Project Proposal: Reproducing and Extending Catamaran

A Feasibility Study of Parallelizing Memory Safety Checks on Modern LLVM

Dongwook Kim (20201034)
UNIST
South Korea

Woohyeok Kang (20211004)
UNIST
South Korea

Abstract

Dynamic memory safety tools like SoftBoundCETS and MoveC are critical for C/C++ development but impose significant runtime overhead (46–224%) due to heavy metadata operations. The 2023 ISSTA paper, "Catamaran," proposes a novel approach to mitigate this by identifying and parallelizing these memory-check meta-operations using thread-level parallelism (TLP). This project aims to reproduce the core performance claims for these metadata-heavy tools and explore their feasibility on modern LLVM infrastructures. Addressing this limitation directly fulfills the "reproduction and improvement" requirement by mitigating the dependency on an older LLVM foundation.

CCS Concepts

• **Software and its engineering** → **Source code generation**; • **Computer systems organization** → *Multicore architectures*; • **General and reference** → *Compiler statistics.*

Keywords

Parallel Computing, Compilers, LLVM, Memory Safety, Reproducibility

1 Introduction

The runtime overhead of dynamic memory safety checks is a critical problem, particularly for metadata-heavy approaches. Tools like SoftBoundCETS [2] and MoveC [1] maintain complex metadata to track pointer bounds, leading to significant performance penalties. Simpler shadow-memory tools like AddressSanitizer (ASan) [3] are faster but cover fewer error types.

The Catamaran paper [4] addresses this overhead directly. Its key insight is that metadata operations (such as `_lookup` and `_update`) are often independent of the main program's data flow and can therefore be executed concurrently. Using a custom LLVM pass to analyze dependencies and a runtime system to schedule checks on separate threads, Catamaran achieves substantial speedups (46–224%). This project explores similar parallelization techniques within the context of modern LLVM, aligning with CSE412's focus on compiler-level parallelism and software optimization.

2 Core Methodology

Catamaran enforces comprehensive memory safety through a parallelized metadata management framework that operates in conjunction with compiler-inserted instrumentation. The core idea is to decouple metadata operations from the main program execution and execute them concurrently in a safe, dependency-aware manner.

2.1 Instrumentation and Metadata Operations

During compilation, Catamaran performs a def-use analysis over all pointer variables to identify *definition*, *propagation*, and *use* points. For each pointer definition or propagation, the compiler inserts a metadata operation `_update(key, md)` to maintain the spatial and temporal information of the pointer's target object. Before each dereference, Catamaran inserts a `_check(ptr, md)` operation to ensure that the access is both spatially in-bounds and temporally valid. Metadata lookup operations (`_lookup(key)`) retrieve the associated object metadata for further use. These instrumentation primitives form the basis for Catamaran's runtime metadata enforcement.

2.2 Parallel Task Graph Construction

Catamaran constructs a directed acyclic graph (DAG) of all metadata operations within each function, known as the *intra-point parallel task graph*. Each node in the graph represents a metadata operation, and edges represent dependence relations derived from def-use chains and aliasing information. Dependency rules (R1–R6) classify read-after-write (RAW), write-after-read (WAR), and write-after-write (WAW) hazards between metadata operations that access the same key or depend on pointer-defining instructions. Algorithm 2 then determines the maximal safe execution regions of each task by computing its dependency frontier and the farthest common dominator of all dependent successors.

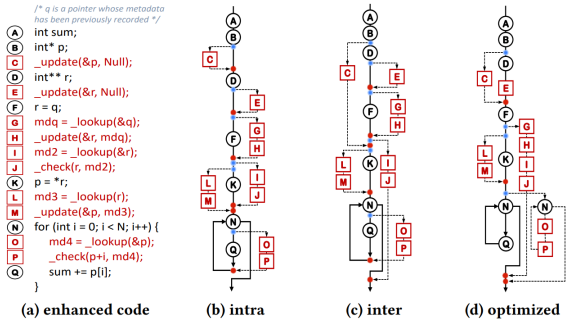

Figure 1: Intra-point parallel task graph construction in Catamaran. Each node represents a metadata operation (`_lookup`, `_update`, or `_check`), while edges encode dependency relations derived from def-use analysis and Rules R1–R6. Independent subgraphs can be scheduled concurrently on parallel check units.

2.3 Parallel Execution and Runtime Support

At runtime, Catamaran dispatches independent metadata operations to parallel execution units, called *metadata check units* (MCUs).

These units perform bounds and temporal checks concurrently with the main program execution, thus overlapping safety enforcement with useful computation. The runtime system ensures correctness by enforcing synchronization barriers at the boundaries of dependency regions computed during compilation.

2.4 Comparison with Sequential Instrumentation

Traditional metadata-based sanitizers such as MoveC and SoftBoundCETS execute each check operation (`_lookup`, `_update`, `_check`) sequentially along the program's critical path. This inline execution model introduces stalls, as the main thread must wait for each metadata operation to complete before proceeding. As a result, the runtime overhead scales roughly linearly with the number of checks inserted by the compiler.

Catamaran introduces a fundamental shift in this model by offloading independent metadata operations to parallel worker threads. This enables the main application thread to continue execution while safety checks are processed asynchronously. Figure 2 contrasts these two execution paradigms.

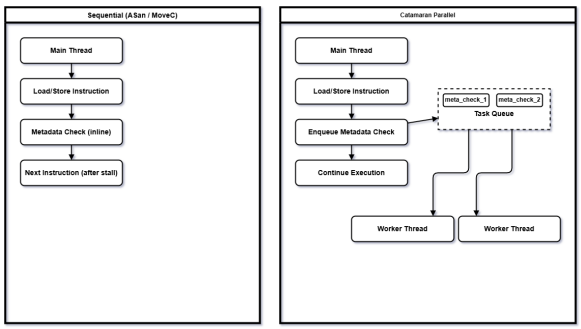

Figure 2: Comparison between sequential metadata execution (left) and Catamaran's parallel offloading model (right). The latter overlaps memory safety enforcement with normal program execution, reducing perceived latency.

2.5 Summary

By converting sequential instrumentation into a parallel task execution model, Catamaran achieves low-overhead enforcement of both spatial and temporal memory safety. Its methodology preserves the precise semantics of MoveC while exploiting inter- and intra-function parallelism through compile-time analysis and runtime scheduling.

3 Project Objectives

This project is organized into two phases to address both reproduction and improvement.

3.1 Objective 1: Reproduction

We aim to reproduce Catamaran's performance results, focusing on metadata-intensive tools. Using the publicly available Zenodo artifact, we will:

(1) Build the older LLVM 3.4 toolchain required for MoveC or SoftBoundCETS.
(2) Run available benchmarks (or MiBench as a substitute) to measure runtime overhead reduction.
(3) Verify that the parallelized version maintains equivalent bug-detection accuracy.

3.2 Objective 2: Extension (Improvement)

Catamaran's dependency on a decade-old LLVM toolchain limits its applicability. Our improvement focuses on adapting its concept to modern LLVM (version 18).

We will conduct a feasibility study and develop a simplified proof-of-concept LLVM pass that:

(1) Identifies independent function calls simulating metadata lookups.
(2) Rewrites them to spawn tasks using lightweight concurrency (e.g., `std::async` or a thread pool).

This approach evaluates the practicality of applying Catamaran's ideas within the constraints of current compiler infrastructures.

4 Risk Analysis and Mitigation

- **Risk:** Building the older LLVM 3.4 environment may be difficult due to deprecated dependencies, potentially delaying Objective 1.
- **Mitigation:** We will time-box Objective 1 (2–3 weeks). If the build remains infeasible, we will shift focus to Objective 2. This ensures a meaningful outcome (a modern LLVM 18 proof-of-concept) even if full reproduction is not possible.

5 Proposed Timeline

Table 1: Proposed Project Timeline.

Weeks	Task
1	Literature review of Catamaran, MoveC, and ASan. Set up development environments (LLVM 3.4 and LLVM 18).
2	Attempt reproduction using the older LLVM toolchain. Run benchmarks and verify functionality.
3	Begin modern LLVM 18 proof-of-concept implementation. Develop simplified parallel-extraction pass.
4	Evaluate and compare results. Prepare final report and presentation.

6 Expected Outcome

We expect the results to provide insights into how parallel sanitizer execution scales with modern multicore hardware. Although our proof-of-concept may not match the full performance of Catamaran, it will clarify which parts of its design remain effective and which require rethinking in current LLVM infrastructures.

7 Conclusion

This project will deepen understanding of compiler-level parallelization and reproducibility challenges. By reproducing Catamaran's ideas and extending them to modern LLVM, we aim to explore how

legacy research can evolve with today's toolchains while reducing the runtime overhead of memory safety enforcement.

References

[1] Peng Chen, Jie Zhang, Feng Qin, and Chengnian Wu. 2019. Detecting Use-after-Free Bugs via Pointer Mutation. In *Proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. ACM, 791–803.

[2] Santosh Nagarakatte, Jianzhou Zhao, Milo M. K. Martin, and Steve Zdancewic. 2015. Everything you want to know about pointer-based checking. *ACM Transactions on Architecture and Code Optimization (TACO)* 11, 1 (2015), 1–34.

[3] Konstantin Serebryany, Derek Bruening, Alexander Potapenko, and Dmitry Vyukov. 2012. AddressSanitizer: A fast address sanity checker. In *Proceedings of the 2012 USENIX Annual Technical Conference (USENIX ATC)*. 309–318.

[4] Yiyu Zhang, Hyeonsu Kim, Donghyeon Kim, and Jaehyuk Kim. 2023. Catamaran: Low-Overhead Memory Safety Enforcement via Parallel Acceleration. In *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*. ACM, 418–429.