ScPy: A SuperCollider Extension for Performing Numerical Computation via Embedded Python

Noah Weninger Abram Hindle

August 23, 2016

Abstract

SuperCollider, a language for sound synthesis and algorithmic composition of audio, supports a wide range of synthesis, effect and analysis algorithms. However, available operations are limited to those implemented explicitly as Unit Generators (UGens). Since UGens are written in C/C++ and loaded as plugins during the SuperCollider server boot process, it is impossible to live code UGens, which limits the user to creating sound as a composition of existing UGens during a performance. Many of the vector operations required for efficiently creating complex audio effects are notably missing or tedious to use. To overcome this, we present ScPy, a UGen which embeds Python within SuperCollider to enable the use of the optimized vector operations provided by the NumPy library.

1 Introduction

Although a number of open-source projects which interface other languages with SuperCollider exist, they are primarily SuperCollider clients, which have features to control synths and send other automation messages. One notable exception is OctaveSC, which embeds GNU Octave within SuperCollider. However, it is designed primarily for performing operations on control rate data arrays and its author notes performance issues which prevent it from being used on audio rate data.

One of the goals of this project is to enable more flexible experimentation with FFT and phase vocoder based operations. SuperCollider contains a total of 32 built in phase vocoder operations, which provide the ability to produce many common effects. There are additionally some extra operations available as user created extensions. However, many conceivable effects are impossible.

ScPy enables users to overcome these limitations. By embedding the Python programming language within SuperCollider, it is possible to do numerical processing operations which would be too slow to perform in real time with SuperCollider alone. Through access to the NumPy library, previously difficult or impossible phase vocoder operations become trivial. Although our usage of

this library for testing purposes has mostly focused on FFT based operations, support exists for passing arbitrary data buffers to Python, which can be used to process numerical data for many other purposes as well.

2 Methodology

During the initial planning stage of the project, we wanted to essentially provide a set of fundamental operations on spectral data which could be composed together to perform any conceivable effect. It was also a requirement that these operations would be syntactically concise in order to enable easy and fast experimentation. Ideally, the system would be able to handle complex effects in real-time.

Although designing a domain-specific language (DSL) for this task was briefly considered, we eventually decided to use Python for a number of reasons. First, Python includes an excellent C API which makes embedding it within other languages trivial in comparison to many other options. Second, Python's syntax is easily readable and concise – it clearly adheres to our goals. Finally, Python has a massive number of community built libraries for performant scientific and numerical computing. With a few small exceptions, these libraries contained every one of the fundamental spectral operations we could think of.

It may seem as though embedding an interpreted language within an interpreted language could offer no performance advantage. There is certainly some performance hit when switching languages and transferring data. However, the biggest advantage comes not strictly from performance but from the diversity of operations which become available with highly optimized implementations. NumPy is used internally by ScPy for handling data arrays, so the entire NumPy library of mathematical functions comes at no cost. Any other Python library can easily be imported as well. Limited experiments have been done with the SciPy library. It would also be possible to use Theano to do processing on the GPU for extra performance. Implementing an equivalent to these popular Python libraries in pure SuperCollider is certainly possible, but it would require a very large amount of work and would see little widespread adoption since SuperCollider is ultimately a niche language.

3 Implementation

Use of ScPy is through two UGens: `Py` and `PyOnce`. Both have one required argument, a block of Python code. A map of variable names may optionally be provided to bind SuperCollider variables to Python variables. Additionally, there is a `DoneAction` argument which specifies an action to occur in SuperCollider after the Python code has finished executing.

Since UGens may only be used as part of a synth, `PyOnce` is provided as a wrapper which hides this detail from the user and allows the execution of

Python code outside of a synth. It is useful for doing initialization work. Since the current implementation executes Python code in a single global namespace, a typical usage patten is for state to be defined in a `PyOnce` block which is later accessed and modified in a `Py` block.

Listing 1: SuperCollider boilerplate for no-op FFT effect with ScPy.

```
(s.waitForBoot {
    var buf = { Buffer.alloc(s, 512) }.dup;
    var hop = 1/4;

    PyOnce("
        def fn(x):
            return x
    ", (hop:hop));

    s.freeAll;
    {
        var in = AudioIn.ar([1, 2]);
        var x = FFT(buf.collect(_.bufnum), in, hop);
        Py("
            out(x, fn(array(x)))
        ", (x:x));
        Out.ar(0, IFFT(x));
    }.play(s);
})
```

ScPy is written mostly in C++, but with a small SuperCollider class library to connect the C++ back-end, and a small Python library that provides some useful operations. Due to differences between these languages, connecting them was somewhat awkward in some cases.

One issue involves the process of passing data between SuperCollider and C++. Since UGens are designed to work primarily with data streams, adding support for passing in many types of data came with a number of challenges. Essentially, all data must be serialized into an array of floats. Strings must be converted to an array of ASCII character codes, and prepended with their length. Arguments passed through to the Python code can have one of many types, and therefore type information must be encoded as well. However, just passing the type of a variable is not enough to know what we can do with it, since we also want to know if it inherits from a class we can use.

4 Evaluation

5 Future Work

At the time of writing, ScPy only supports handling audio rate data via SuperCollider's `Buffer` objects. In order to make some use cases more ergonomic, ScPy could certainly be extended to support raw audio rate input. For example, this would make time domain audio data even simpler to work with than frequency domain data is currently. It would also enable SuperCollider's FFT UGen to be easily re-implemented in Python for greater flexibility.

6 Conclusion

7 Bibliography