CIRE: LLVM Analysis for Floating-Point Rounding Error Affected by Precision and Optimizations

Cayden Lund
cayden.lund@utah.edu
University of Utah
Salt Lake City, Utah, USA

Tanmay Tirpankar
tanmaytirpankar@utah.edu
University of Utah
Salt Lake City, Utah, USA

Ganesh Gopalakrishnan
ganesh@cs.utah.edu
University of Utah
Salt Lake City, Utah, USA

Abstract

CIRE is a static analysis tool that addresses a critical challenge in numerical computing: understanding how compiler optimizations and precision settings affect floating-point rounding errors. Unlike traditional approaches that rely on expensive runtime testing across enormous input spaces, CIRE provides conservative but tight error bounds by analyzing LLVM IR code directly using symbolic automatic differentiation. The tool revolutionizes error estimation by building computational graphs from LLVM IR, applying reverse-mode automatic differentiation symbolically, and using global optimization to maximize error expressions over specified input intervals.

CCS Concepts

• **Software and its engineering** → **Compilers**; *Software verification and validation*; • **Hardware** → *Error detection and error correction.*

Keywords

floating-point analysis, compiler optimization, LLVM, automatic differentiation, numerical precision, static analysis

ACM Reference Format:
Cayden Lund, Tanmay Tirpankar, and Ganesh Gopalakrishnan. 2025. CIRE: LLVM Analysis for Floating-Point Rounding Error Affected by Precision and Optimizations. In *Proceedings of Super Computing (SC25).* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

Numerical programmers face a fundamental trade-off between performance and accuracy. Two primary "knobs" affect performance: compiler optimizations and numerical precision choices. However, these same adjustments unpredictably impact floating-point rounding errors. Traditional error estimation methods require shadow-value executions—running high-precision versions alongside optimized versions with identical inputs—which cannot scale to realistic input domains (e.g., a program with two FP32 inputs requires examining nearly 2^{64} input combinations).

SC25, St. Louis, MO

ACM ISBN 978-x-xxxx-xxxx-x/YYYY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

CIRE (developed by researchers at the University of Utah) is a static analysis tool that addresses this critical challenge in numerical computing by providing conservative but tight error bounds through direct analysis of LLVM IR code.

2 CIRE's Approach

CIRE revolutionizes error estimation through static analysis of LLVM code using symbolic automatic differentiation. The tool performs the following operations:

(1) **Builds computational graphs from LLVM IR** that capture operator dependencies and optimization effects from LLVM code
(2) **Applies reverse-mode automatic differentiation** symbolically, treating input variables as free variables to produce error expressions
(3) **Uses global optimization** (via the Ibex optimizer) to maximize error expressions over specified input intervals

This approach guarantees coverage of all points in given input intervals while running much faster than exhaustive testing methods.

3 Key Findings

3.1 Optimization Impact Varies

Contrary to common assumptions, the `-ffast-math` compiler flag doesn't consistently worsen numerical accuracy. CIRE's analysis reveals that optimization effects depend heavily on the specific program structure. Many benchmarks actually achieve better accuracy with aggressive optimizations due to operation reduction, while some others experience degradation.

3.2 Language Differences Matter

Significant variations exist between C++ (compiled with Clang) and Rust (compiled with Rustc) for identical mathematical operations. One notable difference is in fused-multiply-add (FMA) operations: Clang automatically generates FMA instructions, while Rustc requires explicit use of the `mul_add()` function. These compiler differences create measurable impacts on both performance and error bounds.

3.3 Mixed-Precision Complexity

The research reveals that mixed-precision programming creates complex trade-offs that don't scale predictably. Strategic allocation of single-precision versus double-precision to different subexpressions can dramatically affect both runtime and accuracy. For

example, in the Verhulst benchmark, switching single-precision allocation from one subexpression to another improved performance by 3x and error by 10x.

4 Evaluation Results

CIRE was evaluated on the FPBench benchmark suite, analyzing 75 out of 131 programs that produced compatible LLVM IR. The tool demonstrated:

- **Scalability**: Can analyze expressions with over 10^5 arithmetic operations
- **Speed**: Significantly faster than traditional shadow-value approaches
- **Accuracy**: Error estimates are conservative but empirically within a factor of 2 of true error

Performance analysis across different compiler configurations (Clang -O1, -O3, -O3 -ffast-math, Rustc opt-level=1, opt-level=3) revealed non-uniform effects where optimization benefits vary significantly between programs.

5 Technical Innovation

CIRE extends previous work (FPTaylor, SATIRE) by being the first tool to analyze LLVM representations directly, enabling support for any programming language that targets LLVM. The tool handles:

- **Mixed-precision computations** through LLVM's `fptrunc` and `fpext` instructions
- **Optimization effects** by analyzing transformed LLVM code
- **Multiple precision levels** with different unit-roundoff values

6 Limitations and Future Work

CIRE currently focuses on loop-free programs, as loops present fixed-point computation challenges in static analysis. However, many programs are naturally loop-free (like arithmetic libraries), and finite loop unrolling can provide insights into error trends.

Other limitations include handling of subnormal numbers and potential singularities in optimizer queries, which the authors plan to address in future versions.

7 Impact and Applications

CIRE enables simultaneous optimization of both performance and accuracy, providing valuable feedback for:

- **Compiler developers** seeking to understand optimization effects on numerical accuracy
- **Numerical programmers** making informed precision and optimization choices
- **Mixed-precision programming** where strategic precision allocation can optimize the performance-accuracy trade-off

The tool opens possibilities for integration with error-improvement tools like HERBIE, allowing programmers to explore more aggressive optimizations while maintaining accuracy bounds.

8 Conclusion

CIRE represents a significant advance in floating-point error analysis, providing the first practical tool for understanding how compiler optimizations and precision choices affect numerical accuracy in real-world programs. By analyzing LLVM IR directly, it offers unprecedented insights into the complex interplay between performance optimization and numerical precision across different programming languages and compiler configurations.