Comparing the Performance of WebAssembly Source Languages and Compilers on WebAssembly Servers

Problem Statement

WebAssembly (Wasm) is a low-level bytecode format designed to execute code efficiently across different platforms, especially within web browsers. It provides a runtime environment that allows applications written in various programming languages to achieve near-native performance when running on the web. As WebAssembly adoption grows, understanding its performance characteristics across different programming languages becomes increasingly important. While it is widely regarded for its portability and efficiency, how different programming languages translate into Wasm and how their performance compares in terms of execution speed and loading times is still an open question. Our project explores this aspect by investigating the potential discrepancies in performance after conversion to WebAssembly.

Project Plan

This study focuses on evaluating WebAssembly's performance when generated from multiple source languages, specifically C, C++, C#, and Java. We will implement computationally intensive tasks such as matrix multiplication, recursion, and file operations in these languages, then compile them into WebAssembly. By analyzing execution and loading times in both native and WebAssembly environments, we aim to determine how well WebAssembly preserves performance across different language implementations. Additionally, we will investigate whether some languages have inherent advantages when compiled to WebAssembly. The goal is to gain insights into WebAssembly's capabilities and assess its suitability for high-performance applications across different programming ecosystems.

Key Steps in Our Plan

- ☐ Gain a solid understanding of WebAssembly, its architecture, and how it executes code.
- ☐ Write benchmark programs in C, C++, C#, and Java for selected computational tasks.
- ☐ Convert these programs into WebAssembly using appropriate toolchains and compilers.
- ☐ Measure execution speed, loading time, and memory usage in both native and Wasm environments.
- ☐ Utilize profiling tools such as Chrome DevTools, WasmTime, and other performance analyzers.
- ☐ Compare performance results to identify trends and variations between different source languages.

- [ ] Document findings and compile a detailed report on WebAssembly’s efficiency across multiple programming languages.