Stale Profile Matching

AMIR AYUPOV, MAKSIM PANCHENKO, SERGEY PUPYREV
Agenda

1. Motivation
2. Profile staleness
3. Stale matching
4. Evaluation
5. Conclusion
PGO/BOLT pipeline
PGO/BOLT pipeline

• Max PGO/BOLT effect: profiled binary = optimized binary ("zero gap")
PGO/BOLT pipeline

- Max PGO/BOLT effect: profiled binary = optimized binary ("zero gap")
PGO/BOLT pipeline

• Max PGO/BOLT effect: profiled binary = optimized binary ("zero gap")

• Compromise for CI/CD: “some gap”
PGO/BOLT pipeline

• Max PGO/BOLT effect: profiled binary = optimized binary (“zero gap”)

• Compromise for CI/CD: “some gap”

Ideal pipeline (“Zero Gap”)

CI/CD + Continuous Profiling
PGO/BOLT pipeline

- Max PGO/BOLT effect: profiled binary = optimized binary ("zero gap")
- Compromise for CI/CD: "some gap"

Ideal pipeline ("Zero Gap")

CI/CD + Continuous Profiling
Stale profile characteristics

Sources of BOLT profile staleness:
1. Source code changes
2. Codegen differences (padding, offsets)
3. Inlining => waterfall effect
Stale profile characteristics

Sources of BOLT profile staleness:
1. Source code changes
2. Codegen differences (padding, offsets)
3. Inlining => waterfall effect
Stale profile characteristics

Sources of BOLT profile staleness:
1. Source code changes
2. Codegen differences (padding, offsets)
3. Inlining => waterfall effect
Stale profile characteristics

Sources of BOLT profile staleness:
1. Source code changes
2. Codegen differences (padding, offsets)
3. Inlining => waterfall effect
Stale profile characteristics

Sources of BOLT profile staleness:
1. Source code changes
2. Codegen differences (padding, offsets)
3. Inlining => waterfall effect
Stale profile characteristics

Sources of BOLT profile staleness:
1. Source code changes
2. Codegen differences (padding, offsets)
3. Inlining => waterfall effect
Stale profile characteristics

Sources of BOLT profile staleness:
1. Source code changes
2. Codegen differences (padding, offsets)
3. Inlining => waterfall effect
Problem statement
Problem statement

Binary $B$: set of functions with CFG
Problem statement

Binary $B$: set of functions with CFG
Profile $P$: set of node/edge counts, metadata (names, hashes)
Problem statement

Binary $B$: set of functions with CFG

Profile $P$: set of node/edge counts, metadata (names, hashes)

Assume $B_{old}$, $B_{new}$ and $P_{old}$
Problem statement

Binary $B$: set of functions with CFG
Profile $P$: set of node/edge counts, metadata (names, hashes)

Assume $B_{old}$, $B_{new}$ and $P_{old}$

Goal: infer counts in $B_{new}$ using $P_{old}$
Problem statement

Binary $B$: set of functions with CFG
Profile $P$: set of node/edge counts, metadata (names, hashes)

Assume $B_{old}$, $B_{new}$ and $P_{old}$

Goal: infer counts in $B_{new}$ using $P_{old}$
3. STALE MATCHING

Problem statement

Binary $B$: set of functions with CFG
Profile $P$: set of node/edge counts, metadata (names, hashes)

Assume $B_{old}$, $B_{new}$ and $P_{old}$

Goal: infer counts in $B_{new}$ using $P_{old}$

Stale matching

1. Assign profile: match functions in $B_{new}$ and $P_{old}$ by name
**Problem statement**

Binary $B$: set of functions with CFG
Profile $P$: set of node/edge counts, metadata (names, hashes)
Assume $B_{old}$, $B_{new}$ and $P_{old}$
Goal: infer counts in $B_{new}$ using $P_{old}$

**Stale matching**

1. Assign profile: match functions in $B_{new}$ and $P_{old}$ by name
2. Matching for nodes: hash-based, strict to loose
Problem statement

Binary $B$: set of functions with CFG
Profile $P$: set of node/edge counts, metadata (names, hashes)
Assume $B_{old}$, $B_{new}$ and $P_{old}$
Goal: infer counts in $B_{new}$ using $P_{old}$

Stale matching

1. Assign profile: match functions in $B_{new}$ and $P_{old}$ by name
2. Matching for nodes: hash-based, strict to loose
3. Inference: Profile inference revisited, POPL 2022
03 STALE MATCHING
03 STALE MATCHING

Matching:
- oxA1
- oxB2
- oxC3
- oxD4
- oxF6
- oxEF

Inference:
- 0xA1
- 0xB2
- 0xC3
- 0xXX
- 0xYY
- 325
Evaluation
Evaluation

Open-source benchmarks
Evaluation

Open-source benchmarks
• Compilers: Clang, GCC
Evaluation

Open-source benchmarks
• Compilers: Clang, GCC
• DBs: RocksDB, MySQL
Evaluation

Open-source benchmarks

• Compilers: Clang, GCC
• DBs: RocksDB, MySQL
• Browser: Chromium
Evaluation

Open-source benchmarks
• Compilers: Clang, GCC
• DBs: RocksDB, MySQL
• Browser: Chromium

Internal workloads
Evaluation

Open-source benchmarks
• Compilers: Clang, GCC
• DBs: RocksDB, MySQL
• Browser: Chromium

Internal workloads

Relative speedup over no-BOLT

<table>
<thead>
<tr>
<th></th>
<th>clang</th>
<th>gcc</th>
<th>rocksdb</th>
<th>mysql</th>
<th>chromium</th>
</tr>
</thead>
<tbody>
<tr>
<td>fresh profile</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
</tbody>
</table>

fraction of the maximum BOLT speedup
Evaluation

Open-source benchmarks
- Compilers: Clang, GCC
- DBs: RocksDB, MySQL
- Browser: Chromium

Internal workloads

Prior art: 0.15-0.45x of fresh profile perf


**Evaluation**

Open-source benchmarks
- Compilers: Clang, GCC
- DBs: RocksDB, MySQL
- Browser: Chromium

Internal workloads

<table>
<thead>
<tr>
<th>Tool</th>
<th>Fresh Profile</th>
<th>Stale Profile</th>
<th>Stale Profile Matching</th>
</tr>
</thead>
<tbody>
<tr>
<td>clang</td>
<td>1.0</td>
<td>0.8</td>
<td>0.6</td>
</tr>
<tr>
<td>gcc</td>
<td>1.0</td>
<td>0.8</td>
<td>0.6</td>
</tr>
<tr>
<td>rocksdb</td>
<td>1.0</td>
<td>0.8</td>
<td>0.6</td>
</tr>
<tr>
<td>mysql</td>
<td>1.0</td>
<td>0.8</td>
<td>0.6</td>
</tr>
<tr>
<td>chromium</td>
<td>1.0</td>
<td>0.8</td>
<td>0.6</td>
</tr>
</tbody>
</table>

Prior art: 0.15-0.45x of fresh profile perf
Stale Matching: 0.64-0.79x
Evaluation

Open-source benchmarks
- Compilers: Clang, GCC
- DBs: RocksDB, MySQL
- Browser: Chromium

Internal workloads

Relative speedup over no-BOLT

Prior art: 0.15-0.45x of fresh profile perf
Stale Matching: 0.64-0.79x
Conclusion
Conclusion

1. Contributions
   • Investigated profile staleness problem, formal model
   • Two-stage algorithm
   • Implementation in LLVM BOLT
   • Evaluation on open-source binaries and internal workloads
Conclusion

1. Contributions
   • Investigated profile staleness problem, formal model
   • Two-stage algorithm
   • Implementation in LLVM BOLT
   • Evaluation on open-source binaries and internal workloads

2. Future directions
   • Adapt to PGO, PLO
   • Make compiler output more stable