Search for Extraterrestrial Intelligence: GPU Accelerated TurboSETI

Luigi Cruz[‡*], Wael Farah[‡], Richard Elkins[‡]

Abstract—A common technique adopted by the Search For Extraterrestrial Intelligence (SETI) community is monitoring electromagnetic radiation for signs of extraterrestrial technosignatures using ground-based radio observatories. The analysis is made using a Python-based software called TurboSETI to detect narrowband drifting signals inside the recordings that can mean a technosignature. The data stream generated by a telescope can easily reach the rate of terabits per second. Our goal was to improve the processing speeds by writing a GPU-accelerated backend in addition to the original CPU-based implementation of the de-doppler algorithm used to integrate the power of drifting signals. We discuss how we ported a CPU-only program to leverage the parallel capabilities of a GPU using CuPy, Numba, and custom CUDA kernels. The accelerated backend reached a speed-up of an order of magnitude over the CPU implementation.

Index Terms—gpu, numba, cupy, seti, turboseti

1. Introduction

The Search for Extraterrestrial Intelligence (SETI) is a broad term utilized to describe the effort of locating any scientific proof of past or present technology that originated beyond the bounds of Earth. SETI can be performed in a plethora of ways: either actively by deploying orbiters and rovers around planets/moons within the solar system, or passively by either searching for biosignatures in exoplanet atmospheres or "listening" to technologically-capable extraterrestrial civilizations. One of the most common techniques adopted by the SETI community is monitoring electromagnetic radiation for narrowband signs of technosignatures using ground-based radio observatories. This search can be performed in multiple ways: equipment primarily built for this task, like the Allen Telescope Array (California, USA), renting observation time, or in the background while the primary user is conducting other observations. Other radio-observatories useful for this search include the MeerKAT Telescope (Northern Cape, South Africa), Green Bank Telescope (West Virginia, USA), and the Parkes Telescope (New South Wales, Australia). The operation of a radio-telescope is similar to an optical telescope. Instead of using optics to concentrate light into an optical sensor, a radio-telescope operates by concentrating electromagnetic waves into an antenna using a large reflective structure called a "dish" ([Reb82]). The interaction between the metallic antenna and the electromagnetic wave generates a faint electrical current. This effect is then quantized by an analog-to-digital converter as voltages and transmitted to a processing logic to extract useful information from it. The data stream generated by a radio telescope can easily reach the rate of terabits per second because of the ultra-wide bandwidth radio spectrum. The current workflow utilized by the Breakthrough Listen, the largest scientific research program aimed at finding evidence of extraterrestrial intelligence, consists in pre-processing and storing the incoming data as frequency-time binary files ([LCS+19]) in persistent storage for later analysis. This post-analysis is made possible using a Python-based software called TurboSETI ([ESF+17]) to detect narrowband signals that could be drifting in frequency owing to the relative radial velocity between the observer on earth, and the transmitter. The offline processing speed of TurboSETI is directly related to the scientific output of an observation. Each voltage file ingested by TurboSETI is often on the order of a few hundreds of gigabytes. To process data efficiently without Python overhead, the program uses Numpy for near machine-level performance. To measure a potential signal's drift rate, TurboSETI uses a de-doppler algorithm to align the frequency axis according to a pre-set drift rate. Another algorithm called "hitsearch" ([ESF+17]) is then utilized to identify any signal present in the recorded spectrum. These two algorithms are the most resource-hungry elements of the pipeline consuming almost 90% of the running time.

2. Approach

Multiple methods were utilized in this effort to write a GPU-accelerated backend and optimize the CPU implementation of TurboSETI. In this section, we enumerate all three main methods.

2.1. CuPy

The original implementation of TurboSETI heavily depends on Numpy ([HMvdW+20]) for data processing. To keep the number of modifications as low as possible, we implemented the GPU-accelerated backend using CuPy ([OUN+17]). This open-source library offers GPU acceleration backed by NVIDIA CUDA and AMD ROCm while using a Numpy style API. This enabled us to reuse most of the code between the CPU and GPU-based implementations.

2.1. Numba

Some computationally heavy methods of the original CPU-based implementation of TurboSETI were written in Cython. This approach has disadvantages: the developer has to be familiar with Cython syntax to alter the code; the code requires additional logic

* *Corresponding author: lfcruz@seti.org*
‡ *SETI Institute*

Double-Precision (float64)				
Impl.	Device	File A	File B	File C
Cython	CPU	0.44 min	25.26 min	23.06 min
Numba	CPU	0.36 min	20.67 min	22.44 min
CuPy	GPU	0.05 min	2.73 min	3.40 min

TABLE 1
Double precision processing time benchmark with Cython, Numba and CuPy implementation.

Single-Precision (float32)				
Impl.	Device	File A	File B	File C
Numba	CPU	0.26 min	16.13 min	16.15 min
CuPy	GPU	0.03 min	1.52 min	2.14 min

TABLE 2
Single precision processing time benchmark with Numba and CuPy implementation.

to be compiled at installation time. Consequently, it was decided to replace Cython with pure Python methods decorated with the Numba ([LPS15]) accelerator. By leveraging the power of the Just-In-Time (JIT) compiler from Low Level Virtual Machine (LLVM), Numba can compile Python code into assembly code as well as apply Single Instruction/Multiple Data (SIMD) acceleration instructions to achieve near machine-level speeds.

2.2. Single-Precision Floating-Point

The original implementation of the software handled the input data as double-precision floating-point numbers. This behavior would cause all the mathematical operations to take significantly longer to process because of the extended precision. The ultimate precision of the output product is inherently limited by the precision of the original input data which in most cases is represented by an 8-bit signed integer. Therefore, the addition of a single-precision floating-point number decreased the processing time without compromising the useful precision of the output data.

3. Results

To test the speed improvements between implementations we used files from previous observations coming from different observatories. Table 1 indicates the processing times it took to process three different files in double-precision mode. We can notice that the CPU implementation based on Numba is measurably faster than the original CPU implementation based on Cython. At the same time, the GPU-accelerated backend processed the data from 6.8 to 9.3 times faster than the original CPU-based implementation.

Table 2 indicates the same results as Table 1 but with single-precision floating points. The original Cython implementation was left out because it doesn't support single-precision mode. Here, the same data was processed from 7.5 to 10.6 times faster than the Numba CPU-based implementation.

To illustrate the processing time improvement, a single observation containing 105 GB of data was processed in 12 hours by the original CPU-based TurboSETI implementation on an i7-7700K Intel CPU, and just 1 hour and 45 minutes by the GPU-accelerated backend on a GTX 1070 Ti NVIDIA GPU.

4. Conclusion

The original implementation of TurboSETI worked exclusively on the CPU to process data. We implemented a GPU-accelerated backend to leverage the massive parallelization capabilities of a graphical device. The benchmark performed shows that the new CPU and GPU implementation takes significantly less time to process observation data resulting in more science being produced. Based on the results, the recommended configuration to run the program is with single-precision calculations on a GPU device.

REFERENCES

[ESF+17] J. Emilio Enriquez, Andrew Siemion, Griffin Foster, Vishal Gajjar, Greg Hellbourg, Jack Hickish, Howard Isaacson, Danny C. Price, Steve Croft, David DeBoer, Matt Lebofsky, David H. E. MacMahon, and Dan Werthimer. The breakthrough listen search for intelligent life: 1.1–1.9 ghz observations of 692 nearby stars. *The Astrophysical Journal*, 849(2):104, Nov 2017. URL: https://ui.adsabs.harvard.edu/abs/2017ApJ...849..104E/abstract, doi:10.3847/1538-4357/aa8d1b.

[HMvdW+20] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020. URL: https://doi.org/10.1038/s41586-020-2649-2, doi:10.1038/s41586-020-2649-2.

[LCS+19] Matthew Lebofsky, Steve Croft, Andrew P. V. Siemion, Danny C. Price, J. Emilio Enriquez, Howard Isaacson, David H. E. MacMahon, David Anderson, Bryan Brzycki, Jeff Cobb, Daniel Czech, David DeBoer, Julia DeMarines, Jamie Drew, Griffin Foster, Vishal Gajjar, Nectaria Gizani, Greg Hellbourg, Eric J. Korpela, and Brian Lacki. The breakthrough listen search for intelligent life: Public data, formats, reduction, and archiving. *Publications of the Astronomical Society of the Pacific*, 131(1006):124505, Nov 2019. URL: https://arxiv.org/abs/1906.07391, doi:10.1088/1538-3873/ab3e82.

[LPS15] Siu Kwan Lam, Antoine Pitrou, and Stanley Seibert. Numba: A llvm-based python jit compiler. In *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, LLVM '15, New York, NY, USA, 2015. Association for Computing Machinery. URL: https://doi.org/10.1145/2833157.2833162, doi:10.1145/2833157.2833162.

[OUN+17] Ryosuke Okuta, Yuya Unno, Daisuke Nishino, Shohei Hido, and Crissman Loomis. Cupy: A numpy-compatible library for nvidia gpu calculations. In *Proceedings of Workshop on Machine Learning Systems (LearningSys) in The Thirty-first Annual Conference on Neural Information Processing Systems (NIPS)*, 2017. URL: http://learningsys.org/nips17/assets/papers/paper_16.pdf.

[Reb82] Grote Reber. *Cosmic Static*, pages 61–69. Springer Netherlands, Dordrecht, 1982. URL: https://doi.org/10.1007/978-94-009-7752-5_6, doi:10.1007/978-94-009-7752-5_6.