PREPRINT · GW-2026-0005

Typed SSA NativeIR for Semantics-Preserving JIT and Static AOT in Gray

A portable compiler backend for Linux x86-64 and Apple Silicon AArch64 execution

Author: Annabella Proctor
ORCID: https://orcid.org/0009-0000-0799-1757
Affiliation: Grayworth
Publisher: Grayworth Technical Publications
Manuscript date: August 5, 2026
Version: 1.0
DOI: https://doi.org/10.5281/zenodo.21759781
ISBN-13: 9798191550893

Abstract

This paper presents Gray's typed NativeIR, a bounded control-flow representation used to compile a numeric subset of Gray bytecode to native code on both x86-64 and AArch64 architectures. The system introduces typed control-flow graphs, SSA values, Phi nodes, register classes, architecture-specific lowering, and a shared backend path for Live JIT, Static execution, and native executable artifacts. Native calls, structured exits, source identity, deterministic artifacts, cooperative cancellation, and refusal-safe VM fallback are retained as part of the compiler-runtime contract. The result establishes Gray's native execution path as a portable backend rather than a collection of benchmark-specific accelerators, while clearly identifying the current boundary of supported native code.

Keywords: Gray; NativeIR; SSA; JIT; AOT; x86-64; AArch64; compiler backend

1. Introduction

Gray's first native wins established that interpreter dispatch could be removed from an eligible hot region without abandoning source identity, structured exits, exact results, or cancellation. The next architectural question is broader: whether native execution is a portable compiler backend rather than a single specialized acceleration path. Typed NativeIR answers that question by placing supported bytecode into an explicit typed control-flow representation before architecture-specific emission.

The backend is designed for the way Gray is shipped. Live JIT, Static execution, and native executable artifacts share the same intermediate form and semantic boundary. Unsupported functions remain in VM execution. Supported functions carry enough type, register-class, control-flow, source-identity, and cancellation information to lower to native code while preserving the runtime contract expected by server programs.

This paper documents the backend as an engineering contribution rather than as an unrestricted language-speed claim. Its scope is the current supported native subset on Linux x86-64 and Apple Silicon AArch64. Its importance is that Gray now has a real portable compiler architecture, with refusal and fallback as first-class correctness behavior.

2. NativeIR design

NativeIR represents eligible Gray code as typed basic blocks linked by explicit control-flow edges. Values are in SSA form, with Phi nodes at merge points and register classes assigned before architecture-specific lowering. The representation gives the emitter a stable vocabulary for integer and floating-point values, branch targets, native calls, structured exits, and runtime safepoints.

The representation is intentionally bounded. It does not claim to cover arbitrary dynamic Gray programs. Instead, the compiler proves eligibility for a function or region, emits NativeIR only when the value and control contract is known, and refuses otherwise. Refusal keeps the VM path authoritative for code outside the native subset.

3. Execution modes

Live JIT and Static AOT use the same NativeIR semantics even though they acquire machine code differently. Live mode can lower and install code during execution. Static mode can embed or load deterministic native material as part of the deployment artifact. Native executable work uses the same control and value model so expansion of the backend benefits all execution policies.

The shared path matters because it prevents separate benchmark-only implementations from drifting away from production behavior. Native code must return through the Gray runtime ABI, preserve source-function identity for diagnostics, honor structured exits, and poll for cooperative cancellation at the declared boundaries.

4. Architecture support

The x86-64 and AArch64 emitters lower the same typed IR into platform-specific instructions and calling conventions. Register classes provide the bridge between IR-level values and machine resources. The paper treats platform parity as a qualification target: Linux and macOS require different binaries and emitter details, but the programmer-facing semantics should remain the same.

Apple Silicon support is especially important because Gray development and server qualification both depend on macOS and Linux workflows. AArch64 emission turns NativeIR from a Linux-only optimization into a cross-platform compiler foundation.

5. Boundaries and fallback

Fallback is not an error path hidden from the design; it is part of the contract. When a function contains unsupported operations, ambiguous value ownership, an unavailable native call boundary, or a control shape not yet admitted to the backend, Gray executes it through the VM. That preserves correctness while allowing the native subset to expand behind tests.

The current boundary should be read as a support matrix, not as a limitation of the architecture. Typed arithmetic, selected calls, structured exits, and cancellation-capable loops establish the core path. Future work can add richer dynamic values, collection operations, string behavior, and broader server handlers without redefining the NativeIR contract.

6. Artifact identity and reproducibility

Grayworth's performance records use an evidence-first publication rule. A headline value is admitted only after the runner has retained the command contract, executable identity, fixture identity, environment description, raw measurements, and a machine-readable completion marker. The report generator consumes verifier-accepted evidence rather than terminal excerpts or manually selected screenshots. This rule matters because a benchmark is not merely a timer around a program: it is a claim about which program ran, under what resource limits, against which request stream, and with what correctness checks. Preserving those facts makes later reanalysis possible even while Gray itself continues to change.

The HTTP experiments use an open-loop, constant-throughput load model through wrk2. In a closed-loop client, a slow response delays the next request and can make an overloaded service appear to have acceptable latency because the client stops offering work at the intended schedule. wrk2 records latency from the time a request should have been sent, exposing that coordinated-omission effect. Requested rate, achieved rate, delivery ratio, and corrected latency are therefore reported together. Throughput without latency can reward an ever-growing queue; latency without achieved delivery can describe a server that simply did not process the offered work.

Every HTTP route is preflighted for exact status, body, and content type before load begins. Warmup is separated from the accepted interval, and the server process is monitored throughout the interval. A point fails on socket errors, non-success status codes, unexpected process exit, changed fixture identity, or missing completion evidence. The one-hour matrix additionally samples resident memory and open descriptors after a settling window. Flat sampled values do not prove the absence of every leak, but they do show that the accepted workload did not produce monotonic resource growth at the recorded sampling resolution.

CPU placement is treated as part of the experiment rather than an incidental host detail. Server, load generator, controller, and database roles use declared CPU sets where the retained runner supports them. The same-host comparison constrains every server fixture to one CPU, 512 MiB of memory, and 256 processes inside one digest-pinned Ubuntu image; wrk2 runs outside that container on disjoint client CPUs. This isolates the application-facing runtime path while holding the operating-system image, cgroup contract, network path, and load generator constant.

A controlled-delivery point in the comparison is defined before interpretation as at least 95 percent of the requested rate with corrected p99 below 100 ms. It is a reporting heuristic for locating a useful throughput knee, not a universal service-level objective. The complete curve remains primary evidence because a single threshold can hide how abruptly queues form after saturation. The papers therefore distinguish the highest point satisfying that contract from the maximum achieved throughput and from the later one-hour operating point selected below the knee.

Fresh-process and interleaved sampling are used where process startup, dynamic compilation, or thermal drift could otherwise bias the result. Output is checked before a timing sample is accepted. Medians are reported for repeated compute diagnostics because they resist a single scheduling interruption, while minimum and maximum values show the observed spread. HTTP curves retain every rate point rather than reporting only the favorable region. Negative results, including the original blocking PostgreSQL tail latency and Gray's own collapse at 75,000 requested requests per second, remain in the record because they identify the architectural boundary that the next experiment must address.

The publication language separates observation from interpretation. An observed value is copied from retained output; an architectural explanation is supported by implementation structure, profiling, or an intervention; a projection is labeled as future work. Grayworth publication identifiers remain stable internal record keys, while each version 1.0 preprint also carries its reserved DOI for the archival record. A later artifact release may add source bundles without changing these canonical record URLs, DOI identities, or original measurements.

Reproducibility does not require disclosing unrelated commercial implementation details. It does require enough identity to tell whether two records describe the same executable and fixture. For that reason, the appendices retain cryptographic hashes, runtime modes, requested rates, duration, connection counts, thread counts, database versions, and named artifact directories. Public benchmark and report files can be released progressively toward Gray v1. Until then, the paper distinguishes currently downloadable manuscript and metadata files from repository artifacts named for later release. No unavailable artifact is described as independently archived or externally replicated.

Table 1. NativeIR support summary

Area	Status
Typed SSA control-flow graph	Implemented for the supported subset
Phi nodes	Supported at merge points
Register classes	Used for typed machine lowering
x86-64 lowering	Supported
AArch64 lowering	Supported
Live JIT	Shared NativeIR path
Static AOT	Shared NativeIR path
Fallback	VM remains authoritative outside the subset

7. Conclusion

Typed NativeIR moves Gray beyond isolated native accelerators. It gives the runtime a portable typed backend for supported code, connects JIT and AOT execution through one compiler representation, and keeps fallback explicit when a region is not safe to lower. That combination is the foundation needed for broader competitive compiler work.

The result is not a declaration that every Gray program is native or that every workload is faster than established languages. It is a defensible compiler architecture: typed IR, architecture-specific emitters, shared execution policy, deterministic artifacts, cancellation, source identity, structured exits, and refusal-safe VM execution.

8. References

1. Gil Tene. wrk2: a constant throughput, correct latency recording variant of wrk. https://github.com/giltene/wrk2
2. PostgreSQL Global Development Group. Asynchronous Command Processing in libpq. https://www.postgresql.org/docs/current/libpq-async.html
3. The Go Authors. Package net/http. https://pkg.go.dev/net/http
4. The Go Authors. Diagnostics. https://go.dev/doc/diagnostics
5. Node.js Project. Don't Block the Event Loop or the Worker Pool. https://nodejs.org/en/learn/asynchronous-work/dont-block-the-event-loop
6. PHP Documentation Group. FastCGI Process Manager configuration. https://www.php.net/manual/en/install.fpm.configuration.php
7. M. Anton Ertl and David Gregg. The Structure and Performance of Efficient Interpreters. https://jilp.org/vol5/v5paper12.pdf
8. LLVM Project. Garbage Collection Safepoints in LLVM. https://llvm.org/docs/Statepoints.html
9. Revised6 Report on the Algorithmic Language Scheme. https://r6rs.org/final/html/r6rs/r6rs.html

10. Grayworth. Gray comparison methodology and retained dataset. https://grayworth.com/compare/methodology
11. Grayworth. Gray performance research record. https://grayworth.com/research
12. Grayworth. Formal HTTP comparison data. https://grayworth.com/data/http-comparison-results.json

Appendices

Appendix A

Appendix A records publication identifiers. Grayworth publication ID: GW-2026-0005. DOI: 10.5281/zenodo.21759781. ISBN-13: 9798191550893.

Appendix B

Appendix B records catalog surfaces available at publication time: Open Library OL62420310M, StoryGraph 41a81f65-2816-401e-857e-43ca58b64ebb, and Hardcover 2840285.

Appendix C

Appendix C records the central non-claim. NativeIR is a bounded backend for supported Gray code. Unsupported code remains correct in VM execution and should not be counted as native coverage until the support matrix admits it.