Simplifying Streaming Algorithms

Matthew Builes [1,3] Konstantinos Mamouras [2]

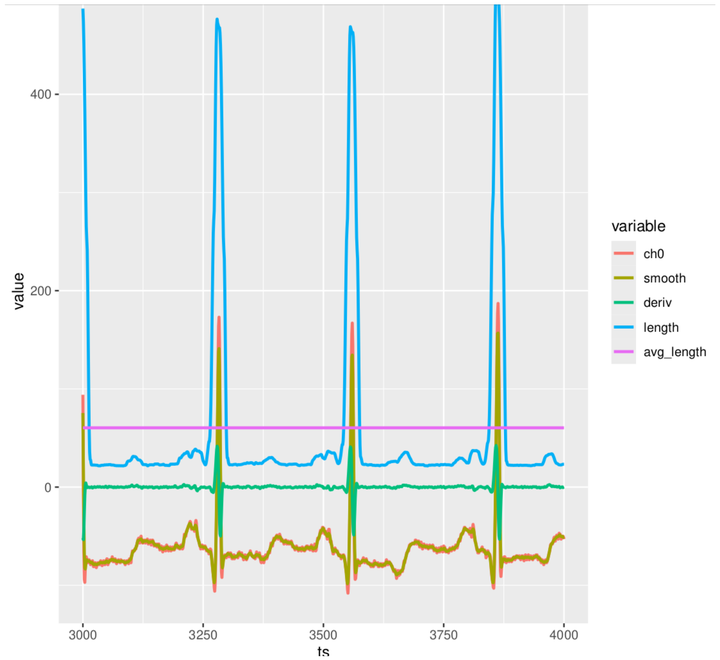
[1] Lawrence E. Elkins High School, Missouri City, TX

[2] Rice University, Houston, TX

[3] Gifted and Talented Mentorship Program, Fort Bend ISD

Introduction

The Internet of Things (IoT) refers to the network of physical devices that interact with their environment through sensors and can communicate and exchange data with other devices. These devices are seeing many new applications, including wearable devices, smart buildings, and healthcare technology. Their ability to collect and transmit information about the physical world and make real-time decisions makes them invaluable, but they suffer from many of the same pitfalls. Many IoT devices are small sensors that are constrained by processing power, storage capabilities, and energy consumption. Additionally, these devices receive continuous input, meaning any computations performed must be in the form of a streaming algorithm. Unlike regular algorithms that have access to all of the data required, streaming algorithms process data sequentially, and due to the theoretically infinite size of the input stream, must be extremely fast and take up low storage. This puts pressure on the programmer to work within these constraints.



Results

To provide examples for the use of the domain specific language, I created two algorithms: a data compression algorithm and a heart rate monitor. The data compression algorithm was built with six query blocks: three for compression, and three for decompression. The compression was achieved by reducing the range of the data and packing the information of multiple numbers together in a way that could still be read and reversed by the decompression algorithm. When tested on Electrocardiogram (ECG) data from the MIT-BIH database, the algorithm achieved over 50% lossless compression. The heart rate detection algorithm, also tested on the MIT-BIH database, worked by applying the curve length function over a sliding window of time to find the highest peaks in the data. These peaks were classified as heart beats, and the average time between these beats was outputted, allowing for the calculation of the heart rate. Notably, both of these algorithms ran extremely quickly and used low amounts of memory, which would translate to high energy efficiency in real world applications, like on a smart watch with limited battery life.

Methodology

To simplify the work of the IoT programmer, I endeavored to create a domain specific language for creating streaming algorithms. Unlike general programming languages like Python or Java that can be used for a wide variety of tasks, domain specific languages are created to give users the tools for one kind of task. The domain specific language I created simplifies the creation of streaming algorithms for IoT programming through query blocks. A query block is an object that takes in an input, performs a transformation, and gives an output. The language provides basic functions as query blocks, such as common operations performed over a sliding window, and allows the user to define more specific query blocks based on their needs. Most importantly, the language allows the user to string together the output of one query into the input of another and even run two queries in parallel using the same input. This allows for complex algorithms to be created by building on simple query blocks.

Next Steps

Currently, the domain specific language is implemented inside of Rust, a low level programming language that allows for fast computations and memory safety guarantees. In the future, though, creating a compiler that translates the domain specific algorithm's code directly into machine code would make the algorithms run even faster. Additionally, the functionalities of the language would need to be expanded to allow for more use cases. Currently, the language is capable of handling one input stream at a time, but other, more complex systems might need to receive various input streams at the same time from different devices, and the information from these streams might not be received at the same rate. The domain specific language will need to include the flexibility to handle these use cases to further simplify programming for users.