Fluidity Smart-Contract Language

Introduction:

Fluidity is a turing-complete specification only programming language for writing smart-contracts. Fluidity enables blockchains to execute smart-contracts with semantics convenient to enable high-performance and scalable execution.

Background:

The current state-of-the-art for smart-contracts is the EVM (Ethereum Virtual Machine), a virtual machine designed for sequential execution on the Ethereum blockchain. As of late, many other blockchains have adopted the EVM for compatibility reasons, e.g., Solana, TRON, Moonbeam, etc. They use the EVM to enable migration of dApps despite alternative execution strategies being necessary for maximum performance. Because Fluidity is both a high-level language and designed to be interpreted on a blockchain, it may be compiled to EVM for compatibility or run itself if blockchains were to adopt it. The fact that Fluidity is a specification only axiomatic language enables it to be run with various execution strategies, catering to the blockchain it is run on.

Motivation:

Current languages for writing smart-contracts compile to a virtual machine causing a disconnect between what the developer is writing and what is being run on the blockchain. Fluidity fixes this by enabling blockchains to execute it directly. Blockchains running in asynchronous environments are significantly handicapped when running sequential EVM code because execution is often reduced to a single thread. Fluidity code can be interpreted in an asynchronous, parallel, or concurrent execution model, enabling greater performance and scalability on some blockchains. On Ethereum with a sequential execution model, the simple task of sorting 256 integers with selection sort requires 17M gas while the current limit for each block is 8M. Quicksort hits the limit after only 2,000 elements. Other mechanisms for scalability restrict blockchain security and capability by enforcing design restrictions. Fluidity does enforce any blockchain technology but enables existing technologies to be optimized without broader compatibility issues. Once Fluidity is completed, it will enable truly scalable execution for smart-contracts in a way never before seen. The benefactors will be anyone utilizing a blockchain that supports Fluidity smart-contracts, most prominently for those executing microtransactions.

Project Description:

The project will entail the construction of an interpreter in Zig (methodology explained in the annexe). The expectation is to integrate the project into the web3 ecosystem by speaking at conferences and integrating Fluidity into existing blockchains. The codebase will be regularly updated and published to a Github repository to involve the web3 community for the continued development of Fluidity on their and my behalf. The development will continue beyond the twelve-week fellowship period, likely involving the merging of Fluidity into other existing projects as a dependency.

Timeline:

The first few days will be the development of the parser. The language is simple syntactically, so this should be a quick task (I will likely write this prior); another day will be writing tests for the parser and bug fixes. The following five weeks will be developing an abstract machine for sequentially executing Fluidity. The following four weeks will be the development of the compiler for Fluidity to the abstract machine implemented in the prior five weeks. The last two weeks will be the development of the standard library to accompany Fluidity.