TISE Research Project Proposal
Running WebAssembly Compilers on the Web

VJS Pranavasri (vjs.pranavasri@research.iiit.ac.in)

September 19, 2022

Area of Interest

My area of interest is broadly under the umbrella of compilers. There are multiple unsolved problems and optimisations to be done across multiple version of compilers. However for this research I am aiming to look at a particular subset of them, compilers that compile to web assembly.

Problem

The problem I am trying to look at is being able to run codes natively on browser. WebAssembly is meant to be solving this problem, but it requires a compilation done in advance before actually hosting it.
I plan to research and find out ways in which a webassembly compiler will be able to compile High Level language codes to webassembly on the browser itself. This would allow for anyone to run(compile/interpret) any language natively on browser without having to have it compiled in advance. As part of the research I'll be looking at different possible ways to implement the compiler.

Importance

Multiple online coding and education platforms like Virtual Labs, would need an exclusive backend server running if they intend on having code compilation. WebAssembly has brought in many new innovations, and if we could utilize it's power to allow the compilation on browser itself it would save multiple such organisations a huge cost of having a backend server also at the same time, sideloading all the work to a client's browser.

Current State of the Art

The current state of the art when it comes to webassembly compilers are compilers like pyiodie, emscrippten and other such compilers[5] which compile high level languages to wasm(webassembly binary format). This can be further be loaded and called directly on javascript through the webassembly run time which is shipped with all major browsers (Chrome, Firefox, Safari etc.)
Another similar implementation is WasmFiddle[4] which is a backend service that compiles C/C++ to wasm and allows for running on javascript. Both of these mentioned solutions haave the following problems

- The high level language program (for ex C/C++) needs to be pre compiled before actually hosting on the web.

- There is a backend that is doing the converting and sending the wasm result back.

What can help me in the process are the research done in perfomance, memory and security of WASM, as those are important aspects to consider when moving a complete compiler onto a browser whose memory is very different form a traditional filesystem. The paper[3] talks about performance comparisions that can help decide on trade off's. Also the paper[2] does a comprehesive study on memory safety of web assembly. These would be good points to start looking at.

Proposed Solution

There are multiple ways to approach the solution. As part of the research I would want to compare these solutions and provide data which could be used to decide the feasibility and how best it could be implemented. Alongside also possibilly give a design idea.
For the scope of the research we'll be talking in terms of implementing a web assembly compiler for C/C++, that will allow to compile C/C++ code to WASM on the browser.
When we think of running a compiler in the browser the most obvious thought would be somehow to run gcc on browser that can help us compile. There are 3 ways of possibile ways:

1. Running a complete linux environment on JS[1]. This was already implemented, and this gives us a complete linux environment on browser, but it comes with a downside of network bandwith, it needs to download all packages before it can actually work.

2. As LLVM/Clang are self compilable, we can use this property and compile clang to wasm. After which we will be having a wasm program that takes C code as input and goves binary as output, this should open us upto using this binary on JS.

3. This final approach is one of the most promising approach, is JupyterLite[6]. It is an implementaion of JupyterNotebook in WASM[7] that allows complete running of notebook on Browser. The best part of this solution is the power of JupyterNotebook to run any language kernel. So this alone can allow almost every language to be compiled and run as long as a proper kernel for that language is written for JupyterLite.

I wish to compare these solutions memory wise, complexity wise performance wise or come up with a design for a fully working implementation of one of these approaches,

How is it different

All the existing solutions discussed only talk about being able to run existing modules/app in browser by compiling them to webassembly. The idea I have is to completely port the whole process to the browser allowing all the new code taken as input from the web page environment can be compiled live on the browser runtime on the client side without any backend dependency.

Timeline

1. Week-1 **(19/08-24/09)**: Do an indepth literature study, get a concrete understanding of exisitng solutions and come up with Research Questions

2. Week-2 **(25/09-02/10)**: Formally define my problem statement for research and come up with a rough draft of the solution or approach.

3. Week-3 **(03/10-09/10)**: Have a rough draft of the complete solution comparisions/design ready.

4. Week-4 **(10/10-16/10)**: Have a first draft of the paper ready for review.

5. Week-5 **(17/10-23/10)**: Take all inputs and discuss how to improve and start working on the final draft of the paper.

This would be the ideal timeline. I will try to incrementally keep updating the timeline and the goals as and when there are developments with the research.

References

[1] Fabrice Bellard. JSLinux. https://bellard.org/jslinux/.

[2] Craig Disselkoen, John Renner, Conrad Watt, Tal Garfinkel, Amit Levy, and Deian Stefan. Position paper: Progressive memory safety for webassembly. In *Proceedings of the 8th International Workshop on Hardware and Architectural Support for Security and Privacy*, HASP '19, New York, NY, USA, 2019. Association for Computing Machinery.

[3] Abhinav Jangda, Bobby Powers, Emery D. Berger, and Arjun Guha. Not so fast: Analyzing the performance of webassembly vs. native code. In *Proceedings of the 2019 USENIX Conference on Usenix Annual Technical Conference*, USENIX ATC '19, page 107–120, USA, 2019. USENIX Association.

[4] PaulNewling. WasmFiddle/wasm_fiddle. https://github.com/WasmFiddle/wasm_fiddle.

[5] Alan Romano, Xinyue Liu, Yonghwi Kwon, and Weihang Wang. An empirical study of bugs in webassembly compilers. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 42–54, 2021.

[6] Jeremy Tuloup. JupyterLite. https://jupyter.org/try-jupyter/lab/.

[7] Jeremy Tuloup. JupyterLite: Jupyter WebAssembly Python. https://blog.jupyter.org/jupyterlite-jupyter-%EF%B8%8F-webassembly-%EF%B8%8F-python-f6e2e41ab3fa, July 2021.