An Introduction to Real-time Digital Signal Processing through Parallelism

Sherwyn Sen
CS 159

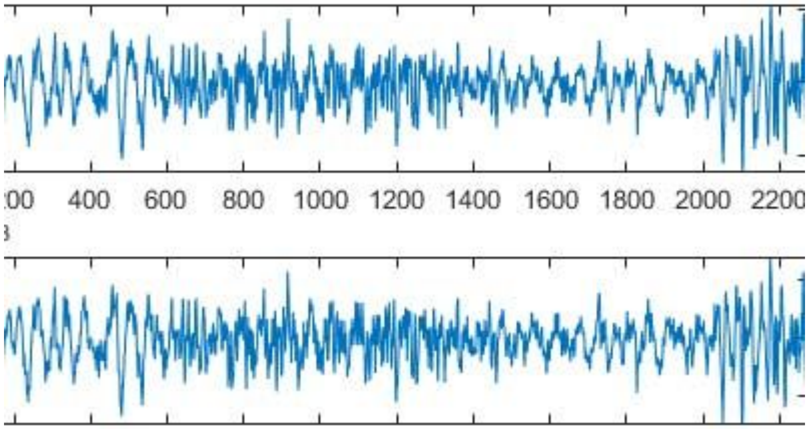
DISCLAIMER

The code I will show in the presentation is not the actual code. It is simplified for the sake of explanation.

REAL CODE (feel free to look at this during the presentation)
https://github.com/ika-musuko/parallelizing-dsp/tree/master/wave_analyzer

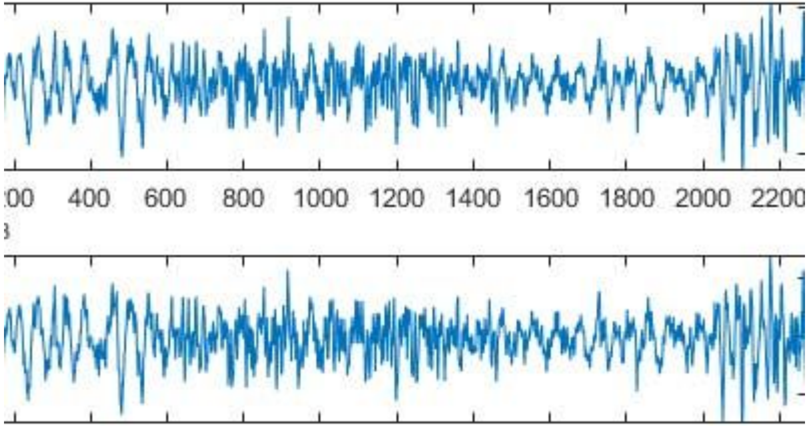
What is Digital Signal Processing?

- Take a signal and modify it
- Signals
  - Motion
  - Sound
  - Image Data



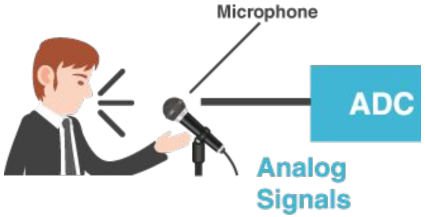
What is Digital Signal Processing?

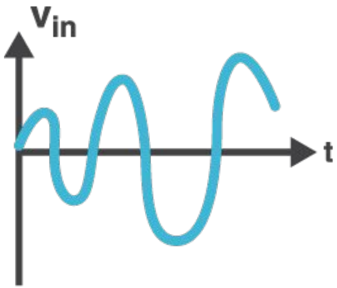
- Take a signal and modify it
- Signals
  - Motion
  - Sound
    - Easiest to explain
  - Image Data



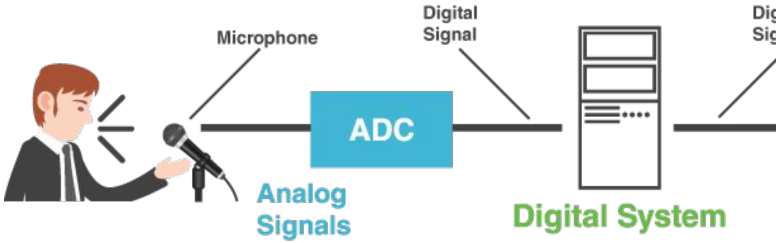
How Signals Get Processed By Computers

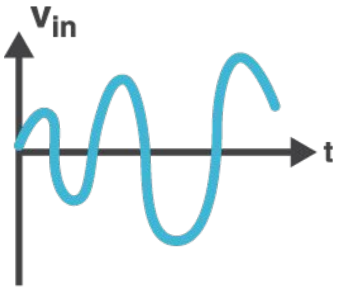
How Signals Get Processed By Computers

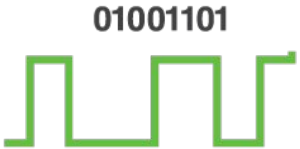




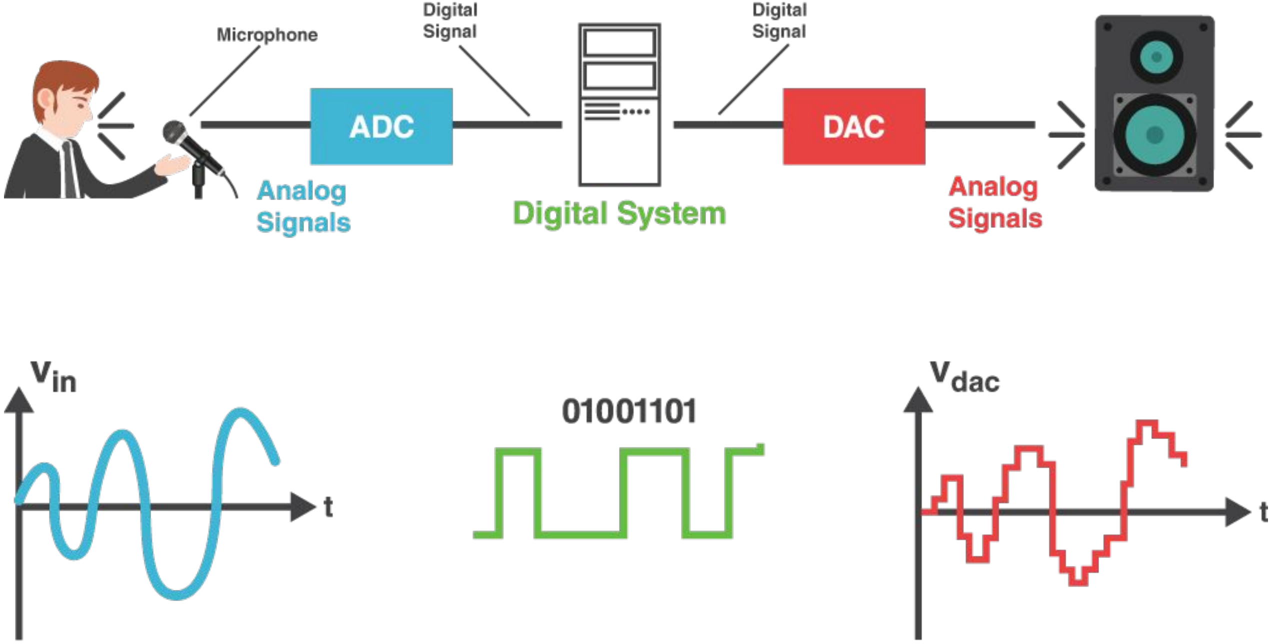
How Signals Get Processed By Computers





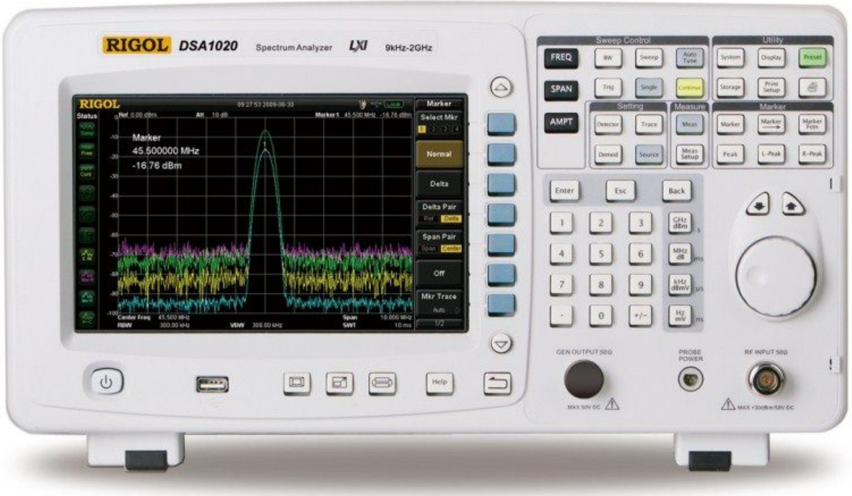


How Signals Get Processed By Computers



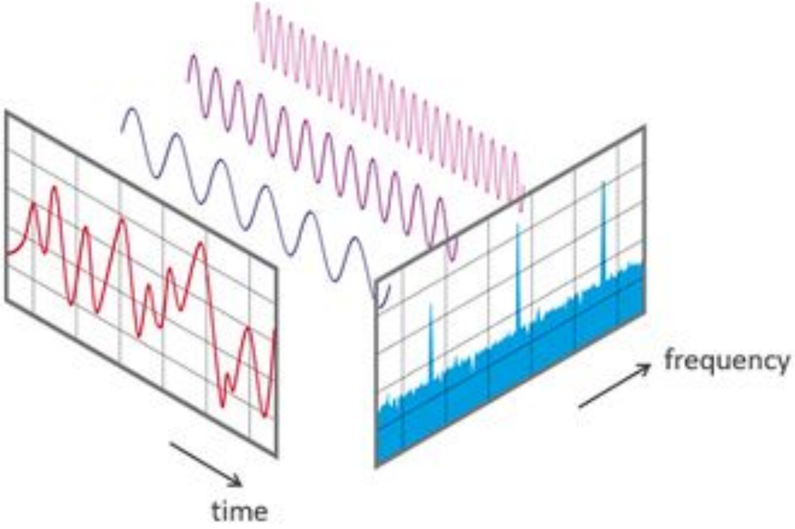
How are we going to Parallelize DSP?

- Let’s make a spectrum analyzer! (in software)



What is a Spectrum Analyzer?

- Measures magnitude of an input signal based on its frequencies



How does a Spectrum Analyzer work?

- Fourier Transform

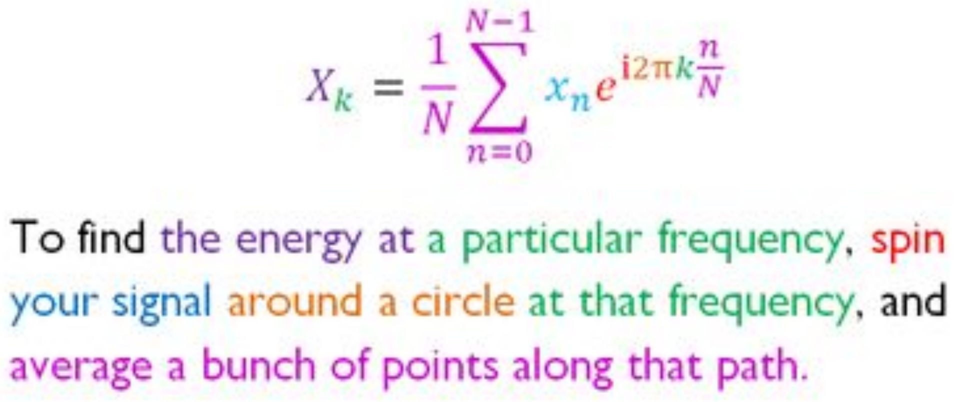
How does a Spectrum Analyzer work?

- Fourier Transform

$$X_k = \frac{1}{N}\sum_{n=0}^{N-1} x_n e^{\mathrm{i}2\pi k\frac{n}{N}}$$

How does a Spectrum Analyzer work?

- Fourier Transform



$$X_k = \frac{1}{N}\sum_{n=0}^{N-1} x_n e^{i2\pi k\frac{n}{N}}$$

To find the energy at a particular frequency, spin your signal around a circle at that frequency, and average a bunch of points along that path.

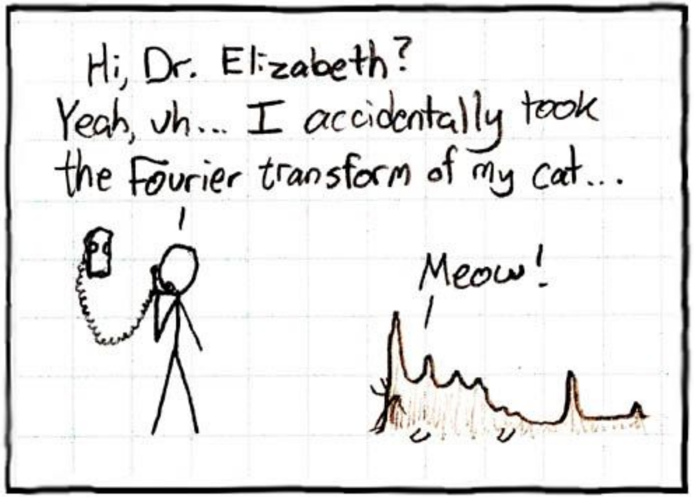
How does a Spectrum Analyzer work?

- Fourier Transform

$$X_k = \frac{1}{N}\sum_{n=0}^{N-1} x_n e^{i2\pi k\frac{n}{N}}$$

To find the energy at a particular frequency, spin your signal around a circle at that frequency, and average a bunch of points along that path.

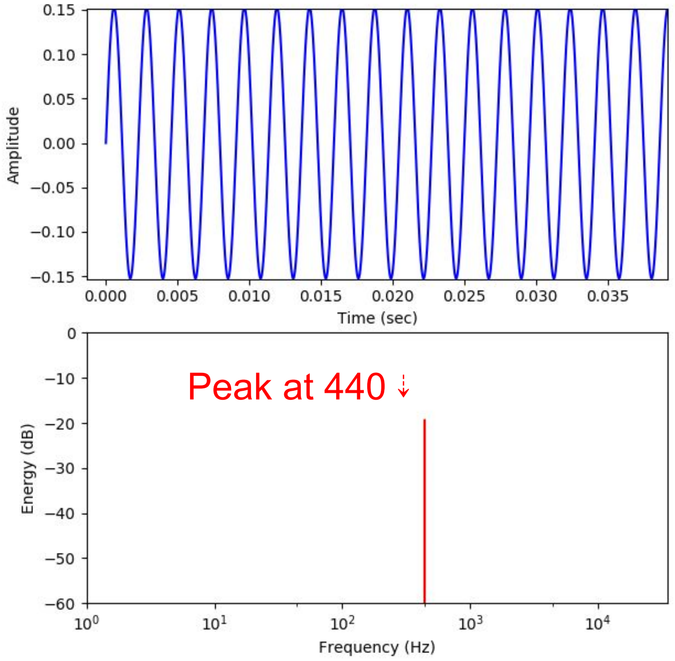
For a more in-depth explanation of the Fourier Transform, consult But what is the Fourier Transform? A visual introduction *by 3blue1brown*


Hi, Dr. Elizabeth?
Yeah, uh... I accidentally took the Fourier transform of my cat...
Meow!

Basic Spectrum Analysis (not real time)

“Result of the Fourier Transform”

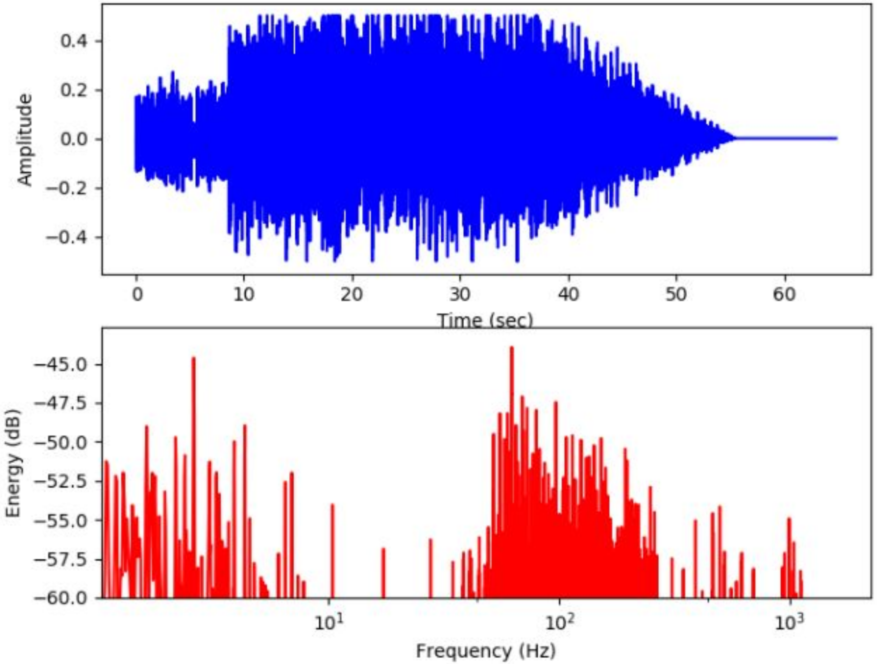
440 Hz Sine Wave



Peak at 440 ⇣

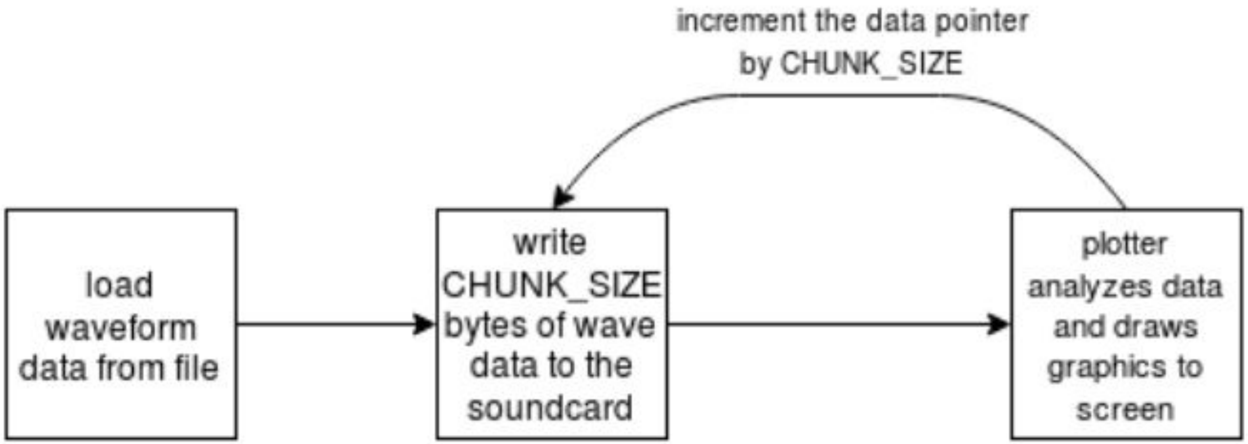
fft_plotter.py

Basic Spectrum Analysis (not real time)



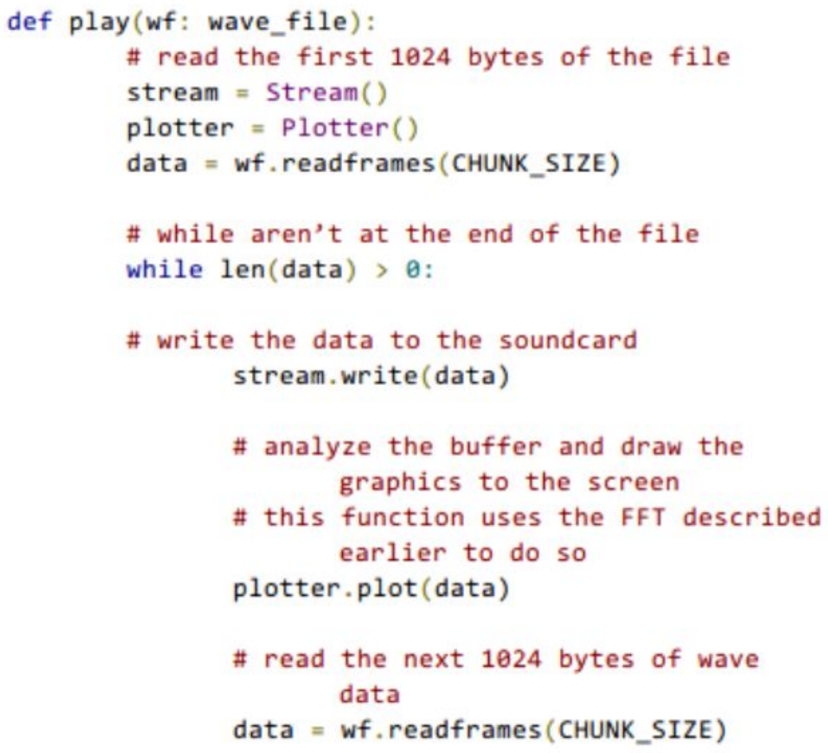
fft_plotter.py

Simple approach to attempt real-time analysis



increment the data pointer by CHUNK_SIZE
load waveform data from file
write CHUNK_SIZE bytes of wave data to the soundcard
plotter analyzes data and draws graphics to screen

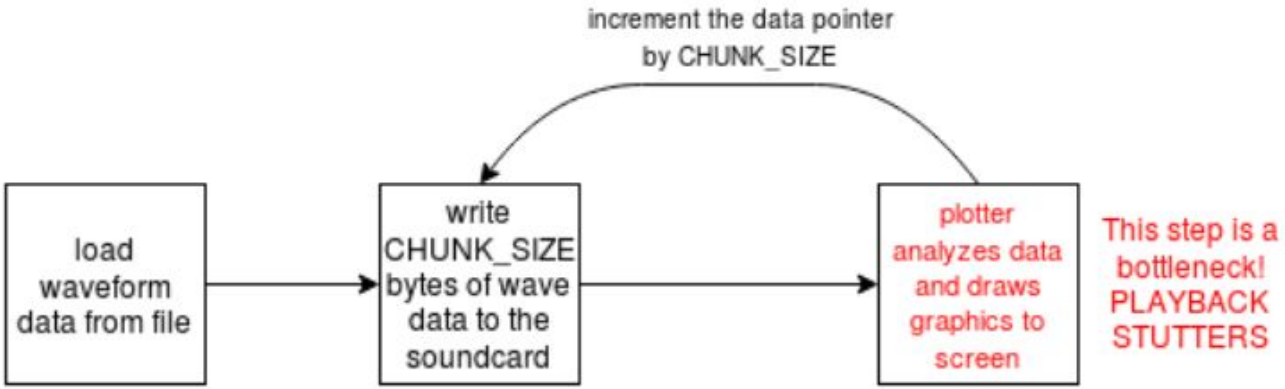
Simple approach to attempt real-time analysis



```
def play(wf: wave_file):
        # read the first 1024 bytes of the file
        stream = Stream()
        plotter = Plotter()
        data = wf.readframes(CHUNK_SIZE)

        # while aren't at the end of the file
        while len(data) > 0:

        # write the data to the soundcard
                stream.write(data)

                # analyze the buffer and draw the
                        graphics to the screen
                # this function uses the FFT described
                        earlier to do so
                plotter.plot(data)

                # read the next 1024 bytes of wave
                        data
                data = wf.readframes(CHUNK_SIZE)
```

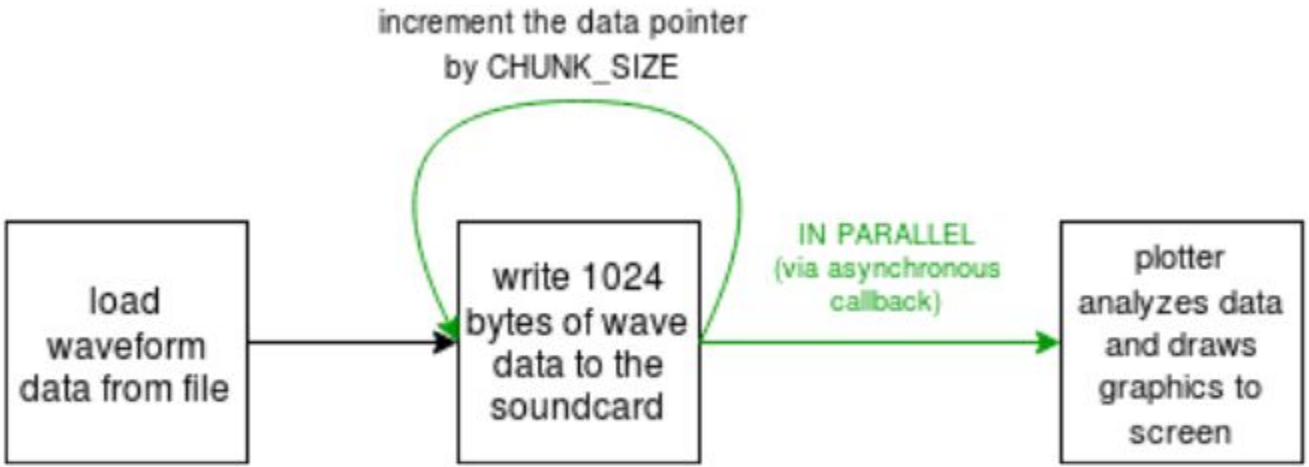
okay! let’s try it...

What happened???


increment the data pointer by CHUNK_SIZE
load waveform data from file
write CHUNK_SIZE bytes of wave data to the soundcard
plotter analyzes data and draws graphics to screen
This step is a bottleneck! PLAYBACK STUTTERS

bad_analyzer.py

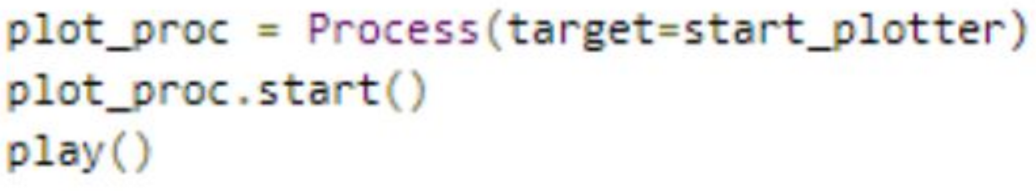
Basic Parallel Analysis - Strategy


increment the data pointer
by CHUNK_SIZE
load waveform data from file
write 1024 bytes of wave data to the soundcard
IN PARALLEL
(via asynchronous callback)
plotter analyzes data and draws graphics to screen

better_analyzer.py

Basic Parallel Analysis - Strategy

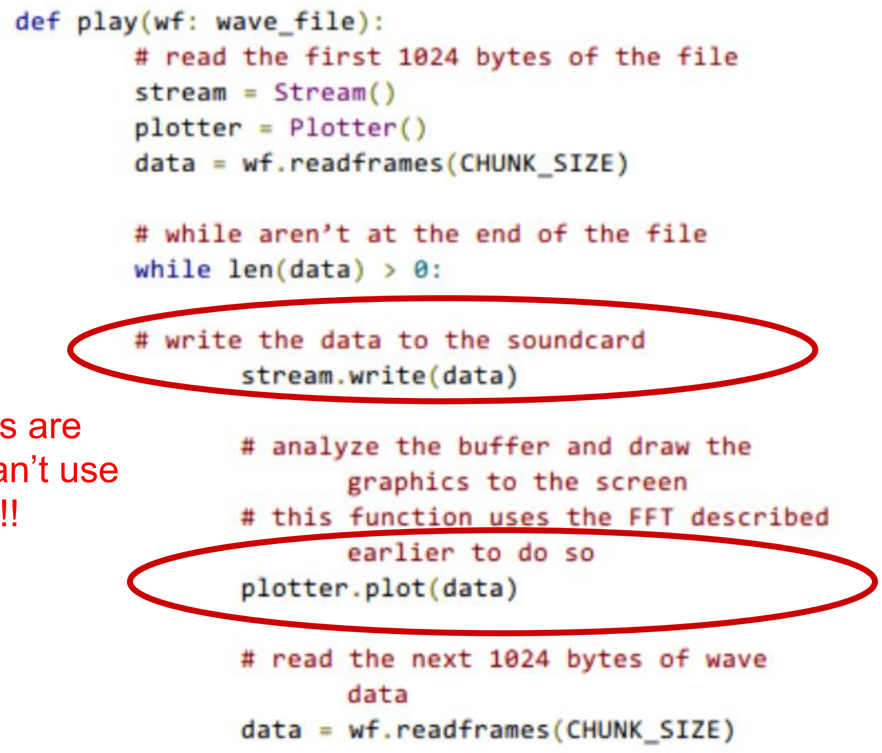
Splitting the player and plotter into processes



```
plot_proc = Process(target=start_plotter)
plot_proc.start()
play()
```

better_analyzer.py

Basic Parallel Analysis Strategy - Problems



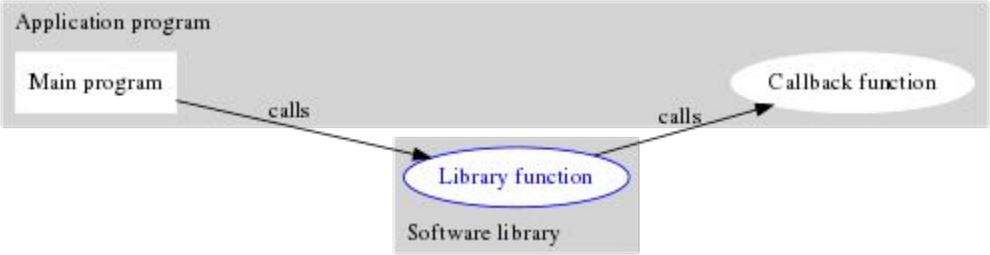
```
def play(wf: wave_file):
        # read the first 1024 bytes of the file
        stream = Stream()
        plotter = Plotter()
        data = wf.readframes(CHUNK_SIZE)

        # while aren't at the end of the file
        while len(data) > 0:

        # write the data to the soundcard
               stream.write(data)

               # analyze the buffer and draw the
                      graphics to the screen
               # this function uses the FFT described
                      earlier to do so
               plotter.plot(data)

               # read the next 1024 bytes of wave
                      data
               data = wf.readframes(CHUNK_SIZE)
```

!!! These functions are blocking I/O !!! Can't use with parallelism !!!!

bad_analyzer.py

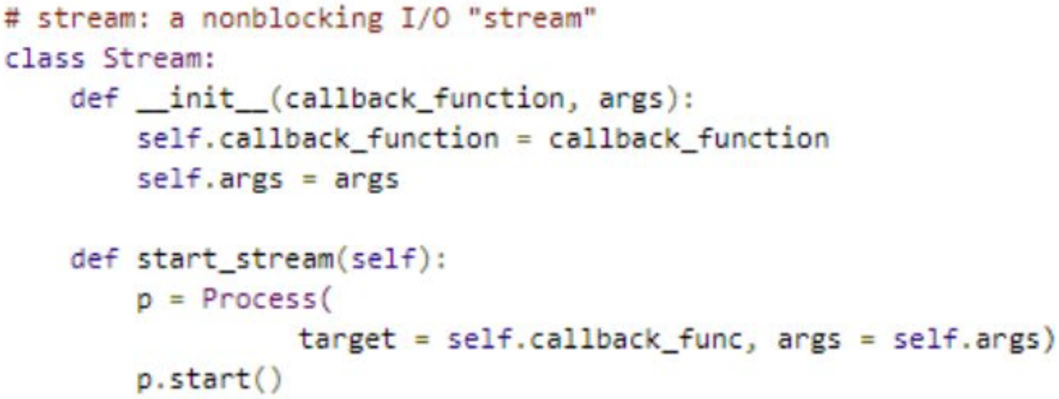
The Solution? Asynchronous Callback Functions

- Callback functions
  - Functions passed to another function or object
  - Called when some sort of event happens
  - Asynchronous callbacks done in parallel using threads/processes
  - Well known examples: Javascript `setTimeout/setInterval`



The Solution? Asynchronous Callback Functions

- “Stream” example

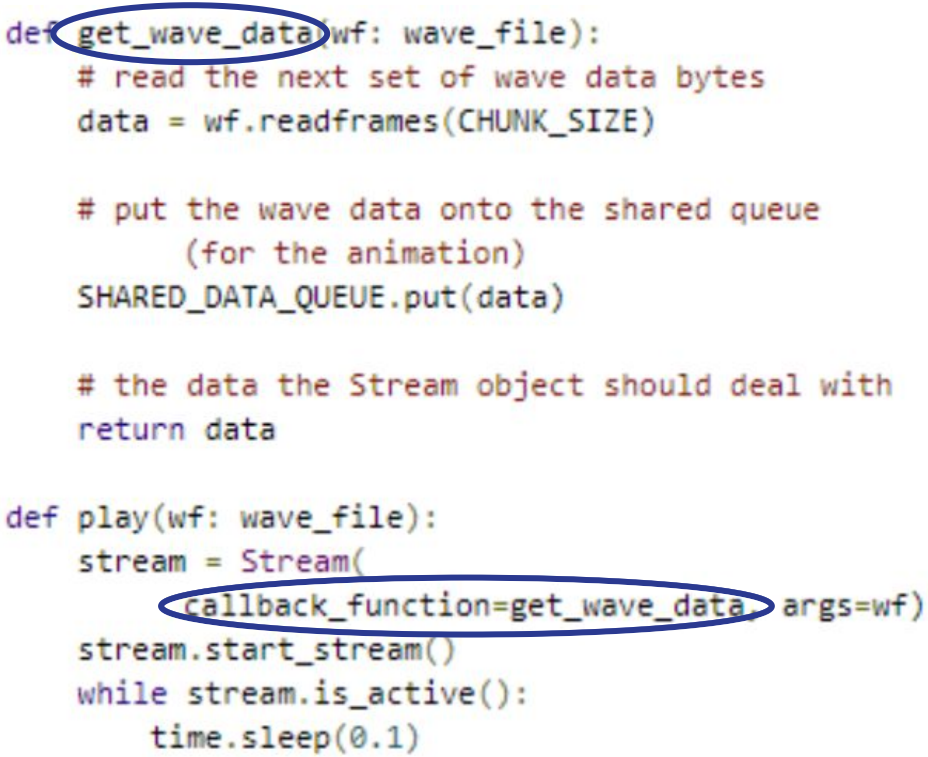


```
# stream: a nonblocking I/O "stream"
class Stream:
    def __init__(callback_function, args):
        self.callback_function = callback_function
        self.args = args

    def start_stream(self):
        p = Process(
                  target = self.callback_func, args = self.args)
        p.start()
```

for audio stream example, see PortAudio’s `portaudio_startstream.c`

The Solution? Asynchronous Callback Functions

audio playback

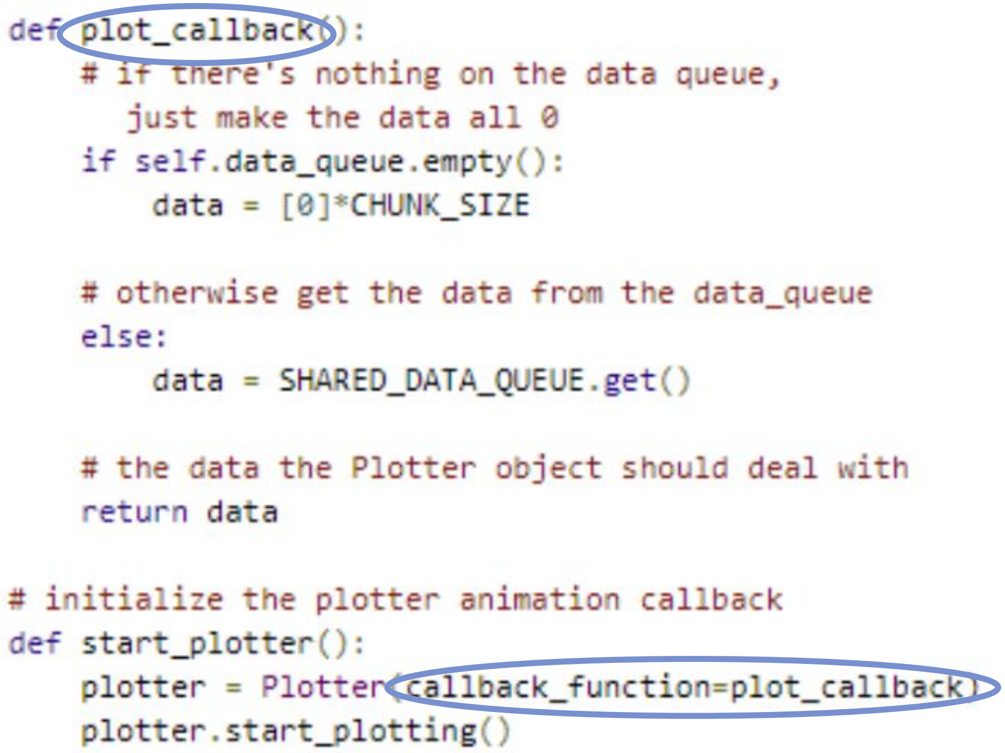


```
def get_wave_data(wf: wave_file):
    # read the next set of wave data bytes
    data = wf.readframes(CHUNK_SIZE)

    # put the wave data onto the shared queue
          (for the animation)
    SHARED_DATA_QUEUE.put(data)

    # the data the Stream object should deal with
    return data

def play(wf: wave_file):
    stream = Stream(
          callback_function=get_wave_data, args=wf)
    stream.start_stream()
    while stream.is_active():
        time.sleep(0.1)
```

better_analyzer.py

The Solution? Asynchronous Callback Functions

plotting on the graph

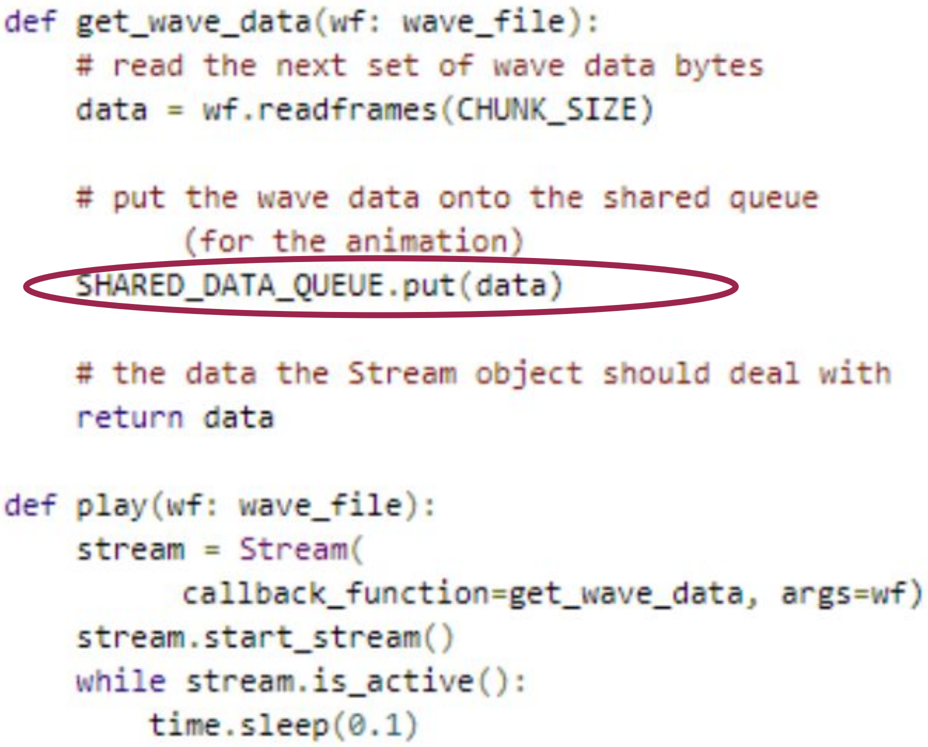


```
def plot_callback():
    # if there's nothing on the data queue,
      just make the data all 0
    if self.data_queue.empty():
        data = [0]*CHUNK_SIZE

    # otherwise get the data from the data_queue
    else:
        data = SHARED_DATA_QUEUE.get()

    # the data the Plotter object should deal with
    return data

# initialize the plotter animation callback
def start_plotter():
    plotter = Plotter(callback_function=plot_callback)
    plotter.start_plotting()
```

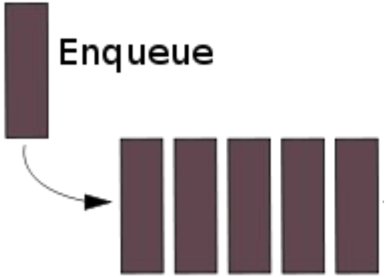
better_analyzer.py

Shared Data Queue - Audio Playback Side

audio playback



```
def get_wave_data(wf: wave_file):
    # read the next set of wave data bytes
    data = wf.readframes(CHUNK_SIZE)

    # put the wave data onto the shared queue
          (for the animation)
    SHARED_DATA_QUEUE.put(data)

    # the data the Stream object should deal with
    return data

def play(wf: wave_file):
    stream = Stream(
          callback_function=get_wave_data, args=wf)
    stream.start_stream()
    while stream.is_active():
        time.sleep(0.1)
```

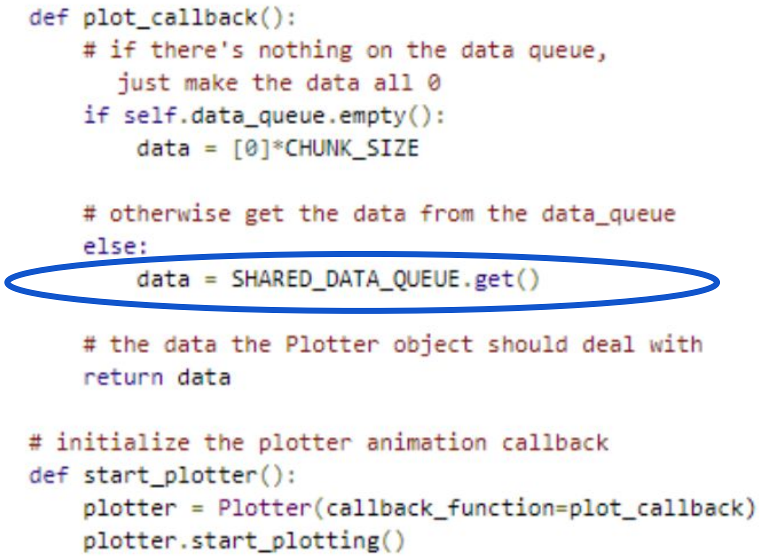


PUT audio data chunks onto the queue before playing them

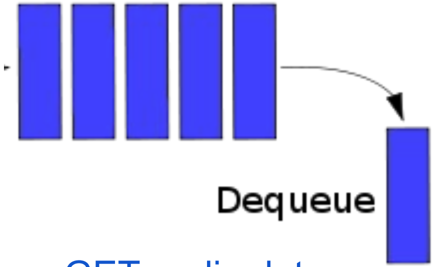
better_analyzer.py

Shared Data Queue - Graph Plotting Side

plotting on the graph



```
def plot_callback():
    # if there's nothing on the data queue,
      just make the data all 0
    if self.data_queue.empty():
        data = [0]*CHUNK_SIZE

    # otherwise get the data from the data_queue
    else:
        data = SHARED_DATA_QUEUE.get()

    # the data the Plotter object should deal with
    return data

# initialize the plotter animation callback
def start_plotter():
    plotter = Plotter(callback_function=plot_callback)
    plotter.start_plotting()
```



GET audio data chunks from the queue and graph them

better_analyzer.py

QUESTION

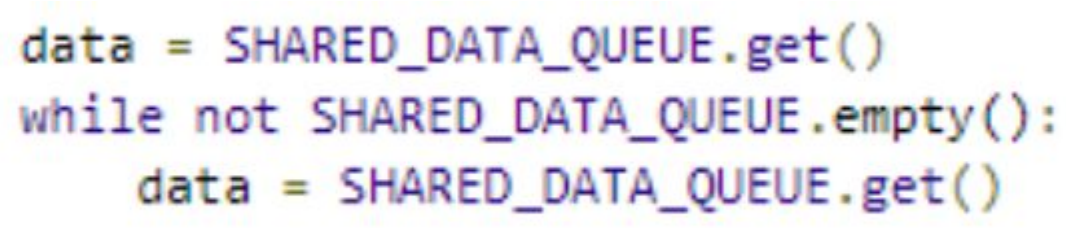
Why not just use a shared data variable between the two processes?

ROUND 2

Almost there...What's the problem?

- The chunk producing audio player is much faster than the chunk consuming graph plotter
- Cause of the LAG!!

Solution: Queue Eating



```
data = SHARED_DATA_QUEUE.get()
while not SHARED_DATA_QUEUE.empty():
    data = SHARED_DATA_QUEUE.get()
```

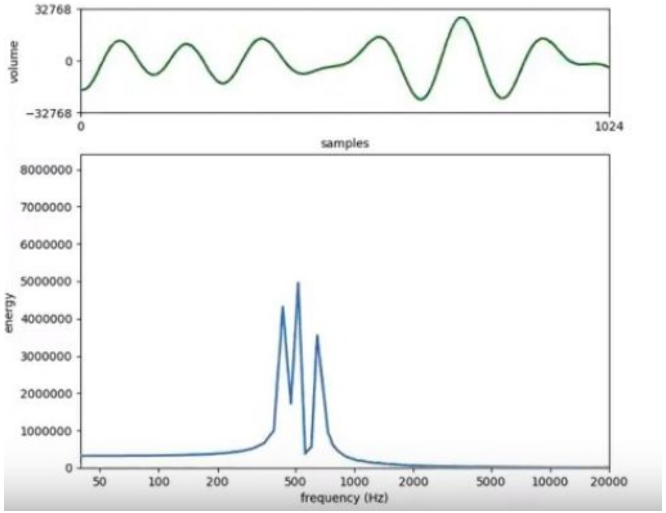
- Plotter pops off all of the items in the queue (“eats” the items)
    - Relatively fast operation compared to processing the chunk
- Only processes the last data chunk on the queue
- *The screen needs to show “now”, not 20 chunks ago!*

wave_analyzer.py

WILL THIS WORK?

Conclusion

- Took a basic digital signal processing routine
- Identified the bottleneck
- Partitioned the tasks
- Parallelized the partitioned tasks



What can be done from here?

- Use multiple wave analyzers, each with their own inputs to create a simple mixer
  - The “master” channel could have its own analyzer as well
- Create an interface for applying effects to the audio in real time
  - Filters
  - Delays
  - Reverb
- Possibilities are endless