Data-Parallel Operations I

Parallel Programming in Scala

Aleksandar Prokopec

Parallel Collections

In Scala, most collection operations can become data-parallel.

The `.par` call converts a sequential collection to a parallel collection.

```
(1 until 1000).par
  .filter(n => n % 3 == 0)
  .count(n => n.toString == n.toString.reverse)
```

Parallel Collections

In Scala, most collection operations can become data-parallel.

The `.par` call converts a sequential collection to a parallel collection.

```
(1 until 1000).par
  .filter(n => n % 3 == 0)
  .count(n => n.toString == n.toString.reverse)
```

However, some operations are not parallelizable.

Non-Parallelizable Operations

Task: implement the method `sum` using the `foldLeft` method.

```
def sum(xs: Array[Int]): Int
```

Non-Parallelizable Operations

Task: implement the method `sum` using the `foldLeft` method.

```
def sum(xs: Array[Int]): Int = {
  xs.par.foldLeft(0)(_ + _)
}
```

Does this implementation execute in parallel?

Non-Parallelizable Operations

Task: implement the method `sum` using the `foldLeft` method.

```
def sum(xs: Array[Int]): Int = {
  xs.par.foldLeft(0)(_ + _)
}
```

Does this implementation execute in parallel?

Why not?

Non-Parallelizable Operations

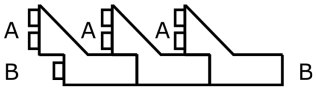
Let's examine the `foldLeft` signature:

```
def foldLeft[B](z: B)(f: (B, A) => B): B
```

Non-Parallelizable Operations

Let's examine the `foldLeft` signature:

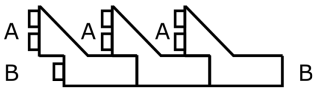
```
def foldLeft[B](z: B)(f: (B, A) => B): B
```



Non-Parallelizable Operations

Let's examine the `foldLeft` signature:

```
def foldLeft[B](z: B)(f: (B, A) => B): B
```



Operations `foldRight`, `reduceLeft`, `reduceRight`, `scanLeft` and `scanRight` similarly must process the elements sequentially.

The `fold` Operation

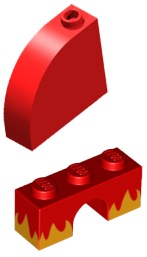
Next, let's examine the `fold` signature:

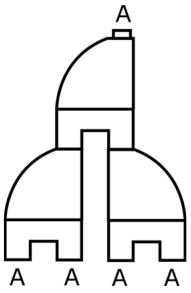
```
def fold(z: A)(f: (A, A) => A): A
```

The `fold` Operation

Next, let's examine the `fold` signature:

```
def fold(z: A)(f: (A, A) => A): A
```





The `fold` operation can process the elements in a reduction tree, so it can execute in parallel.