Cupy를 활용한 GPU 기반 시간-주파수 분석 알고리즘 처리 속도 분석

윤우진

LIG넥스원

woojin.yun@lignex1.com

Analysis of Processing Time For Time-Frequency Representation Algorithm based on GPU using Cupy

Woojin Yun

요 약

최근 전자전 ELINT(Electronic Signal Intelligence) 분야에서 광대역 수신 기술이 발전함에 따라 고속 ADC(Analog Digital Converter) 샘플링을 통해 수집된 대용량의 신호 데이터들을 빠르게 처리하기 위한 대규모 데이터 병렬 처리 기술이 필요해지고 있다. 본 논문에서는 Python 환경에서 GPU(Graphic Processing Unit) 연산을 간단하게 수행할 수 있는 Cupy 라이브러리 기반 시간-주파수 분석 방법을 소개하고, CPU 기반 방법과 처리 속도를 비교하였다.

Ⅰ. 서 론

최근 전자전 ELINT 환경에서는 주파수 및 위상 변조가 적용된 복잡한 위협 신호들이 등장하고 있다. 해당 위협들은 광대역 전자전 신호 수신 장비의 ADC를 통해 고속 샘플링으로 대용량의 신호 데이터가 수집되며, 해당 데이터를 고속으로 처리하기 위한 대규모 데이터 병렬 처리 기술이 필요해지고 있다 [1]. 최근 이러한 경향에 따라 GPU를 사용하여 데이터를 병렬 처리할 수 있는 다양한 소프트웨어 라이브러리들이 출시되고 있다. 본 논문에서는 Python 환경에서 GPU 연산을 수행할 수 있는 Cupy 라이브러리를 소개하고, GPU 기반의 단시간 푸리에 변환(Short-Time Fourier Transform, STFT)을 알고리즘의 처리 성능을 분석한다. STFT는 시간에 따른 신호의 주파수 및 위상 변화를 나타낼 수 있는 알고리즘으로, 신호 검출 및 변조 인식을 위한 시간-주파수 분석 기법으로 사용될 수 있다. 본 논문에서는 샘플 길이에 따라 임의의 I/Q(In-phase/Quadrature) 신호를 생성하여 GPU 기반 STFT와 CPU(Central Processing Unit) 기반 STFT 수행 시간을 비교하였다.

Ⅱ. 본론

본 논문에서는 여러 시간-주파수 분석 기법 중 STFT를 선정하여 알고리즘 수행 시간 분석을 진행하였다.

2.1 STFT

STFT는 시간에 따라 변화하는 신호의 주파수 성분을 분석하기 위한 효과적인 시간-주파수 분석 방법이다. STFT는 신호를 겹치는 작은 세그먼트로 나누고 각 세그먼트에 대해 FFT(Fast Fourier Transform)를 적용함으로써 수행되며, 시간에 따른 신호의 주파수 변화를 이미지 형태로 나타낼 수 있다. 식(1)은 STFT의 연산 식이다. 사용자는 윈도우 함수, 윈도우 크기 및 오버랩 비율을 설정하여 원하는 해상도의 STFT 결과를 얻을 수 있다 [2],[3]

$$STFTx(t)(f,\tau) = \int x(t)w(t-\tau)e^{-j2\pi ft}dt \qquad (1)$$

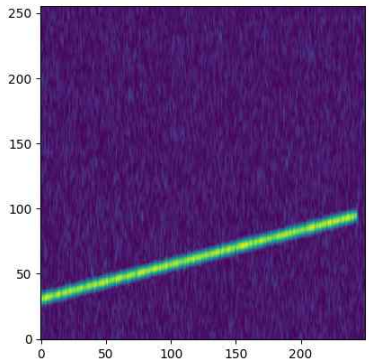

그림 1 LFM 신호 STFT 이미지

그림 1은 50us 길이의 임의의 LFM(Linear Frequency Modulation) 신호를 생성하여 156MHz 샘플링 속도로 얻은 I/Q(Inphase/Quadrature) 신호 데이터를 윈도우 크기 64 오버랩 비율 50%, FFT 사이즈를 256으로 설정하여 2차원 스펙트로그램으로 나타낸 결과이다. 이와 같이 시간-주파수 분석기법으로 수신 I/Q 신호를 변환하면 , 주파수 또는 위상 변화를 2차원 이미지 형태로 나타낼 수 있어 인공지능 학습 데이터 및 전자전 레이더 신호분석에 활용될 수 있다.

2.2 Cupy

Cupy는 GPU 가속을 위해 설계된 오픈 소스 Python 라이브러리로, 대규모 배열 연산을 빠르게 수행할 수 있다 [4]. Python Numpy API(Application Programming Interface)를 대부분 따르고 있어 사용하기가 편리하며, 기본 함수로 STFT 및 FFT를 내장하고 있다.

2.2.1 Cupy STFT 동작 과정

Cupy를 활용한 GPU 기반 STFT는 대략 세 부분에 걸쳐 수행된다. 첫 번째는 Host 메모리 상의 입력 데이터를 GPU로 복사하는 부분으로, Host to Device로 표현하였다. 두 번째로는 GPU 메모리에 할당된 입력 데이터를 통해 GPU기반 STFT 연산이 수행되는 과정으로, Execution on

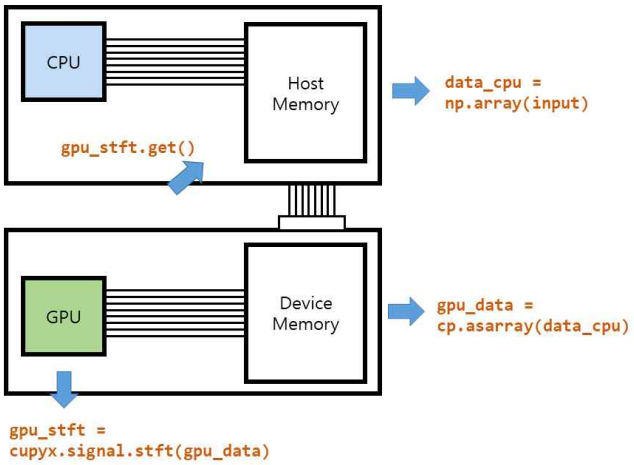

그림 2 GPU 기반 STFT 동작 과정

GPU로 표현하였다. 마지막으로 GPU 메모리 상에 할당된 STFT 연산 결과 값을 다시 Host 메모리로 복사하는 과정으로, Device to Host로 표현하였다. 해당 내용은 그림 2에 도식화 하였다.

1) Host to Device(HtoD)

GPU에서 연산을 수행하기 위해서는 우선 입력 데이터가 GPU 메모리 상에 존재해야 한다. 따라서 Host 메모리 공간의 데이터를 GPU 메모리 공간으로 복사하는 과정이 필요하다. 해당 과정을 Host to Device라고 하며, 본 실험에서는 입력 I/Q 데이터를 Python Numpy 배열로 생성한 뒤, Cupy의 asarray 함수를 통해 해당 배열을 Cupy 배열로 변환하면 간단히 GPU 메모리 공간으로 복사할 수 있다.

2) Execution on GPU (Execution)

Host to Device 과정을 통해 GPU 메모리 공간에 할당된 입력 I/Q 데이터로 STFT 연산을 수행하는 과정이다. Cupy 내장 함수 stft를 사용하여 입력 값으로 1단계에서 생성했던 Cupy 배열 I/Q 데이터를 입력해주면 STFT가 수행되고, 수행 결과는 Cupy 배열로 출력되어 새로운 GPU 메모리 공간에 할당된다.

3) Device to Host(DtoH)

2단계에서 GPU 메모리에 할당된 STFT 연산 결과 데이터를 다시 호스트 메모리로 복사하기 위한 과정이다. 생성된 Cupy 배열에서 get 함수를 사용하면 Numpy 배열을 출력하게 되며 Host 메모리 상에 해당 데이터가 복사된다.

2.3 수행 시간 비교

GPU 기반 STFT 연산 시간 분석을 위해 CPU 기반 STFT 연산 시간과 비교하였다. CPU 기반 STFT는 Python scipy 라이브러리의 STFT 함수를 사용하였다 [4]. 실험에 사용된 GPU는 NVIDIA RTX-A4500이며, CPU는 Intel Xeon W-2295를 사용하였다. GPU 연산의 경우 HtoD, Execution, DtoH 과정을 포함한 시간을 측정하였으며, CPU 연산의 경우 STFT 함수가 실행되는 시간을 측정하였다. 추가적으로 데이터의 수가 단일 또는 배치 단위 일 때로 구분하여 각 데이터의 샘플 수 별 수행 시간을 비교하였다. 배치 단위 데이터의 경우 샘플 수 별로 100개의 임의의 I/Q 데이터를 생성하여 실험을 진행하였으며 표 1은 단일 입력의 경우의 수행 시간을 비교한 결과를, 표 2는 배치 단위 입력의 경우 수행 시간을 비교하여 나타내었다.

Ⅲ. 결론

단일 데이터의 경우 16384 개 이하의 샘플을 처리하는데 있어 CPU가 GPU대비 빠른 연산 속도를 보였다. 반면 32768개 이상 샘플부터는 GPU의 처리속도가 더 빨랐다. 이는 GPU 연산 시 필연적으로 발생하는 장치 간 메모리 복사 오버헤드 때문에 낮은 샘플 수 에서는 오히려 CPU의 처리 속도가 것으로 보인다. 하지만 샘플 수가 커질수록 GPU의 연산 속도가 더 빨라지는 것을 확인할 수 있었다. 다중 데이터의 경우 GPU와

표 1. 단일 데이터 수행 시간 비교(ms)

샘플 수	CPU	GPU
4096	1.46	4.26
8192	2.45	6.19
16384	4.63	6.34
32768	9.83	5.76
65536	32.92	8.31

표 2. 다중 데이터 수행 시간 비교(ms)

샘플 수	CPU	GPU
4096	2.16	0.36
8192	4.3	0.71
16384	8.66	1.45
32768	17.16	2.93
65536	34.25	5.97

CPU의 처리 속도 차이가 더 크게 나타났다. 반면 배치 단위 데이터의 경우 모든 샘플 수에서 GPU가 CPU대비 더 빠른 처리속도를 보이는 것을 확인할 수 있었다. 65536개 샘플의 경우 GPU가 CPU 대비 약 6배 빠른 처리 속도를 보여주었다. 이는 샘플 수 및 데이터의 차원이 증가할수록 GPU의 병렬 연산의 효과가 더욱 크게 나타난다고 해석할 수 있다. 이렇듯 GPU 기반의 다양한 신호처리 알고리즘이 개발된다면, 방대해지는 신호 데이터를 효과적으로 빠르게 처리할 수 있을 것이다.

참 고 문 헌

[1] Kim, S. W., J. H. Kim, and U. S. Jeong. "Development trend of digital receiving technology for electronic warfare." The Journal of Korean Institute of Electromagnetic Engineering and Science 32.2 (2021): 21-27.

[2] Oppenheim, Alan V. Discrete-time signal processing. Pearson Education India, 1999.

[3] Griffin, Daniel, and Jae Lim. "Signal estimation from modified short-time Fourier transform." IEEE Transactions on acoustics, speech, and signal processing 32.2 (1984): 236-243.

[4] Nishino, R. O. Y. U. D., and Shohei Hido Crissman Loomis. "Cupy: A numpy-compatible library for nvidia gpu calculations." 31st confernce on neural information processing systems 151.7 (2017).

[5] Virtanen, Pauli, et al. "SciPy 1.0: fundamental algorithms for scientific computing in Python." Nature methods 17.3 (2020): 261-272.[1] Davies R. W." The Data Encryption standard in perspective,"Computer Security and the Data Encryption Standard, pp. 129-132.