Research Proposal

Serverless computing is an emerging approach to cloud computing that allows programmers to easily deploy and scale small functions that respond to external events. A programmer provides the source code of a serverless function to a serverless provider, and that provider automatically manages function deployment and execution to meet invocation needs. While the name "serverless computing" implies an absence of servers, the reality is that programmers are simply freed from the responsibility of server management. This means that web developers no longer need to manage servers, and those without access to server-side hardware can still host their applications.

However, serverless computing has several issues. Most serverless functions are written in dynamic languages, such as JavaScript. This is troubling because 1) modern JavaScript JIT compilers consume a large amount of time and memory during warmup, 2) the use of dynamic languages requires sandboxing of function instances in containers or Virtual Machines which incurs slowdowns, and 3) multiple function instances within the same sandbox do not have truly isolated memory. For example, recent work by Shahred et all[1] has shown that starting and stopping these sandboxes incurs up to a 20x slowdown compared to native function execution. If these functions were written in a language more well-suited to serverless computing, such as Rust, then these issues could be eliminated. Rust is unique in that it offers robust language-level memory isolation, removing the need for function sandboxing.

Broader Impacts. Although Rust is more well-suited to serverless computing, it has a notoriously steep learning curve and JavaScript is already widely used. This introduces a dilemma where functions written in Rust would provide performance improvements to providers but would place the burden of implementation on developers. A solution to this problem would create an accelerator that incurs serverless computing performance improvements without adding additional burden to developers. This would mean that developers would benefit from lower latency at a lower cost, and serverless providers would lower hardware utilization and power consumption.

Research Plan

Intellectual Merit. I plan to research methods of serverless computing acceleration that allow programmers to benefit from performance improvements and language-level memory safety of Rust, and that do so in a language-neutral way without placing overhead on developers. To do this, I will implement a serverless function accelerator that compiles JavaScript to safe Rust code. But JavaScript language features and Rust language features have fundemental differences, and the research challenge is to create a mechanism that can compile these JavaScript features to their Rust counterparts without asking programmers to develop fundementally different code. A complete accelerator of this kind will demonstrate a mechanism for resolving language feature discrepancies.

We have implemented an initial serverless function accelerator that compiles a beginning subset of JavaScript to Rust. This accelerator captures JavaScript execution using a runtime tracing mechanism and compiles it to an interprocedural representation based on trace trees. These trace trees are compiled to produce safe and efficient Rust code, and type inference is used to confidently

[1]Mohammad Shahrad, Jonathan Balkind, and David Wentzlaf. "Architectural Implications of Function-as-a-Service Computing". In: *IEEE/ACM 52nd International Symposium on Microarchitecture*. 2019.

add static types when possible to the originally dynamically typed program. This will serve as the basis for my continuing research.

Now that an initial accelerator is built, I can use it as a tool to explore new compiler techniques and generate performance improvements. One known possible improvement is the consideration of recursion. The initial accelerator ignores recursion and builds traces linearly. By handling this more efficiently, I can lower memory use and incur faster compilation times. In order to do this, I will need to add recursion detection to the tracing mechanism. Previous JavaScript tracing JIT's have ignored recursion, but Pycket[2] is a tracing JIT for Racket that detects recursion using call graphs and continuation frames. I will look to this work as I implement such a system for JavaScript. I will complete this over the third year of my PhD and it will result in a peer-reviewed publication.

Because the accelerator uses the trace trees as an interprocedural representation, alternative runtime tracing systems could be used with alternative languages. To build to this, I will formalize the tracing mechanism using operational semantics. Guo et al.[3] present a formalization of compiling using traces. While they focus on reasoning about the soundness of trace-based optimizations, I will focus on how using traces resolves language feature discrepancies. Completing this will allow me to investigate the theoretical foundations behind my work. At the same time, I will release an open source implemention of the accelerator so that users can benefit from performance improvements. This will likely be done using Apache OpenWhisk,[4] as it is a popular open source serverless platform. Releasing the software package and formalism together will allow users to implement alternative runtime tracing systems for languages they wish to use. This work will take place over the fourth year of my PhD and will result in a peer-reviewed publication and open source software package.

Then I will want to resolve the differences between user-defined JavaScript classes and user-defined Rust structs and enums. Using the tracing mechanism formalism, I will reason about how to express JavaScript's prototype system and hidden class system in a language-neutral way, and will implement this into the accelerator. Once I am able to compile JavaScript classes to the trace trees, this significantly opens up the subset of JavaScript compatible with the accelerator. I will explore methods of tracing and compiling external JavaScript packages with the serverless functions, and in particular, I will target the JavaScript serverless function API's provided by serverless providers. This will allow users to use external cloud services while still incurring performance benefits. I will complete this over the fifth year of my PhD and it will result in a peer-reviewd publication.

To conclude my PhD, I will perform an analysis of modern serverless platforms, comparing the accelerator implementation in Apache OpenWhisk to Google Cloud Functions, AWS Lambda, and any new serverless providers. This analysis will allow me to compare the results of the accelerator across latency, CPU utilization, memory consumption, and power consumption and demonstrate how these elements can be achieved without placing additional burdens on programmers. This will result in a peer-reviewed publication and the successful completion of my PhD.

[2]Spenser Bauman et al. "Pycket: a tracing JIT for a functional language". In: *ACM SIGPLAN Notices*. Vol. 50. 9. ACM. 2015, pp. 22–34.

[3]Shu-yu Guo and Jens Palsberg. "The essence of compiling with traces". In: *ACM SIGPLAN Notices*. Vol. 46. 1. ACM. 2011, pp. 563–574.

[4]*Apache OpenWhisk*. `https://openwhisk.apache.org`. Accessed Oct 20 2019. 2018.