Ryan Eberhardt

ryan@reberhardt.com https://reberhardt.com

Skills

TypeScript, C, C++, Python, Java, Rust, Cuda, OpenMP, React [Native], AngularJS, Sass, MySQL, PostgreSQL

Education

STANFORD UNIVERSITY **BS JUNE 2019, MS JUNE 2021**
Received BS in Computer Science (Computer Systems track). GPA: 4.08/4.0
Received MS in Computer Science (Computer and Network Security track). GPA: 4.0/4.0

Selected Experience

SITE RELIABILITY ENGINEER, CODA **SEPTEMBER 2021 - PRESENT**

- Responded to production incidents, conducted post-mortem review, instituted processes to avoid bugs. As an example, after we had two site outages caused by dynamic configuration changes, I instituted a variety of improvements to ensure it never happens again, and we haven't had a similar outage in the year since.
- Designed and implemented a platform to perform a systematic risk assessment of DDoS vulnerability based on cache performance, rate limits, and database query cost. Identified subtle vulnerabilities that were missed during manual audit and implemented appropriate protections.
- Improved production observability: Implemented tooling for profiling production nodes and attaching a live debugger. Implemented application tracing using OpenTelemetry. Improved log collection and analysis tooling.
- Currently leading initiative to improve our integration testing: Surveying engineers to identify and remedy pain points in the testing process (people won't write tests if it's painful), improving test execution speed, adding debugging facilities for tests failed in CI, eliminating flakiness in tests.

LECTURER, STANFORD UNIVERSITY **JUNE 2018 - SEPTEMBER 2021**

- Designed and taught *CS 110L: Safety in Systems Programming* from scratch, focused on practical approaches for writing better software. Course surveys static and dynamic analysis tools for C/C++ programs, and teaches Rust as a vehicle for exploring language constructs that improve memory safety, thread safety, and correctness.
- Wrote seven programming assignments from scratch, including implementing a mini GDB, and another where students compete to implement the best load balancer (I built accompanying performance-testing infrastructure).
- Course website: reberhardt.com/cs110l; Blog post: reberhardt.com/110l-blog
- Taught *CS 110: Principles of Computer Systems*, focused on filesystems, concurrency, and networking. Received 4.7/5 rating on student evaluations (highest rating for this class since 2015). Website: reberhardt.com/cs110
- Developed extensive tooling to help students visualize concurrency and kernel data structures (e.g. see Cplayground.com below) and wrote many autograders and code review tools.

RESEARCH AND DEVELOPMENT INTERN, TRAIL OF BITS **JUNE - DECEMBER 2020**

- Developed a prototype GPU-based fuzzer for embedded software that is able to achieve roughly 8x executions per second per dollar compared to libFuzzer on Google Compute Engine. Blog post: reberhardt.com/fuzzer-blog
- Isolated two bugs in clang/LLVM and one in Nvidia's ptxas assembler, worked with upstream vendors to fix these.

COMPUTATIONAL SCIENCE RESEARCH INTERN, SANDIA NATIONAL LABORATORIES **MAY - AUGUST 2015**

- Developed high-performance sparse linear algebra kernels for CPUs, GPUs, and Xeon Phi accelerators, contributing to the Kokkos parallel computing project. Designed a block sparse matrix-vector kernel outperforming the implementations in Intel MKL by 3x and Nvidia cuSPARSE by 4x. Research published in IPDPSW '16.

COFOUNDER & LEAD BACKEND DEVELOPER, NAVIER INC. **2013-2015**

- Created a platform enabling web developers to interact with consumers to improve prototypes of apps. Designed distributed, secure infrastructure with Xen and Docker to host users' applications.

Selected Personal Projects

CPLAYGROUND.COM **DECEMBER 2018 - PRESENT**
An online sandbox that allows users to quickly test and visualize C/C++ systems code, built with NodeJS and React. The CPlayground debugger allows for individual process and thread control, enabling users to more easily step through concurrent code and understand race conditions. Visualization tools illustrate the use of thread synchronization primitives in a user's program. Running programs are executed in Docker containers, and stdin/stdout are streamed to the client over a websocket. The debugger is implemented using GDB and a custom kernel module.
Blog posts: reberhardt.com/cplayground-diagrams (high level) and reberhardt.com/cplayground-kernel (kernel)