Scala Performance Considerations

Nermin Serifovic

ferrari48.com

About me

- ~ 10 years of experience building Java enterprise apps
- currently a Principal Software Engineer at Constant Contact
- Scala enthusiast since 2009
- co-founded Boston Area Scala Enthusiasts

@higherkinded

Methodology

Stopwatch benchmarking

Followed usual micro-benchmark writing rules:

- Warm-up phase
- Test subroutine outside of `main` method
- Series of measurements

HotSpot VM in server mode

Scala optimise flag made no difference

Scala performance

“Run-time performance is *usually* on par with Java programs”

Tight loops - Java

```
int s = 0

for (int i = 1; i <= 2000; i++)
  for (int j = 1; j <= 2000; j++)
    for (int k = 1; k <= 2000; k++)
       s += 1
```

90 µs

Tight loops - Scala

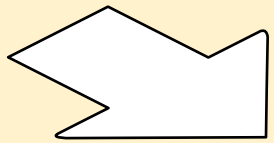
```
var s = 0

for (i <- 1 to 2000;
     j <- 1 to 2000;
     k <- 1 to 2000)
   s += 1
```

1.3 s

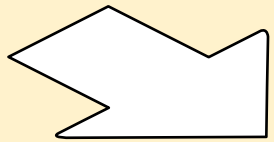
```
(1 to 2000).foreach(
  i => (1 to 2000).foreach(
    j => (1 to 2000).foreach(k => s += 1)
```

```
k => s += 1
```



```
new Function1[Int, Unit]
{
  def apply(k: Int): Unit
  {
    s += 1
  }
}
```

```
1 to 2000
```



```
Predef.intWrapper(1).to(2000)
```

```
trait Function1[
  @specialized(scala.Int, scala.Long,
               scala.Float, scala.Double) -T1,
  @specialized(scala.Unit,
               scala.Boolean,scala.Int,
               scala.Float,
               scala.Long,scala.Double) +R]
      extends AnyRef
```

```
def foreach[@specialized(Unit) U](f: Int => U)
```

Results in:

```
Predef.intWrapper(1).to(2000).
  foreach(new Function1[Int,Unit]
    { def apply(i: Int): Unit = {
      Predef.intWrapper(1).to(2000).
        foreach(new Function1[Int,Unit]
          { def apply(j: Int): Unit = {
            Predef.intWrapper(1).to(2000).
              foreach(new Function1[Int,Unit]
                { def apply(k: Int): Unit = {
                    s += 1
                  }})
          }})
    }})
```

Lots of overhead!

Ticket #1338: Optimize simple for loops

Tight loops - Scala

```
var s = 0
var i = 1; var j = 1; var k = 1

    while (i <= 2000) { j = 1
      while (j <= 2000) { k = 1
        while (k <= 2000) { k += 1 }
        j += 1 }
      i += 1 }
```

60 µs

For loop filtering

```
var a: Long  = 0

for (i <- 1 to 2000;
     j <- 1 to 2000;
     k <- 1 to 2000
     if ((i+j+k) % 777 == 0)) a += 1
```

96 s

For loop filtering

```
var a: Long  = 0

for (i <- 1 to 2000;
     j <- 1 to 2000;
     k <- 1 to 2000) {
  if ((i+j+k) % 777 == 0) a += 1
}
```

25 s

```
for (i <- 1 to 2000) if (i % 777 == 0) a += 1
```

Translates to:

```
Range(1, 2000).foreach(
  i => if (i % 777) a += 1)
```

Further translates to:

```
Range(1, 2000).foreach(
  new Function1[Int, Unit] {
    def apply(i: Int): Unit = {
      if (i % 777 == 0) a += 1
    }
  }
```

```
for (i <- 1 to 2000 if (i % 777 == 0)) a += 1
```

Translates to:

```
Range(1, 2000).withFilter(_ % 777 == 0)
  .foreach(_ => a += 1)

def withFilter(p: A => Boolean):
    FilterMonadic[A, Repr]
```

Further translates to:

```
Range(1,2000).withFilter(
  new Function1[Object, Boolean] {
    def apply(i: Object): Boolean = {
      apply(Int.unbox(i)) }

    def apply(i: Int): Boolean = {
      i % 777 == 0 }
  }).foreach(
    new Function1[Int, Unit] {
      def apply(i: Int): Unit = { a += 1}
    })
```

Code spends 78% of time executing:

```
scala.runtime.BoxesRunTime.boxToInteger
```

Optimizing Higher-Order Functions in Scala

search for: **scala optimization**

by Iulian Dragos

Performance management

- Fine-grained optimization
- Efficiency of the overall design
- Algorithms choice
- Measure, use a profiler
- When in doubt, decompile and/or disassemble.

Performance management

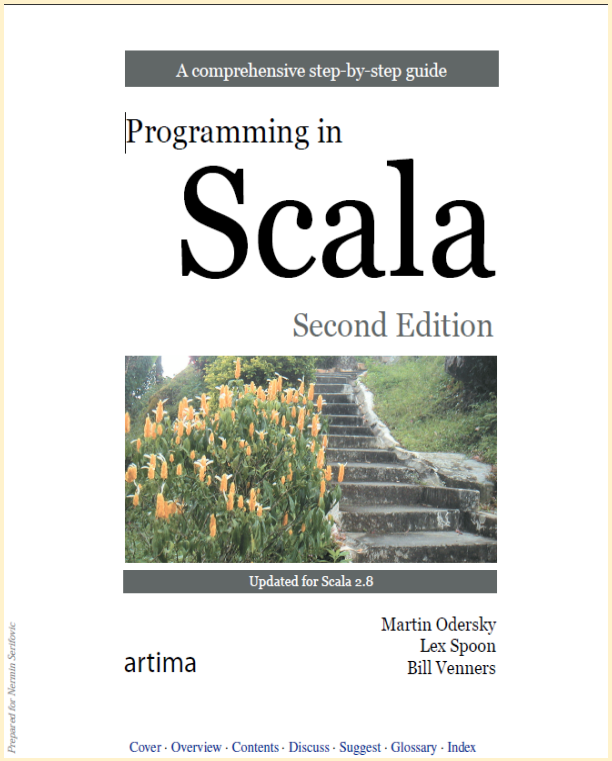
Scala is a good fit for areas which have big performance needs.

Examples:

- server side components
- infrastructure
- concurrency

Performance-critical blocks of code

Using the right collection



mutable vs. immutable

performance characteristics of different collection types

	head	tail	apply	update	prepend	append	insert
immutable							
`List`	C	C	L	L	C	L	-
`Stream`	C	C	L	L	C	L	-
`Vector`	eC	eC	eC	eC	eC	eC	-
`Stack`	C	C	L	L	C	L	-
`Queue`	aC	aC	L	L	L	C	-
`Range`	C	C	C	-	-	-	-
`String`	C	L	C	L	L	L	-
mutable							
`ArrayBuffer`	C	L	C	C	L	aC	L
`ListBuffer`	C	L	L	L	C	C	L
`StringBuilder`	C	L	C	C	L	aC	L
`MutableList`	C	L	L	L	C	C	L
`Queue`	C	L	L	L	C	C	L
`ArraySeq`	C	L	C	C	-	-	-
`Stack`	C	L	L	L	C	L	L
`ArrayStack`	C	L	C	C	aC	L	L
`Array`	C	L	C	C	-	-	-

Table 24.10 · Performance characteristics of sequence types

	lookup	add	remove	min
immutable				
`HashSet/HashMap`	eC	eC	eC	L
`TreeSet/TreeMap`	Log	Log	Log	Log
`BitSet`	C	L	L	eC^a
`ListMap`	L	L	L	L
mutable				
`HashSet/HashMap`	eC	eC	eC	L
`WeakHashMap`	eC	eC	eC	L
`BitSet`	C	aC	C	eC^a

Table 24.11 · Performance characteristics of set and map types

Tail call optimization

```
def sum(x: Int): Int = {
    if (x == 1) x
    else x + sum(x - 1)
  }

sum(1000)
```

770 µs

```
 @tailrec
 def sum(x: Int, total: Int = 0): Int = {
    if (x == 0) total
    else sum(x - 1, total + x)
  }
```

74 µs

Type Specialization

```
class My [A] {
  def iden(x: A): A = x
}

val a = new My[Int]
for (i <- 1 to 1000000) a.iden(i)
```

40 ms

Type Specialization

```
class My [@specialized(Int) A] {
  def iden(x: A): A = x
}

val a = new My[Int]
for (i <- 1 to 1000000) a.iden(i)
```

17 ms

Structural types

```
implicit def toInOperand[A](a: A) = new {
  def in(seq: Seq[A]) = seq.contains(a) }

def test {
  var a: Long = 0

  for (i <- 1 to 1000000) {
    if (i in List(7,77,777)) a += 1
    if (i.toString in
      List("8","88","888"))  a += 1
    if ((i * 2) in
      List (6, 66, 666)) a += 1
  }}
```

455 ms

Structural types

```
class InOperand[A](a: A) {
  def in(seq: Seq[A]) = seq.contains(a)
}

implicit def toInOperand[A](a: A) = new
InOperand(a)

def test {
  var a: Long = 0
  for (i <- 1 to 1000000) {
    if (i in List(7,77,777)) a += 1
    if (i.toString in
       List("8","88","888")) a += 1
    if ((i * 2) in List(6, 66, 666)) a += 1}}
```

403 ms

Structural types

```
implicit def toInOperand[A](a: A) = new {
    def in(seq: Seq[A]) = seq.contains(a) }
```

gets translated into:

```
def toInOperand[A](a: A): Object = {
  new Object { def in(seq: Seq[A]) =
    seq.contains(a) }
}
```

Functional Programming

```
val r = (1 to 1000000).
  map(_ * 2).
    filter(_ > 100000).
      reduceLeft(_ + _)
```

96 ms

Functional Programming

```
val r = (1 to 1000000).view.
  map(_ * 2).
    filter(_ > 100000).
      reduceLeft(_ + _)
```

120 ms

for Range:

```
override def foreach[@specialized(Unit) U]
(f: Int => U) {
    if (length > 0) {
        val last = this.last
        var i = start
        while (i != last) {
            f(i)
            i += step
        }
        f(i)
    }
}
```

```
def map [B] (f: (Int) => B): Range[B]
```

implemented as:

```
for (x <- this) b += f(x)
```

which gets translated to:

```
this.foreach(x => b += f(x))
```

In case of .view code spends 25% of time in scala.collection.immutable.Range.length:

```
final def apply(idx: Int): Int = {
      if (idx < 0 || idx >= length) throw new
IndexOutOfBoundsException(idx.toString)
      start + idx * step}


def length: Int = fullLength.toInt

protected def fullLength: Long =
  if (end > start == step > 0 && start != end)
    ((last.toLong - start.toLong) /
      step.toLong + 1) else 0
```

Functional Programming

```
var i = 1
var sum = 0

while (i <= 1000000) {
  if ((i * 2) > 100000) sum += (i * 2)
  i += 1
}
```

48 ms

Traits vs. Abstract Classes

```
javap -c

   ...
   12: invokeinterface  #26,  2; //InterfaceMethod
Similarity.isNotSimilar:(Ljava/lang/Object;)Z
   ...
```

```
   ...
   12: invokevirtual  #24; //Method PointA.
isNotSimilar:(Ljava/lang/Object;)Z
   ...
```

Thank you!

References

- Odersky, Spoon, Venners: Programming in Scala, 2nd edition
- Wampler, Payne: Programming Scala
- http://blog.danmachine.com/2011/01/moving-from-java-to-scala-one-year.html
- http://skillsmatter.com/podcast/scala/scala-performance-improvements-of-a-factor-of-4-to-30-boxing-and-specialization
- http://www.artima.com/underthehood/invocationP.html
- http://blog.richdougherty.com/2009/04/tail-calls-tailrec-and-trampolines.html
- ww.ibm.com/developerworks/java/library/j-jtp03253.html
- http://scala-programming-language.1934581.n4.nabble.com/For-loops-dead-slow-in-Scala-td1942400.html
- http://stackoverflow.com/questions/2529736/groovy-scala-java-under-the-hood
- http://stackoverflow.com/questions/2794823/is-scala-functional-programming-slower-than-traditional-coding
- http://stackoverflow.com/questions/4001347/scala-compiler-says-my-method-is-recursive-in-case-when-implicits-and-anonymous-c/4002686#4002686
- http://stackoverflow.com/questions/4577144/which-scala-features-are-internally-implemented-using-reflection
- http://stackoverflow.com/questions/4962618/do-abstract-classes-in-scala-really-perform-better-than-traits
- http://stackoverflow.com/questions/4982529/how-to-recognize-boxing-unboxing-in-a-decompiled-scala-code
- http://stackoverflow.com/questions/504103/how-do-i-write-a-correct-micro-benchmark-in-java