Revamping Sampling-Based PGO

with Context-Sensitivity and Pseudo-Instrumentation

Wenlei He   Hongtao Yu   Lei Wang   Taewook Oh

Meta

Agenda

1. Motivation
2. PGO architectural overview
3. Pseudo-instrumentation
4. Context-sensitive sample PGO
5. CSSPGO production results

Data centers need PGO at scale

PGO delivers 10-20% CPU performance uplift, but there is tension…

Easy to operate at scale ⟷ Better performance

Sample-PGO (AutoFDO)

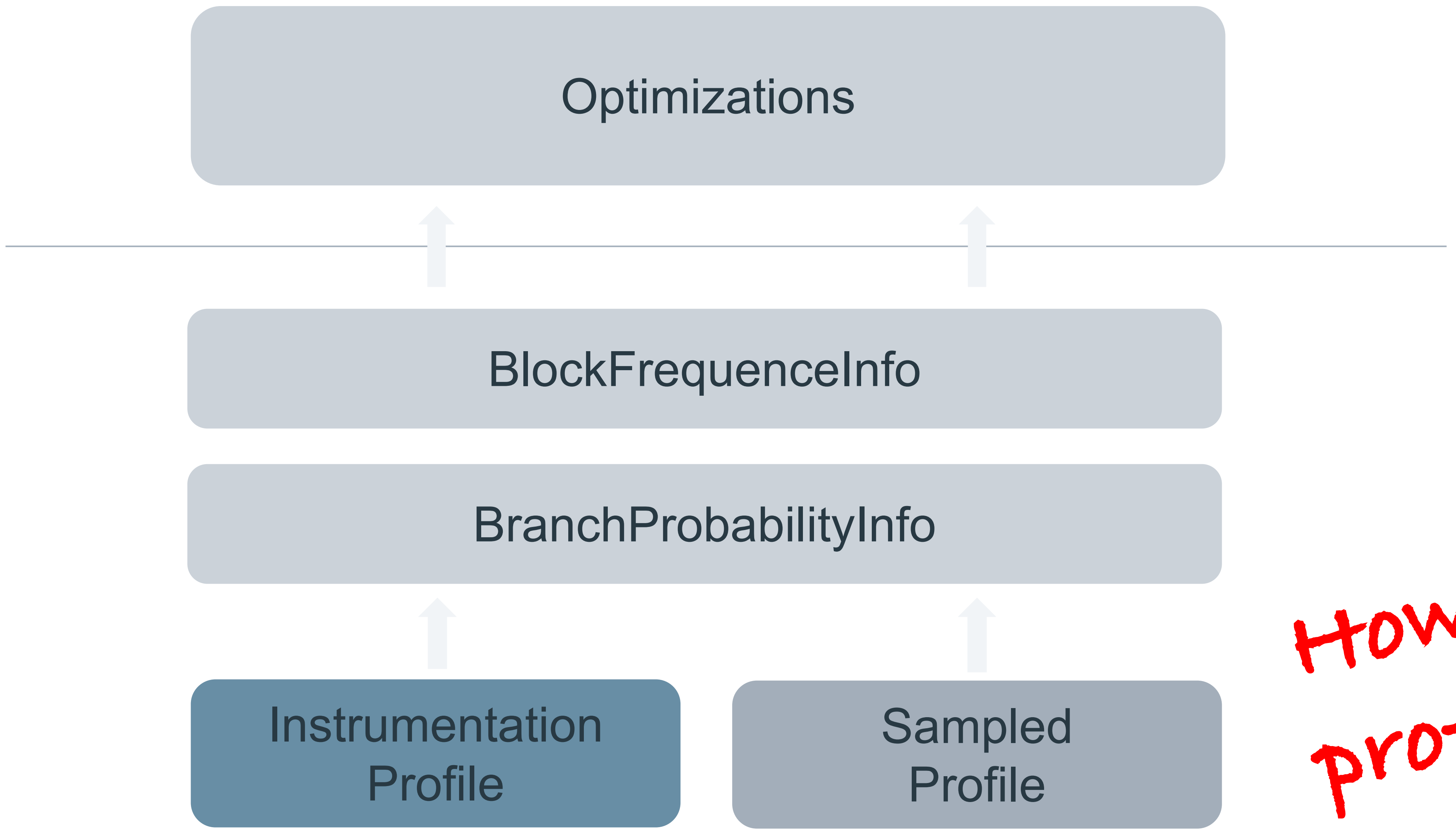
Zero profiling overhead

Instrumentation PGO (IR-PGO)

Extra 2-3% performance
Up to 2x training overhead

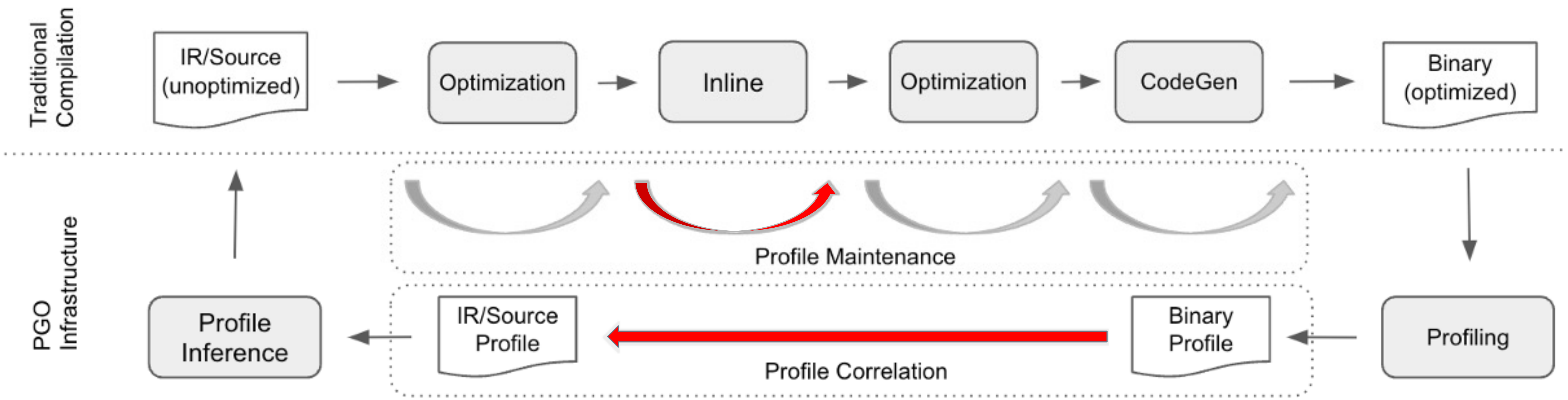
How to get the best of both?

Performance = f(Profile Quality, Optimization)

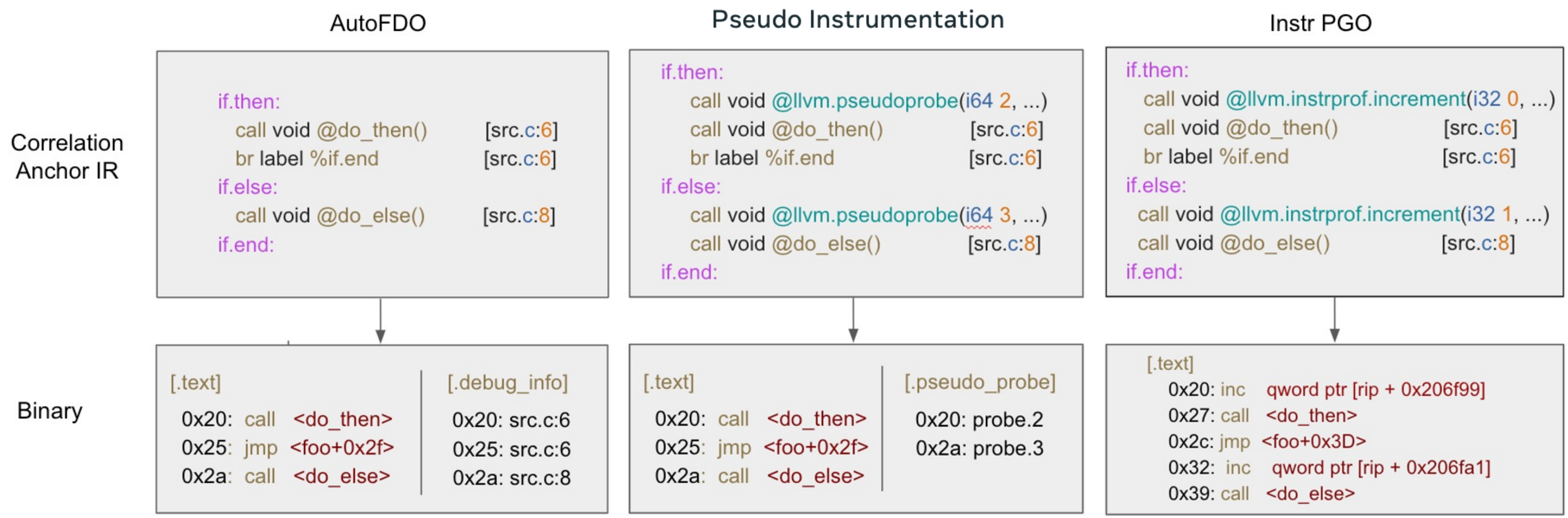


How to improve profile quality

Profile Quality



Profile Correlation



Pseudo-Instrumentation

llvm > llvm-project > llvm > include > llvm > IR > Intrinsics.td > ...

```
// The pseudoprobe intrinsic works as a place holder to the block it probes.
// Like the sideeffect intrinsic defined above, this intrinsic is treated by the
// optimizer as having opaque side effects so that it won't be get rid of or moved
// out of the block it probes.
def int_pseudoprobe : DefaultAttrsIntrinsic<[], [llvm_i64_ty, llvm_i64_ty, llvm_i32_ty, llvm_i64_ty],
                                            [IntrInaccessibleMemOnly, IntrWillReturn]>;
```

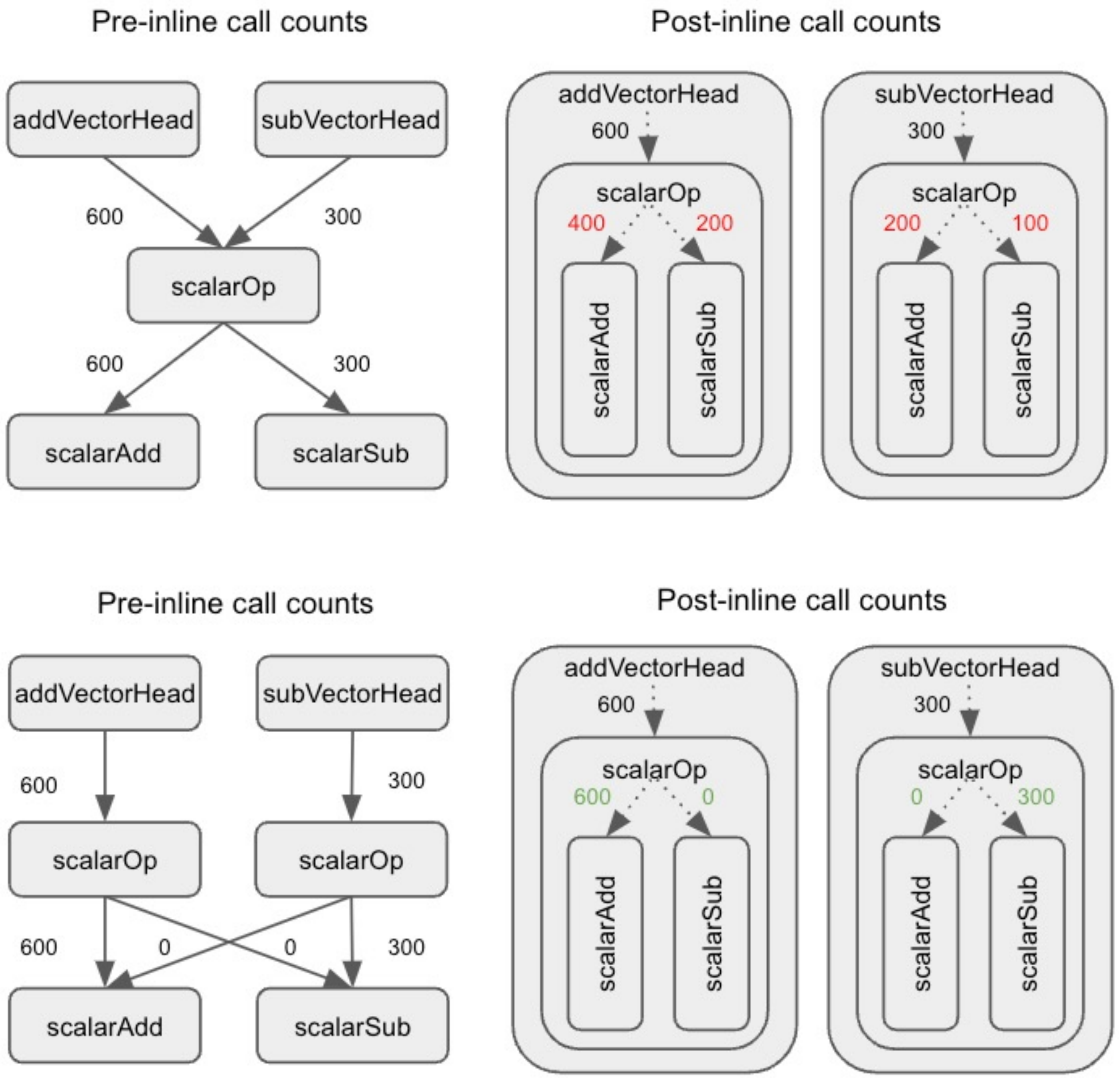
Comparison on HHVM	AutoFDO	Pseudo Instrumentation	Instrumentation PGO
Block overlap	88.2%	92.3%	100%
Profiling overhead	0%	0.04%	73.06%

Post-inline Profile Quality

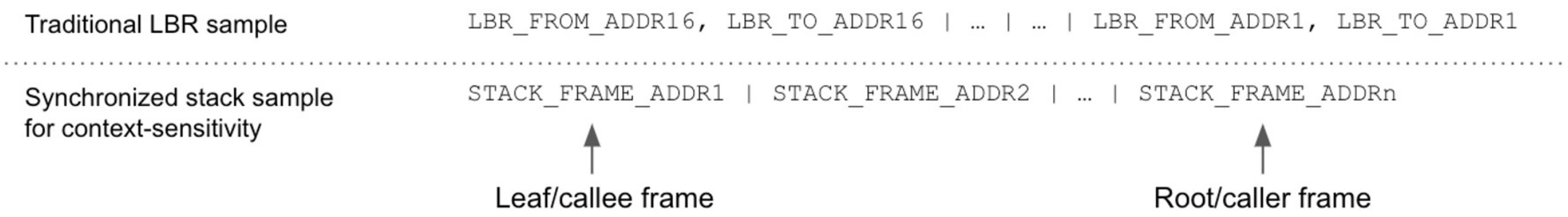
```
Elem AddVectorHead(Vector V1, Vector V2) {
  return scalarOp(V1[0], V2[0], OpAdd);
}
Elem subVectorHead(Vector V1, Vector V2) {
  return scalarOp(V1[0], V2[0], OpSub);
}
Elem scalarOp(Elem E1, Elem E2, Opcode Op) {
  switch (Op) {
  case OpAdd:
    return scalarAdd(E1, E2);
  case OpSub:
    return scalarSub(E1, E2);
  }
}
```

Context-insensitive

Context-sensitive



Context-sensitive Sample Profiling



Traditional LBR sampling:

- To and From of adjacent branches form sampled range
- Raw LBR profiles converted into a set of sample address ranges

Context-sensitive sampling:

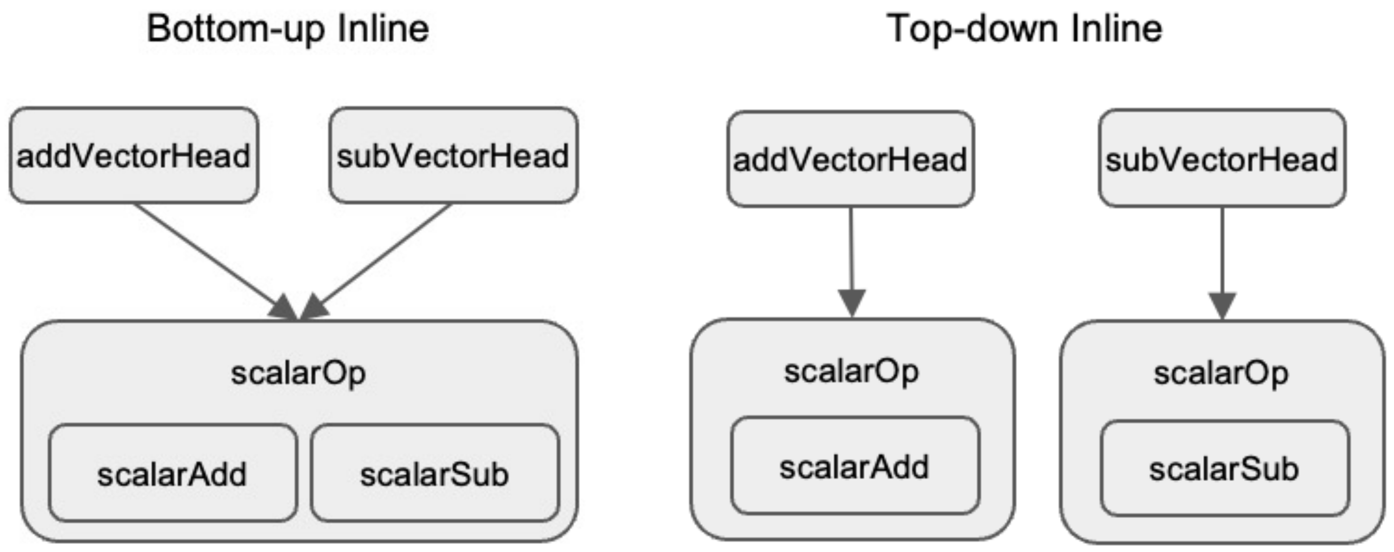
- Synchronized: leaf frame of stack sample align with last branch in LBR (leverage PEBS)
- Sampled stack identifies context for LBR leaf
- Virtual unwind over calls/returns in LBR to adjust stack and recover context for all ranges in LBR

```
Profile for scalarAdd
```

```
Profile for addVectorHead->scalarOp->scalarAdd
Profile for subVectorHead->scalarOp->scalarAdd
```

Context-sensitive Inlining

- Specialization based on context
  - Top-down priority-based inliner in sample loader



- Merge not-inlined context profile across modules
  - Global pre-inliner during profile generation, using binary function size as cost proxy

Context-sensitive Inlining

	AutoFDO		CSSPGO		
Goal	Preserve context profile from prior inlining	Inline decision driven by profile	Preserve context profile across module	Context-sensitive inline decision with specialization	Cold or size inlining
Solution	Early replay Inliner	CGSCC Inliner	Global pre-inliner	Early top-down CS Inliner	CGSCC Inliner

CSSPGO: Putting things together

- Pseudo Instrumentation: low overhead accurate profile correlation
- Context-sensitive sample profiling: better post-inline profile quality
- Context-sensitive PGO inliner and pre-inliner

CSSPGO

More details: [CGO 2024 paper] & LLVM upstream

Production usage at Meta (21 data center regions globally)

PGO Type	Instr. PGO	CSSPGO	Other
CPU cycles %	~10%	~75%	~15%

HHVM

AdRanker, AdRetriever,
AdFinder, HaaS, …

Revamping Sampling-Based PGO with Context-Sensitivity and Pseudo-instrumentation

Wenlei He, *Meta Inc.*, USA, wenlei@meta.com
Hongtao Yu, *Meta Inc.*, USA, hoy@meta.com
Lei Wang, *Meta Inc.*, USA, wlei@meta.com
Taewook Oh, *Meta Inc.*, USA, twoh@meta.com

ization (CGO) | 979-8-3503-9509-9/24/$31.00 ©2024 IEEE | DOI: 10.1109/CGO57630.2024.10444807

***Abstract*—The ever increasing scale of modern data center demands more effective optimizations, as even a small percentage of performance improvement can result in a significant reduction in data-center cost and its environmental footprint. However, the diverse set of workloads running in data centers also challenges the scalability of optimization solutions. Profile-guided optimization (PGO) is a promising technique to improve application performance. Sampling-based PGO is widely used in data-center applications due to its low operational overhead, but the performance gains are not as substantial as the instrumentation-based counterpart. The high operational overhead of instrumentation-based PGO, on the other hand, hinders its large-scale adoption, despite its superior performance gains.**

In this paper, we propose CSSPGO, a context-sensitive sampling-based PGO framework with pseudo-instrumentation. CSSPGO offers a more balanced solution to push sampling-based PGO performance closer to instrumentation-based PGO while maintaining minimal operational overhead. It leverages pseudo-instrumentation to improve profile quality without incurring the overhead of traditional instrumentation. It also enriches profile with context-sensitivity to aid more effective optimizations through a novel profiling methodology using synchronized LBR and stack sampling. CSSPGO is now used to optimize over 75% of Meta's data center CPU cycles. Our evaluation with production workloads demonstrates 1%-5% performance improvement on top of state-of-the-art sampling-based PGO.

***Index Terms*—Profile Guided Optimization, Feedback Directed Optimization, Sampling, Instrumentation, Context-sensitive Profiling, Compiler**

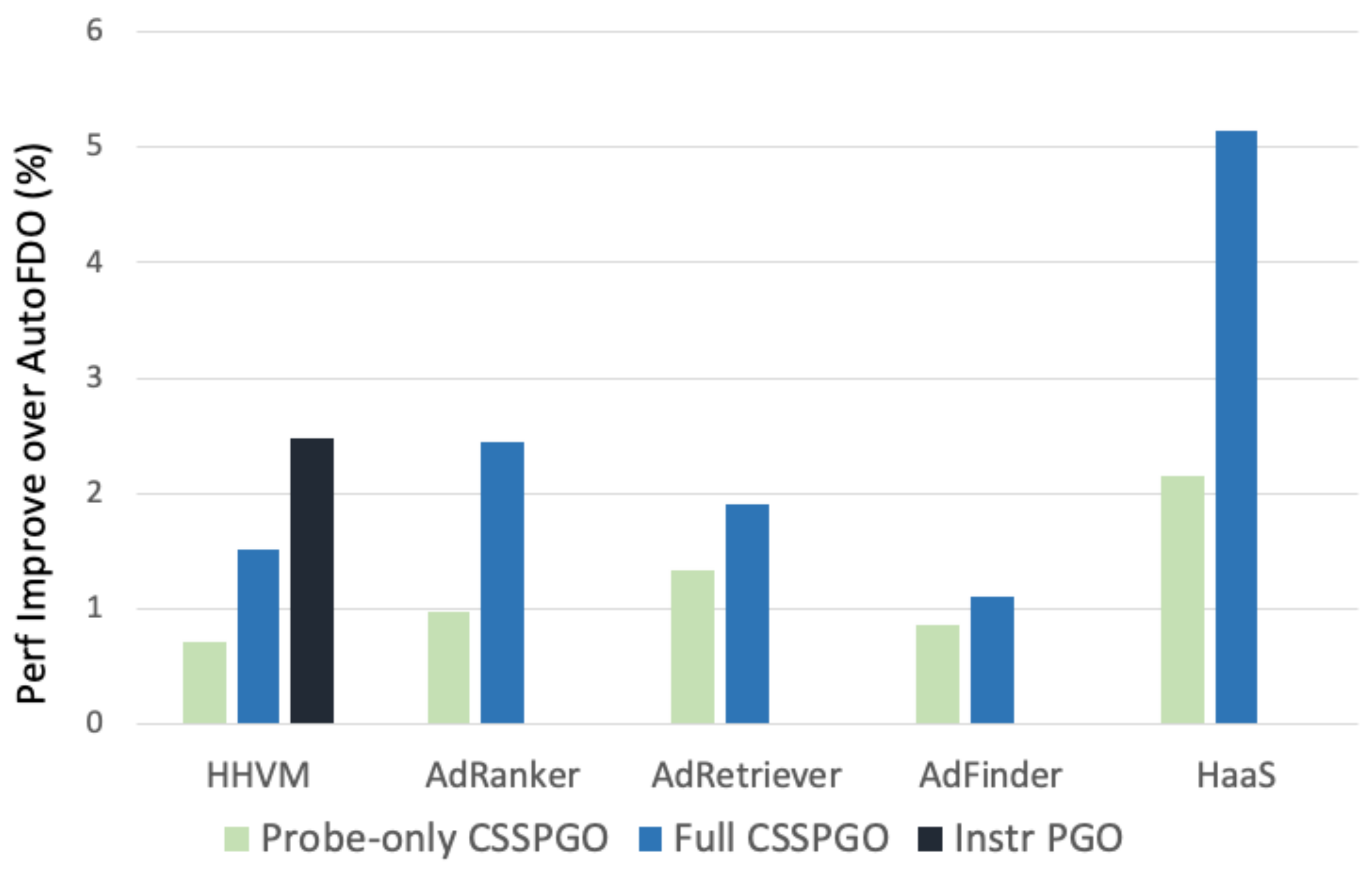
prohibitive for large-scale adoption. Sampling-based PGO, on the other hand, has low entry barriers but it does not deliver the same performance as instrumentation-based PGO.

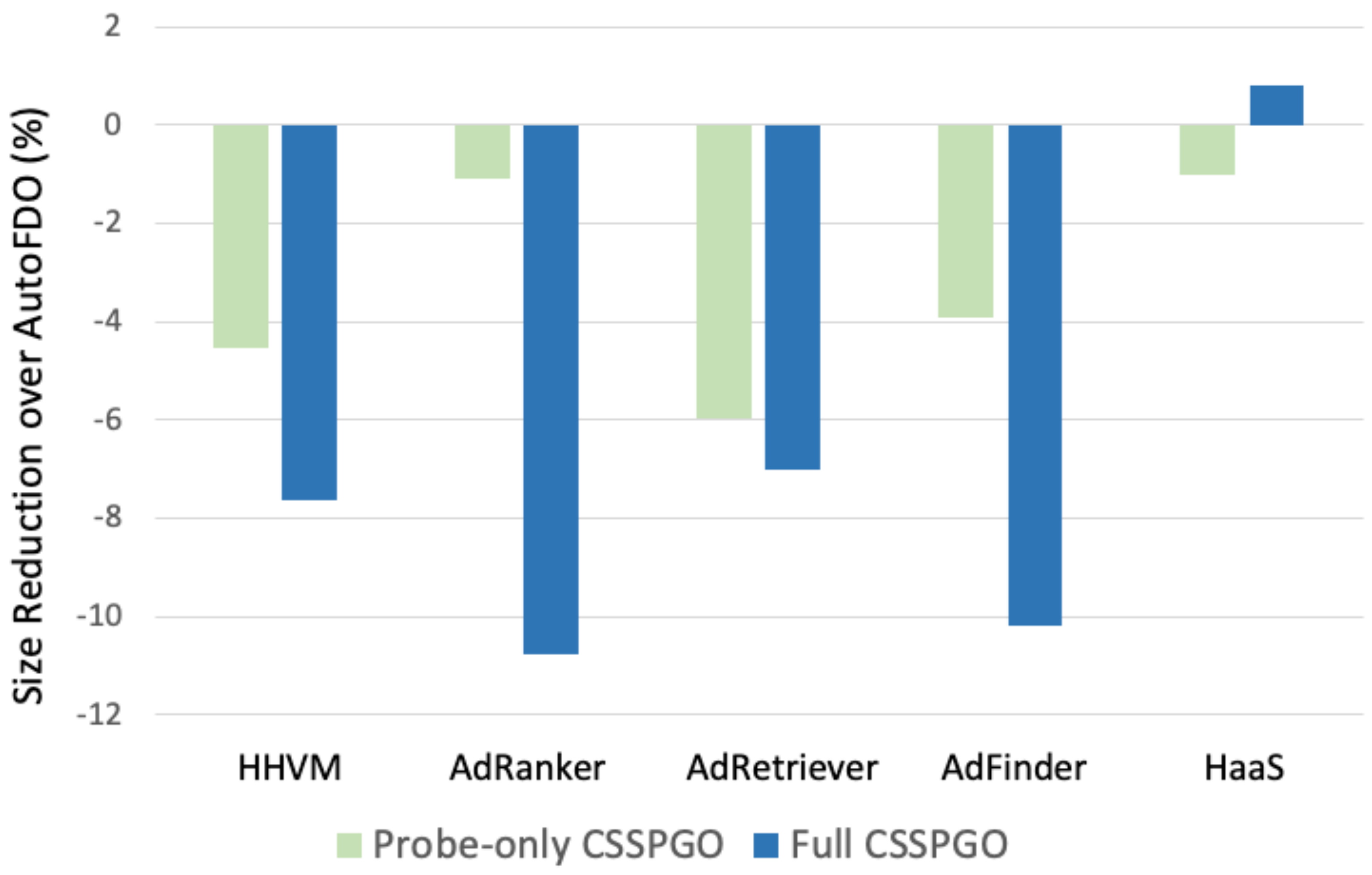
Context-sensitive sampling-based PGO with pseudo-instrumentation (CSSPGO) proposed in this paper provides an alternative solution with better performance than traditional sampling-based PGO while maintaining low operational overhead.

A. Motivation

Large data centers often run a diverse set of workloads. Given that most of the workloads are compute-bound, optimizing CPU performance with compiler optimizations and PGO in particular has proven to be very effective. Within Meta, there are thousands of different back-end services running to serve its users. This requires optimizations to be service-agnostic and have low-operational cost to be used across the entire server fleet, and PGO is no exception. While the most performant variant of PGO is instrumentation PGO, it comes with significant operational complexity. Instrumentation adds non-trivial run-time overhead, so profiling instrumented binary requires special setup for each service, and the instrumented binary usually cannot be run in production environment. Such limitation significantly hinders its adoption.

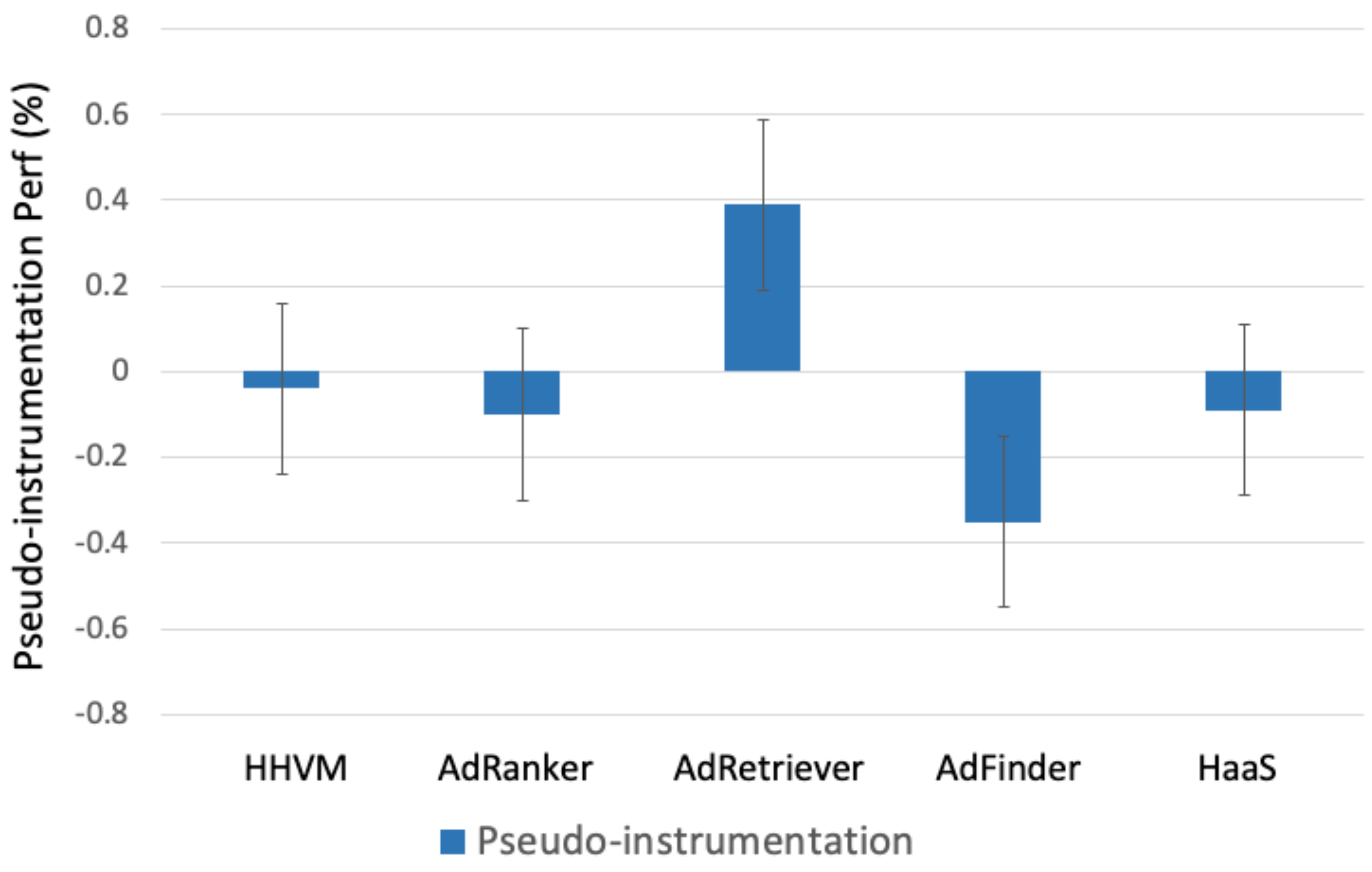
Performance & code size

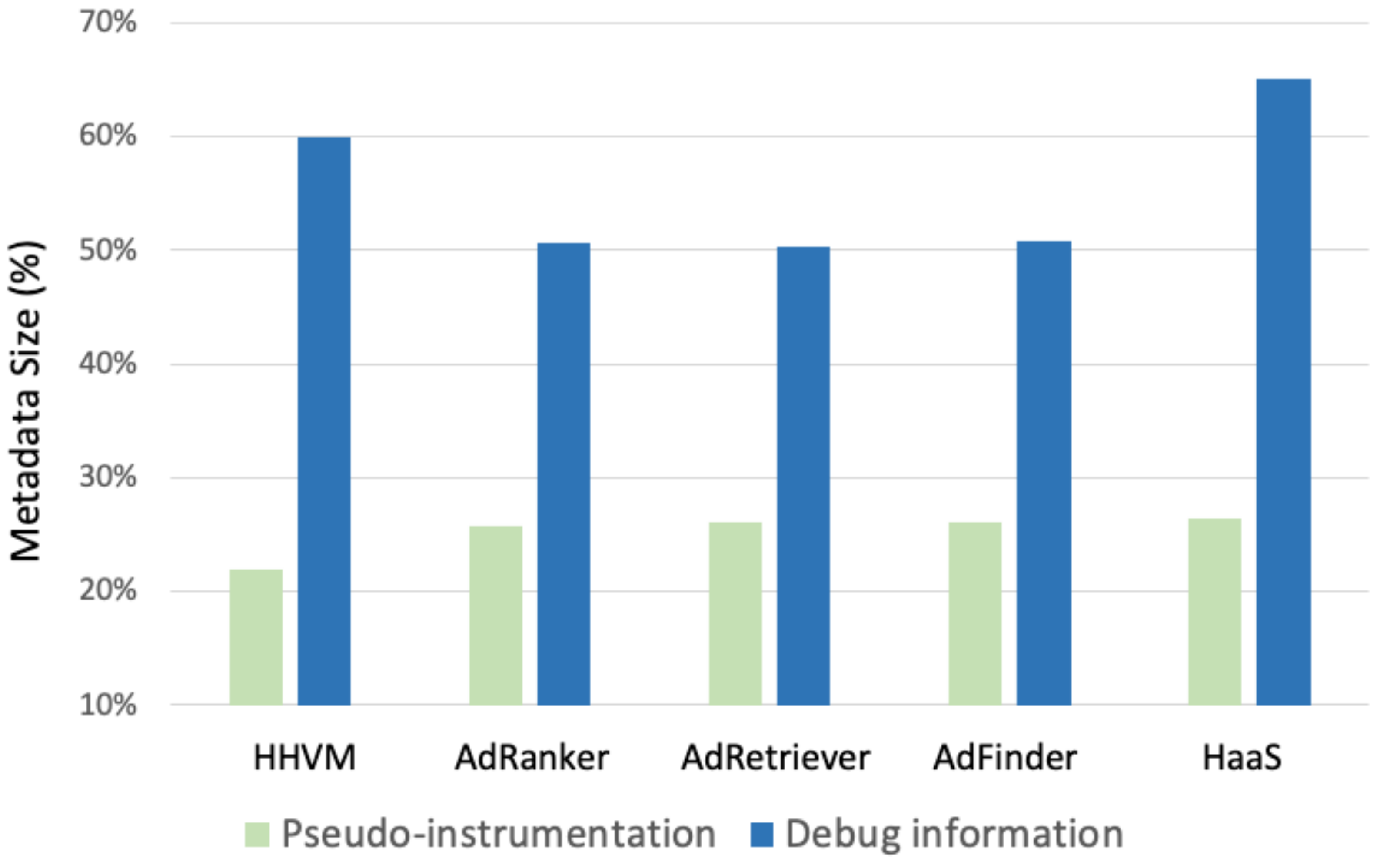




Additional 1-5% performance on top of AutoFDO (60% Instr. PGO benefit) for Meta's server workloads with smaller code size

Overhead





Additional 1-5% performance on top of AutoFDO (60% Instr. PGO benefit) for Meta's server workloads with smaller code size while maintaining ~0% profiling overhead and transparent workflow

Conclusion & Next steps

- Recap
  - Introduced CSSPGO that consists of pseudo-instrumentation and sampling context-sensitive profiling & inlining
  - Demonstrated 1-5% perf on top of AutoFDO for Meta's production data center workloads
- Aspirational challenges
  - If we derive dynamic instructions count from MBFI at the end, is it close to ground truth?
  - If replace all zero count blocks with int3 trap (assuming IR-PGO), will the program run?
  - If we turn off CGSCC inlining, will sample loader inliner or module inliner capture all beneficial inlining?
- How do we get closer
  - Tightening up profile maintenance (updating profile metadata for optimizations)
  - Shifting more profile guided inlining from CGSCC to PGO friendly inliner
  - Infrastructure / verifier to make sure profiles gets updated properly for optimizations

Questions?

Meta