Efficient Layered Method Execution in ContextAmber

COP 2015

Matthias Springer, Jens Lincke, Robert Hirschfeld

Hasso Plattner Institute, Software Architecture Group

July 5, 2015

Introduction

- Implemented **ContextAmber**: layer-based COP library, written in Smalltalk, compiled to JavaScript
- Optimizations for ContextAmber: make **layered method execution faster**
- Running example: Vector Graphics Debugging

Amber

Smalltalk brought to the web

Which COP is it?

- Layer-based COP for class-based object-oriented programming
- Layer activation **globally** (+scoped) and **per object**
- **Explicit layer activation** only
  (i.e. no declarative layer activation or `activeLayer` method override)

Problem: Why is ContextAmber slow?

What happens when a layered method is invoked:

1. Compute which layers are active for the receiver

global:	(L1, L2, L3, L4)
O1:	(+L5, -L2, -L3) = (L1, L4, L5)
O2:	(-L1 + L1) = (L2, L3, L4, L1)

2. Repeatedly do:
   - 2.1 Find next partial method

     $L2 \longrightarrow L3 \longrightarrow L4 \longrightarrow L1$

   - 2.2 Dispatch to partial method

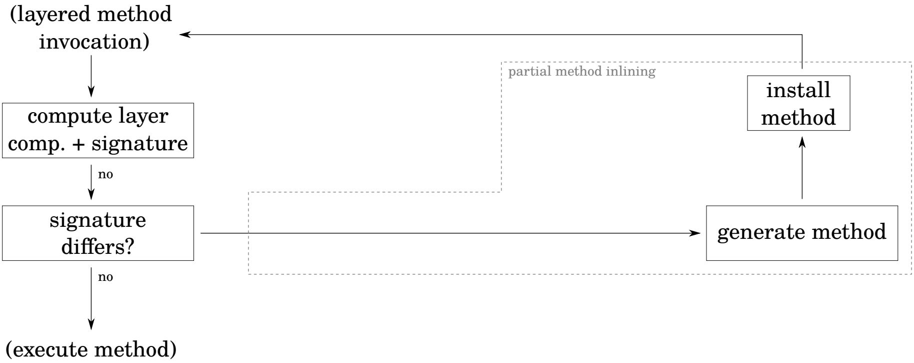
Solution

- **Cache active layers** on a per-object basis
- **Aggressive inlining**: remove all partial method dispatches
- **Inlined method caching**

What’s Next?

Biggest overhead: **looking up** and **dispatching** to next partial method

Partial Method Inlining



(layered method invocation)

compute layer comp. + signature

no

signature differs?

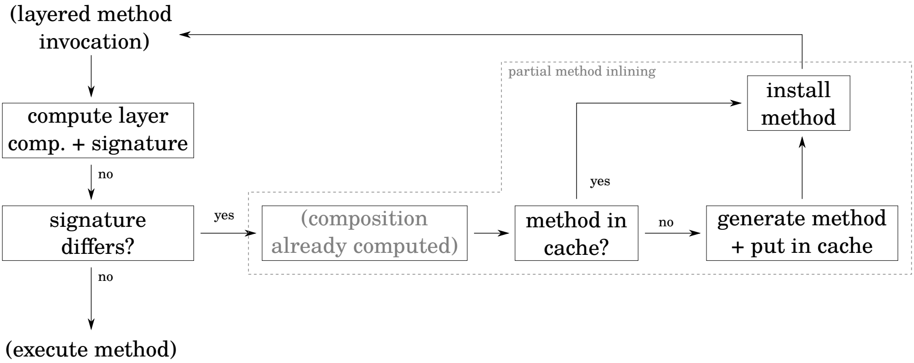
no

(execute method)

What’s Next?

Biggest overhead: **inlining methods**
every time the layer composition changes

Method Caches


(layered method invocation)
compute layer comp. + signature
no
signature differs?
yes
no
(execute method)
partial method inlining
(composition already computed)
method in cache?
yes
no
generate method + put in cache
install method

What's Next?

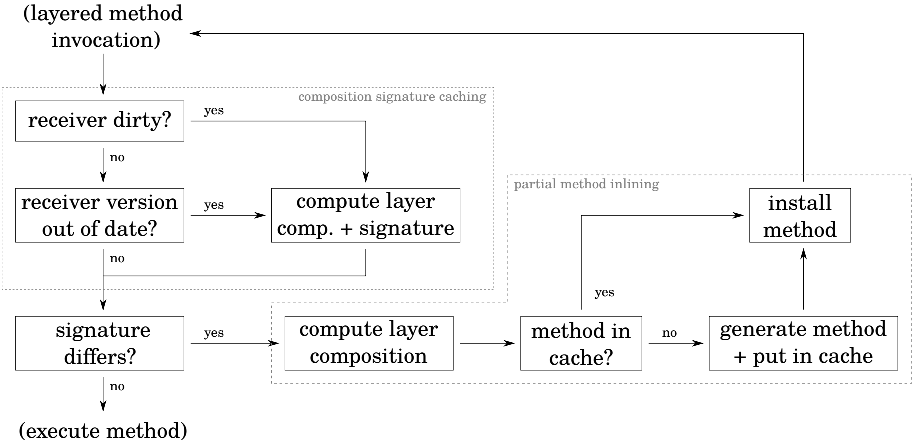
Biggest overhead: **calculating the current layer composition** on every layered method execution

Layer Composition Changes

When does the the layer composition change?

- Layer activated for an object
  → single object affected → dirty bit
- Layer activated globally
  → multiple objects affected → version number

Layer Composition Caching


(layered method invocation)
composition signature caching
receiver dirty?
yes
no
receiver version out of date?
yes
compute layer comp. + signature
no
signature differs?
yes
no
(execute method)
partial method inlining
compute layer composition
method in cache?
yes
no
generate method + put in cache
install method

What's Next?

Biggest overhead: (probably) **JIT trace invalidation** every time a new layered method is installed

Instance-specific Method Inlining

Every **object** has its **own inlined method**.

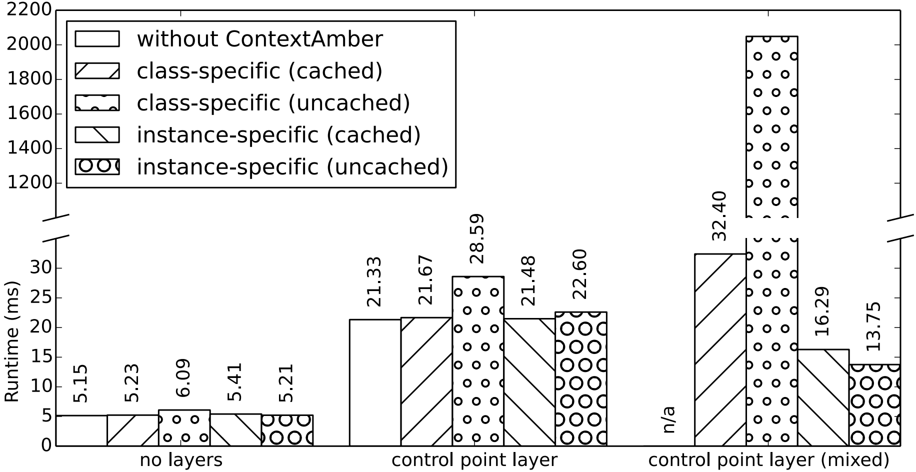
- Layer composition change: nothing changed
  (different layer composition → different inlined method)
- Invoke `a.method` and `b.method`,
  and `a` and `b` have different layer compositions:
  no JIT trace invalidation anymore

What's Next?

Performance is very close
to performance without COP

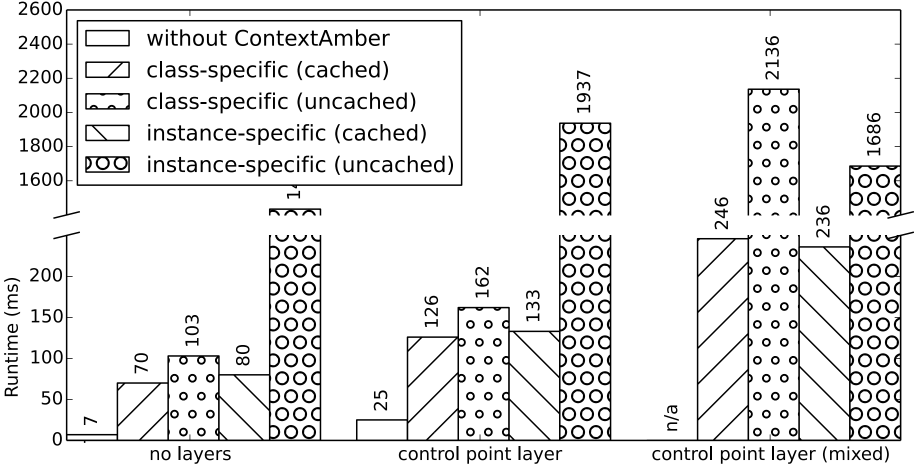
Benchmarks

Average, without first frame



Benchmarks

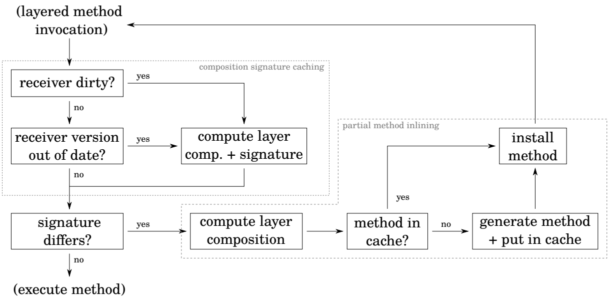
First frame only



Future Work

- Methods are taken from a cache mapping composition signatures to inlined methods
- One method only is ever installed
- Next step: make method lookup aware of layer compositions
  - receiver type × composition signature → target method
  - Preseve JIT traces even if layer composition changes

Summary


(layered method invocation)
composition signature caching
receiver dirty?
yes
no
receiver version out of date?
yes
compute layer comp. + signature
no
signature differs?
yes
no
(execute method)
compute layer composition
partial method inlining
method in cache?
yes
no
install method
generate method + put in cache

Method Inlining

Method Caching

Layer Composition Caching