논문 20-45-03-16 The Journal of Korean Institute of Communications and Information Sciences '20-03 Vol.45 No.03
https://doi.org/10.7840/kics.2020.45.3.577

GPGPU 병렬 프로그래밍 기법을 활용한 DSSS 신호의 Tuned Save 구현

윤 현 철◆°, 이 현 휘*, 강 현 진*, 김 재 윤*, 문 병 호*, 김 선 교**

Tuned Save of DSSS Signal Using Parallel Programming Skills of General Purpose Computing on Graphics Processing Units

Hyun-Chul Yoon◆°, Hyeon-Hwi Lee*, Hyun-Jin Kang*, Jae-Yun Kim*,
Byung-Ho Moon*, Seon-Kyo Kim**

요 약

최근 고속 신호처리 시스템이 발달하면서 대용량의 광대역 I/Q를 저장하고 이를 분석하기 위한 장비의 기술 개발이 활발하다. 특히, 광대역 I/Q를 분석하기 위한 알고리즘 연산의 시간소모가 크기 때문에 FPGA, DSP 등의 하드웨어에 의한 개발이 대부분이다. 본 논문에서는 대용량의 DSSS신호가 저장된 광대역 I/Q 데이터 신호처리를 위해 전체 I/Q 데이터를 분석하는 것이 아닌, 분석에 필요한 부분만 저장하는 Tuned Save 기능을 구현하였다. 또한, 고가의 하드웨어 장비가 아닌 PC 기반 기술인 GPGPU를 활용, 복잡도가 큰 연산에 직접 적용하여 Tuned Save 기능을 구현하였다. 성능 평가를 통해 데이터의 누락없이 CPU 대비 고속의 Tuned Save 기능을 제공함을 확인하였다.

키워드 : 직접확산변조, 부분 저장, GPGPU, CUDA, 신호처리, 병렬 처리 프로그래밍
Keywords : DSSS, Tuned Save, GPGPU, CUDA, Signal Processing, Parallel Programming

ABSTRACT

Recently, development of high-speed signal processing systems, technology development of equipment for storing and analyzing large-capacity broadband I/Q is active. In particular, since the algorithm operation for analyzing wideband I/Q is time consuming, development by hardware such as FPGA and DSP is most common. In this paper, we implemented the Tuned Save function, which stores only the parts needed for analysis, rather than analyzing the entire I/Q data for processing the wideband I / Q data signal in which large-capacity DSSS signals are stored. In addition, the tuned save function is implemented by directly applying to GPGPU, a PC-based technology, rather than expensive hardware equipment. The evaluation performance show that it provides fast tuned save function compared to CPU without data loss.

※ 본 연구는 국방과학연구소 광대역저피탐 DSSS 신호탐지 및 분석시험장치대 과제에 의해 수행되었습니다.
◆° First and Corresponding Author : LIG Nex1, hyunchul.yoon@lignex1.com, 정회원
* LIG Nex1, {hyeonhwi.lee, hyunjin.kang, jykim0118}@lignex1.com, 정회원; bhmoons123@lignex1.com
** Agemcy for Defense Development, seonkyo.kim@gmail.com, 정회원
논문번호 : 201911-275-0-SE, Received October 31, 2019; Revised December 31, 2019; Accepted December 31, 2019

Ⅰ. 서 론

고속 신호처리 시스템이 발달하면서 대용량의 광대역 신호의 저장에 대한 요구가 증가되고 있다. 특히, 막대한 통신 신호정보를 필요로 하는 군용 통신 시스템의 경우, 대량의 정보를 분석하기 위해 대용량의 광대역 및 협대역 신호를 저장하고, 이를 분석하기 위한 장비에 대한 연구가 활발하게 진행되고 있다. 특히, 신호분석에 가장 많은 연산을 처리하는 신호처리 부분의 경우, FPGA[1,2]를 이용한 병렬 신호처리에 대한 연구가 많이 진행되어왔다.

한편, FPGA를 활용한 Personal Computer(PC) 기반의 통신 분석 시스템을 구축하는 경우, 고가의 FPGA 시스템을 별도 구축해야하고, PC에서 신호처리 데이터를 읽기 위한 별도의 인터페이스를 설계해야 한다. PC 기반의 CPU의 경우, 단일 처리 성능은 우수하지만, 반복연산이 많은 신호처리 분야의 경우에는 GPU 대비 성능이 낮아진다[3]. 따라서, PC 기반의 고속신호처리 시스템을 구축하기 위해서는 병렬처리를 위한 별도의 시스템 구축이 필요하다.

본 논문에서 고속의 FPGA의 고속 병렬 처리[4]를 대체할 GPGPU[5] 방식의 CUDA Programming[5]을 이용한 PC 기반의 통신신호분석 시스템[6]을 구축하는 방안을 제시한다.

GPGPU를 이용한 통신신호분석 시스템을 구축할 경우, 신호처리 흐름에 따라 기능을 구분할 수 있다. 먼저, 광대역 I/Q로 저장된 미상의 통신신호에 대한 분석을 위해서는 저장된 광대역 신호에서 분석에 필요한 신호를 선별하는 작업을 수행한다. 전체 광대역 신호에서 분석에 필요한 부분의 I/Q 데이터만 추출하는 기술인 Tuned Save를 수행하기 위해서는 분석에 필요한 주파수를 중심주파수로 이동시키는 Mixer[7] 과정을 거치게 된다. Mixer를 통해 이동한 주파수는 원하는 대역의 주파수만 통과시키기 위한 방법인 Low Pass Filter(LPF)[7]를 통과한다.

본 논문에서는 앞서 언급한 병렬처리 시스템을 구축하기 위해 Tuned Save 기능의 각 단계를 CUDA를 이용하여 구현하였으며 CPU와의 성능 비교를 통해 알고리즘을 검증하였다.

본 논문의 구성은 다음과 같다. Ⅱ장에서는 Mixer, LPF의 단계별 알고리즘을 제시하고, 수학적 정의를 설명한다. 한편, Ⅲ장에서는 제안하는 CUDA 적용 방안에 대한 성능 평가를 통해 CPU와 GPU로 구현된 알고리즘 각각의 성능 분석을 수행한다. 그 후, 논문의 결론을 맺는다.

Ⅱ. Tuned Save 구조

2.1 Tuned Save 흐름

광대역 I/Q 데이터의 저장은 특정 주파수 영역의 신호 검출 여부를 판단하여 정해진 대역폭만큼의 대역 신호를 전체 저장한다. 저장된 대역폭의 전체 영역을 분석하는 경우도 있지만, 특정 주파수 영역의 신호만 분석하기 원할 경우에는 그 대역의 신호만 별도 추출하여 분석하면 연산량이 그만큼 줄어들게 된다. 따라서, 광대역 I/Q 데이터의 효율적인 분석을 위해서는 전체 대역폭의 신호를 분석하는 것이 아닌, 분석에 필요한 부분만 분석하는 기능인 Tuned Save 가 필수적이다. 그림 1은 Tuned Save의 기본적인 흐름을 보여준다.

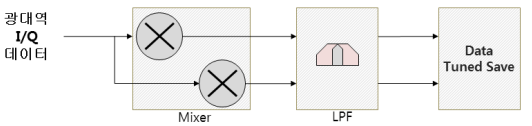


그림 1. Tuned Save 흐름도
Fig. 1. Flow chart of Tuned save.

2.2 Mixer

광대역 I/Q 데이터는 분석에 필요한 부분을 특정 대역폭만큼 잘라내어 저장하기 위해 Mixer를 수행한다.

Mixer는 분석에 필요한 주파수 영역의 신호를 중심주파수로 옮겨주는 방식이다. Mixer에 대한 수학적 정의는 식 (1)과 같다.

$$\begin{aligned} y_{re}[i] &= x[i] \times \cos(2\pi f_c t) \\ y_{im}[i] &= x[i] \times \sin(2\pi f_c t) \end{aligned} \tag{1}$$

$y_{re}[i]$는 Mixer 의 Real 성분, $y_{im}[i]$는 Mixer 의 Image 성분, $x[i]$는 입력된 광대역 I/Q 데이터, f_c는 중심주파수, t는 시간의 흐름에 대한 Index를 의미한다.

본 논문에서는 입력되는 광대역 I/Q 신호에 대한 Mixer를 수행하기 위하여 Mixer의 주요 연산인 cos과 sin 연산을 CPU와 GPU로 각각 구현하였다. Mixer를 통해 중심주파수를 이동한 I/Q 데이터는 다음 단계인 LPF를 거치게 된다. 그림 2는 Mixer에 대한 수학적인 수식을 도식화하여 표현하였다.

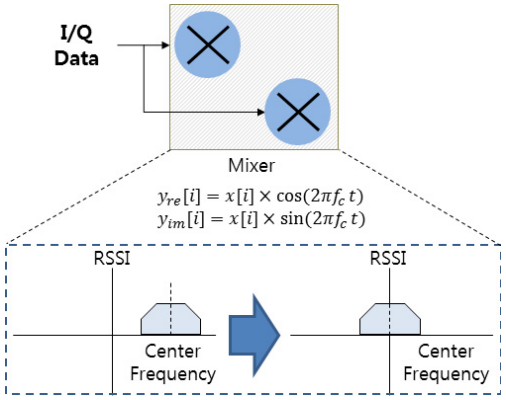


그림 2. Mixer 수학적 정의
Fig. 2. Function of Mixer

2.3 LPF

Mixer를 통해 이동한 데이터는 저역 주파수 성분만을 통과시키기 위한 방법인 LPF를 수행한다. 본 논문에서는 LPF 중 FIR필터[7]을 사용하였다. FIR 필터의 수학적인 정의는 식 (2)와 같다. 그림 3은 FIR필터에 대한 수학적인 수식을 도식화하여 표현하였다.

$$y[i] = \sum_{j=0}^{K} x[i] \times h[j-i] \tag{2}$$

여기서, $x[i]$는 입력 데이터, $h[j-i]$는 필터계수, K는 필터 차수이다[7]. 본 논문에서는 필터계수로 Hanning Window[8], Equiripple[9]를 각각 사용하였고, 필터 차수는 200 탭으로 설정하였다. 본 논문에서는 Mixer를 통과한 데이터의 LPF를 수행하기 위하여 LPF의 주요 연산인 다중 반복문 곱셈 연산을 CPU와 GPU로 각각 구현하였다. 여기에서 필터계수로 사용되는 Hanning Window와 Equiripple은 CPU로 구현하였다. 연산양이 많지 않거나 연산에 사용되는 데이터 표본 수가 적은 경우에는 CPU의 연산처리 속도가 GPU를 앞서기 때문이다[10].

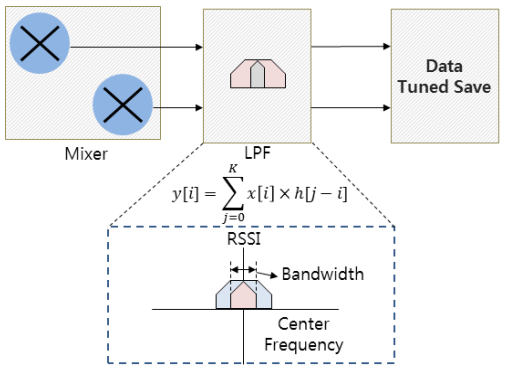


그림 3. FIR 필터의 수학적 정의
Fig. 3. Function of FIR Filter

2.4 GPGPU

다수의 프로세스를 보유한 시스템에서 알고리즘을 분리하여 처리하는 병렬처리 방식은 다양한 시스템에 사용되고 있다. 이미지 프로세싱 분야에서 렌더링 등의 특수 연산 목적으로 사용되던 GPU를 범용 연산에 활용하는 GPGPU가 제시되면서 GPGPU에 대한 연구가 지속되고 있다. 최근, GPU는 다중 연산처리에서 CPU를 뛰어넘는 연산 능력을 갖고 있으며, 딥러닝을 포함하여 다양한 산업 분야에서 연구가 활발히 진행되고 있다[11].

Ⅲ. 구현 및 성능평가

본 장에서는 앞서 설명한 광대역 I/Q 데이터의 Tuned Save 기능의 단계별 알고리즘에 대하여 성능평가를 통해 검증한다. 성능평가에 사용한 광대역 I/Q 데이터는 샘플률 26.25MHz의 톤 간격이 820kHz인 2FSK 신호가 주입된 30초 저장 데이터이다. 그림 4는 원 데이터의 스펙트럼을 표현하였다. 성능 평가 과정은 다음과 같다.

먼저, 2FSK 신호 중, 중심주파수 기준으로 +410kHz에 위치한 피크 신호를 -410kHz 만큼 Mixer를 통과한다. Mixer를 통과한 데이터는 LPF를 거치게 되는데, Mixer를 통과한 데이터의 중심주파수 기준으로 ±200kHz 대역폭의 신호만 Tuned Save 하는 기능을 구현하였다.

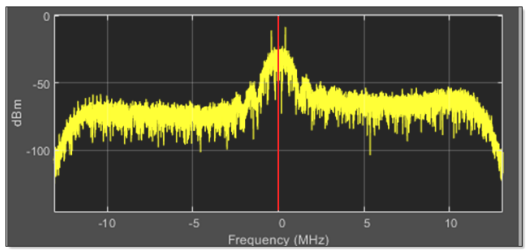


그림 4. Raw-Data Spectrum
Fig. 4. Raw-Data Spectrum

3.1 Mixer 구현

본 논문에서는 Mixer를 CPU와 GPU로 각각 구현하였다. 저장된 I/Q 데이터에서 피크신호의 주파수를 중심주파수로 이동하기 위해 주파수를 -410kHz 만큼

이동시킨다. 그림 5는 CPU를 이용하여 구현한 Mixer를 통과한 I/Q 데이터의 스펙트럼이다. 그림 6은 GPU를 이용하여 구현한 Mixer를 통과한 I/Q 데이터의 스펙트럼 결과이다.

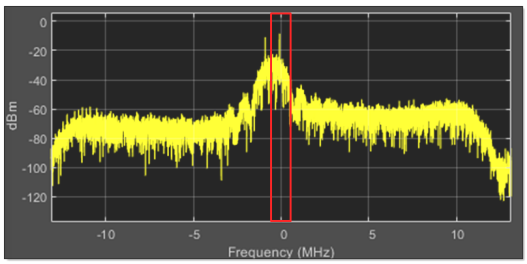


그림 5. After Mixer Using CPU
Fig. 5. After Mixer Using CPU

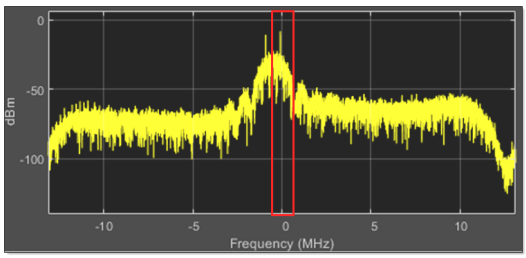


그림 6. After Mixer Using GPU
Fig. 6. After Mixer Using GPU

3.2 필터계수 구현

Mixer를 통과한 I/Q 데이터는 CPU와 GPU로 구현된 FIR 필터를 통과한다. 이때, 지정한 대역폭 만큼의 데이터를 획득하기 위해 필터 계수인 Hanning Window와 Equiripple을 사용하였다. 필터계수는 그림 7, 그림 8로 표현하였다. 그림에서 보는바와 같이, 필터 종류에 따라 원 신호의 특성을 더욱 잘보존할 수 있다. 필터 계수는 앞서 설명한 대로 CPU를 이용하여 구현하였다. 필터 계수 특성 상, 가변 필터의 경우는 계수가 변하기 때문에 지속적으로 연산해야 하나, 본 논문의 성능 평가의 경우는 필터 계수를 고정하였고,

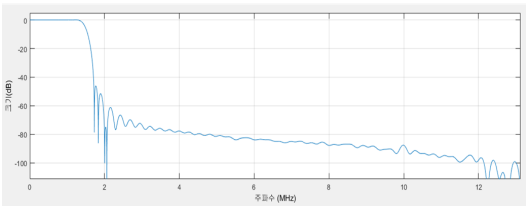


그림 7. Hanning Window
Fig. 7. Hanning Window

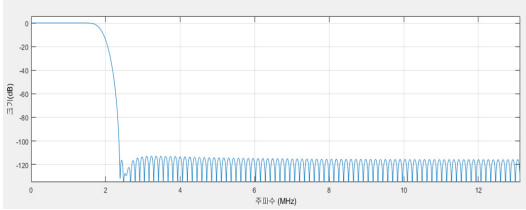


그림 8. Equiripple
Fig. 8. Equiripple

필터 계수를 고정 변수로 지정하여 연산에 적용하였다. 그 결과, 필터 계수를 고정한 상태로 두 필터를 계수로 사용하였을 경우, 연산양의 변화는 없었으며, 결과 데이터의 차이만 존재했다. 성능평가를 위해 두 필터를 비교하였으며, 원 신호 보존의 경우, Hanning Window보다 Equiripple의 성능이 우수함을 성능 평가를 통해 확인하였다. 다만, 필터 계수를 가변으로 할 경우, Equiripple의 연산[9]이 더 많기 때문에 필터 계수 연산에 소모되는 절대시각이 증가한다.

3.3 FIR CPU vs GPU

앞서 설명한 필터계수를 포함한 FIR 필터의 CPU와 GPU로 각각 구현하였다. 이때, 중심주파수 기준의 ±200kHz 대역폭의 신호만 Tuned Save 하는 기능을 구현하였다.

한편, Tuned Save 기능을 수행한 최종 I/Q 데이터

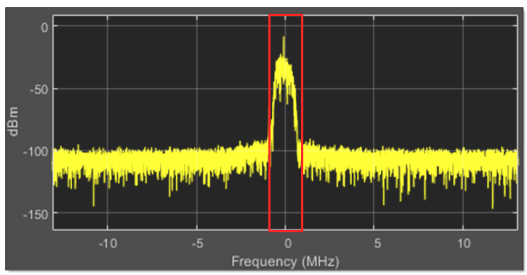


그림 9. After FIR Filter Using CPU
Fig. 9. After FIR Filter Using CPU

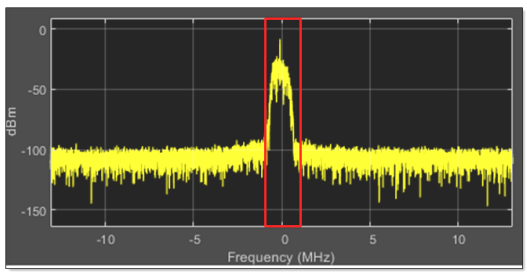


그림 10. After FIR Filter Using GPU
Fig. 10. After FIR Filter Using GPU

는 원 신호인 2FSK의 우측 신호를 기준으로 ±200kHz 대역폭의 신호만을 획득한 후, I/Q 파일로 저장하게 된다.

본 논문에서는 대역폭을 ±200kHz 로 고정하였지만, 가변 대역폭의 경우도 대응 가능하도록 알고리즘을 구현하였다.

3.4 성능평가

본 장에서는 앞서 제안 및 설명한 Tuned Save의 단계별 구현 기능에 대하여 성능 평가를 수행하였다. Tuned Save 기능을 확인하기 위한 입력 데이터는 앞서 설명한 대로 성능평가에 사용한 광대역 I/Q 데이터는 샘플률 26.25MHz의 톤 간격이 820kHz인 2FSK 신호가 주입된 30초 저장 데이터이다. 본 성능 평가에서는 30초 데이터를 분할하여 10초, 20초, 30초 각각 입력데이터로 사용하여 성능을 평가한다. 데이터를 분할하여 성능 평가를 수행하는 이유는 입력 데이터의 크기에 따른 연산 속도의 차이를 확인하기 위함이다. 성능평가에 사용된 하드웨어 사양은 표 1과 같다.

성능평가에 사용된 PC는 Windows 10 pro 64bit 이며, File I/O의 성능 향상을 위해 SSD를 사용하였다.

성능평가의 첫 부분은 Mixer이다. Mixer는 CPU와 GPU로 구현된 알고리즘을 수행한 후, 입력데이터 시각만큼의 연산 수행 시각을 측정하였다. 그 결과, 입력데이터가 10초인 경우, CPU 대비 GPU의 성능이 약 165배 향상됨을 확인하였다. 입력데이터가 30초인 경우도 CPU 대비 GPU의 성능이 약 163배 향상되었고, 본 논문에서 제안하는 기법이 우수한 성능을 제공함을 확인할 수 있다. 표 2는 CPU와 GPU로 구현된 Mixer의 성능 평가 결과이다.

한편, Mixer를 수행한 데이터는 다음 단계인 LPF를 수행하게 된다. 필터 계수의 선정에 따른 성능 분석을 위하여 Hanning window와 Equiripple을 각각 수행하여 성능평가 하였다. 본 논문에서는 필터계수를 가변으로 설정하지 않고 고정으로 두었으며, 최초 1회만 필터계수를 생성하였고, 반복되는 필터계수와의 연산은 특정 변수로 저장하여 연산 수행 회수를 감소시켰다. 그 결과, 입력데이터가 10초이고 필터계수가 Hanning window인 경우, CPU 대비 GPU의 성능이 약 106배 향상됨을 확인하였다. 마찬가지로, 입력데이터가 10초이고 필터계수 Equiripple인 경우, CPU 대비 GPU의 성능이 약 112배 향상되었고, 본 논문에서 제안하는 기법이 우수한 성능을 제공함을 확인할 수 있다.

표 1. 하드웨어 사양
Table 1. Specification of Hardware

	CPU	GPU
OS	Windows 10 64b	
SSD	SSD 1TB	
Model	Intel scalable 5122	Titan RTX
Clock(GHz)	3.6	1.7
Core	4	4608
RAM(GB)	64	24

표 2. CPU와 GPU로 구현한 Mixer 성능 평과 결과
Table 2. Evaluation Performance of Mixer between CPU and GPU

	Input Data (sec)	CPU	GPU	CPU 대비 성능(배)
Mixer	10	4.1970	0.0255	164.5882
	20	8.3868	0.0508	165.0945
	30	12.4439	0.0762	163.3058

표 3. CPU와 GPU로 구현한 LPF 성능 평가 결과
Table 3. Evaluation Performance of LPF between CPU and GPU

	Input Data (sec)	CPU	GPU	CPU 대비 성능(배)
LPF Hanning window	10	68.6325	0.6476	105.9798
	20	138.1833	1.1549	119.6496
	30	205.8389	1.6587	124.0965
LPF Equiripple	10	69.0420	0.6156	112.1540
	20	137.4896	1.1659	117.9257
	30	205.9767	1.7114	120.3557

표 3은 CPU와 GPU로 구현된 Hanning window, Equiripple을 필터계수로 갖는 LPF의 성능 평가 결과이다.

필터계수를 가변으로 설정하지 않고 고정된 계수로 사용하고, 내부 변수로 관리하며 반복되는 LPF 연산을 수행할 경우, 필터계수에 따른 연산 속도의 차이는 크지 않음을 본 논문의 성능평가에 의해 확인할 수 있다.

표 4는 본 논문에서 제안하는 GPU를 이용한 Tuned Save 기능의 성능평가 결과이다. 필터계수에 따라 각각 성능평가를 수행하였으며, 필터계수가 Hanning Window이고 입력데이터가 30초인 경우, CPU 대비 GPU의 성능이 약 125배, Equiripple인 경

표 4. Tuned Save 성능 평가 결과
Table 4. Evaluation Performance of Tuned Save

	Input Data (sec)	CPU	GPU	CPU 대비 성능(배)
Tuned Save Hanning window	10	72.8295	0.6731	108.2001
	20	146.5701	1.2057	121.5643
	30	218.2828	1.7349	125.8187
Tuned Save Equiripple	10	73.239	0.6411	114.2396
	20	145.8764	1.2167	119.8951
	30	218.4206	1.7876	122.1865

우, CPU 대비 GPU의 성능이 약 122배 향상되었다. GPU를 활용함으로써 연산 수행 속도가 크게 향상됨을 성능평가를 통해 확인하였다. 또한, 입력 데이터의 크기가 커질수록 CPU 대비 GPU의 성능이 향상됨을 확인하였다.

Ⅳ. 결 론

본 논문에서는 PC 기반의 GPGPU를 활용하여 DSSS신호가 저장된 광대역 I/Q 데이터의 Tuned Save 기능을 GPU를 활용하는 방안을 제안하였다. PC 기반의 소프트웨어로 구현하는 경우에 대하여 CPU와 GPU로 각각의 알고리즘을 구현하였고, 두 장치간의 성능을 비교하였다. 그 결과, GPU로 Tuned Save 기능을 구현한 경우, Tuned Save 결과는 같고, 연산 수행 속도가 크게 향상됨을 성능 평가 결과를 통해 확인하였다. 성능 평가를 통해 제안한 Tuned Save 기능은 원 신호에 대한 누락 없이 정확한 구현 결과를 제공함을 확인할 수 있었다. 따라서, 제안한 GPU를 이용한 Tuned Save 구현 기능은 신호 처리 분야 등, 반복 연산 수행이 많은 분야 등에 직접 활용 가능할 것으로 판단된다. 한편, 가변 대역폭, 가변 필터 계수 등, 고정되지 않은 상황에서 효율적으로 알고리즘을 수행할 수 있는 Tuned Save 구현 방안을 고려할 수 있다.

References

[1] E. Monmasson, L. Idkhajine, M. Cirstea, I. Bahri, A. Tisan, and M. Naouar, "FPGAs in industrial control applications," *IEEE Trans. Ind. Informat.*, vol. 7, no. 2, pp. 224-243, May 2011.

[2] P. S. B. Nascimento, H. E. P. de Souza, F. A. S. Neves, and L. R. Limongi, "FPGA implementation of the generalized delayed signal cancelationphase locked loop method for detecting harmonic sequence components in three-phase signals," *IEEE Trans. Ind. Electron.*, vol. 60, no. 2, pp. 645-658, Feb. 2013.

[3] D.-C. Kwon and B.-Y. Kang, "CPU and GPU performance analysis for convolution neural network," *JKIIT*, vol. 15, no. 8, pp. 11-18, 2017.

[4] R. Woods, et al., *FPGA-based implementation of signal processing systems*, John Wiley & Sons, 2008.

[5] J. Sanders and E. Kandrot, *CUDA by example - An introduction to generalpurpose GPU programming*, Addison-Wesley, 2011.

[6] H. C. Yoon, et al., "Circular buffer system design for efficient I/Q data store and analysis," in *Proc. KICS Winter Conf.*, pp. 227-228, 2019.

[7] B. Sklar, *Digital Communications*, 2nd Ed., 2004.

[8] P. Podder, T. Z. Khan, M. H. Khan, and M. Rahman, "Comparative performance analysis of hamming, hanning, and blackman window," *Int. J. Comput. Appl.*, vol. 96, no. 18, 2014.

[9] A. Enis CETIN, Omer N. GEREK, and Y. Yardimci, "Equiripple FIR filter design by the FFT algorithm," *IEEE Sign. Process. Mag.*, vol. 14, no. 2, pp. 60-64, 1997.

[10] S. Asano, T. Maruyama, and Y. Yamaguchi, "Performance comparison of FPGA, GPU and CPU in image processing," in *2009 IEEE, Int. Conf. Field Programmable Logic and Appl.*, pp. 126-131, 2009.

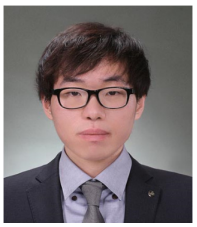
[11] D. Xie, L. Zhang, and L. Bai, "Deep learning in visual computing and signal processing," *Appl. Computat. Intell. and Soft Comput.*, vol. 2017, 2017.

윤 현 철 (Hyun-Chul Yoon)

2010년 2월 : 선문대학교 컴퓨터 공학과 학사 졸업
2012년 2월 : 한양대학교 전자컴퓨터통신공학과 석사 졸업
2012년 1월~현재 : LIG넥스원 전자전연구소 선임연구원

<관심분야> 전자전신호처리, 디지털통신, GPGPU
[ORCID:0000-0001-8325-5620]

이 현 휘 (Hyeon-Hwi Lee)



2013년 2월 : 한동대학교 전산전자공학과 학사 졸업
2013년 1월~현재 : LIG 넥스원 전자전연구소 선임 연구원

<관심분야> 병렬처리, 디지털 신호 처리
[ORCID:0000-0001-5315-0625]

강 현 진 (Hyun-Jin Kang)

2006년 2월 : 중앙대학교 전자공학과 학사 졸업
2008년 2월 : 중앙대학교 전자공학과 석사 졸업
2011년 1월~현재 : LIG넥스원 전자전연구소 선임연구원

<관심분야> 전자전신호처리, 디지털통신
[ORCID:0000-0003-1770-2634]

김 재 윤 (Jae-Yun Kim)

2002년 2월 : 한양대학교 전자공학과 학사 졸업
2004년 2월 : 한양대학교 전자전기제어 계측공학과 석사 졸업
2004년 1월~현재 : LIG넥스원 전자전연구소 수석연구원

<관심분야> 통신 및 전자전 신호처리
[ORCID:0000-0002-4692-8467]

문 병 호 (Byung-Ho Moon)

1997년 2월 : 대구대학교 전자공학과 학사 졸업
2019년 2월 : 한양대학교 통신정보공학과 석사 졸업
1997년 1월~현재 : LIG넥스원 전자전연구소 수석연구원

<관심분야> 통신 및 전자전 체계
[ORCID:0000-0003-1208-200X]

김 선 교 (Seon-Kyo Kim)

2010년 8월 : 연세대학교 컴퓨터공학과(공학사)
2013년 2월 : 연세대학교 컴퓨터과학과(공학석사)
2013년 12월~현재 : 국방과학연구소 선임연구원

<관심분야> ES 신호분석 SW
[ORCID:0000-0001-8714-2404]