Revamping Sampling-Based PGO with Context-Sensitivity and Pseudo-Instrumentation
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)
Profile Quality

02 PGO Architectural Overview
Profile Correlation

**AutoFDO**
- if.then:
  - call void @do_then() [src.c:6]
  - br label %if.end [src.c:6]
- if.else:
  - call void @do_else() [src.c:8]
- if.end:

**Pseudo Instrumentation**
- if.then:
  - call void @llvm.pseudoprobe(i64 2, ...)
  - call void @do_then() [src.c:6]
  - br label %if.end [src.c:6]
- if.else:
  - call void @llvm.pseudoprobe(i64 3, ...)
  - call void @do_else() [src.c:8]
- if.end:

**Instr PGO**
- if.then:
  - call void @llvm.instrprof.increment(i32 0, ...)
  - call void @do_then() [src.c:6]
  - br label %if.end [src.c:6]
- if.else:
  - call void @llvm.instrprof.increment(i32 1, ...)
  - call void @do_else() [src.c:8]
- if.end:

**Binary**
- 0x20: call <do_then>
- 0x25: jmp <foo+0x2f>
- 0x2a: call <do_else>

**Debug Info**
- 0x20: src.c:6
- 0x25: src.c:6
- 0x2a: src.c:8

**Pseudo Probe**
- 0x20: probe.2
- 0x2a: probe.3

**Instr PGO**
- qword ptr [rip + 0x206f99]
- inc 0x27: call <do_then>
- jmp <foo+0x3d>
- inc 0x32: qword ptr [rip + 0x206fa1]
- call <do_else>
Pseudo-Instrumentation

// 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]>;

<table>
<thead>
<tr>
<th>Comparison on HHVM</th>
<th>AutoFDO</th>
<th>Pseudo Instrumentation</th>
<th>Instrumentation PGO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block overlap</td>
<td>88.2%</td>
<td>92.3%</td>
<td>100%</td>
</tr>
<tr>
<td>Profiling overhead</td>
<td>0%</td>
<td>0.04%</td>
<td>73.06%</td>
</tr>
</tbody>
</table>
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

Pre-in-line call counts

Post-in-line call counts
Context-sensitive Sample Profiling

Traditional LBR sample

\[
\text{LBR}_\text{FROM ADDR16, LBR}_\text{TO ADDR16} | \ldots | \text{LBR}_\text{FROM ADDR1, LBR}_\text{TO ADDR1}
\]

Synchronized stack sample for context-sensitivity

\[
\text{STACK FRAME ADDR1} | \text{STACK FRAME ADDR2} | \ldots | \text{STACK FRAME ADDRn}
\]

Leaf/callee frame

Root/caller frame

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

**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

More details: CGO 2024 paper & LLVM upstream

Production usage at Meta (21 data center regions globally)

<table>
<thead>
<tr>
<th>PGO Type</th>
<th>Instr. PGO</th>
<th>CSSPGO</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU cycles %</td>
<td>~10%</td>
<td>~75%</td>
<td>~15%</td>
</tr>
</tbody>
</table>

HHVM
AdRanker, AdRetriever, AdFinder, HaaS, ...
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?