AutoFDO: Automatic Feedback-Directed Optimization for Warehouse-Scale Applications

Dehao Chen, David Xinliang Li, Tipp Moseley

Presenter: Yifan Zhao

FDO: Feedback-directed Optimization

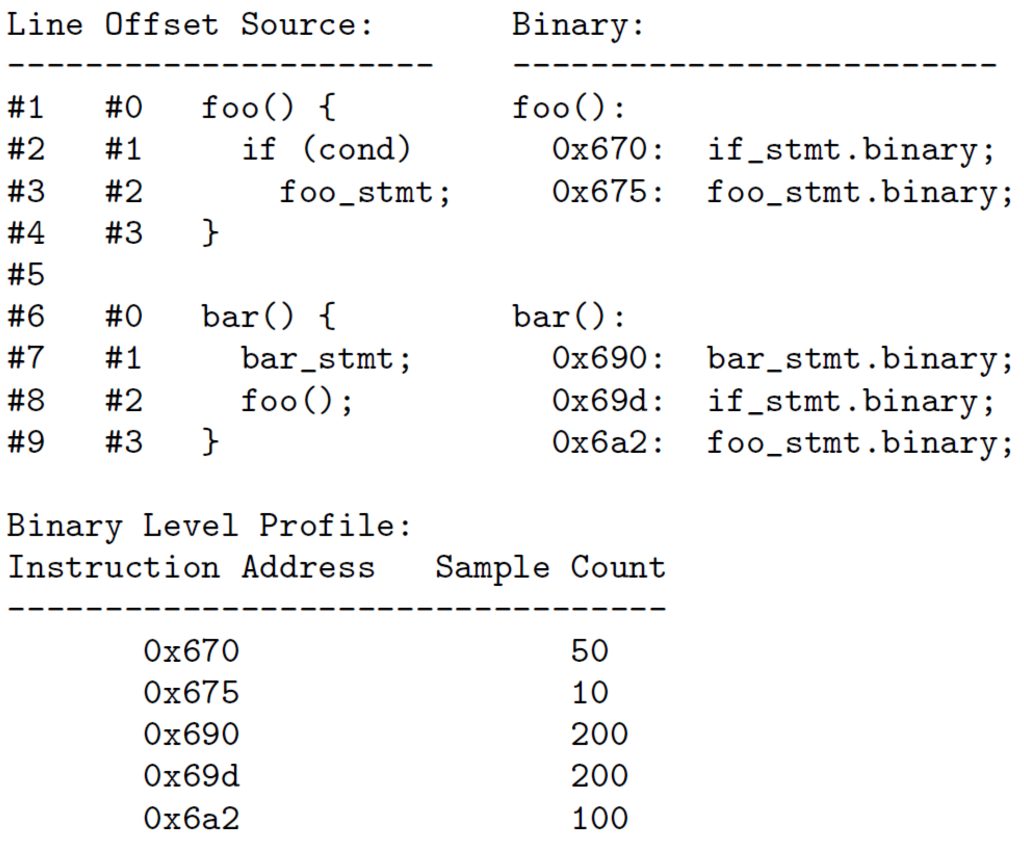
- Steps:
  - Compile with instrumentation
  - Run a benchmark to generate *representative profile*
  - Recompile with the profile
- Representative profile requires representative input
- Many other problems with Google datacenter

Problems

- Binaries change rapidly between releases
  - Best if tolerating stale profiles
- Programs deal with sensitive data
  - Profile could expose everything to an attack
- Instrumentation overhead triggers timeouts on servers
  - Instrumented code runs differently from “release” code
  - Lead to even less representative profiles

Solution and Key Insights

- Hardware sampling
  - Instruction frequency
  - Branch taken frequency

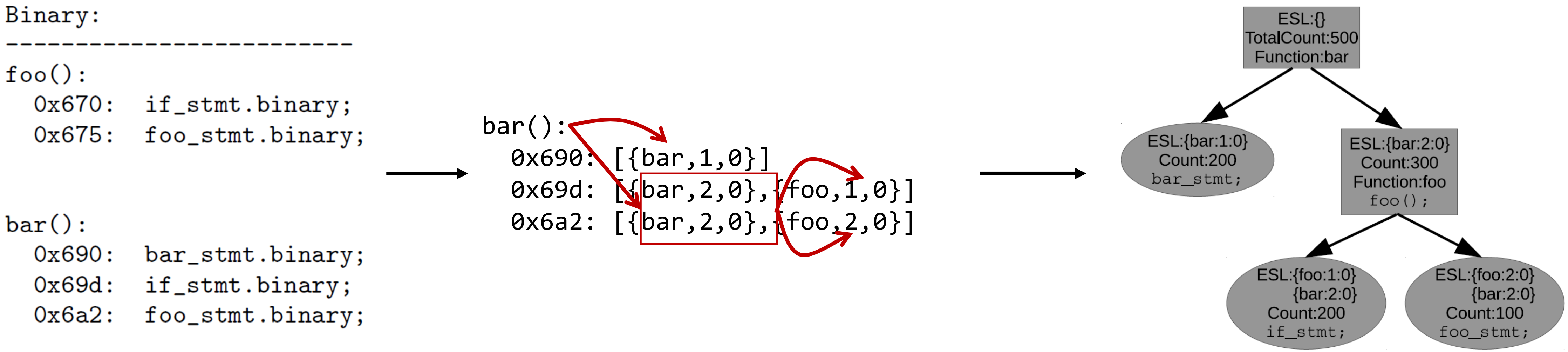


```
Line Offset Source:          Binary:
----------------------       -------------------------
#1    #0    foo() {          foo():
#2    #1      if (cond)        0x670:  if_stmt.binary;
#3    #2        foo_stmt;      0x675:  foo_stmt.binary;
#4    #3    }
#5
#6    #0    bar() {          bar():
#7    #1      bar_stmt;        0x690:  bar_stmt.binary;
#8    #2      foo();           0x69d:  if_stmt.binary;
#9    #3    }                  0x6a2:  foo_stmt.binary;

Binary Level Profile:
Instruction Address   Sample Count
----------------------------------
       0x670                 50
       0x675                 10
       0x690                200
       0x69d                200
       0x6a2                100
```

Solution and Key Insights

- Hardware sampling
- Profile mapping
  - Source location
  - Extended source location (ESL)
  - ESL on the tree



```
Binary:
-------------------------
foo():
  0x670:  if_stmt.binary;
  0x675:  foo_stmt.binary;


bar():
  0x690:  bar_stmt.binary;
  0x69d:  if_stmt.binary;
  0x6a2:  foo_stmt.binary;
```

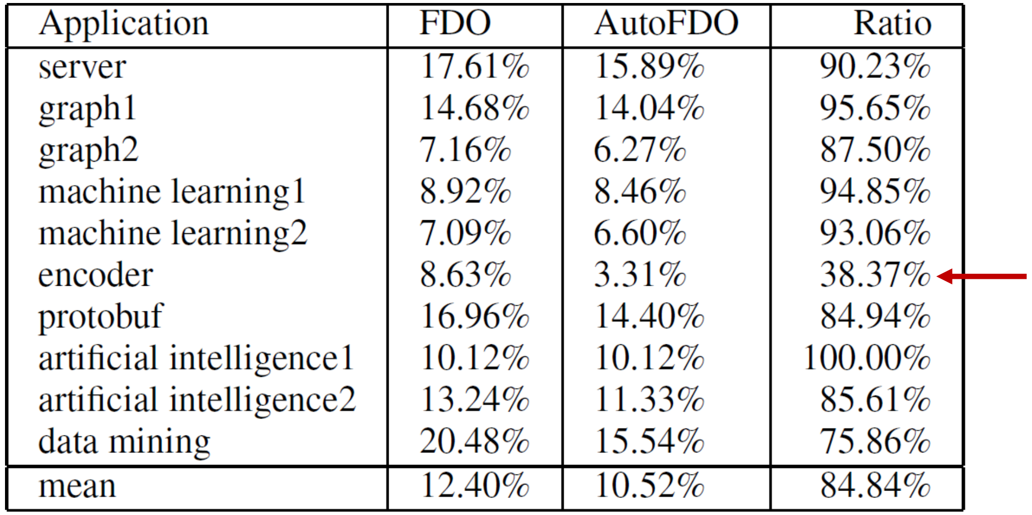
```
bar():
  0x690: [{bar,1,0}]
  0x69d: [{bar,2,0},{foo,1,0}]
  0x6a2: [{bar,2,0},{foo,2,0}]
```

Solution and Key Insights

- Hardware sampling
- Profile mapping
- Top-first profile collection
    - Get a 10-second profile from 10% of machines each day
    - Low coverage in general -- 0.001% of observable cycles
    - Enough coverage for top workloads

Evaluation

- AutoFDO, vs FDO
  - Achieve >90% of the FDO speedup
  - Gap mostly due to inaccurate debug info (not sampling)
  - Mostly happen to loop nesting

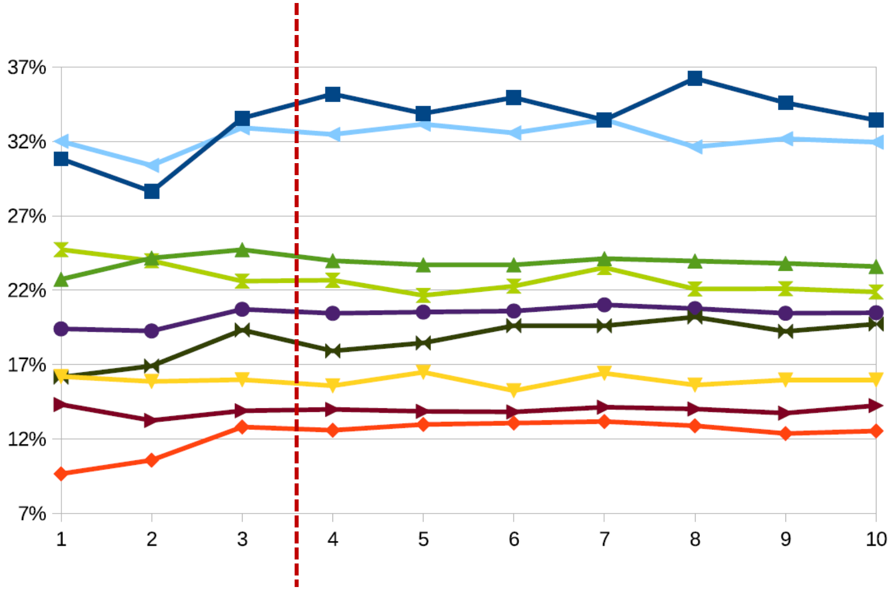


Application	FDO	AutoFDO	Ratio
server	17.61%	15.89%	90.23%
graph1	14.68%	14.04%	95.65%
graph2	7.16%	6.27%	87.50%
machine learning1	8.92%	8.46%	94.85%
machine learning2	7.09%	6.60%	93.06%
encoder	8.63%	3.31%	38.37%
protobuf	16.96%	14.40%	84.94%
artificial intelligence1	10.12%	10.12%	100.00%
artificial intelligence2	13.24%	11.33%	85.61%
data mining	20.48%	15.54%	75.86%
mean	12.40%	10.52%	84.84%

Mostly
nested loops

Evaluation

- AutoFDO, vs FDO
- Iterative AutoFDO
  - Performance varies in some apps
    - Again due to loop nesting
  - Usually not improving beyond 3 iterations
    - After all hot callees are inlined



Evaluation

- AutoFDO, vs FDO
- Iterative AutoFDO
- Stale profiles
  - Line number in function outperforms line number in file
  - Inserting a function does not break the former
  - 50% of the speedup with 6-months old profile

Strengths and Weaknesses

- Design strengths
  - Handling degraded debugging information
  - Integration into compiler
  - Deferring symbolization
  - Support of iterative FDO
- Shortcomings
  - Obscuring bugs in apps (more than a typical -O2 flag)
  - If stability comes before average performance
  - Breaking *release integration*

Open Questions

- AutoFDO is *automated* FDO, not quite *advanced* FDO
  - Issues of traditional FDO remains
  - Security, input dependence, etc.
- AutoFDO samples and transforms text segment
  - How about extending FDO to data transformation
- Cross-module optimization
  - AutoLIPO / ThinLTO