CoSSJIT: Combining Static Analysis and Speculation in JIT Compilers *

ADITYA ANAND†, VIJAY SUNDARESAN‡, DARYL MAIER‡ AND MANAS THAKUR†

IIT BOMBAY

Singapore SPLASH 2025

PROBLEM STATEMENT

- JIT compilers prioritize performance over precision while performing analyses.
- **Recent Solutions**: Use precise static AOT analysis results to deliver aggressive optimizations in JIT compilers.
- Relying completely on AOT results can take away what a JIT compiler is better at i.e. speculation based on run-time profiles.
- Static AOT analysis doesn't consider possibility of having runtime profile based speculative optimizations.

MOTIVATING EXAMPLE

```
1. class A {
     . . .
4.   void foo(A z) {
5.       A x = new A(); // O5
6.       A y = new A(); // O6
7.       x.f = new A(); // O7
8.       if(z instanceof A) {
9.         . . .
11.      } else {
12.        global = y; Escape
13.      }
14.      z.bar(x);
15.    } /* method foo */
16. } /* class A */
17. class B extends A {
18.    void bar(A p2) { . . . }
19. } /* class B */
20. class C extends A {
21.    void bar(A p3) {
22.      global = p3;   Escape
23.    }
24. } /* class C */
25. class D extends A {
26.    void bar(A p4) {
27.      // p4's pointee
28.      // doesn't escape
29.      p4.f = new A(); // O29
30.      p4.foobar(p4.f);      Escape
31.      . . .
33.    } /* method bar */
33.    void foobar(A p5) { . . . }
34.  } /* class D */
35. class E extends A { . . . }
```

Polymorphic Callsite:

⊓ : {B, C, D}
[O5],[O7] [Escaping]

Branching:

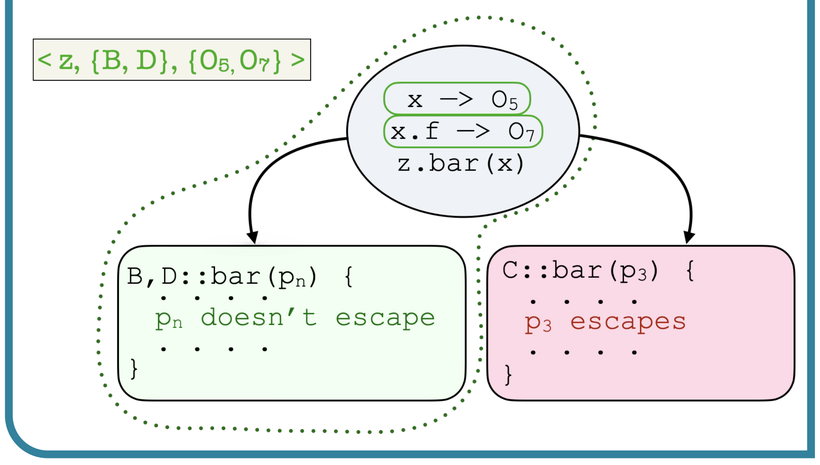
⊓ : {8, 11} [O6] [Escaping]

Method Inlining:

[O29] marked as [Escaping]

Insight: Enrich static analysis results with possibility of run-time speculation and enable the JIT compiler to perform more stack allocations.

POLYMORPHIC CALLSITE

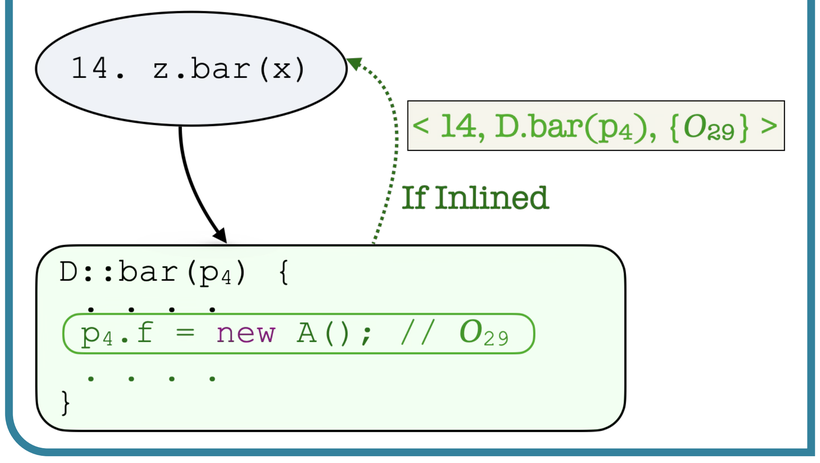


BRANCHING

```
A::foo(z) {
  . . . .
6.   Y = new A(); // O6
8.   if(*) {
       . . .
11.  } else {
12.      // Object O6 escapes.
13.  }
   }
```

< {8}, {O6} >

METHOD INLINING



RESOLVING SPECULATIVE CONDITIONS IN THE JIT

1. Polymorphic Callsite

```
for each O in JIT_identified_objects(m):
    (CHm[c] ⊆ SAm[O][c]) ∨ ( ∀t ∈ SAm[O][c] Σ CPm(t) > ST)
```

CHTable: c → α

Static Res: α, β | 0

Profile: α val1 | β val2

2. Branching Conditions

```
for each O in JIT_identified_objects(m):
    (∀ b ∈ SAm[O][br] with Σ BPm(b) > ST)
```

Static Res: b1,b2 | 0

Profile: b1 val1 | b2 val2

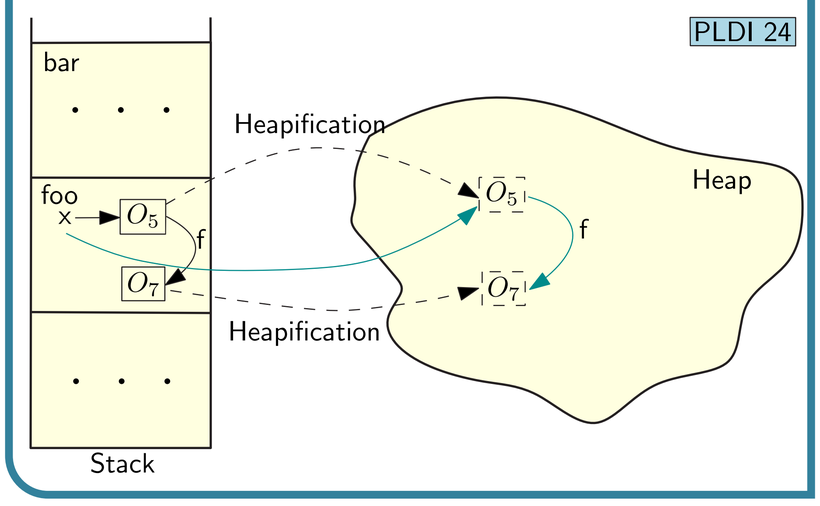
3. Inlining Conditions

Static Res: c, <m> | O1 O2 On

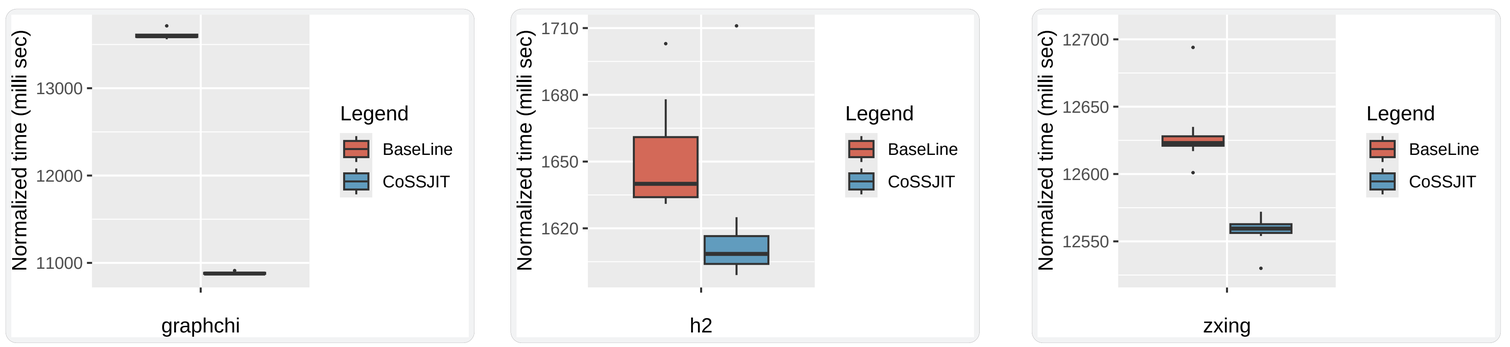
ITable: c → <m>

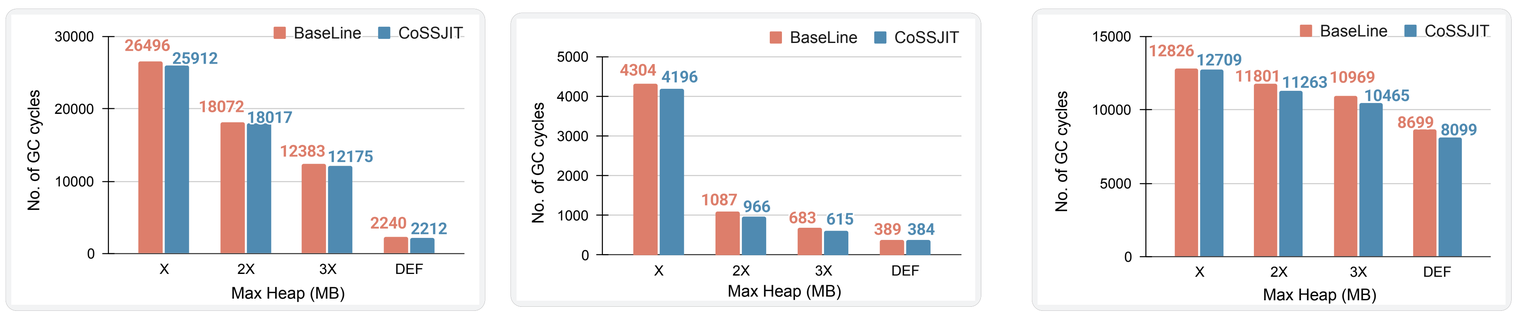
```
for each callsite c ∈ CallSitesm:
    Callersc = SAm[c]
    if ∃ n such that (n ∈ ITm[c] ∧ n ∈ Callersc):
        Ostatic = statically_marked_objects(m)
        mark all O ∈ Ostatic
```

FALLBACK (HEAPIFICATION)



PERFORMANCE IMPROVEMENT





STACK ALLOCATION

graphchi				
BaseLine		CoSSJIT		
Stack-Objects	Stack-Bytes	Stack-Objects	Stack-Bytes	Heapification
349M	8.3 GB	1041.1M	20.1 GB	0.006M

h2				
BaseLine		CoSSJIT		
Stack-Objects	Stack-Bytes	Stack-Objects	Stack-Bytes	Heapification
33M	0.5 GB	525M	12.7 GB	6M

zxing				
BaseLine		CoSSJIT		
Stack-Objects	Stack-Bytes	Stack-Objects	Stack-Bytes	Heapification
24.2M	0.9 GB	153.9M	52.8 GB	0.4M

OOPSLA 2025, CO-LOCATED WITH ICFP/SPLASH 2025

† Author addresses: {adityaanand, manas}@cse.iitb.ac.in. Department of CSE, IIT Bombay, Mumbai, India.
† Author addresses: {vijaysun, maier}@ca.ibm.com. IBM Canada Lab, Canada.

* Anand et al. "CoSSJIT: Combining Static Analysis and Speculation in JIT Compilers."
In Proceedings of the ACM on Programming Languages **(OOPSLA)**, Singapore, October 16-18, 2025.

PLATO