Post-Link Outlining for Code Size Reduction

Shaobai Yuan, Jihong He, Yihui Xie, Feng Wang, Jie Zhao

Hunan University, Changsha, China

CC 2025

Outline

Motivation

- Code size reduction is critical for resource-constrained environments
- Traditional compile-time optimizations focus primarily on performance rather than code size.
- Outlining is a transformation that extracts repeated instruction sequences into separate functions, trading performance for code size reduction.

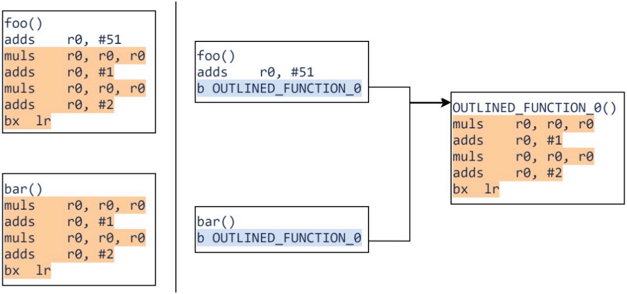


Figure: outlining

Motivation

- **Compile-time Outlining:**
  - LLVM's `-moutline`, `-Oz` enable aggressive outlining.
  - But limited by local scope.
- **LTO or Linker Outlining:**
  - Integrates global knowledge at link time.
  - May not consider dynamic information.
  - Some approaches require changing the build pipeline significantly.
  - Rely on LLVM
- **BOLT (Binary Optimization and Layout Tool):**
  - A post-link optimizer that can reorder code, integrate PGO, etc.
  - **Opportunity:** No deep changes to compilation flow.
- **Goal:** Develop a post-link outlining(PLOS) approach to achieve further code size reduction by leveraging a *whole-program* perspective without altering the standard LLVM/GCC build flow.

Contribution

- **Contribution:** Our PLOS approach extends BOLT to add a post-link outlining, enabling:
  - Fine-grained repeated sequence detection.
  - Careful stack frame management.
  - Nested outlining (supporting multiple layers).
  - Integration with Profile-Guided Optimization (PGO).

Overall Flow of PLOS

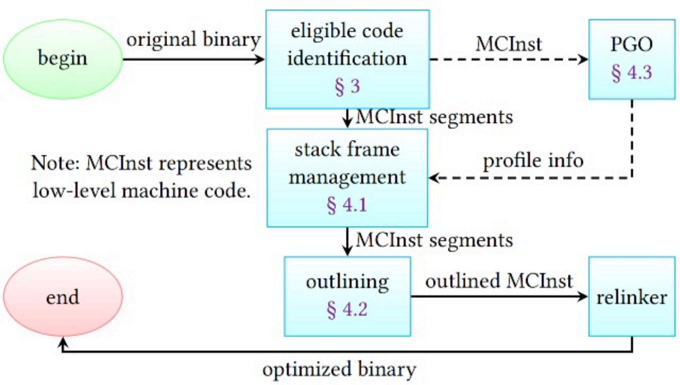


Figure: *

PLOS Flow: Disassemble → Identify repeated sequences → Outline → Relink

Key Techniques

(1) Stack Frame Management

- Generate minimal prologue/epilogue in the outlined function.
- Properly adjust stack pointer alignment (e.g., 16-byte for AArch64).
- Offset stack accesses when needed, enabling bigger extraction scopes.
- Tail call optimization: converting `bl` to `b` if call is last in the function.

(2) Nested Outlining

- Further outline newly created outlined functions themselves if repeated code persists.
- Manage second-level or deeper extractions carefully (stack offsets, tail calls again).
- Post-link shrink-wrapping: remove unnecessary prologues/epilogues if the new function flow is simple.

Key Techniques

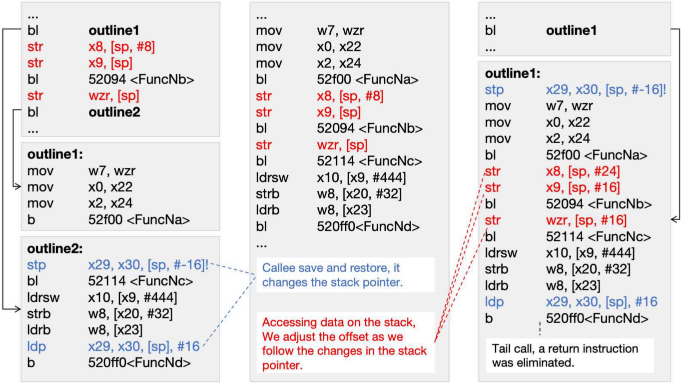
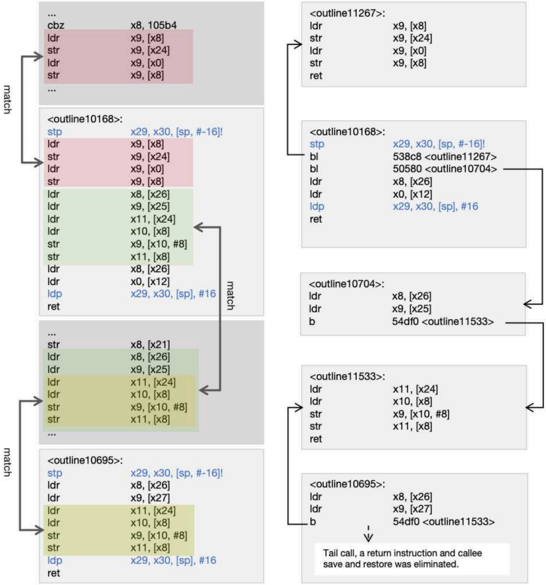


Figure: Stack Frame Management

Key Techniques: Nested-Outlining



PGO Integration

- BOLT can analyze runtime profiles under typical workloads.
- PLOS outlines *cold* code segments more aggressively, while leaving hot segments intact to reduce performance overhead.
- Achieve a balance between code size savings and performance.
- No major changes to standard compilation flow. Post-link stage is fully decoupled from the main build pipeline.

Code Size Reduction

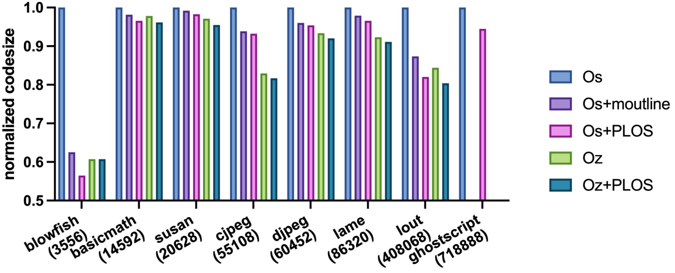


Figure: Code size reduction for Mibench (normalized to '-Os').

- **PLOS achieves a mean code size reduction of 10.88%, up to 43.53%. (compare to -Os)**
- **PLOS achieves a mean code size reduction of 1.76%, up to 4.75%. (compare to -Oz) (Outlining is applied at compile time.)**

Performance Trade-off

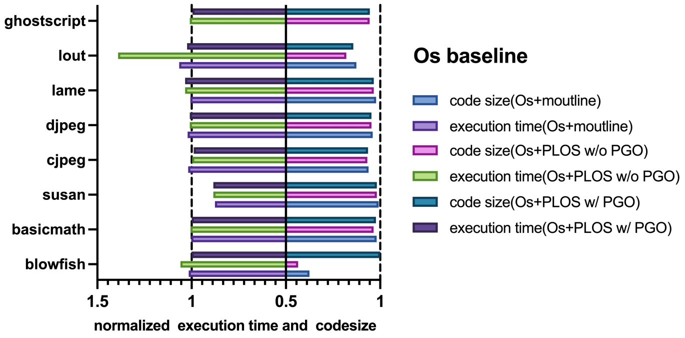


Figure: Performance trade-off for Mibench (normalized to '-Os').

With PGO, performance degradation remains below 3% while preserving code size benefits.

Experimental Comparison: PLOS vs. SOTA

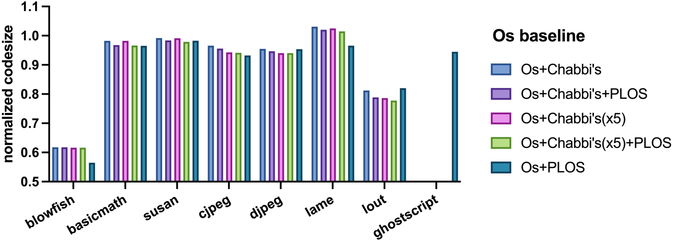


Figure: Comparison of Code Size Reduction: LTO vs. SOTA

- **PLOS achieves a mean code size reduction of 2.88% and up to 8.56%.**

Future Work

- **Enhanced Cost Model** we aim to explore more sophisticated or machine-learning-based models to better balance code size reduction and performance under various scenarios.
- **Finer-Grained Profile-Guided Optimization** By using Finer-Grained profiling (e.g., at the basic-block or instruction level), we could preserve performance for hot paths more accurately, while aggressively outlining cold paths to further shrink code size.
- **Support for Additional Architectures** In the future, we plan to extend the stack-frame management and offsetting logic to platforms like x86 and RISC-V, verifying the method's versatility and scalability
- **Combining with Other Post-Link Techniques** We intend to explore integrating PLOS with these existing optimizations to further enhance both code size reduction and performance.

Thank you!

Q & A