January 2026

APGO

Architecture-agnostic Profile-Guided Optimization

"Profile Once, Optimize Anywhere"

Lei Qiu[1,3] **Yikang Fan**[1,2] **Yanxia Wu**[2] **Fang Lyu**[1]

[1]SKLP, Institute of Computing Technology, CAS

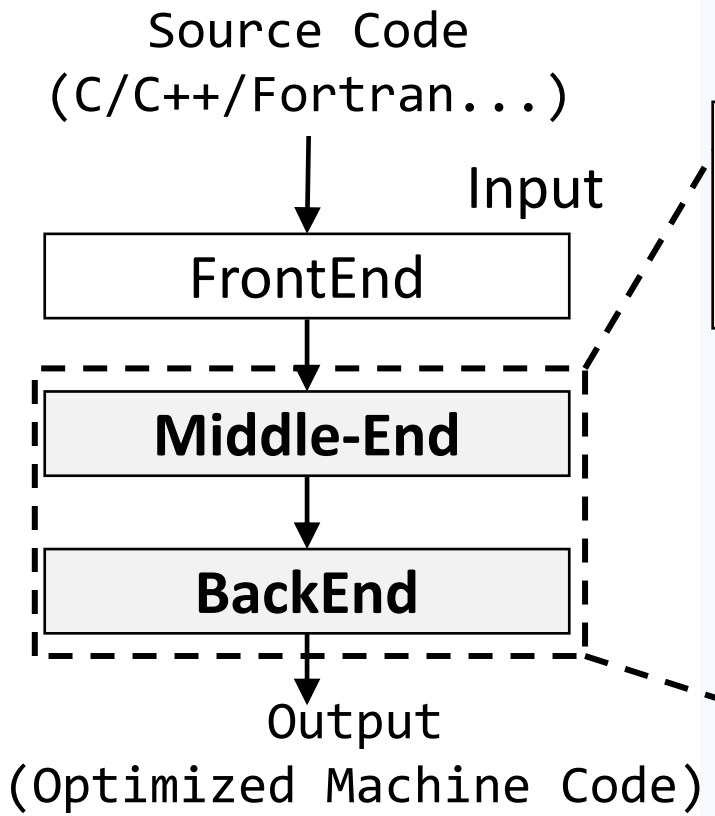
[2]Harbin Engineering University, China

[3]University of Chinese Academy of Sciences, Beijing, China

Presentation Overview

- Introduction to Profile-Guided Optimization (PGO)
- The Problem: Cross-Architecture PGO Challenges
- APGO Solution: Architecture-Agnostic Framework
- Experimental Results & Performance Analysis
- Conclusion & Future Work

What is Profile-Guided Optimization (PGO)?



Traditional Compilation

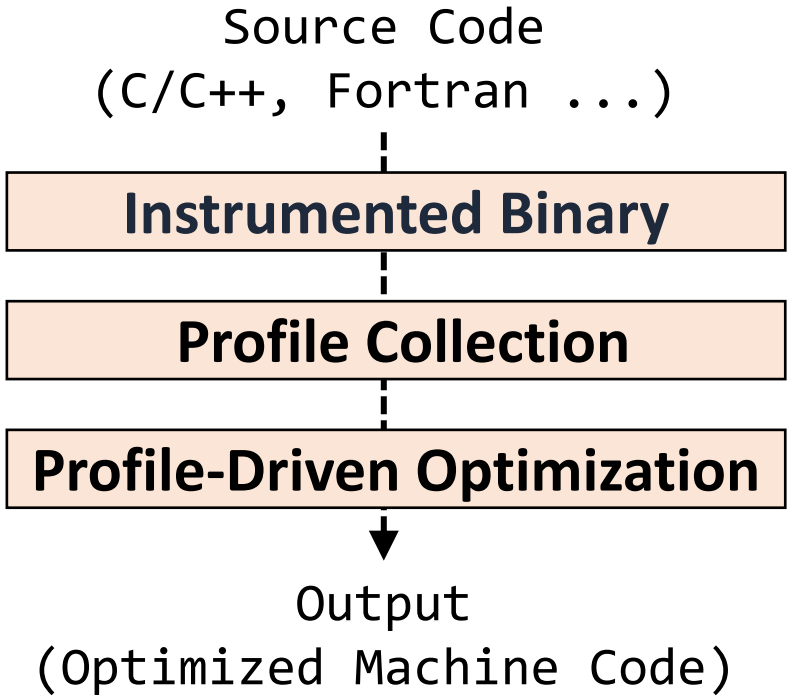
Expert-designed Generic Heuristics

- **Statically** estimate cost-benefit trade-offs of optimizations

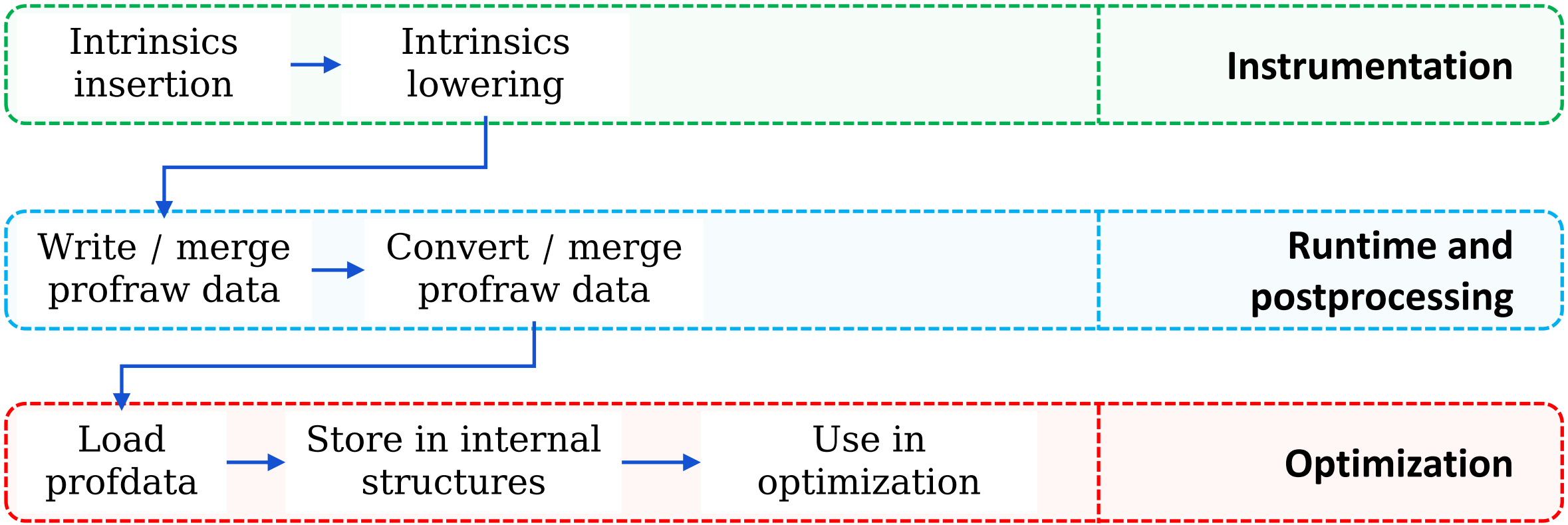
Require manual retuning for different architectures.

Unable to accurately capture program-specific runtime behavior

Profile-Guided Optimization



Implementation details of PGO in LLVM



PGO Instrumentation: Example of CallSite-Aware Profiling (2020 LLVM Developers' Meeting)

Why PGO Matters?

- ✓ Incorporates execution profiles from real runs
- ✓ Replaces static assumptions with feedback-driven decisions
- ✓ Enables more targeted and effective optimizations on specific programs and platforms

Deliver performance improvement of 5%-30% across a wide range of workloads[1]

Better Inlining Decisions

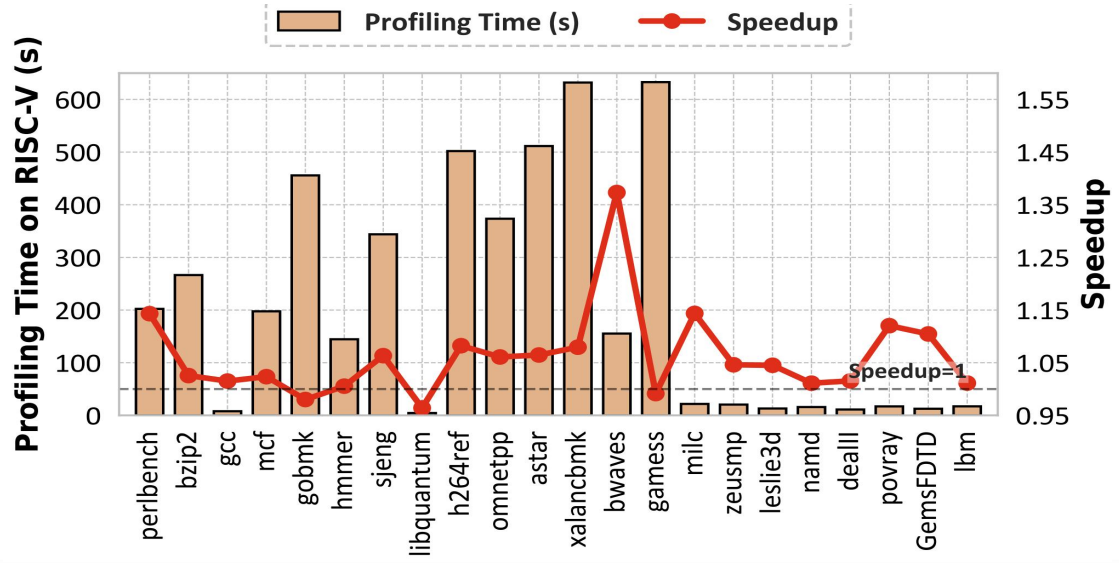
Code Layout Optimization

Loop Optimization

Register Allocation

...

Performance Benefits of PGO on RISC-V (SpacemiT Key Stone K1)



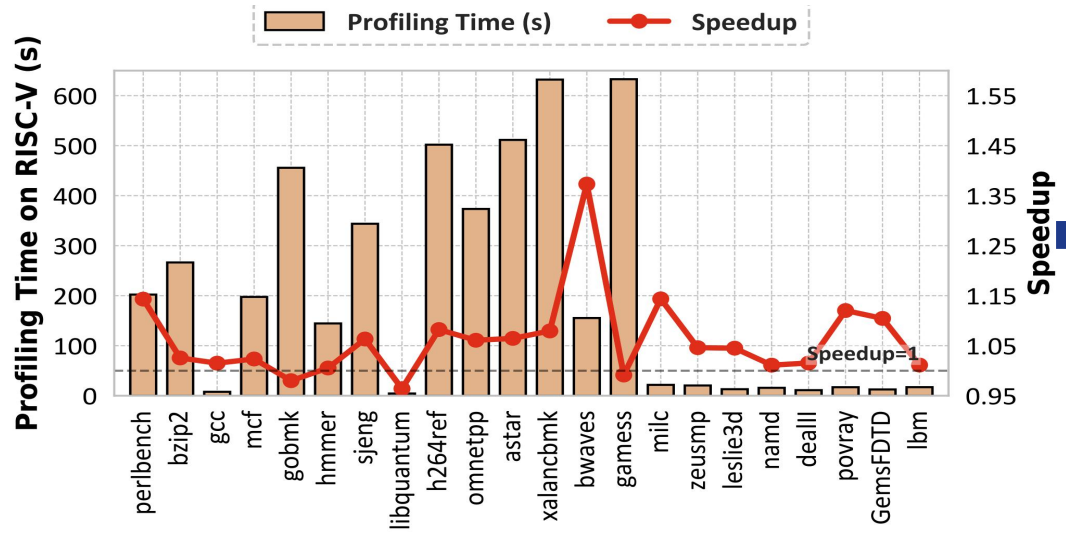
[1] Examining and Reducing the Influence of Sampling Errors on Feedback-Driven Optimizations (TACO'2016)

Limitations of Traditional PGO

PGO relies on ***architecture-specific*** profiles, necessitating separate profiling runs for each target, even for the same source code.

High Profiling Overhead

Collecting profiles on **resource-constrained devices** (e.g., IoT) or **emerging platforms with slow simulation** (e.g., RTL) is prohibitively expensive.



For SpacemiT Key Stone K1, profiling overhead typically **exceeds 100 seconds** per benchmark.

Key Observation: Profile Consistency Across Architectures

Identical Values

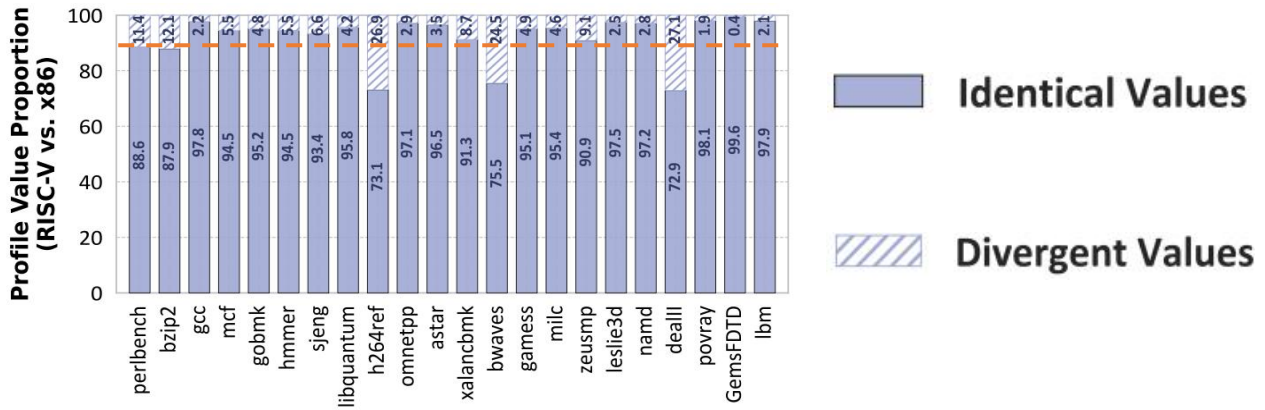
>90 %

of profile data points remain **identical** across different architectures

x86 (Intel Core i9-11900K)

RISC-V (SpacemiT Key Stone K1))

Comparison of profile values between RISC-V and x86



→ The "Profile Once" Strategy

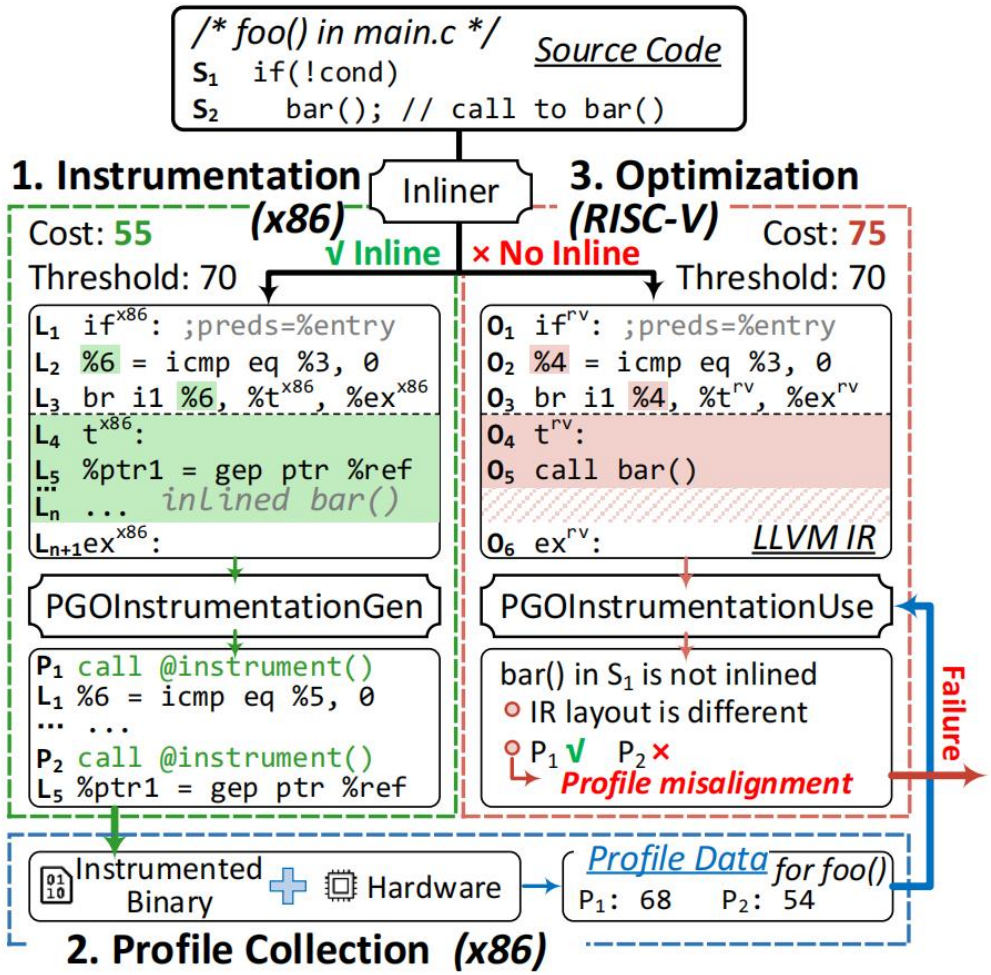
Since the vast majority of behavior is architecture-agnostic, we can **reuse** matched regions directly.

Strategy 1
Reuse Matched Regions

+

Strategy 2
Synthesize Differences

Why Cross-Architecture PGO Fails



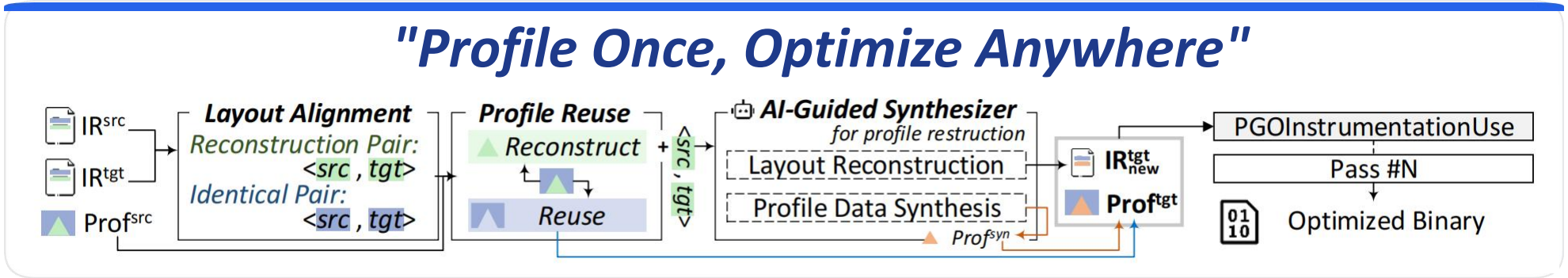
Architecture Dependency & IR Mismatch

Architecture-specific optimizations (like **inlining** on x86 vs. RISC-V) drastically alter the IR layout, making cross-architecture profile application prone to failure.

Inefficiency of Existing Solutions

Current LLVM adopt a rigid approach: they simply **discard** profile data whenever function layouts differ, resulting in low profile utilization and suboptimal performance.

APGO Overview



Treat profile transfer as a region-level alignment problem

01 Layout Alignment

Identify structurally identical regions.

02 Direct Reuse

Direct transfer for matched regions.

03 AI-Guided Synthesis

Reconstruct missing profile data.

Key Innovation: AI-Guided Synthesizer

- Layout Reconstruction: Transfer beneficial optimizations from profiling to target architecture
- Profile Data Synthesis: Generate missing profile values through architecture-aware mapping

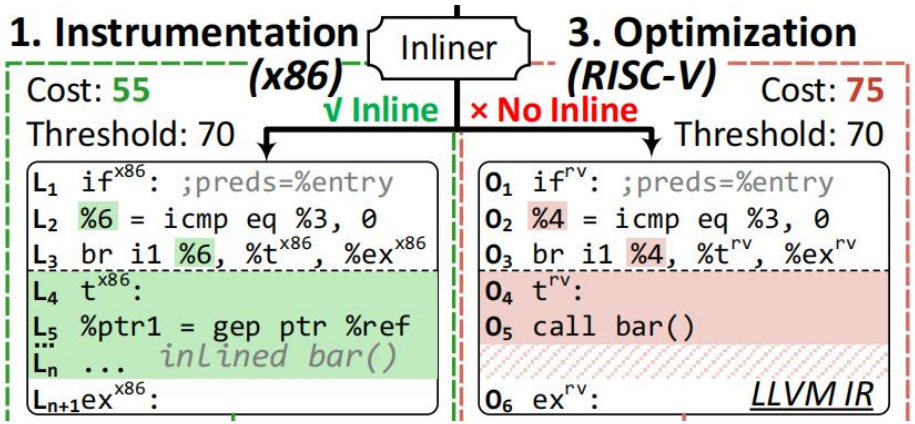
Core Techniques: Layout Alignment

Region Pair Construction

- Use Levenshtein edit distance for basic block alignment
- Aggregate aligned blocks into contiguous regions
- Create one-to-one region correspondences

Identical Pairs Recognition

- Hash control flow patterns & instruction sequences to ignore syntactic variations (e.g., variable naming)
- Partition into **identical vs. reconstruction pairs**



Example Output:

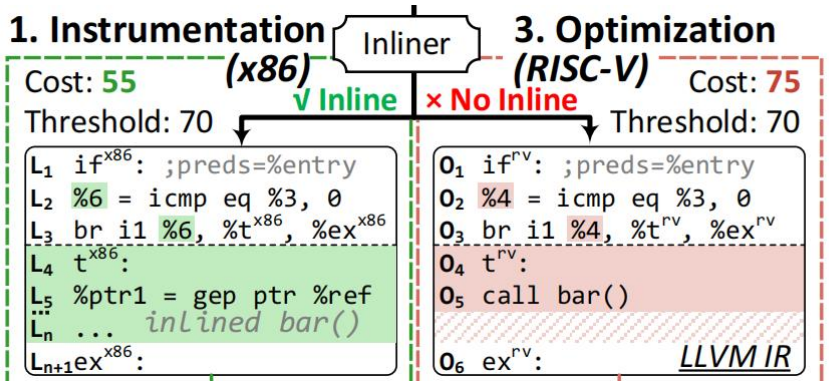
Identical Pairs: <{if^{x86}}, {if^{rv}}> → Direct profile reuse

Reconstruction Pairs: <{t^{x86}}, {t^{rv}}> → AI-guided synthesis needed

Core Techniques: AI-Guided Synthesizer

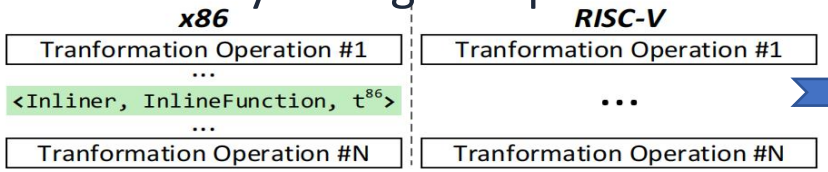
Layout Reconstruction

Reconstruction Pairs: <{t^{x86}}, {t^{rv}}>



Divergence Tracking

- Track optimization paths
- Identify divergent operations



We extend LLVMContext::diagnose mechanism in LLVM to capture detailed transformation records.

Benefit Validation: LLM evaluates performance impact

Prompt: Assess whether "inlining the bar function" preserves performance benefits for the following LLVM IR on RISC-V.
- LLVM IR: t^{rv}
- Architecture Characteristics:
  InstructionCost=..., InlineCallPenalty=..., CallerAllocaCost=...
- Execution Frequency: 54 / 68

Response: Inlining bar in t^{rv} is beneficial.

Transformation: Apply beneficial optimizations

Profile Data Synthesis

LLM generates architecture-aware transformation formulas.

Prompt: Derive mapping formulas to map instrumentation points between profiling and target architectures.
- Profiling: t^{x86} with instrumentation point P_2 placed before L_5
- Target: t^{rv}_{new} with instrumentation point...
- Divergent Transformation Operations: ...
- Divergent Architecture Characteristics: $MispredictPenalty^{86} = 14$, $MispredictPenalty^{rv} = 8$, ...

Response: $V_2^{rv} = V_2^{x86}$, where V_i^j denotes the profile value at instrumentation point i on architecture j.

$$V_2^{rv} = V_2^{x86}$$

Apply formulas to generate target profile values.

Experimental Setup

Platforms

- Profiling: Intel Core i9-11900K (8-core, 3.50GHz x86)
- Target 1: SpacemiT Key Stone K1 (8-core, 1.6GHz RISC-V)
- Target 2: T-Head Yitian 710 (128-core, 2.75GHz ARM)

Benchmarks

- 27 from SPEC CPU2006
- 5 from NPB (parallel)

Comparison Baselines

- Base: LLVM -O3 (no PGO)
- CrossPGO: Default LLVM
- NativePGO: Same-arch profile

Benchmark	Suite	Benchmark	Suite
perlbench	SPECint2006	zeusmp	SPECfp2006
bzip2	SPECint2006	leslie3d	SPECfp2006
gcc	SPECint2006	namd	SPECfp2006
mcf	SPECint2006	dealII	SPECfp2006
gobmk	SPECint2006	soplex	SPECfp2006
hmmer	SPECint2006	povray	SPECfp2006
sjeng	SPECint2006	GemsFDTD	SPECfp2006
libquantum	SPECint2006	lbm	SPECfp2006
omnetpp	SPECint2006	IS	NPB
astar	SPECint2006	EP	NPB
xalancbmk	SPECint2006	CG	NPB
bwaves	SPECfp2006	MG	NPB
gamess	SPECfp2006	FT	NPB
milc	SPECfp2006		

Configuration: LLVM 19.1.0 | -O3 -flto -ffast-math | Vectorization disabled for clarity

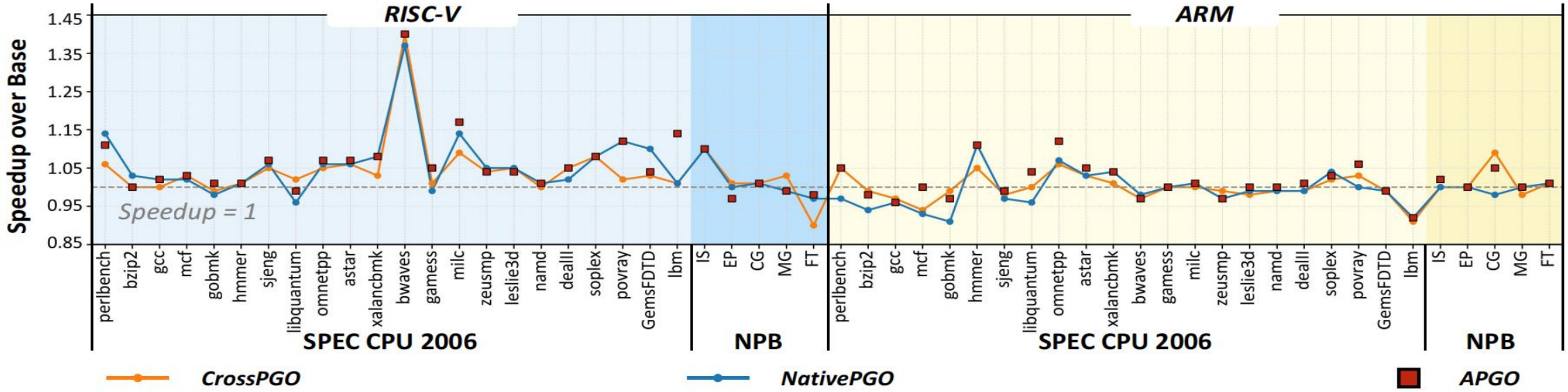
Performance Results: Single-Core

54 workloads (27 benchmarks × 2 architectures)

34/54
workloads achieve
comparable performance vs. NativePGO

14/54
workloads
outperform NativePGO

13%
peak improvement
over NativePGO

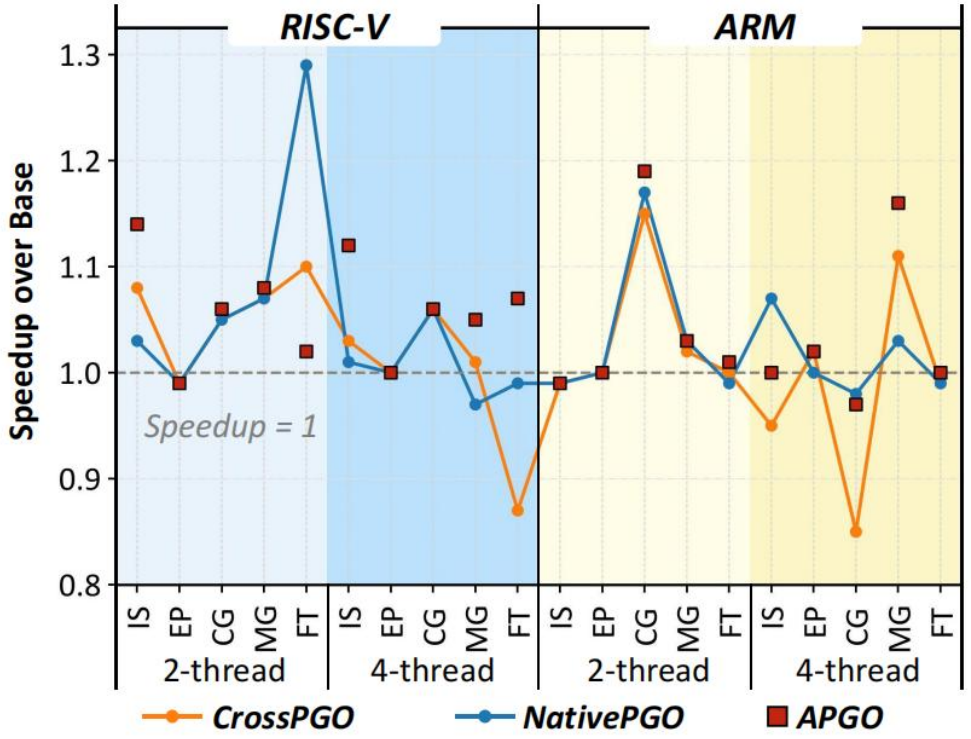


Performance Results: Multi-threaded

20 workloads from NPB benchmarks (2-thread & 4-thread)

11/20
workloads match
NativePGO performance

6/20
workloads exceed
NativePGO (up to 11.85%)



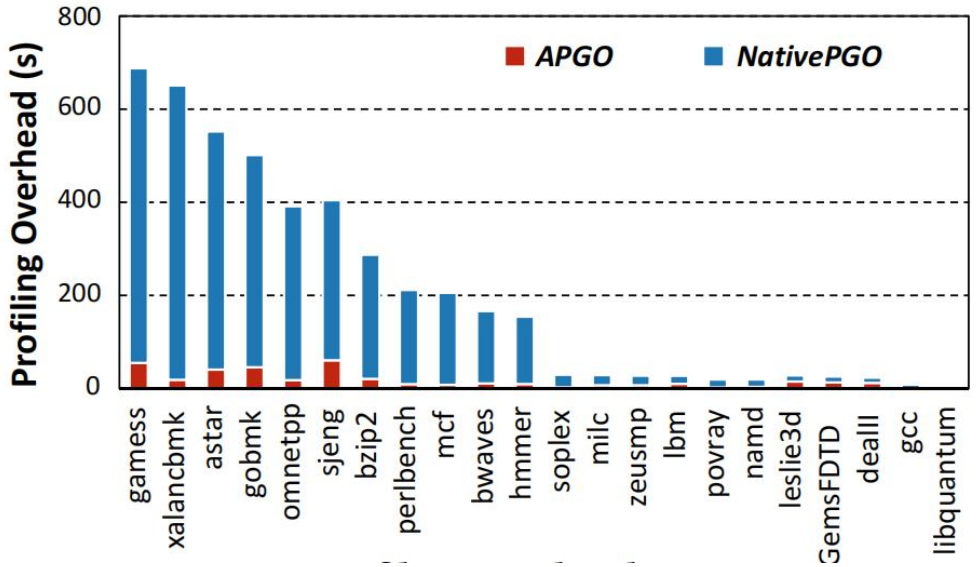
Profiling Overhead Analysis

APGO dramatically reduces profiling time by using high-performance x86.

32.78×

profiling time reduction

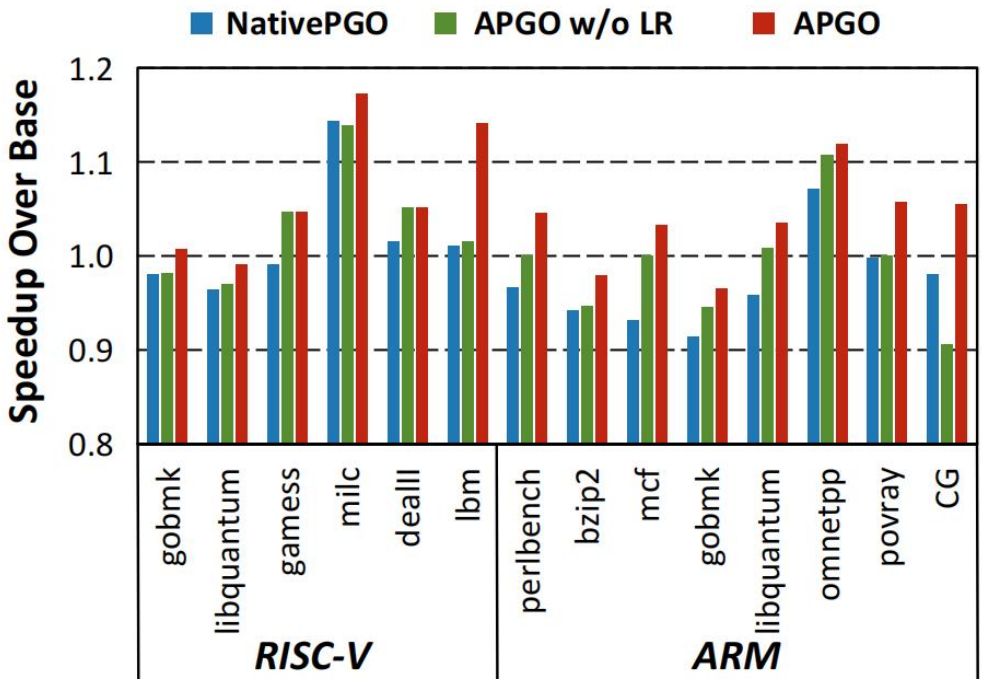
xalancbmk benchmark: 632.10s → 18.71s



Why This Matters:

- Resource-constrained embedded devices can leverage powerful profiling infrastructure
- Slow simulation platforms no longer bottleneck the optimization process
- Developers can profile once on development machines and deploy optimized code everywhere

Ablation Study: Component Analysis



Profile Data Synthesis

APGO w/o Layout Reconstruction achieves modest gains through LLM-based profile mapping alone

Layout Reconstruction

Majority of performance gains stem from transferring beneficial x86 optimizations to target architectures

Example: dealII on RISC-V

APGO w/o Layout Reconstruction: 1.05× speedup

Complete APGO: 1.12× speedup

Improvement: 1.12 / 1.05 = 1.067 (6.7% additional gain from layout reconstruction)

Current Limitations & Future Work

Vectorization Challenges

Current evaluation disables vectorization to simplify alignment. Future work needed to handle x86's fixed-width SIMD vs. ARM/RISC-V's variable-length vector extensions

Architecture Diversity

Evaluated on RISC-V and ARM. Extending to more diverse architectures (GPU, FPGA, specialized accelerators) requires further investigation

Future Directions

- Specialized vectorization profile transfer techniques
- Extension to heterogeneous computing (GPU, TPU, FPGA)

Conclusion

APGO enables ***"Profile Once, Optimize Anywhere"***

- ✓ Up to 32.78× profiling speedup
- ✓ Comparable performance to native PGO
- ✓ Architecture-aware optimization transfer
- ✓ AI-powered profile reconstruction

Questions?

APGO: Profile Once, Optimize Anywhere

qiulei21b@ict.ac.cn

LLVM-CGO 2026 | Sydney, Australia