Adaptive GPU-Accelerated Software Satellite Beacon Processing for Geospace Environmental Sensing

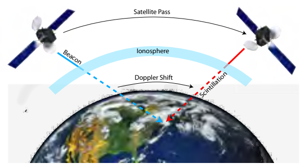
John Grasel - Harvey Mudd College
MIT Haystack REU Summer 2010

Abstract

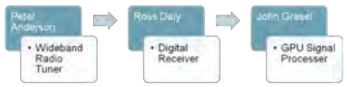
Radio beacons on satellites can be used in conjunction with ground receivers to study the ionosphere. The flexibility of new wideband tuners and digital receiver platforms requires a modular, adaptable software chain to optimally process and interpret beacon overflight data. A python-based system was developed to track the beacon, filter noise, and convert the signal to baseband. The slow and intrinsically parallel nature of the process led to large performance gains when methods were ported to the Graphical Processing Unit (GPU) using a Python wrapper of NVIDIA's CUDA programming language. This poster will discuss methodologies to port algorithms to GPU execution as well as show results for representative beacon overflights in the Westford, MA vicinity.

Introduction

The large-scale structure ionosphere consists of layers of ions and electrons, but within these are small-scale regions of irregular electron density. These density structures cause RF (radio frequency) signals propagating through them to experience random changes in amplitude and changes in phase, called scintillations. Atmospheric scientists use satellite beacon signals to study scintillation, the amplitude and phase variation of a radio signal imposed by ionospheric variations. Scintillation causes a loss of signal integrity which directly affects Global Positioning System (GPS) and the High Frequency (HF) communication band signals. In addition, electron density information, a key component of space weather, can be extracted from scintillation data.



A passing satellite beacon results in a dopper shift and scintillation

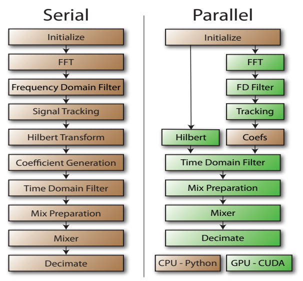


Layout of the Beacon Chain

The recording and processing of beacon data is done with a beacon chain because of its linked but modular nature. As a satellite beacon like the 150 MHz beacon on DMSP F15 transits overhead, its frequency is Doppler-shifted. The RF signal is converted into electrical current by an antenna. The signal is mixed to an frequency (IF) and filtered around the beacon frequency. The analog to digital converter (ADC) samples the analog signal, and the digital receiver downsamples the signal to a lower frequency. The data is fed into an Ethernet backend, where a network computer begins the software signal processing.

The traditional process for studying scintillation has been through dedicated hardware, but faster computers and powerful scientific libraries have allowed more work to be shifted to software written in Python. In addition to decreasing development time, the use of software allows for data processing to be highly configurable. Not only can software be cloned easily, but advances in computer hardware will increase software's performance with little or no work to the programmer. Finally, in software, data can be reprocessed or processed with using configurations.

Design



Too much data is collected from a satellite pass to process it all at once, so a Beacon Block represents a portion of the data. The voltage data is stored in a two-dimensional array whose rows contain voltage data. The width of the Beacon Block, along with the sample rate, thus determines frequency and time resolution.

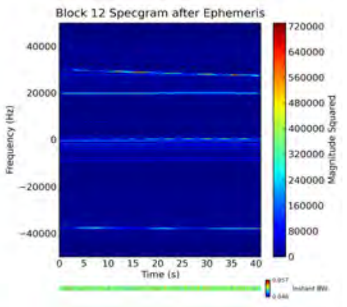
Components

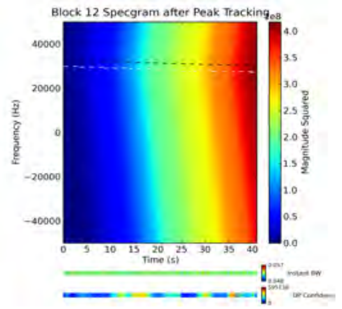
Instantaneous Bandwidth

A measure of the width of frequencies in a signal. Used as an indicator of when the beacon is no longer being tracked.

Ephemeris

The expected frequency of the satellite over time is calculated to filter noise sources before peak tracking. The software package PyEphem was used to calculate the satellite positions, and the frequencies were derived from these.



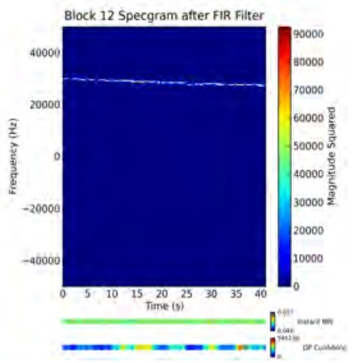


Peak Tracking

The satellite beacon is tracked via a method called Dynamic Programming (DP). The spectrogram of the signal is considered as a three-dimensional surface, the line tracking the peak is the one that stays at the highest power for the longest time. DP allows for this line to be calculated in n-squared time, and it lets other parameters such as the results of previous Beacon Blocks increase the accuracy of the tracking. In addition, the calculation yields a confidence indicator that accurately indicates the instantaneous strength of the tracked signal.

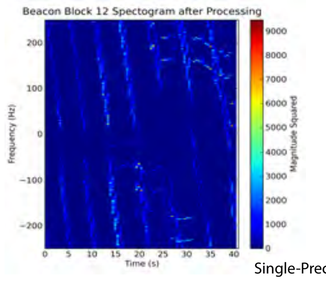
Hilbert Transform

The FIR filter selects both positive and negative frequencies, so a Hilbert Transform removes the negative frequency components. This performs the transform in the frequency domain.



Preselector Filter

A Finite Impulse Response (FIR) bandpass filter removes signal components around the tracked peak. An FIR filter's output is a weighted sum of the current and a finite number (known as the order) of previous values of the input. The weights are generated using a SciPy package. A higher order filter has sharper rolloff, but it requires more computation and has increased delay.



Mixer

Multiplying a signal of known frequency by a complex exponential of opposite frequency centers the signal on baseband.

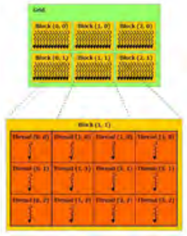
Decimation

Integer downsampling, or the selection of the first sample out of every M points, reduces the signal's bandwidth by a factor of 1/M.

NVIDIA Tesla GPU

Processing Cores: 448
Single-Precision Performance: 1.03 TFlops
Memory: 3GB DDR5
Transfer Bandwidth: 144 GB/s

CUDA and PyCuda



Unlike the CPU, which specializes in logic and program flow, the GPU is specialized for compute-intensive, highly parallel computation. CUDA is a C-subset designed to give the programmer access to the GPU's features in an easily- scalable manner. Functions are called kernels, and one kernel runs on the GPU at a time. A kernel runs on a 2-D grid of blocks. Each block is a 3-D container for threads. Threads within a block share local memory and can easily be synchronized. Blocks are required to be executable in any order. The kernel code is written for an arbitrary thread in an arbitrary block, and CUDA executes the kernel for each thread in each block in parallel.

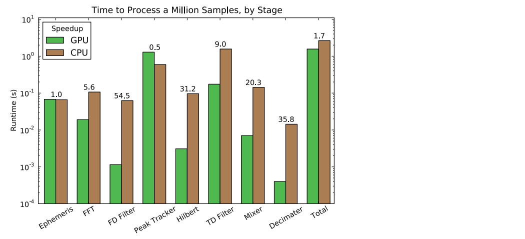
PyCuda is a Python wrapper for the CUDA C library. It opens all of the CUDA library to Python bindings, and includes other convenient features:

- GPUArrays, which are like Numpy arrays that can be passed directly to CUDA functions
- Automatic GPU-accelerated element-wise math for GPUArrays, like scalar and vector products and trig functions
- Simple syntax for Elementwise Kernels and Reductions

Although many of the functions in the center were easily ported to the GPU, some functions had to be completely reworked to be parallelizable. For example, the CPU mixer iterates along the data keeping track of the signal's phase, but the GPU has to precalculate phases.

Performance Results

Porting individual functions to the GPU resulted in modest speedups eroded by slow memory transfer between the CPU and GPU. The GPU-enabled processing is 70% faster than the CPU version, and it processes data at over 600,000 samples per second.



Writing Software for the GPU

- Prototyping applications in Python makes debugging easier
- Converting code from Python map, reduce and list comprehensions to Cuda is simple – write a Kernel that works on one element, and ensure the thread and block dimensions span your data
- Must have enough parallel elements to make memory transfers worthwhile
- Practically no existing CUDA libraries except for FFT
- Metaprogramming, or Python code that writes CUDA kernels as they are needed, speeds execution without sacrificing readability
- Avoid as much code branching, which includes conditionals, as possible

Summary

A working configurable software radio processor was developed in Python and accelerated with PyCuda. The peak tracking with Dynamic Programming proved very effective, and the use of PyCuda sped up the processing to a rate 6 greater than necessary to perform real-time signal processing. CUDA is simple and powerful enough language to become mainstream in data-intensive applications, and PyCuda is a very effective way to access it.

Acknowledgements

Dr. Phil Erickson, Bill Rideout, Dr. Frank Lind
KT Paul, Ching Lue, and all of the Haystack Staff

Sources

• CUDA C Programming Guide 3.1. http://developer.nvidia.com/object/cuda_3_1_downloads.html
• Guturu, Harendra. Beacon Signal Acquisition and Processing Using Software Radio. 2007.
• Ristic, Branko. Algorithms for Instantaneous Bandwidth Estimation. 1996.
• Bernhardt, Paul. New satellite-based systems for ionospheric tomography and scintillation region imaging. 2006.
• Martin, Kenneth. Complex Signal Processing is Not Complex. 2004.