Updated: 2021-03-31

6.1 Parallel Collections

Northeastern University

Parallel Collections

- In module 4.8 (Futures), we mentioned parallel collections very briefly.
  - But there are times when they can be useful.
  - Here, we will talk about them in more detail.
  - Note that parallel collections are no longer part of the standard Scala library (as they were in 2.12).
  - You need to include in your *build.sbt*:

```
"org.scala-lang.modules" %% "scala-parallel-collections" % "1.0.2"
```

Parallel Collections: Documentation

- For all the detail on parallel collections, see https://github.com/scala/scala-parallel-collections.git
- And also the overview: https://docs.scala-lang.org/overviews/parallel-collections/overview.html
- For the API, you need to look here: https://javadoc.io/doc/org.scala-lang.modules/scala-parallel-collections_2.13/latest/scala/collection/index.html

What are the parallel types?

- ParIterable:
  - ParMap:
    - immutable.ParMap,
    - mutable.ParMap,
    - immutable.ParHashMap,
    - mutable.ParHashMap,
    - ParTrieMap
  - ParSeq:
    - ParRange
    - immutable.ParSeq,
    - mutable.ParSeq,
    - ParArray:
    - ParVector
  - ParSet:
    - mutable.ParSet
    - immutable.ParSet
    - mutable.ParHashSet
    - immutable.ParHashSet

Creating parallel collections

- The easiest thing to do is to import the collection conversions and invoke the *par* method on a sequential type to get the following parallel types:

Sequential	Parallel
mutable	
Array	ParArray
HashMap	ParHashMap
HashSet	ParHashSet
TrieMap	ParTrieMap
immutable	
Seq	ParSeq
Vector	ParVector
Range	ParRange
HashMap	ParHashMap
HashSet	ParHashSet

An example (in repo)

```scala
package edu.neu.coe.csye7200.asstwc.par
import scala.collection.parallel.CollectionConverters._
import scala.collection.parallel.immutable
import scala.language.postfixOps

object Parallel extends App {
  val m = 10
  val n = 10000000
  val expected: BigInt = (BigInt(2) * n * n * n + 3L * n * n + n) / 6
  println(s"Benchmark of sum of squares: N = $n with $m repetitions")
  val xs: List[Int] = LazyList from 1 take n toList
  val ys: List[BigInt] = xs map (x => BigInt(x) * x)
  val zs: immutable.ParSeq[BigInt] = ys.par
  val timeN = benchmark("Non-parallel", m, ys.sum)
  val timeP = benchmark("Parallel", m, zs.sum)
  println(s"speed up with parallelization is by a factor of ${((timeN / timeP - 1) * 100).toInt}%")

  def benchmark(message: String, m: Int, z: => BigInt) = {
    val (sum, time) = m times z
    if (sum == expected)
      println(s"$message: average time: $time mSecs")
    else {
      println(s"$message: error: $sum, expected: $expected")
    }
    time
  }
}
```

Results

- On my machine, I get results such as the following:

```
Benchmark of sum of squares: N = 10000000 with 10 repetitions
Non-parallel: average time: 433.1002735 mSecs
Parallel: average time: 129.0844142 mSecs
Speed up with parallelization is by a factor of 235%
```