Towards Parallelizing Scala Compilations

Eugene Burmako
Twitter, Inc.

11/15/2018

About me

- PhD from Martin Odersky's lab at EPFL (2011-2016)
- Tech lead of the Advanced Scala Tools team at Twitter (2017-present)
- Founder of Scala macros, Scalameta and Rsc

Agenda

- Previously on Rsc
- Towards practical usefulness
- Multi-phase compilation
- Experimental results

Previously on Rsc

Mission statement

An experimental Scala compiler focused on compilation speed

Research goal

5-10x compilation speedup for typical Scala codebases

Axioms

- Rewrite from first principles
- Start small
- Contribute to the community

Example (parse)

```
import M._

class A {
  def foo: Foo = {
    val b: B = ...
    b.c
  }
  def b: B = { ... }
}
```

```
object M extends X {
  class B {
    def c: C = {
      ...
    }
    def a: A = { ... }
  }
}
```

Example (schedule)

```
import M._

class A {
  def foo: Foo = {
    val b: B = ...
    b.c
  }
  def b: B = { ... }
}
```

```
object M extends X {
  class B {
    def c: C = {
      ...
    }
    def a: A = { ... }
  }
}
```

Example (scope)

```
import M._

class A {
  def foo: Foo = {
    val b: B = ...
    b.c
  }
  def b: B = { ... }
}
```

```
object M extends X {
  class B {
    def c: C = {
      ...
    }
    def a: A = { ... }
  }
}
```

Example (outline)

```
import M._

class A {
  def foo: Foo = {
    val b: B = ...
    b.c
  }
  def b: B = { ... }
}
```

```
object M extends X {
  class B {
    def c: C = {
      ...
    }
    def a: A = { ... }
  }
}
```

Example (typecheck)

```
import M._

class A {
  def foo: Foo = {
    val b: B = ...
    b.c
  }
  def b: B = { ... }
}
```

```
object M extends X {
  class B {
    def c: C = {
      ...
    }
    def a: A = { ... }
  }
}
```

Implementation

- Prototype typechecker
- Only supports small subset of Scala
- Only supports source dependencies
- Only supports name resolution

Experimental results

- An implementation of RE2 ported from Java
- ~11kloc of nontrivial but simple enough Scala code
- Prototype Rsc typechecker is ~25x faster than the full Scalac typechecker

Towards practical usefulness

Next steps

- We can resolve names very quickly
- Now what?

Idea #1: Emit bytecode

- Take outlines
- Create JVM classes and definitions
- Take resolved names within method bodies
- Emit bytecode for method bodies

Idea #1: Emit bytecode

- Scalac backend is tightly coupled with Scalac internals
- Dotty midend/backend is tightly coupled with Dotty internals
- Our own backend will significantly increase the scope of the project
- Unlikely to be practically useful in the short term

Idea #2: Emit SemanticDB

```
message TextDocument {
  string uri = 2;
  string text = 3;
  string md5 = 11;
  Language language = 10;
  repeated SymbolInformation symbols = 5;
  repeated SymbolOccurrence occurrences = 6;
  repeated Diagnostic diagnostics = 7;
  repeated Synthetic synthetics = 12;
}
```

Idea #2: Emit SemanticDB

- SemanticDB is used by Metabrowse, Metals, Scalafix and other tools
- A very fast IDE would be very nice
- However, an IDE is much more than just typechecking
- Unlikely to be practically useful in the short term

Idea #3: Emit ScalaSignatures

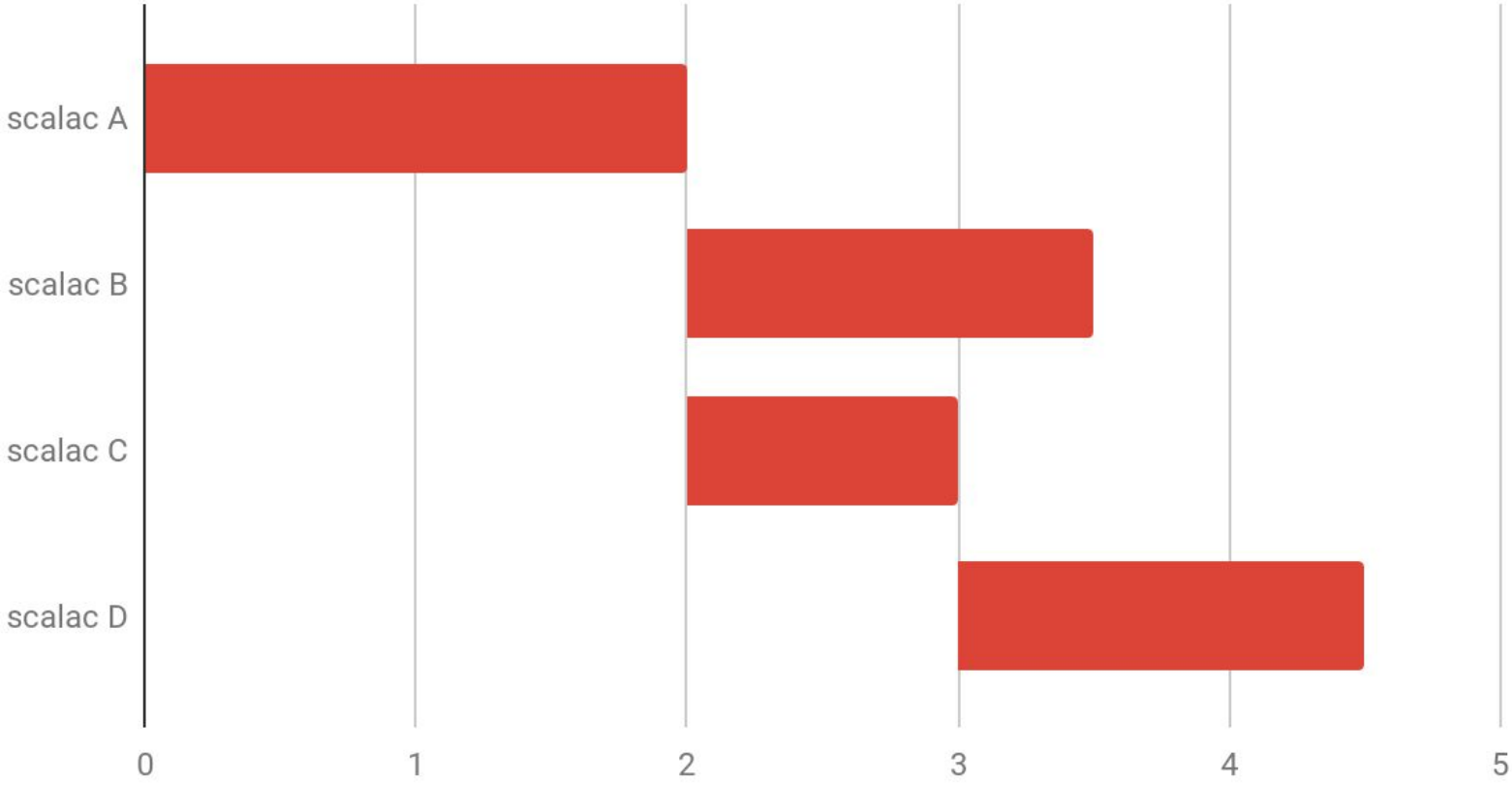
- Take outlines
- Emit ScalaSignatures
- Delegate everything else to Scalac

Multi-phase compilation

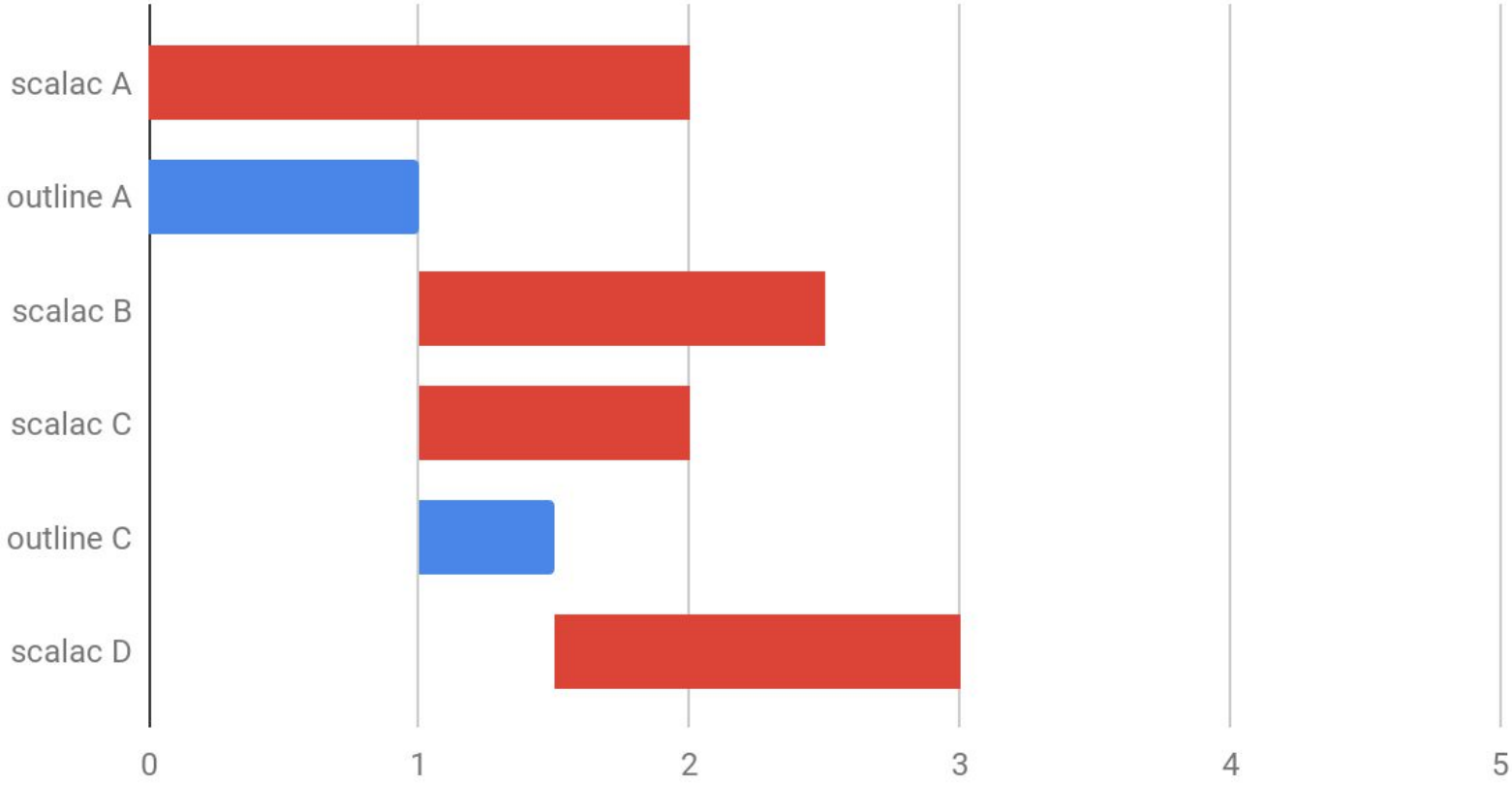
Fictional example

- Project A depends on nothing and compiles in 2s
- Project B depends on A and compiles in 1.5s
- Project C depends on A and compiles in 1s
- Project D depends on C and compiles in 1.5s

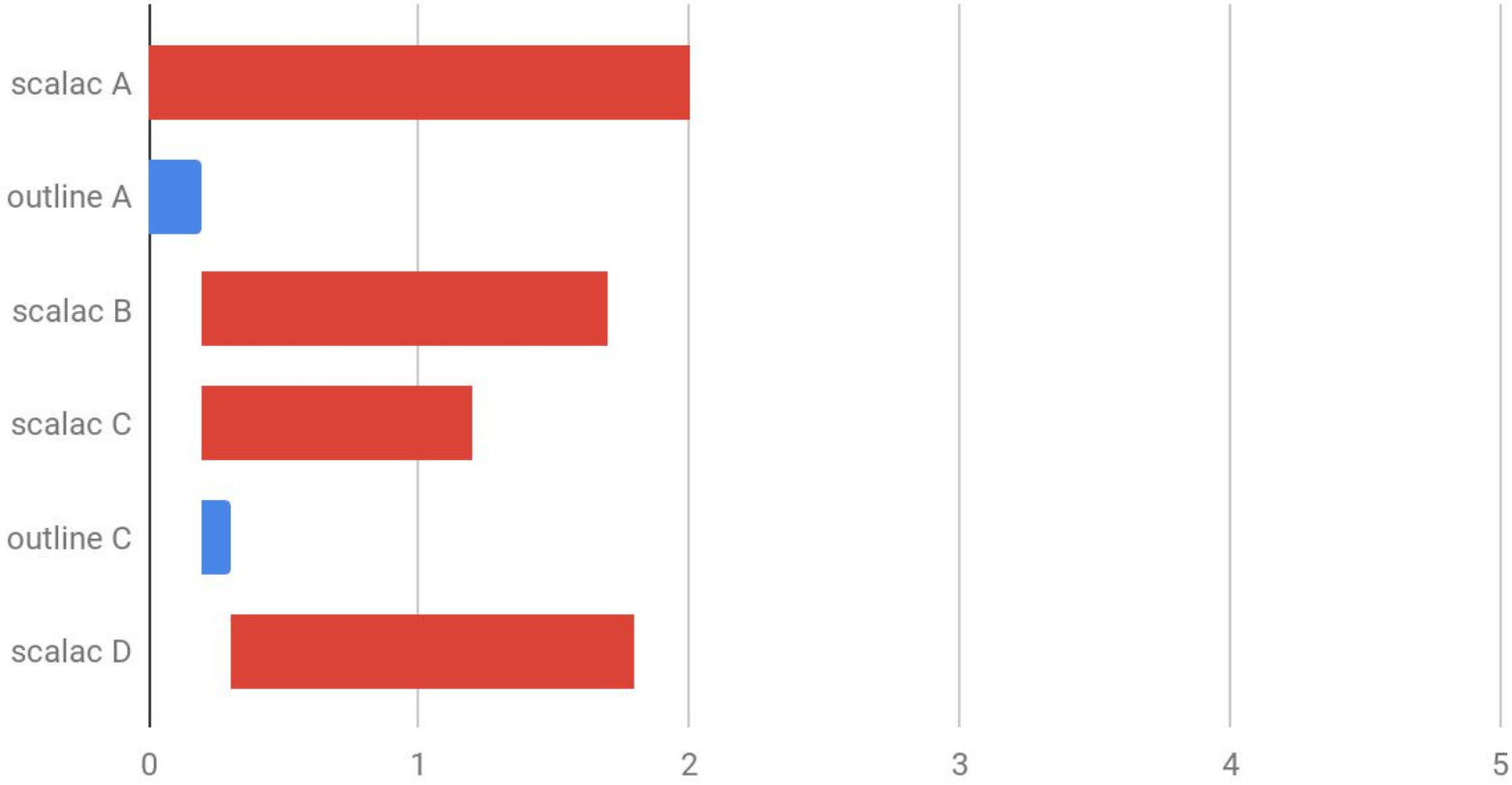
One-phase compilation (4.5s)



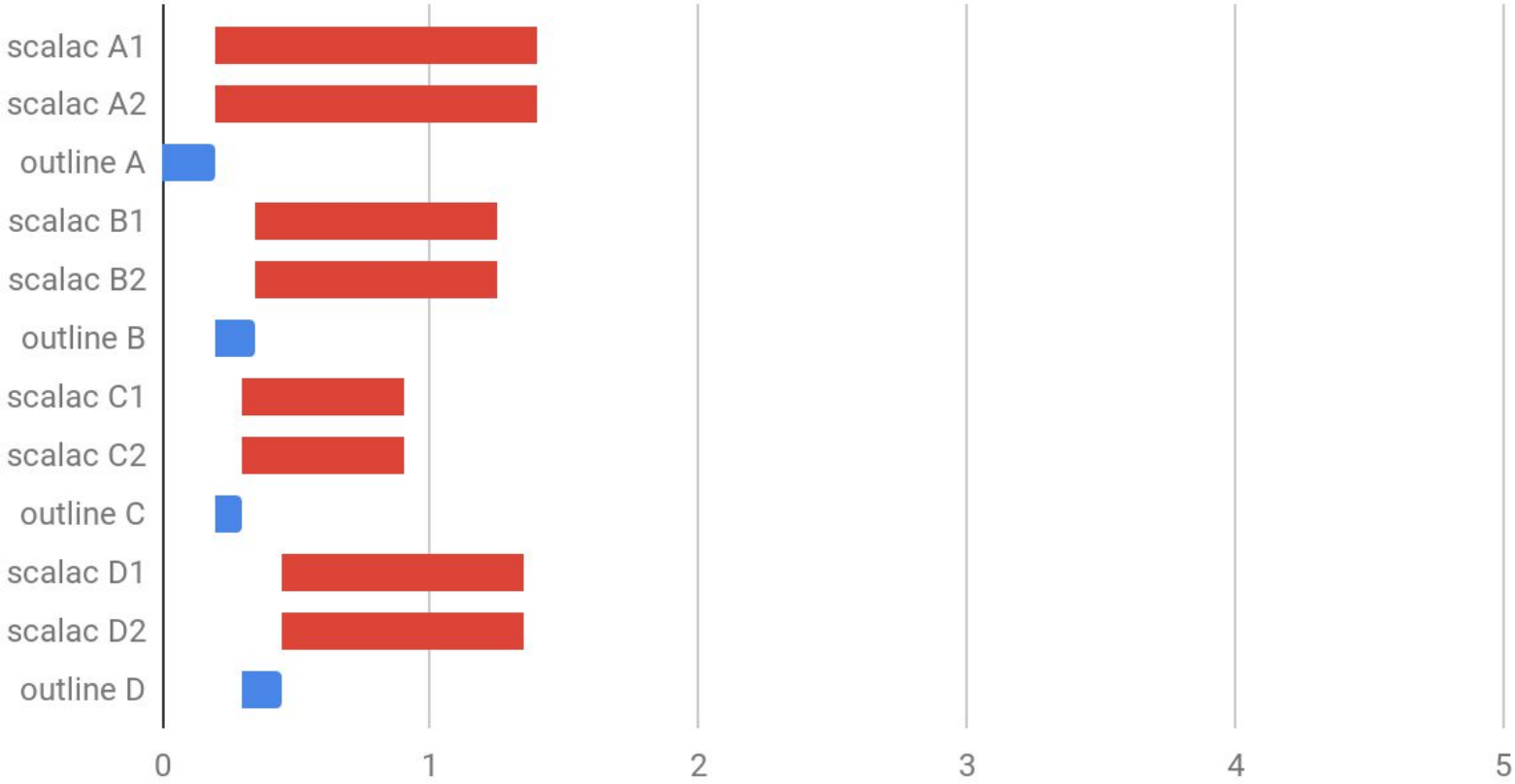
Two-phase compilation, outline takes 50% (3s)



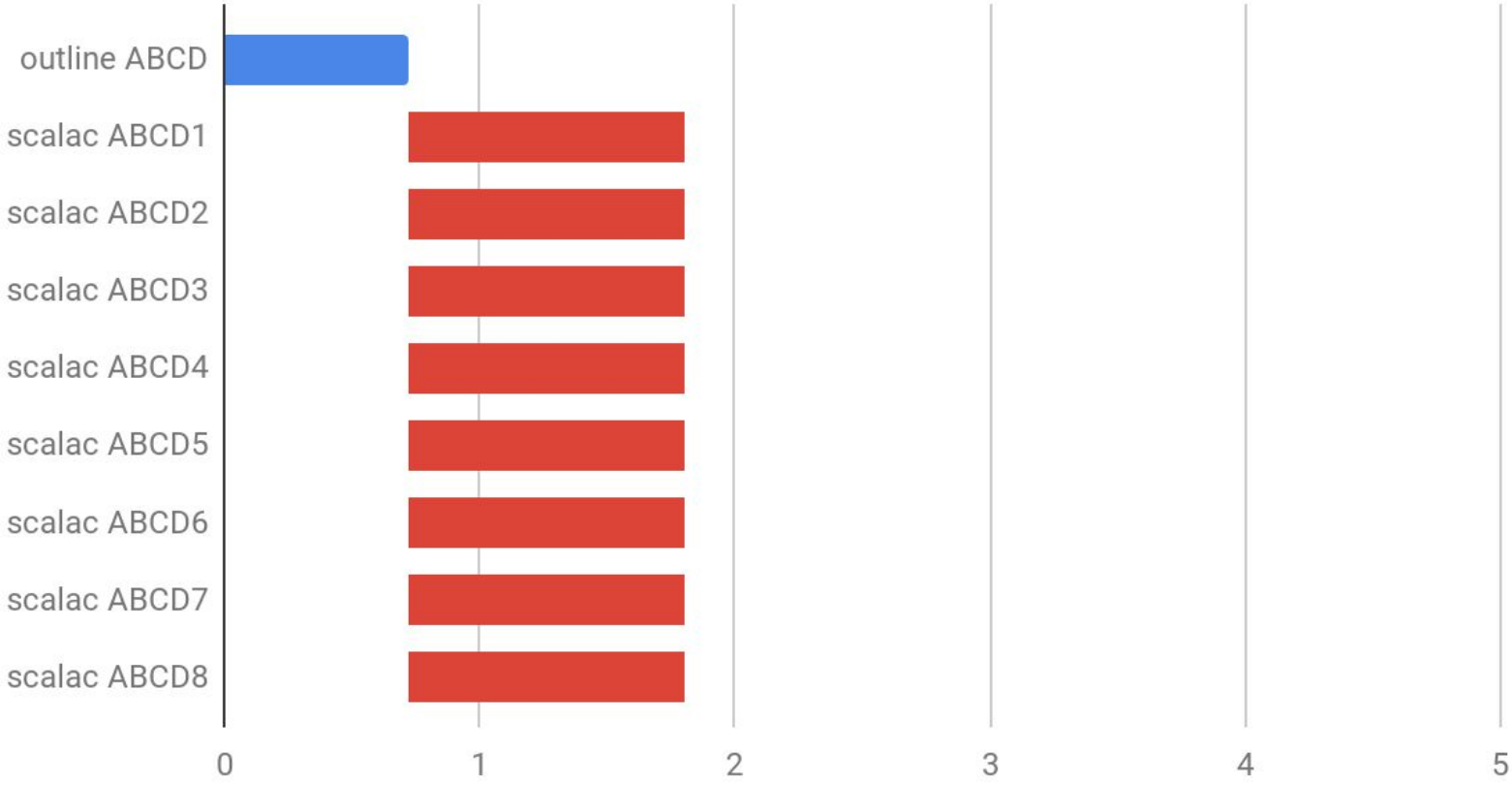
Two-phase compilation, outline takes 10% (2s)



Subtarget parallelism, overhead 20% (1.4s)



Supertarget parallelism, further overhead 20% (1.8s)



Scala support

- Scala targets may need to adhere to certain language restrictions
- If the outliner doesn’t support full-blown typechecking:
  - Explicit return types for public/protected members
  - Explicit type arguments for parent clauses
  - Additional work for compiler plugins and macro annotations

Java support

- Java targets introduce complications
- For Java -> Scala, we need a Java outliner
- For Scala -> Java, we need one of the following:
  - Support for emitting full-blown classfiles (not just ScalaSignatures)
  - Three-phase compilation (outline, compile Scala, compile Java)

Experimental results

Hardware

- Intel Xeon E5-2683 v4 (2x CPUs, 16x cores each, 2x threads each)
- 128GB RAM
- 480GB SSD

Software

- Java 1.8.0_181
- Scala 2.11.12
- Rsc 0.0.0-446-c64e6937
- Warp 0.0.0-173-1dc06b47

Case study

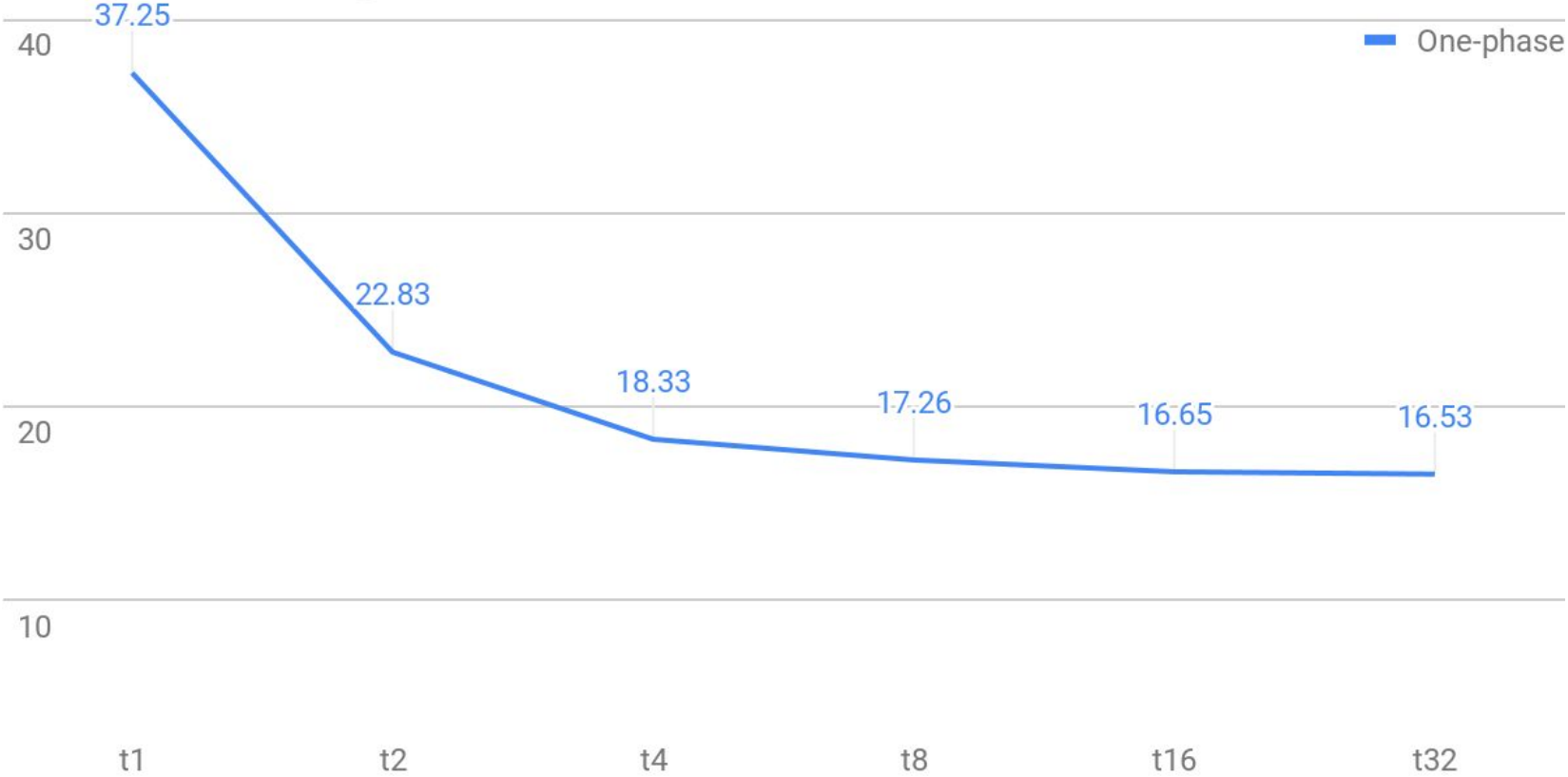
- Internal mirror of https://github.com/twitter/util
- 45 Scala targets (425 files, 69088 loc)
- 15 Java targets (77 files, 5106 loc)

Benchmarks

- Hot one-phase compilation with 1, 2, 4, 8, 16, 32 threads
- Hot three-phase compilation with 1, 2, 4, 8, 16, 32 threads
  - No hyperthreading (doesn’t work well)
  - Subtarget parallelism with the batch size of 5 source files
  - No supertarget parallelism (doesn’t work without parallel outlining)

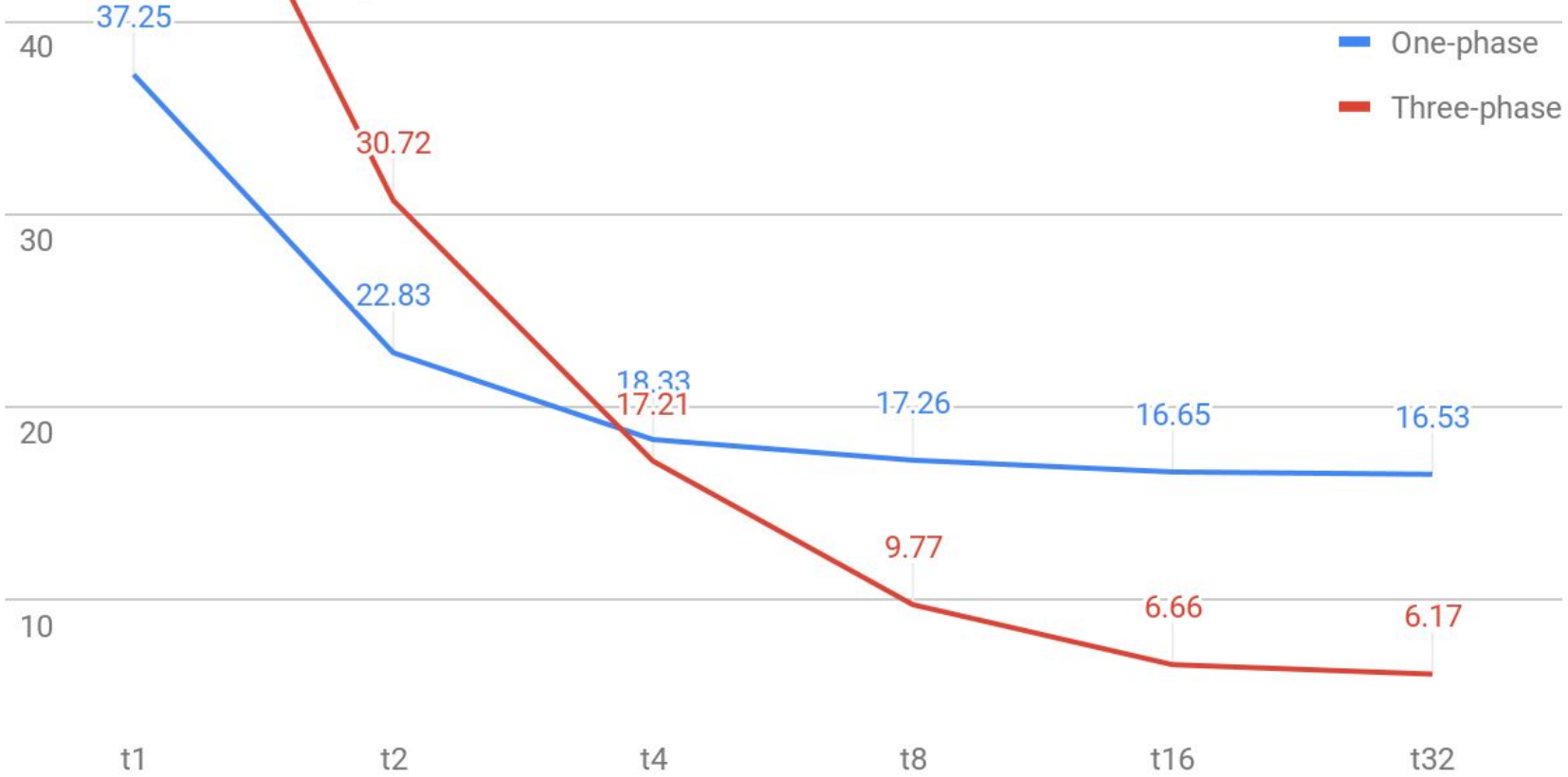
One-phase compilation (2.25x on 32 threads)

Stops scaling well at 4 threads



Three-phase compilation (6.03x on 32 threads)

Stops scaling well at 16 threads



Summary

Related work (in chronological order)

- Limits of Scala typechecking speed: https://github.com/gkossakowski/kentuckymule
- Implement build pipelining: https://github.com/scalacenter/bloop/pull/633
- [WIP] Parallelize the compiler via two-pass compilation: https://github.com/lampepfl/dotty/pull/4767

Credits (in chronological order)

- Martin Odersky (idea of writing a Scala compiler from scratch)
- Denys Shabalin (idea of instant startup time for Scala tools)
- Grzegorz Kossakowski (idea of outlining)
- Advanced Scala Tools team (design and implementation of Rsc)
- Stu Hood (idea of two-phase compilation)
- Build team (integration between Rsc and Pants)

Summary

- We have a very fast prototype outliner for subsets of Scala and Java
- This unlocks a significant degree of parallelism in Scala builds