Scala optimizations

A brief overview of the Scala Performance landscape today

Total Disclaimer

- Everything I say may or may not be true
- Performance discusssions are (almost) always context dependant

Goal of this presentation

- Give examples of some common performance issues in Scala
- Knowlegde of some Scala annotations
  - @elidable
  - @switch
  - @tailrec
- Basic knowlegde of ScalaCL

Agenda

- Scala performance - some common gotchas
- Using annotations
- ScalaCL
- A little bit about the Scala Compiler

Scala Performance – a clear picture?

- In general not a lot documentation exists on Scala performance issues
- Style vs. Scala as a language
- JVM – the winner *changes* all

Some Scala Performance Considerations

- For loops - Range vs while
  - Object creation, method invocation
- Type vals (ducktyping)
  - Using reflection
- Option Some()
  - object creation, memory consumption
- *Implicits*
  - object creation
  - Couldn't create a good use case which gave issues

Annotations for Performance

- @elidable
  - Remove methods when compiling
- @switch
  - Compile verification of efficient pattern match using tableswitch or tablelookup
  - Only int compatible values (char, short, byte)
- @tailrec
  - Compile check to ensure tail call optimization is done, on recursive functions.
- @inline…
- @specialize…

ScalaCL

- Consists of
  - GPU-backed collections
  - Compiler plugin for automatic code optimizations
- Status
  - Very little activity (last update february)
  - Scalaxy - macro optimizations for Scala 2.10 (Code example)
  - ScalaCL - Scala 2.9 only

Compiler arguments - performance

- -optimise
  - Hard to find actual use cases
  - Hotspot is most likely doing same optimizations
- -target (jvm 1.6, jvm 1.7)
  - New backend - experimental
    - Can use method handles (Currently slow)
    - Faster compile time
    - Emits less bytecode

Scala Compile options
- A big chinese super market

- Learn your compiler
  - -Xelide-below <n>
  - -Xlog-reflective-calls
  - -Ydead-code
  - -Yinline…
  - -Ynotnull