Add WebAssembly support in clang-repl

Lev E. Chechulin
l.e.chechulin@gmail.com

Motivation

WebAssembly (Wasm) is a binary instruction format designed as a portable target for compilation and enabling deployment on the web for client and server applications. With increasing interest in WebAssembly, there is a growing need for developer tools that can leverage its capabilities.

Clang-repl is a part of the LLVM compiler infrastructure project that allows developers to incrementally compile and execute code in a Read-Eval-Print Loop (REPL) environment. Xeus-clang-repl is a C++ kernel for Jupyter based on clang-repl, enabling a more interactive and user-friendly experience when working with C++ code in Jupyter notebooks. Currently, clang-repl and xeus-clang-repl do not support WebAssembly generation and execution. By adding this feature, we aim to provide several benefits:

> Broader platform compatibility: WASM is designed as a low-level virtual machine that runs code at near-native speed, providing a compact binary format that is fast to decode and execute. It is supported by all major web browsers, enabling developers to run their C++ code on a wide range of platforms without requiring platform-specific compilation. With Clang-repl and xeus-clang-repl supporting WebAssembly, users can take advantage of this cross-platform compatibility for their C++ code, broadening the possible use cases and target environments.
>
> WebAssembly's design includes strong sandboxing, which isolates the executed code from the host system, providing an additional layer of security. Integrating WebAssembly support into Clang-repl and xeus-clang-repl allows users to run untrusted or experimental C++ code with less risk, since the code will be executed within a sandboxed environment.

Possible approach

Following steps are needed to enable generating WASM in clang-repl: modify clang interpreter to generate WebAssembly, add WebAssembly support to xeus-clang-repl, enable code generation in browser. In more detail:

1. Change `Interpreter.h` and `Interpreter.cpp` to add function that will be responsible for WASM generation. This `EmitWebAssembly` function will parse code and run optimization and code generation passes to return the WASM binary.

2. Add a new command-line option or a configuration setting for xeus-clang-repl that enables WASM generation mode.

3. When the WASM generation mode is enabled, modify the input handling code in xeus-clang-repl to treat all input code as code that should be compiled to WASM. For each input code block, pass it to the `EmitWebAssembly` function, passing the code (as an `llvm::Module` instance) as an argument.

4. Enable WASM execution in Jupyter. I suggest the following approach: xeus-clang-repl should return the WebAssembly binary wrapped in a JavaScript code snippet that will handle the instantiation and execution of the binary.

5. Enhance error handling, update documentation, and perform testing.

Timeline

- May 4 - 28
  Community Bonding Phase.

- May 29 - June 11
  Implement WASM support in clang-repl.

- June 12 - 25
  Add WASM support to xeus-clang-repl.

- June 26 - July 2
  Testing, 'spare week' in case issues on previous stages will be encountered.

- July 3 - July 14
  Midterm evaluation.

- July 15 - Aug 6
  Submitting patches, review process.

- until November 6
  Troubleshooting and resolving issues, improving code quality. Integration and adoption: focus on enabling the WebAssembly generation feature to work effectively within a browser context. Testing and validation, documentation. Writing a blog post.
  Submitting the project.

About me

My name is Lev Chechulin, I have background in Mathematics (Topology and Computational PDEs) and some software engineering experience. By joining Google Summer of Code and working on the project, I aim to develop skills in compiler technologies, contribute to an influential open-source project, and become an active member of the LLVM community.