Introdução à Programação em Scala

João Rafael Moraes Nicola

Programa de Pós-Graduação em Informática
Departamento de Informática
Universidade Federal do Espírito Santo

24 de junho de 2015

UFES

Indice

1. Outra linguagem ?!
2. Scala
   - De Java para Scala
   - Scala além do Java
   - Coleções
3. APIs Scala
4. Paradigma Funcional $\times$ Paradigma Imperativo

UFES

Para quê aprender uma outra linguagem?

- Todas as linguagens de uso geral são igualmente expressivas (Máquina de Turing)
- O mercado de TI no Brasil exige primariamente conhecimento em Java.
- A linguagem de programação em si não é importante. O importante é o processo de desenvolvimento, qualidade das ferramentas, bibliotecas, etc.
- Já me formei, não quero estudar mais.

UFES

Para quê aprender uma outra linguagem?

- Todas as linguagens de uso geral são igualmente expressivas (Máquina de Turing)
- O mercado de TI no Brasil exige primariamente conhecimento em Java.
- A linguagem de programação em si não é importante. O importante é o processo de desenvolvimento, qualidade das ferramentas, bibliotecas, etc.
- Já me formei, não quero estudar mais.

UFES

Para quê aprender uma outra linguagem?

- Todas as linguagens de uso geral são igualmente expressivas (Máquina de Turing)
- O mercado de TI no Brasil exige primariamente conhecimento em Java.
- A linguagem de programação em si não é importante. O importante é o processo de desenvolvimento, qualidade das ferramentas, bibliotecas, etc.
- Já me formei, não quero estudar mais.

UFES

Para quê aprender uma outra linguagem?

- Todas as linguagens de uso geral são igualmente expressivas (Máquina de Turing)
- O mercado de TI no Brasil exige primariamente conhecimento em Java.
- A linguagem de programação em si não é importante. O importante é o processo de desenvolvimento, qualidade das ferramentas, bibliotecas, etc.
- Já me formei, não quero estudar mais.

UFES

Para quê aprender uma outra linguagem?

Expressividade

Todas as linguagens de uso geral são igualmente expressivas

1. Se distinguem sintaticamente
2. Se distinguem pragmáticamente

Para quê aprender uma outra linguagem?

Expressividade

Todas as linguagens de uso geral são igualmente expressivas

1. Se distinguem sintaticamente
2. Se distinguem pragmáticamente

UFES

Para quê aprender uma outra linguagem?

Expressividade

Todas as linguagens de uso geral são igualmente expressivas

1. Se distinguem sintaticamente
2. Se distinguem pragmáticamente

Para quê aprender uma outra linguagem?

Mercado

O mercado de TI no Brasil exige primariamente conhecimento em Java.

1. O mercado muda e os paradigmas também.
2. A 3 anos atrás, Cobol era (ou é ainda) a linguagem mais usada nos ambientes corporativos.
3. Vamos cometer o mesmo erro de nossos pais (na profissão)?

UFES

Para quê aprender uma outra linguagem?

Mercado

O mercado de TI no Brasil exige primariamente conhecimento em Java.

1. O mercado muda e os paradigmas também.
2. A 3 anos atrás, Cobol era (ou é ainda) a linguagem mais usada nos ambientes corporativos.
3. Vamos cometer o mesmo erro de nossos pais (na profissão)?

Para quê aprender uma outra linguagem?

Mercado

O mercado de TI no Brasil exige primariamente conhecimento em Java.

1. O mercado muda e os paradigmas também.
2. A 3 anos atrás, Cobol era (ou é ainda) a linguagem mais usada nos ambientes corporativos.
3. Vamos cometer o mesmo erro de nossos pais (na profissão)?

Para quê aprender uma outra linguagem?

Mercado

O mercado de TI no Brasil exige primariamente conhecimento em Java.

1. O mercado muda e os paradigmas também.
2. A 3 anos atrás, Cobol era (ou é ainda) a linguagem mais usada nos ambientes corporativos.
3. Vamos cometer o mesmo erro de nossos pais (na profissão)?

Para quê aprender uma outra linguagem?

Processo

A linguagem de programação em si não é importante. O importante é o processo de desenvolvimento, qualidade das ferramentas, bibliotecas, etc.

Deve ser por isso que todo mundo ainda programa em FORTRAN e Assembler!

Para quê aprender uma outra linguagem?

Processo

A linguagem de programação em si não é importante. O importante é o processo de desenvolvimento, qualidade das ferramentas, bibliotecas, etc.

Deve ser por isso que todo mundo ainda programa em FORTRAN e Assembler!

Para quê aprender uma outra linguagem?

Preguiça

Já me formei, não quero estudar mais.

Essa desculpa vocês não têm!

Para quê aprender uma outra linguagem?

Preguiça

Já me formei, não quero estudar mais.

Essa desculpa vocês não têm!

UFES

Para quê aprender uma outra linguagem?

"As linguagens que falamos afetam nossas percepções sobre mundo" (Lera Boroditsky, 2011)

"É tentador, se a única ferramenta que você tiver for um martelo, tratar tudo como se fosse um prego." (Abraham Maslow, 1966)

`http://hammerprinciple.com/therighttool`

Índice

1. Outra linguagem ?!
2. Scala
   - De Java para Scala
   - Scala além do Java
   - Coleções
3. APIs Scala
4. Paradigma Funcional × Paradigma Imperativo

Hello World!

Java

```
class HelloWorld {
  public static void main(String[] args) {
    System.out.println("Hello World!")
  }
}
```

Scala

```
object HelloWorld extends App {
    println("Hello World!")
}
```

Hello World!

Java

```
class HelloWorld {
  public static void main(String[] args) {
    System.out.println("Hello World!")
  }
}
```

Scala

```
object HelloWorld extends App {
    println("Hello World!")
}
```

Classes

Java

```
package x.y.z;

public class A extends B implements C, D
```

Scala

```
package x.y.z

class A extends B with C with D
```

Classes

Java

```
package x.y.z;

public class A extends B implements C, D
```

Scala

```
package x.y.z

class A extends B with C with D
```

Interfaces

Java

```
public interface A extends B, C
```

Scala

```
trait A extends B with C
```

UFES

Interfaces

Java

```
public interface A extends B, C
```

Scala

```
trait A extends B with C
```

Atribuição

Java

```
Map<String,Integer> m =
    new HashMap<String,Integer>();
⋮
 final int xyz = 12;
```

Scala

```
var m = new HashMap[String,Integer]()
⋮
 val xyz = 12
```

Atribuição

Java

```
Map<String,Integer> m =
    new HashMap<String,Integer>();
⋮
 final int xyz = 12;
```

Scala

```
var m = new HashMap[String,Integer]()
⋮
 val xyz = 12
```

Sigletons

Java

```
public class T {
  private static T instance = null;
  public static T getInstance() {
    synchronized(this) {
      if(instance == null) {
        instance = new T();
      }
      return instance;
    }
  }
}
...
T obj = T.getInstance();
obj.method();
```

Sigletons

Scala

```
object T
...
T.method()
```

Métodos

Java

```
class C {
  public String metodo(int x, double y, Obj o) {
    ...
  }
}
```

Scala

```
class C {
  def metodo(x : Int, y : Double, o : Obj) : String = {
    ...
  }
}
```

Métodos

Java

```
class C {
  public String metodo(int x, double y, Obj o) {
    ...
  }
}
```

Scala

```
class C {
  def metodo(x : Int, y : Double, o : Obj) : String = {
    ...
  }
}
```

Classes de Domínio

Java

```
class Retangulo implements Serializable,
   Comparable<Retangulo> {
  private final double width;
  private final double height;
  public Retangulo(double width,
                   double height) {
    this.width = width;
    this.height = height;
  }
  public Double getWidth() {
    return width;
  }
  public Double getHeight() {
    return height;
  }
  public String toString() {
    return "Retangulo(" + width +
    "," + height + ")";
  }
  public boolean equals(Object O) {
    ...
  }
  public int compareTo(Retangulo r) {
    ...
  }
  public int hashCode() {
    ...
  }
}
```

Scala

```
case class Retangulo(
  width : Double, height : Double)
```

Classes de Domínio

Java

```
class Retangulo implements Serializable,
   Comparable<Retangulo> {
  private final double width;
  private final double height;
  public Retangulo(double width,
                   double height) {
    this.width = width;
    this.height = height;
  }
  public Double getWidth() {
    return width;
  }
  public Double getHeight() {
    return height;
  }
  public String toString() {
    return "Retangulo(" + width +
    "," + height + ")";
  }
  public boolean equals(Object O) {
    ...
  }
  public int compareTo(Retangulo r) {
    ...
  }
  public int hashCode() {
    ...
  }
}
```

Scala

```
case class Retangulo(
  width : Double, height : Double)
```

Índice

1. Outra linguagem ?!
2. Scala
   - De Java para Scala
   - Scala além do Java
   - Coleções
3. APIs Scala
4. Paradigma Funcional $\times$ Paradigma Imperativo

UFES

Inferência de tipos

Scala

```
val x = "aaaaaa" // x : String

val y = 12 +  x.length // y : Int

val m = Array("abc", "de", "f")
// m : Array[String]

val z = m map (_.length) // z : Array[Int]
...
```

Tuplas

Scala

```
def toPolar(x : Double, y : Double) =
   (math.sqrt(x*x+y*y), math.atan(x,y))
// toPolar :: (Double,Double) => (Double,Double)

val (mag,ang) = toPolar(3,4)
// mag : Double, ang : Double
```

Aplicação parcial (currying)

Scala

```
def quicksort[T](
      le_pred : (T,T) => Boolean)(
      data : Vector[T]) : Vector[T] =
  if(data.length <= 1)
   { data } else {
      val pivot = data.head
      val (le,g) =
        data.tail.partition(le_pred)
      quicksort(le_pred)(le) ++
        Vector(pivot) ++
        quicksort(g)
  }

//quicksort para strings
val quicksortString =
  quicksort[String](_ <= _)

//quicksort para ints (decrescente)
val quicksortInt =
  quicksort[Int](_ >= _)
```

Traits

Múltipla herança (*mixins*):

Scala

```
trait A {
  def metodo1(x : Int) =
    x + metodo2(x)
  def metodo2(x : Int) : Int
}

trait B {
  def metodo2(x : Int) =
    x * metodo3(x)
  def metodo3(x : Int) : Int
}

trait C {
  def metodo2(x : Int) = x * 5
}
```

```
class D extends A with B {
  def metodo3(x : Int) = x - 2
}

println(new D().metodo1(10))
//imprime: 90

println((new A with C).metodo1(10))
//imprime: 60
```

Direitos iguais

- Em Scala, todos os tipos herdam de `scala.Any`
- Não há distinção entre *Boxed* e *Unboxed* types.

Scala

```
(1 to 10) foreach println
//imprimes os números de 1 a 10
```

UFES

Direitos iguais

- Em Scala, todos os tipos herdam de `scala.Any`
- Não há distinção entre *Boxed* e *Unboxed* types.

Scala

```
(1 to 10) foreach println
//imprimes os números de 1 a 10
```

Direitos iguais

- Em Scala, todos os tipos herdam de `scala.Any`
- Não há distinção entre *Boxed* e *Unboxed* types.

Scala

```
(1 to 10) foreach println
//imprimes os números de 1 a 10
```

Casamento de padrões

Scala

```
abstract sealed class Expr
case class Val(n : Int) extends Expr
case class Plus(e1 : Expr, e2 : Expr) extends Expr
case class Minus(e1 : Expr, e2 : Expr) extends Expr
case class Times(e1 : Expr, e2 : Expr) extends Expr
case class Div(e1 : Expr, e2 : Expr) extends Expr

def eval(e : Expr) : Int = e match {
  case Val(n) => n
  case Plus(e1,e2) => eval(e1) + eval(e2)
  case Minus(e1,e2) => eval(e1) - eval(e2)
  case Times(e1,e2) => eval(e1) * eval(e2)
  case Div(e1,e2) => eval(e1) / eval(e2)
}

println(eval(
  Times(Val(5),Plus(Val(2),Val(3)))))
//Imprime: 25
```

Expressões lambda

Scala

```
val f1 = (x : Int, y : Int) => x + y
//f1 : (Int,Int) => Int
val f2 : (Int,Int) => Int = (_ + _)
//f2 = f1
val f3 : List[Int] => Int = {
    case List() => 0
    case List(x) => x
    case List(x,y) => x * y
    case l => l.sum
}
val f5 = (x : Int) => x * 10
val f6 = f3 andThen f5

println(f6(List(1,2,3,4,5)))
//imprime: 150
```

Call-by-name

Scala

```
class Logger {
  var level : Int = 2
  def info(msg : => String) =
    if (level > 3) { println(msg) } else { }
}

def costlyOp() = {
  Thread.sleep(5000)
  "done"
}

val logger = new Logger
logger.info(costlyOp())
//Não imprime nada nem executa
//a expressão do parâmetro

logger.level = 4
logger.info(costlyOp())
//Agora espera 5 segs e imprime "done"
```

Expressões Regulares

Scala

```
val re1 = "(\d\d\d\d)-(\d\d)-(\d\d)"r
val re2 = "(\d\d)/(\d\d)/(\d\d\d\d)"r

def parseDate(t : String) = t match {
  case re1(ano,mes,dia) => (ano,mes,dia)
  case re2(dia,mes,ano) => (ano,mes,dia)
  case _ =>
    throw new RuntimeException(
        s"Invalid date: ${t}")
}

println(parseDate("2009-12-02"))
println(parseDate("02/12/2009"))
```

Outras Características

Parser combinators, *Actors*, *Monads*, diversos tipos de DSLs embutidas (DB, ...), macros *higiênicas*, interpolação de strings, API moderna de coleções, generalização do comando *for*, recursão de cauda, co/contra-variância, parâmetros implícitos, conversões implícitas, *algebraic data types*, compilação para .NET e para Javascript, interpretador REPL (*Read-Evaluate-Print Loop*), *path-dependent types*, sinônimos de tipos, granularidade fina de controle aos membros da classe ...

UFES

Índice

1. Outra linguagem ?!
2. Scala
   - De Java para Scala
   - Scala além do Java
   - Coleções
3. APIs Scala
4. Paradigma Funcional $\times$ Paradigma Imperativo

UFES

Coleções

Scala

```
val x = (0 until 100).to[Vector]
val y = x map (_ + 5)
          filter (_ < 10)
val z1 = (2 to 5) flatMap
          (t => x map (t * _))
val z2 = for {
           a <- 0 until 100
           b = a + 5
           if b < 10
           t <- 2 to 5
         } yield (t * a)

def costlyOp(x : Int) = {
  Thread.sleep(
   (math.random * 100.0).toLong)
  x * 5
}
```

```
def chrono[T](op : => T) = {
  val s = System.nanoTime
  val res = op
  val e =
    (System.nanoTime-s)
     .toDouble / 1e9
  println(
     s"elapsed: ${elapsed} s")
  (elapsed,res)
}

val (st,_) = chrono {
  x map (costlyOp)
    foreach(_ => ())
}
val (pt,_) = chrono {
  x.par map (costlyOp)
    foreach(_ => ()))
}
println(
  s"speedup: ${seqTime/parTime}")
```

Coleções

Scala

```
val m = Map(1 -> "Joao", 2 -> "Pedro",
            3 -> "Andre")
val m1 = m map { case (x,y) => (y,x) }
println(s"m1 = ${m1}")
val s = (2 to 10).collect(m).to[Set]
val s2 = s - Set("Pedro")
println(s"s2 = ${s2}")
val m3 = (1 to 100).groupBy(_ % 5)
println("m3:")
m3 foreach {
  case (k,l) =>
   println(s"   $k: ${l.mkString(",")}")
}
```

Coleções

Scala

```
import java.io._
import scala.io.Source

val pw = new PrintWriter(
    new FileWriter("test.csv"))
List(1 to 5, 6 to 10, 11 to 15).
  foreach(l => pw.println(l.mkString("\t")))
pw.close()

val res = (
  Source.fromFile(new File("test.csv"))
    .getLines
    .map(_.split("\\t").filter(_.trim != ))
    .collect {
       case Array(x1,_,_,_,x2) =>
         (x1.toDouble,x2.toDouble)
     }
    .map { case (x1,x2) => x1 * x2 }
    .reduce(_ + _)
  )
println(res)
```

APIs

- Programs Scala tem acesso à todas APIs disponíveis na JVM
- Frequentemente é mais fácil usar as APIs pelo Scala
- Existe uma grande quantidade de APIs para o Scala que
  1. fazem uso dos recursos do Scala para construir DSLs
  2. exploram o sistema de tipos do Scala
  3. encaixam-se melhor no paradigma funcional

UFES

APIs

- Programs Scala tem acesso à todas APIs disponíveis na JVM
- Frequentemente é mais fácil usar as APIs pelo Scala
- Existe uma grande quantidade de APIs para o Scala que
  1. fazem uso dos recursos do Scala para construir DSLs
  2. exploram o sistema de tipos do Scala
  3. encaixam-se melhor no paradigma funcional

UFES

APIs

- Programs Scala tem acesso à todas APIs disponíveis na JVM
- Frequentemente é mais fácil usar as APIs pelo Scala
- Existe uma grande quantidade de APIs para o Scala que
  1. fazem uso dos recursos do Scala para construir DSLs
  2. exploram o sistema de tipos do Scala
  3. encaixam-se melhor no paradigma funcional

UFES

APIs

- Programs Scala tem acesso à todas APIs disponíveis na JVM
- Frequentemente é mais fácil usar as APIs pelo Scala
- Existe uma grande quantidade de APIs para o Scala que
  1. fazem uso dos recursos do Scala para construir DSLs
  2. exploram o sistema de tipos do Scala
  3. encaixam-se melhor no paradigma funcional

UFES

APIs

- Programs Scala tem acesso à todas APIs disponíveis na JVM
- Frequentemente é mais fácil usar as APIs pelo Scala
- Existe uma grande quantidade de APIs para o Scala que
  1. fazem uso dos recursos do Scala para construir DSLs
  2. exploram o sistema de tipos do Scala
  3. encaixam-se melhor no paradigma funcional

UFES

Frameworks Web

- Play! Framework
- Lift
- Scalatra
- Spray
- ...

UFES

Persistence

- Slick
- Sorm
- Squeryl
- Scala ActiveRecord
- ...

Testing

- ScalaCheck
- ScalaTest
- Specs2 — Software Specifications for Scala.
- Scalastyle
- ScalaMock – Scala native mocking framework
- ...

UFES

Distributed Systems

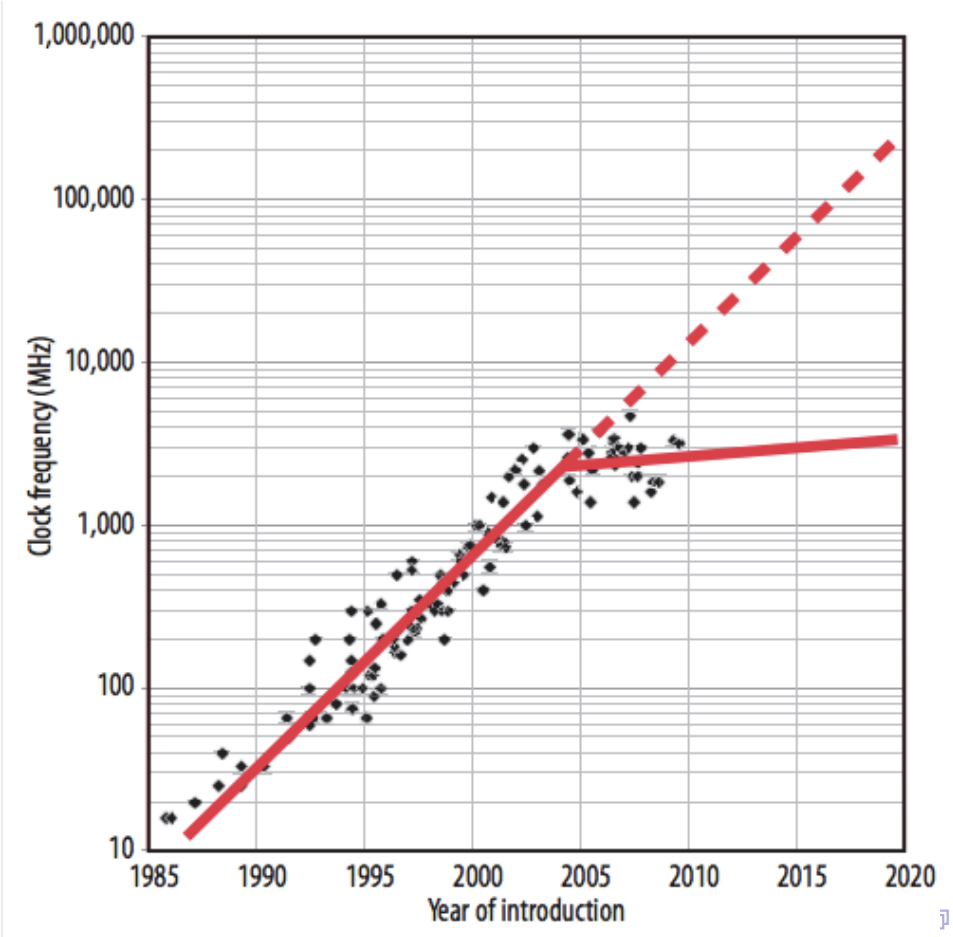
- Akka
- Finagle
- ...

Distributed Systems

- Akka
- Finagle
- ...

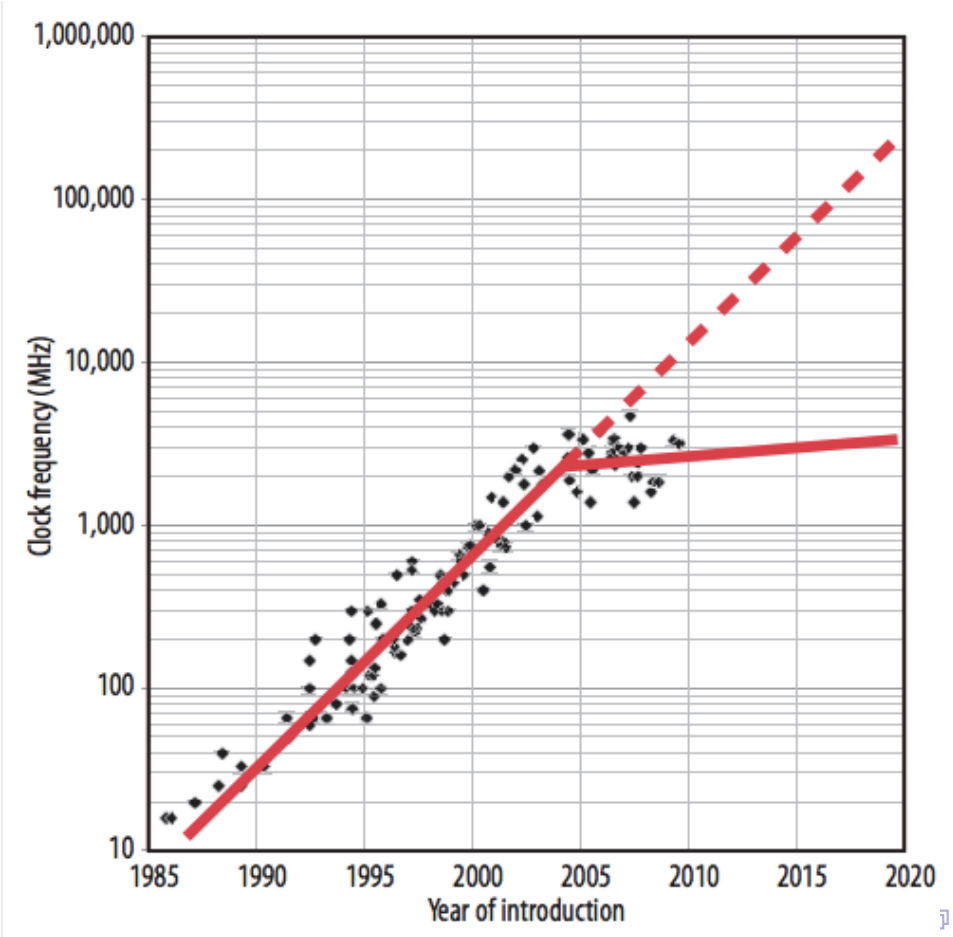
Lei de Moore

"O número de transístores dos chips dobra à cada 18 meses mantendo o mesmo custo" (Gordon Moore, 1965)



Lei de Moore

"O número de transístores dos chips dobra à cada 18 meses mantendo o mesmo custo" (Gordon Moore, 1965)



Paralelismo

- Processadores *Multi-core*
  1. Frequentes em celulares e tablets
  2. Lançamento de 2014: Intel Core i7-5960x, com 8 núcleos.
  3. Xeons com 18 núcleos
  4. SPARCs: 128 threads paralelas
- Limitação na velocidade por núcleo.

Solução: paralelizar o código!

UFES

Paralelismo

- Processadores *Multi-core*
  1. Frequentes em celulares e tablets
  2. Lançamento de 2014: Intel Core i7-5960x, com 8 núcleos.
  3. Xeons com 18 núcleos
  4. SPARCs: 128 threads paralelas
- Limitação na velocidade por núcleo.

Solução: paralelizar o código!

Paralelismo

- Processadores *Multi-core*
  1. Frequentes em celulares e tablets
  2. Lançamento de 2014: Intel Core i7-5960x, com 8 núcleos.
  3. Xeons com 18 núcleos
  4. SPARCs: 128 threads paralelas
- Limitação na velocidade por núcleo.

Solução: paralelizar o código!

UFES

Paralelismo

- Processadores *Multi-core*
  1. Frequentes em celulares e tablets
  2. Lançamento de 2014: Intel Core i7-5960x, com 8 núcleos.
  3. Xeons com 18 núcleos
  4. SPARCs: 128 threads paralelas
- Limitação na velocidade por núcleo.

Solução: paralelizar o código!

UFES

Paralelismo - Dificuldades

- Condições de concorrência
  1. Deadlocks, livelocks, etc.
  2. Sincronização
  3. Não-determinismo
  4. Caos!
- Melhores práticas:
  1. Evitar contenção
  2. Evitar mudança de estado!
  3. Evitar bloqueios
  4. Usar arquiteturas assíncronas
  5. Evitar mudança de estado!
  6. Evitar mudança de estado!

UFES

Paralelismo - Dificuldades

- Condições de concorrência
  1. Deadlocks, livelocks, etc.
  2. Sincronização
  3. Não-determinismo
  4. Caos!
- Melhores práticas:
  1. Evitar contenção
  2. Evitar mudança de estado!
  3. Evitar bloqueios
  4. Usar arquiteturas assíncronas
  5. Evitar mudança de estado!
  6. Evitar mudança de estado!

UFES

Paralelismo - Dificuldades

- Condições de concorrência
  1. Deadlocks, livelocks, etc.
  2. Sincronização
  3. Não-determinismo
  4. Caos!
- Melhores práticas:
  1. Evitar contenção
  2. Evitar mudança de estado!
  3. Evitar bloqueios
  4. Usar arquiteturas assíncronas
  5. Evitar mudança de estado!
  6. Evitar mudança de estado!

UFES

Paralelismo - Dificuldades

- Condições de concorrência
  1. Deadlocks, livelocks, etc.
  2. Sincronização
  3. Não-determinismo
  4. Caos!
- Melhores práticas:
  1. Evitar contenção
  2. Evitar mudança de estado!
  3. Evitar bloqueios
  4. Usar arquiteturas assíncronas
  5. Evitar mudança de estado!
  6. Evitar mudança de estado!

UFES

Paralelismo - Dificuldades

- Condições de concorrência
  1. Deadlocks, livelocks, etc.
  2. Sincronização
  3. Não-determinismo
  4. Caos!
- Melhores práticas:
  1. Evitar contenção
  2. Evitar mudança de estado!
  3. Evitar bloqueios
  4. Usar arquiteturas assíncronas
  5. Evitar mudança de estado!
  6. Evitar mudança de estado!

UFES

Paralelismo - Dificuldades

- Condições de concorrência
  1. Deadlocks, livelocks, etc.
  2. Sincronização
  3. Não-determinismo
  4. Caos!
- Melhores práticas:
  1. Evitar contenção
  2. Evitar mudança de estado!
  3. Evitar bloqueios
  4. Usar arquiteturas assíncronas
  5. Evitar mudança de estado!
  6. Evitar mudança de estado!

UFES

Paralelismo - Dificuldades

- Condições de concorrência
  1. Deadlocks, livelocks, etc.
  2. Sincronização
  3. Não-determinismo
  4. Caos!
- Melhores práticas:
  1. Evitar contenção
  2. Evitar mudança de estado!
  3. Evitar bloqueios
  4. Usar arquiteturas assíncronas
  5. Evitar mudança de estado!
  6. Evitar mudança de estado!

UFES

Paralelismo - Dificuldades

- Condições de concorrência
  1. Deadlocks, livelocks, etc.
  2. Sincronização
  3. Não-determinismo
  4. Caos!
- Melhores práticas:
  1. Evitar contenção
  2. Evitar mudança de estado!
  3. Evitar bloqueios
  4. Usar arquiteturas assíncronas
  5. Evitar mudança de estado!
  6. Evitar mudança de estado!

UFES

Paralelismo - Dificuldades

- Condições de concorrência
  1. Deadlocks, livelocks, etc.
  2. Sincronização
  3. Não-determinismo
  4. Caos!
- Melhores práticas:
  1. Evitar contenção
  2. Evitar mudança de estado!
  3. Evitar bloqueios
  4. Usar arquiteturas assíncronas
  5. Evitar mudança de estado!
  6. Evitar mudança de estado!

UFES

Paralelismo - Dificuldades

- Condições de concorrência
  1. Deadlocks, livelocks, etc.
  2. Sincronização
  3. Não-determinismo
  4. Caos!
- Melhores práticas:
  1. Evitar contenção
  2. Evitar mudança de estado!
  3. Evitar bloqueios
  4. Usar arquiteturas assíncronas
  5. Evitar mudança de estado!
  6. Evitar mudança de estado!

UFES

Paralelismo - Dificuldades

- Condições de concorrência
  1. Deadlocks, livelocks, etc.
  2. Sincronização
  3. Não-determinismo
  4. Caos!
- Melhores práticas:
  1. Evitar contenção
  2. Evitar mudança de estado!
  3. Evitar bloqueios
  4. Usar arquiteturas assíncronas
  5. Evitar mudança de estado!
  6. Evitar mudança de estado!

UFES

Paralelismo - Dificuldades

- Condições de concorrência
  1. Deadlocks, livelocks, etc.
  2. Sincronização
  3. Não-determinismo
  4. Caos!
- Melhores práticas:
  1. Evitar contenção
  2. Evitar mudança de estado!
  3. Evitar bloqueios
  4. Usar arquiteturas assíncronas
  5. Evitar mudança de estado!
  6. Evitar mudança de estado!

UFES

Paralelismo

- Programação orientada à objeto: ênfase em estados
- Programação funcional: incentiva o desenvolvimento de código independente de mudançã de estados
- Scala: o melhor dos dois mundos!

Paralelismo

- Programação orientada à objeto: ênfase em estados
- Programação funcional: incentiva o desenvolvimento de código independente de mudançã de estados
- Scala: o melhor dos dois mundos!

UFES

Paralelismo

- Programação orientada à objeto: ênfase em estados
- Programação funcional: incentiva o desenvolvimento de código independente de mudançã de estados
- Scala: o melhor dos dois mundos!

UFES

Fim


UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
DOCETE OMNES GENTES

UFES