An MLIR-based acceleration method for RISC-V ISA simulator generated by Sail

Mingzhu Yan
Engineer
PLCT Lab
Beijing, China
yanmingzhu@iscas.ac.cn

Shuo Huang
Engineer
PLCT Lab
huangshuo4@gmail.com

Yunxiang Luo
Test Director
PLCT Lab
Beijing, China
luoyunxiang@iscas.ac.cn

***Abstract*—Sail is a domain-specific language (DSL) for Instruction Set Architecture (ISA) that can automatically generate simulators. While RISC-V ISA models implemented with Sail are generally functional, the automatically generated simulators are far less efficient than industrial-grade ones. This paper innovatively proposes a Sail compilation backend based on MLIR, marking the first integration of the MLIR toolchain into Sail IR optimization. This backend leverages MLIR's multi-level abstraction representation and flexible infrastructure, providing excellent scalability and room for further optimization. This paper creates a new dialect for the Sail language, applies MLIR's rich set of existing compiler optimization passes, and attempts integration with LLVM JIT, opening up more possibilities for the development and utilization of RISC-V Sail models.**

***Index Terms*—RISC-V, Formal Verification, MLIR, Sail**

I. Introduction

Modern Instruction Set Architectures (ISAs) are becoming increasingly complex, posing new demands for their formal specification, verification, and simulation. DSLs like Sail[1] are powerful tools for this purpose, allowing engineers to precisely describe ISA semantics. Sail can automatically generate various artifacts, including executable simulators. Notably, Sail-riscv[2], a formal specification of the RISC-V architecture written in Sail, has been adopted by the RISC-V Foundation.

While automatically generating simulators from formal specifications offers significant advantages in correctness and consistency, these generated simulators often have a considerable performance gap compared to highly optimized, manually coded industrial simulators. This paper addresses this challenge by introducing a new Sail compilation backend based on the MLIR[3] framework. Key contributions include: being the first to utilize the MLIR toolchain for Sail optimization, the design and implementation of the Jib dialect, the development and application of specialized MLIR-based compiler optimization techniques, and integration with the LLVM JIT framework. By leveraging MLIR's flexible infrastructure, this paper provides a scalable path for developing domain-specific optimizations for Sail, aiming to significantly accelerate generated ISA simulators and narrow the performance gap with industrial-grade solutions.

II. Methods

To address the optimization limitations of automatically generated Sail simulators and fully leverage MLIR's advantages in the compiler domain, this paper designs and implements an MLIR-based Sail compilation backend. The core idea of this backend is to progressively transform Sail's high-level ISA descriptions into MLIR's multi-level intermediate representation, applying a series of transformations and optimizations in the process.

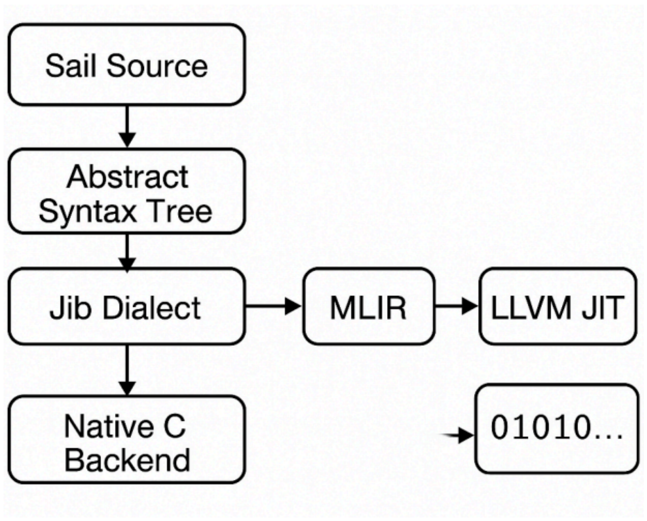

Fig 1: MLIR-based Sail compilation backend

A. Mapping Sail to an MLIR Dialect

this paper creates a new MLIR dialect for the Sail language. Most existing MLIR dialects are based on representations of traditional user programs; the purpose of creating a custom Sail dialect is to precisely represent the syntactic features of Sail used for ISA description within MLIR. This approach allows Sail's unique concepts, such as instruction definitions and register operations, to be directly converted into MLIR IR. This maximizes the preservation of Sail's original semantics, enabling the design of optimization processes specifically tailored for Sail, which is crucial for accurately simulating RISC-V architectural behavior.

B. *Designing and Reusing MLIR's Multi-level Abstraction*

One of MLIR's core advantages is its support for multi-level abstraction representation. After converting Sail IR into an MLIR dialect, the method proposed in this paper can directly reuse MLIR's rich ecosystem of existing compiler optimization Passes. These Passes cover a wide range of optimizations, from general ones (like constant folding and dead code elimination) to more complex control flow and data flow analysis.

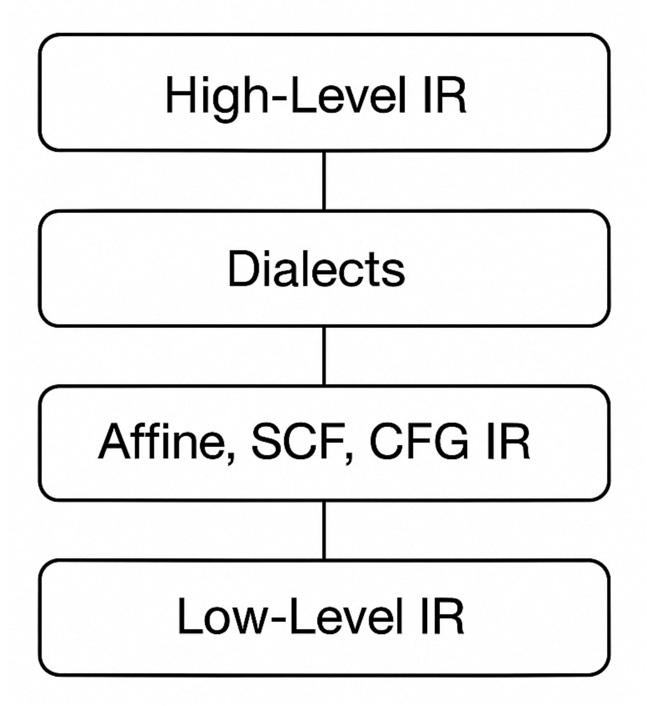

Fig 2: Multi level abstraction representation

In the proposed backend, Sail code is first translated into a high-level Sail dialect IR. Subsequently, through a series of carefully designed lowering Passes, this IR is progressively transformed into lower-level MLIR dialects, eventually converting to generic LLVM IR. This gradual transformation enables the application of targeted optimizations at different abstraction levels. For instance, at the Sail dialect layer, abstract operation optimizations closely related to RISC-V ISA semantics can be performed; at lower generic MLIR dialect or LLVM IR layers, broader general compiler optimizations can be applied. This layered optimization capability offers significant advantages for handling the complexity and diversity of the RISC-V architecture.

C. *Integration with LLVM JIT*

To further enhance the flexibility and dynamism of RISC-V simulators, this paper explores integration with LLVM JIT (Just-In-Time compilation) technology. When the MLIR IR is finally lowered to LLVM IR, the proposed MLIR-based backend can leverage LLVM JIT to dynamically compile and execute code. This integration allows the simulator to perform on-the-fly optimizations based on runtime behavior, such as identifying and optimizing hot code paths. The introduction of LLVM JIT opens up more possibilities for the development and utilization of RISC-V Sail models, making them more advantageous in scenarios requiring dynamic code generation and optimization, which is particularly important for exploring RISC-V software development and performance tuning.

III. Results and Conslusion

This paper successfully constructs an MLIR-based Sail compilation backend, marking the first integration of MLIR, a modern compiler infrastructure, into Sail's compilation pipeline. This work holds significant importance for the continuous development of RISC-V ISA models. The achievements of this work are reflected in two main aspects: Firstly, the proposed method explores the application potential of MLIR in RISC-V ISA modeling languages. It designs and implements multi-level lowering Passes from high-level Sail dialects to lower-level MLIR dialects and even LLVM IR, ensuring the correct transformation and representation of Sail semantics at different abstraction levels. It integrates MLIR's rich set of optimization Passes, demonstrating the tremendous potential of the MLIR framework in Sail code optimization. These optimizations are crucial for generating high-quality RISC-V simulators. Secondly, by designing and reusing MLIR's multi-level abstraction, this paper further leverages MLIR's rich infrastructure, opening up vast space for optimization and development for Sail simulators and the RISC-V ecosystem. It creates a new Sail dialect, enabling a precise mapping of Sail semantics to MLIR IR, laying the foundation for deep optimization of RISC-V Sail models by utilizing the MLIR infrastructure. This enables Sail to better serve the specification and simulation of the RISC-V architecture.

Table 1: Sail vs Sail-MLIR

Feature	**Sail**	**Sail-MLIR**
IR	Single/Limited	Multi Level
General Opts	Limited	Highly Reusable
Extensibility	Complex	Modular
Dynamic Opts	Static Only	JIT Potential

In summary, this paper creates a new compilation backend for the Sail language. By integrating Sail with the MLIR ecosystem, it reveals more possibilities for the development and utilization of RISC-V Sail models, providing a solid foundation for building more flexible, easier-to-optimize, and extensible RISC-V ISA models. This work is not only an innovative exploration in Sail compilation technology but also provides valuable practical experience for future compiler design and RISC-V architecture modeling, thereby accelerating the adoption and application of RISC-V technology.

References

[1] A. Armstrong *et al.*, “ISA semantics for ARMv8-a, RISC-v, and CHERI-MIPS,” *Proc. ACM Program. Lang.*, vol. 3, no. POPL, p. 31, Jan. 2019, doi: 10.1145/3290384.

[2] A. Armstrong, “sail.” [Online]. Available: https://github.com/rems-project/sail

[3] C. Lattner and others, “MLIR: Scaling Compiler Infrastructure for Domain Specific Computation,” *2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, pp. 2–14, 2021, doi: 10.1109/CGO51591.2021.9370308.