Fine-grained Deterministic Parallelization of Static Analyses

Philipp Haller

KTH Royal Institute of Technology
Stockholm, Sweden

Joint work with
Dominik Helm, Guido Salvaneschi, Mira Mezini (TU Darmstadt, Germany), and
Michael Eichberg (German Federal Criminal Police Office)

CASTOR Software Days
Stockholm, October 14th, 2019

Background

- Associate professor at KTH (2014–2018 assistant professor)
- 2005–2014 ***Scala language team***
  - 2012–2014 Typesafe, Inc. (now Lightbend, Inc.)
- Co-author Scala language specification
- Focus on ***asynchronous, concurrent and distributed programming***
  - Creator of Scala actors, co-author of Scala's futures and async/await
  - *Topics:* programming models, compilers, type systems, semantics

The Problem

- Increasing importance of static analysis
  - Bug finding, security analysis, taint tracking, etc.
- Precise and powerful analyses have ***long running times***
  - Infeasible to integrate into nightly builds, CI, IDE, …
  - ***Parallelization difficult:*** advanced static analyses not data-parallel
- Scaling static analyses to ever-growing software systems requires ***maximizing utilization of multi-core CPUs***

The Approach

Ongoing work on checking correctness

- Novel ***concurrent programming model***
  - Generalization of futures/promises
  - Guarantees deterministic outcomes *(if used correctly)*
- Implemented in Scala
  - Statically-typed, integrates functional and object-oriented programming
  - Supported backends: JVM, JavaScript (+ experimental native backend)
- Integrated with OPAL, a state-of-the-art ***JVM bytecode analysis framework***

Example

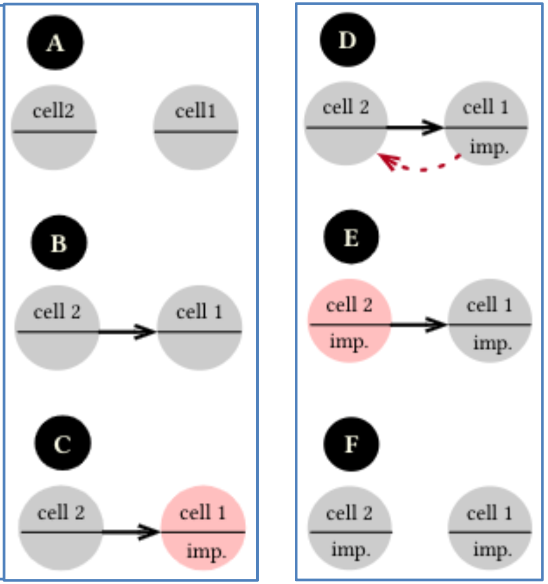
- Two key concepts: ***cells*** and ***handlers***
- Cell completers permit ***writing***, cells only ***reading*** (concurrently)

```
val completer1 = CellCompleter[...]
val completer2 = CellCompleter[...]
val cell1 = completer1.cell
val cell2 = completer2.cell

cell2.when(cell1) { update =>
  if (update.value == Impure) FinalOutcome(Impure)
  else NoOutcome
}
completer1.putFinal(Impure)
```

Example

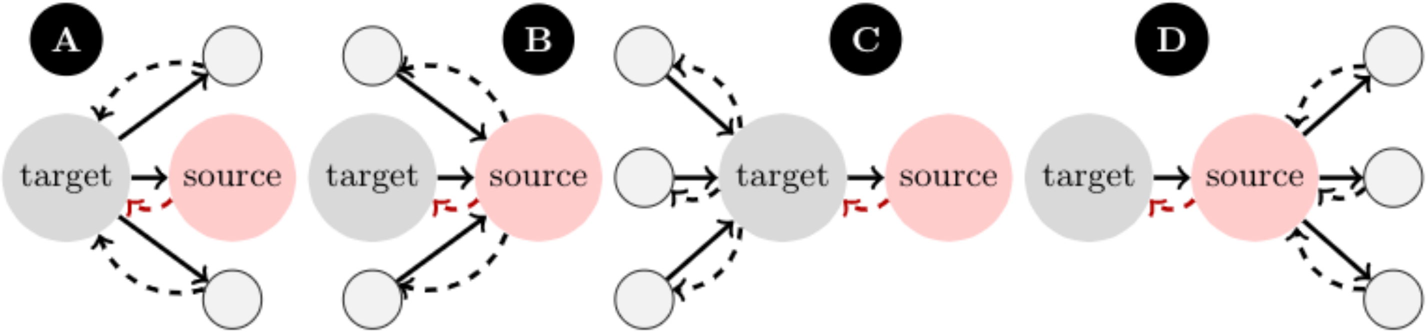
- Two key concepts: ***cells*** and ***handlers***
- Cell completers permit ***writing***, cells only ***reading*** (concurrently)

```
val completer1 = CellCompleter[...]
val completer2 = CellCompleter[...]
val cell1 = completer1.cell
val cell2 = completer2.cell

cell2.when(cell1) { update =>
  if (update.value == Impure) FinalOutcome
  else NoOutcome
}
completer1.putFinal(Impure)
```

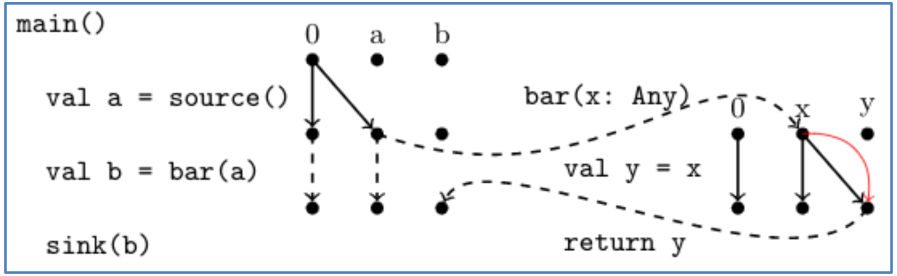


Scheduling Strategies

- ***Priorities for message propagations*** depending on number of dependencies of source/target nodes and dependees/dependers



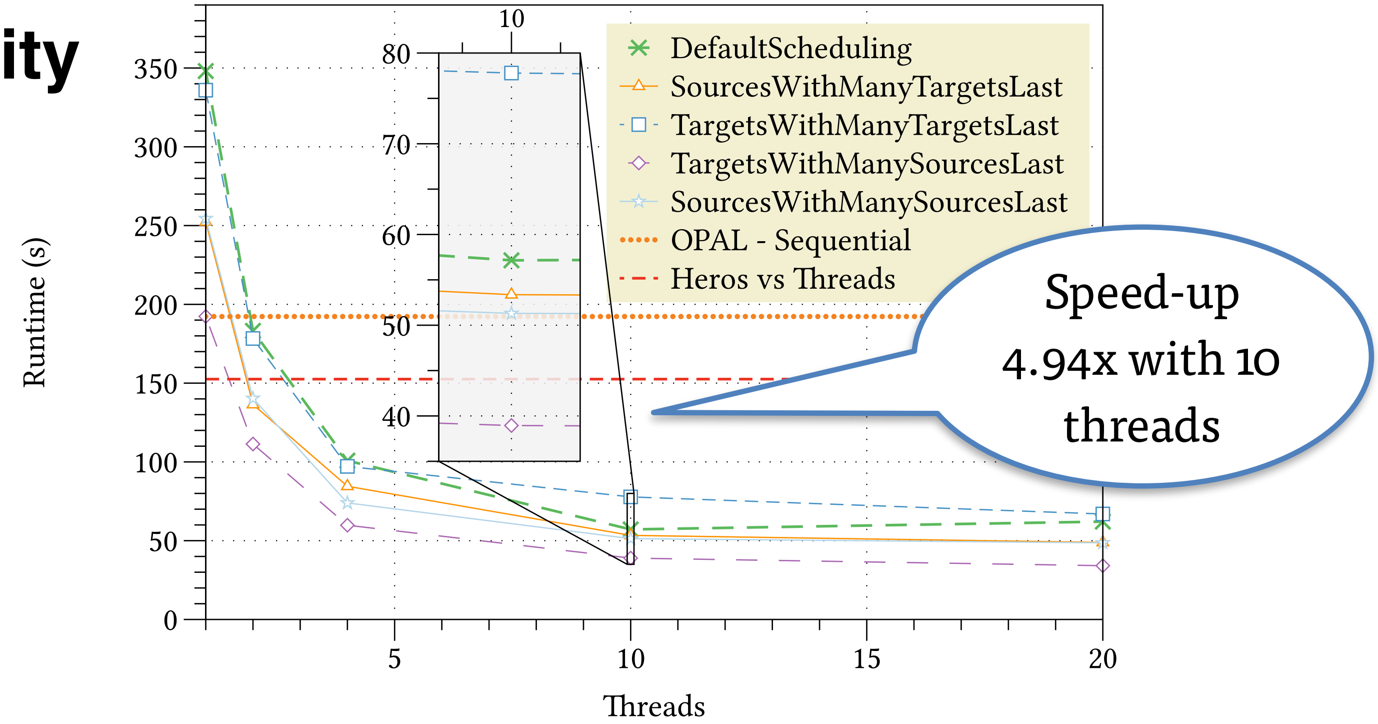
Experimental Evaluation



- Implementation of IFDS[1] analysis framework
- Use IFDS framework to implement ***taint analysis***
  - search for methods in JDK with return type `Object` or `Class` with String parameter that is later used in an invocation of `Class.forName`

[1] Interprocedural Finite Distributive Subset

Scalability



Analysis executed on Intel(R) Core(TM) i9-7900X CPU @ 3.30GHz (10 cores) with 128 GB RAM running Ubuntu 18.04.1 and OpenJDK 1.8_212

Scheduling Strategies

Table 2. Performance of different scheduling strategies. Percentages show speedup of each strategy compared to (a) default strategy and (b) slowest strategy.

Strategy	Run time [s]	Speedup (a)	Speedup (b)
DefaultScheduling	57.15	0.00%	26.54%
SourcesWithManyTargetsLast	53.37	6.62%	31.40%
TargetsWithManySourcesLast	38.94	31.86%	49.94%
TargetsWithManyTargetsLast	77.79	-36.12%	0.00%
SourcesWithManySourcesLast	51.30	10.23%	34.05%

- Using suitable scheduling strategy has big impact on execution time
- Best strategy ***49.94% faster than worst*** strategy, ***31.86% faster than default***

Conclusion

- Deterministic concurrent programming model
  - Supporting pluggable, domain-specific scheduling strategies
- Implemented as a library for Scala
- Experimental results for state-of-the-art IFDS-based taint analysis:
  - Speed-up of 4.94x using 10 threads
  - Significant gains using analysis-specific scheduling strategies
- Open-source code available on GitHub: https://github.com/phaller/reactive-async