Practical Path Profiling for Dynamic Optimizers

Michael Bond, UT Austin
Kathryn McKinley, UT Austin

Why path profiling?

- Processors need long instruction sequences
- Programs have branches
Why path profiling?

- Compiler identifies hot paths across multiple basic blocks

Why path profiling?

- Compiler identifies hot paths across multiple basic blocks
  - Forms and optimizes “traces”
Why path profiling?

- Compiler identifies hot paths across multiple basic blocks
  - Forms and optimizes “traces”

---

Why path profiling?

- Compiler identifies hot paths across multiple basic blocks
  - Forms and optimizes “traces”

---

Less aggressive  More aggressive
Superblocks  Hyperblocks  Dynamo fragments  rePlay frames  MSSP tasks
Ball-Larus path profiling

- Instrumentation measures execution frequency of each path
- Acyclic, intraprocedural paths

Edge profiling

- Hardware or sampling
- Estimate hot paths from edge profile
Ideal for dynamic optimizer

Targeted path profiling [Joshi et al. ’04]

- Profile-guided profiling
- Accuracy good
- Overhead high for dynamic optimizer
Practical path profiling

Outline

- Background
  - Staged dynamic optimization
  - Profile-guided profiling
  - Ball-Larus path profiling
- Practical path profiling
- Methodology
  - Edge profile-guided inlining and unrolling
  - Measuring accuracy with branch-flow metric
- Accuracy and overhead
Staged dynamic optimization

Stage 0

Static optimizations

Sampling-based edge profiler

Edge profile
Staged dynamic optimization

Stage 0
- Static optimizations
- Sampling-based edge profiler

Stage 1
- Local optimizations incl. inlining & unrolling

Sampling-based edge profiler

Staged dynamic optimization

Stage 0
- Static optimizations
- Sampling-based edge profiler

Stage 1
- Local optimizations incl. inlining & unrolling

- Larger routines
- Longer paths
- More challenging platform for path profiling
Staged dynamic optimization

Stage 0
- Static optimizations
- Sampling-based edge profiler

Stage 1
- Local optimizations incl. inlining & unrolling
- Path profiling instrumentation

Staged dynamic optimization

Stage 0
- Static optimizations
- Sampling-based edge profiler

Stage 1
- Local optimizations incl. inlining & unrolling
- Path profiling instrumentation
Staged dynamic optimization

Stage 0
- Static optimizations
- Sampling-based edge profiler

Stage 1
- Local optimizations incl. inlining & unrolling
- Path profiling instrumentation
- Edge profile

Stage 2
- Global optimizations
- Path profile

Edge profile:
- Identifies hot and cold edges
- Provides partial path profile
Profile-guided profiling

Stage 0
- Static optimizations

Stage 1
- Local optimizations incl. inlining & unrolling
- Path profiling instrumentation

Stage 2
- Global optimizations

Edge profile:
- Identifies hot and cold edges
- Provides partial path profile

Ball-Larus path profiling

- Acyclic, intraprocedural paths
  - Handles cyclic routines
- Instrumentation maintains execution frequency of each path
  - Each path computes unique integer in \([0, N-1]\)
Ball-Larus path profiling

- 4 paths $\rightarrow [0, 3]$

Each path sums to unique integer
Ball-Larus path profiling

- 4 paths $\to [0, 3]$
- Each path sums to unique integer

Path 0

Ball-Larus path profiling

- 4 paths $\to [0, 3]$
- Each path sums to unique integer

Path 0
Path 1
Ball-Larus path profiling

- 4 paths $\rightarrow [0, 3]$
- Each path sums to unique integer

Path 0
Path 1
Path 2
Path 3
Ball-Larus path profiling

- **r**: path register
  - Computes path number
- **count**: Stores path frequencies

- Array by default
- Too many paths?
  - Hash table
  - High overhead
Outline

- Background
  - Ball-Larus path profiling
  - Staged dynamic optimization
  - Profile-guided profiling
- Practical path profiling
- Methodology
  - Edge profile-guided inlining and unrolling
  - Measuring accuracy with branch-flow metric
- Accuracy and overhead

Practical path profiling

- Goal: Reduce instrumentation overhead without hurting accuracy
  - Use profile-guided profiling
- Strategies
  - Decrease number of possible paths
  - Avoid instrumenting paths edge profile predicts well
  - Simplify instrumentation on profiled paths
Practical path profiling

- Goal: Reduce instrumentation overhead without hurting accuracy
  - Use profile-guided profiling

- Strategies
  - Decrease number of possible paths
  - Avoid instrumenting paths edge profile predicts well
  - Simplify instrumentation on profiled paths

- Techniques from targeted path profiling
  - Improves techniques
  - Adds new techniques

Strategy 1: Fewer possible paths

- Goal: Hash table → array
- Want to remove cold paths
Strategy 1: Fewer possible paths

- Goal: Hash table → array
- Want to remove cold paths
- Observation: A path with a cold edge is a cold path
- Remove cold edges
  - Local and global criteria
- Paths: 16 → 4
Strategy 1: Fewer possible paths

- Remaining paths potentially hot
- 4 paths $\rightarrow [0, 3]$
Strategy 1: Fewer possible paths

- What if cold edge taken?

Cold edges “poison” path register
  - Set it to \( N \)
  - Cold paths use \([N, 2N-1]\)
Strategy 1: Fewer possible paths

- What if cold edge taken?
- Cold edges “poison” path register
  - Set it to \( N \)
  - Cold paths use \([N, 2N-1]\)

- What if still too many possible paths?

New
Strategy 2: Avoid instrumenting paths

- Consider right half of CFG
  - *Obvious* paths: Each path has an edge unique to it
  - Edge profile provides perfect path profile
Strategy 2: Avoid instrumenting paths

- Consider right half of CFG
  - *Obvious* paths: Each path has an edge unique to it
  - Edge profile provides perfect path profile
- We don’t instrument the right half of the CFG

Synergy: Cold edge removal creates more obvious paths
Strategy 2: Avoid instrumenting paths

- Synergy: Cold edge removal creates more obvious paths
  - Right half is obvious

What if cold edge is part of obvious and non-obvious paths?
Strategy 2: Avoid instrumenting paths

- What if cold edge is part of obvious and non-obvious paths?
- Right half obvious

![Diagram]

Strategy 2: Avoid instrumenting paths

- What if cold edge is part of obvious and non-obvious paths?
- Right half obvious
  - But we haven’t avoided instrumenting it!
Strategy 2: Avoid instrumenting paths

- What if cold edge is part of obvious and non-obvious paths?
- Right half obvious
  - But we haven’t avoided instrumenting it!
- Aggressive instrumentation pushing

Strategy 2: Avoid instrumenting paths

- Overcounts some hot paths
Strategy 2: Avoid instrumenting paths

- Overcounts some hot paths
- Example cold path counts hot path number 1
- Overcount tends to be small

Some paths need profiling

- Correlation between cascading branches
Strategy 3: Simplify instrumentation

- Moderately biased branches

Strategy 3: Simplify instrumentation

- Moderately biased branches
- Put zeros on hotter edges
Strategy 3: Simplify instrumentation

- Moderately biased branches
- Put zeros on hotter edges
  - No instrumentation on hotter edges

Outline

- Background
  - Staged dynamic optimization
  - Profile-guided profiling
  - Ball-Larus path profiling
- Practical path profiling
- Methodology
  - Edge profile-guided inlining and unrolling
  - Measuring accuracy with branch-flow metric
- Accuracy and overhead
Methodology

- Path profiling implemented in Scale [McKinley et al.]
  - Ahead-of-time compiler → deterministic platform
- Edge profile-guided inlining and unrolling precede path profiling

- Alpha binaries for subset of SPEC2000
  - C and Fortran 77 only
  - Scale wouldn’t compile gzip, vortex, gcc
- ref inputs for all runs
Measuring accuracy

- Compare estimated profile with actual profile
  - Wall weight matching* or profile overlap
- Weight paths by flow: amount of execution
  - Previous work measures flow with unit-flow metric
    \[ \text{Flow}(p) = \text{Freq}(p) \]
  - We introduce \textit{branch-flow} metric
    \[ \text{Flow}(p) = \text{Freq}(p) \times \text{NumBranches}(p) \]

Motivating the branch-flow metric

- Programs really execute one very long path
Motivating the branch-flow metric

- Programs really execute one very long path

Ball-Larus path profiling breaks it into multiple acyclic, intraprocedural paths.

Motivating the branch-flow metric

- Programs really execute one very long path
  - Ball-Larus path profiling breaks it into multiple acyclic, intraprocedural paths.
Motivating the branch-flow metric

- Some paths longer than others
  - We care more about longer paths
  - Unit-flow metric unfairly rewards edge profiling

Outline

- Background
  - Staged dynamic optimization
  - Profile-guided profiling
  - Ball-Larus path profiling
- Practical path profiling
- Methodology
  - Edge profile-guided inlining and unrolling
  - Measuring accuracy with branch-flow metric
- Accuracy and overhead
Accuracy

![Accuracy Chart]

Overview of accuracy for various profiling methods.

Overhead

![Overhead Chart]

Comparison of overhead across different profiling methods.
Related work

- **Dynamo** [Bala et al. ’00]
  - Successful path-based dynamic optimizer
  - “Bails out” when no dominant path

- Instrumentation sampling & dynamic instrumentation
  [Arnold & Ryder ’01, Hirzel & Chilimbi ’04, Yasue et al. ’04]
  - Lower overhead by extending profiling time
  - Orthogonal to practical path profiling

- **Hardware-based path profiling** [Vaswani et al. ’05]
  - High accuracy when hot path table large enough

---

Summary

![Summary Diagram]

**Contributions:**
- Inlining and unrolling
- Branch-flow metric
- Practical path profiling
Questions?