A GPU Based DSP Front End

605.417 Introduction to GPU Programming

Michael DiBerardino

Introduction

- Digital Signal Processing front end
  - Gather RF from antenna
  - Typically FPGA
  - Real time streaming required
  - High bandwidth
- Information content low
  - Compared to RF bandwidth
    - 100 MHz+
  - Signal is heavily decimated
    - Voice: 100 MHz to 44.1 KHz

Idea

- GPU's combine parallelism with software
  - Massively parallel
  - Follow traditional software paradigms
  - Faster to program than FPGA's
    - My current FPGA design at work takes 2.5 hours to compile
      - One mistake equates to 2.5 hours of lost time!
- Implement DSP front end in GPU
  - Tune, Filter, Decimate

The Math

- Tune
  - Complex multiplication
  - Generate Sin & Cos
- Filter
  - 1D convolution
    - 1 multiply per thread
  - Parallel summation
    - log2(n)
- Decimate
  - Modulo block id
  - Restrict copies from shared to global memory

The Code - CPU

- Only 2 lines in Octave
  - Involves library calls for decimation
- Implemented in 1 thread

```
#Start timer
tic;

#Frequency shift
arr_baseband = arr .* (cos(2*pi*f_shift.*(0:len-1)/Fs)+sqrt(-1)*sin(2*pi*f_shift.*(0:len-1)/Fs));

#Decimate using 64 tap FIR
arr_dec = decimate(arr_baseband, 2, 64, "fir");

#Stop timer
toc;
```

The Code - GPU

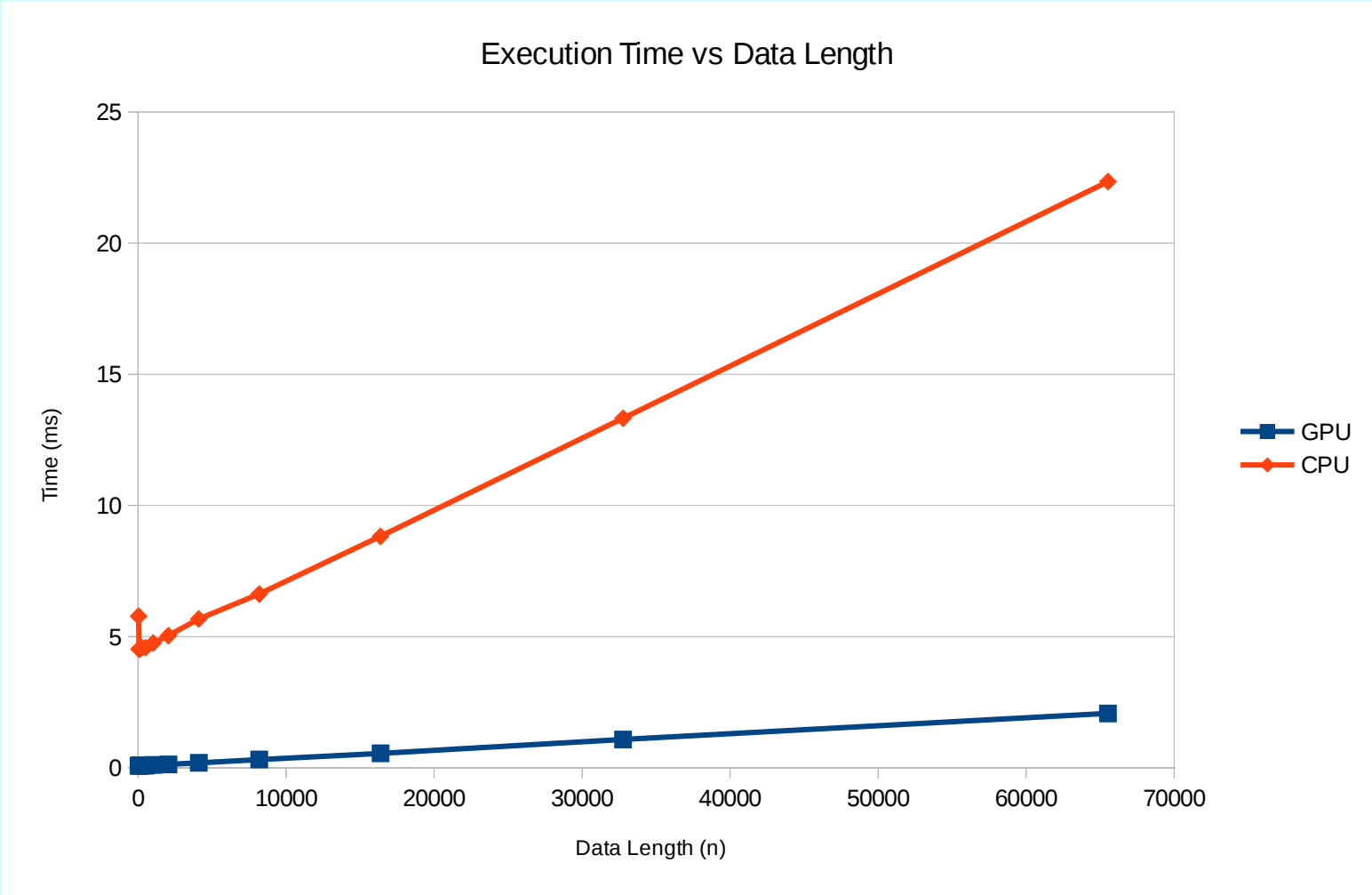
- Kernel is only a few lines long
- Convolution is 6 lines
- Down-sample is 3 lines

```
// FIR based decimation
__global__
void decimate(float * input_buffer, float * output_buffer, const unsigned int decimation_factor)
{
    __shared__ float conv[FIR_SIZE];

    //Who am I?
    const unsigned int thread_id = threadIdx.x;
    const unsigned int block_id = blockIdx.x;

    //Perform the convolution as a copy from global (num samples) to shared (FIR width)
    float sample = 0.0;
    int sample_index = block_id - thread_id;
    if(sample_index >= 0)
        sample = input_buffer[sample_index];
    conv[thread_id] = sample*fir_coef[thread_id];
    __syncthreads();

    //Sum results vector using loop unrolling and shared memory
    sum_array(conv, blockDim.x, thread_id);

    //Decimate
    if(thread_id == 0)
    {
        if((block_id % decimation_factor) == 0)
            output_buffer[block_id / decimation_factor] = conv[0];
    }
}
```

Side by Side

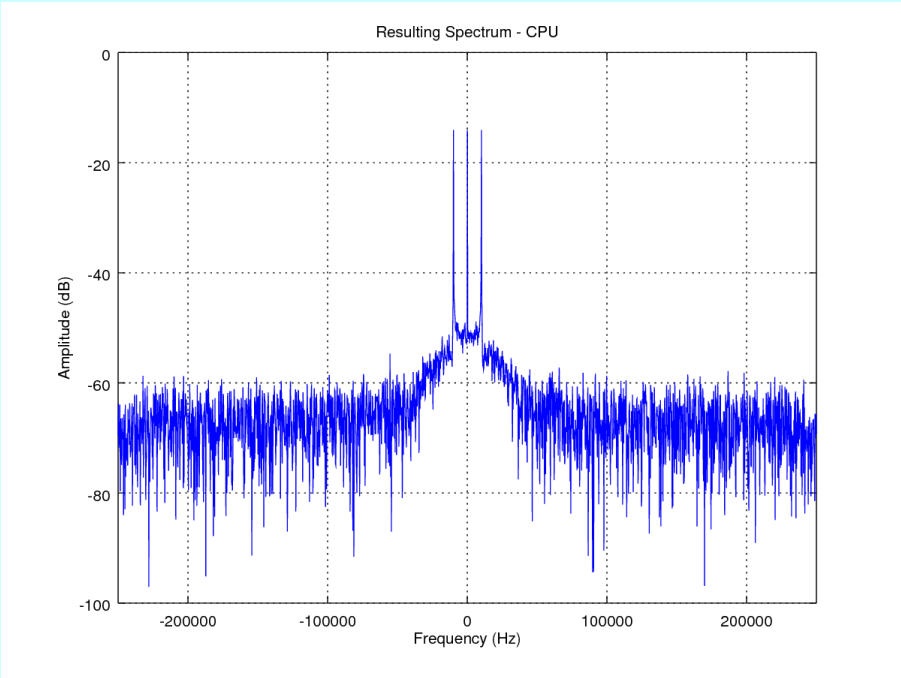
- CPU
  - Octave
    - Interpreted
  - Double precision
  - 1 thread
- GPU
  - Cuda
    - Compiled
  - Single precision
  - 4096 blocks
    - One block per sample
  - 64 threads
    - 1 thread per filter tap
    - Sum 64 threads to get 1 data sample

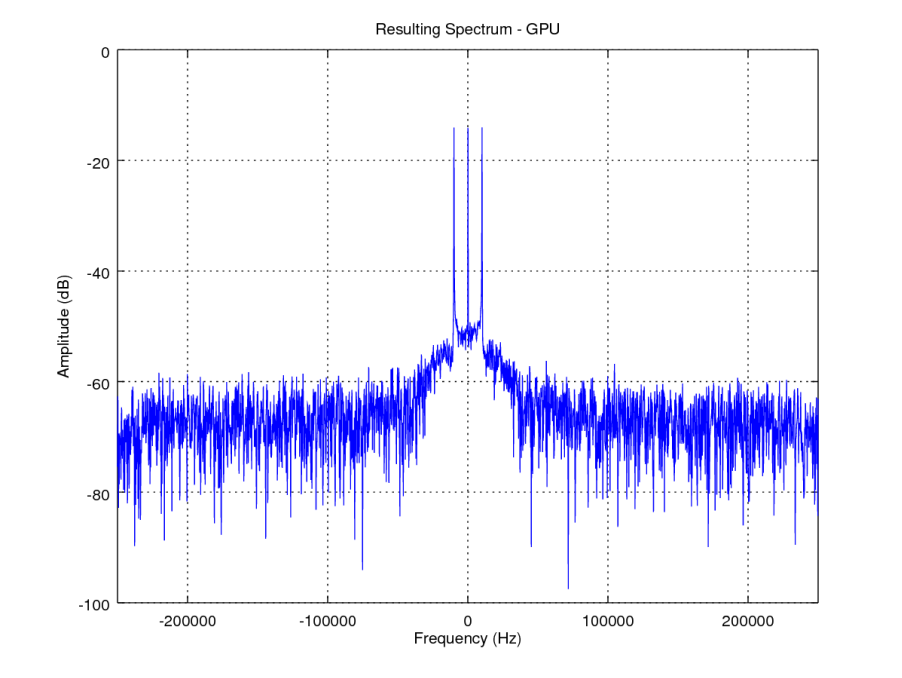
Side by Side


Execution Time vs Data Length
Time (ms)
25
20
15
10
5
0
0
10000
20000
30000
40000
50000
60000
70000
Data Length (n)
GPU
CPU

Side by Side

- CPU result spectrum
  - 3 signals of interest
- GPU result spectrum
  - 3 signals of interest





Conclusion

- Successful implementation
  - GPU is
    - Faster than CPU
      - Roughly 10x
    - Just as accurate as CPU
    - Just as easy to code as CPU
    - Easier to code than FPGA
- CPU & GPU spectra look identical
  - Minor variations in noise floor
    - Possibly different LSB rounding schemes

Future Work

- Change decimation rate on the fly
  - Recalculate filter coefficients
    - Kaiser Window => Bessel functions
- Include FFT
  - Another function frequently handled by FPGA
  - Can be made parallel
- Max length is 65,535
  - Incorporate cohearant, sequential runs
  - Time interleave beginning and ending samples