Design of a Task Parallel Library
  - http://research.microsoft.com/apps/pubs/default.aspx?id=77368
- How C# Threads relate to OS Threads
  - https://github.com/dotnet/coreclr/blob/master/Documentation/botr/threading.md
- About side-effect free programming
  - https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/refactoring-into-pure-functions (Basic)
  - https://davesquared.net/2013/04/side-effect-free-csharp.html (Advanced)