논문 20-45-03-12 The Journal of Korean Institute of Communications and Information Sciences '20-03 Vol.45 No.03
https://doi.org/10.7840/kics.2020.45.3.546

DSSS 신호 스펙트럼 데이터의 2D 랜더링을 위한 GPU 가속 컴퓨팅 성능 분석

우 성 원[◆°], 정 진 아[*], 강 현 석[*], 윤 현 철[**], 박 철 순[***]

GPU-Accelerated Computing Performance Analysis for 2D Rendering of DSSS Signal Spectrum Data

Woo Seong Won[◆°], Jeong Jin Ah[*], Kang Hyeun Seok[*], Yoon Hyun Chul[**], Park Cheol Sun[***]

요 약

본 논문에서는 DSSS 신호의 스펙트럼 데이터 생성 및 2D 렌더링에 대한 GPU가속 컴퓨팅 성능을 분석한다. 모든 과정에서 프로세스 단위로 CPU 직렬 컴퓨팅 및 GPU가속 컴퓨팅 방식에 대해 각각 설계하고 모듈을 구현하였다. 이를 2개의 장치에서 처리 샘플 수에 따른 성능을 비교, 분석한다. 성능 분석 결과 장치 성능 및 처리 샘플 양을 종합적으로 고려할 때 고성능의 GPU 환경과 대용량의 데이터 입력에서 GPU가속 컴퓨팅 모델을 사용한 데이터 처리가 가장 바람직한 것으로 나타났다.

Key Words : GPU-accelerated computing, GPGPU, parallel computing, hardware-accelerated rendering, OpenGL

ABSTRACT

In this paper, we analyze the GPU-accelerated computing performance for spectral data generation and 2D rendering of DSSS signals. All processes were designed and implemented for CPU serial computing and GPU-accelerated computing in process units. It compares and analyzes the performance according to the number of processed samples in two devices. Performance analysis shows that data processing using the GPU-accelerated computing model is most desirable for high-performance GPU environments and large data inputs when considering device performance and processing sample volume.

Ⅰ. 서 론

일반적으로 DSSS(Direct Sequence Spread Spectrum)전송방식은 변조기(modulation) 다음 단에서 확산대역 파형(Spreading Waveform)을 곱하여 원신호의 대역폭을 확장으로써 간섭이나 재밍에 강인하도록 한다.[1] 확장된 대역폭을 가지는 DSSS신호는 대용량의 데이터를 가지게 되며 PC 기반의 통신신호분석 시스템[9]의 구성에 있어 신호분석을 위한 스펙트럼 데이터 생성 및 2D렌더링을 통한 시각 자료의 생성

※ 본 연구는 국방과학연구소 광대역저피탐 DSSS 신호탐지 및 분석시험장치대 과제에 의해 수행되었습니다.
◆° First and Corresponding Author : People-i, swwoo@people-i.co.kr, 정회원
* People-i, jajeong@people-i.co.kr; hskang@people-i.co.kr, 정회원
** LIG Nex1, hyunchul.yoon@lignex1.com
*** Agency for Defense Development, helione@nate.com, 정회원
논문번호 : 201911-274-0-SE, Received October 31, 2019; Revised January 22, 2020; Accepted February 20, 2020

과정에서 CPU를 사용한 직렬 컴퓨팅으로는 분석 결과를 얻기까지 많은 시간이 소요된다. 기존에 신호 분석 컴퓨팅에서는 형변환을 수행하는 부분과 FFT 알고리즘의 연산, 그리고 Log Scale로 변환하는 연산 과정과 GDI를 사용한 2D 렌더링 수행 과정에서 데이터 처리에 많은 시간이 소요되며 실시간 신호 분석을 수행하기 위해서 위 과정에 소요되는 시간을 줄일 필요성을 확인하였다.

이를 개선하기 위해 NVIDIA에서 제공하는 CUDA 라이브러리를 사용한 GPU가속과 GPU가속을 지원하는 2D렌더링 라이브러리인 OpenGL를 사용하여 고속 연산 및 이미지 렌더링을 통해 성능을 개선하는 방안을 고려하였다. 따라서 본 논문에서는 연산 및 렌더링 과정을 단위 프로세로 나누어 처리 시간을 측정 할 수 있도록 구성하였으며 각 프로세스는 CPU 직렬 컴퓨팅 및 GPU가속 컴퓨팅 방식을 선택적으로 수행할 수 있도록 설계하였다. 이를 2개의 장치에서 처리 샘플 수를 달리하여 시험하는 것으로 단위 구간 및 전체 구간에 대한 성능을 비교하였다.

본 논문의 구성은 다음과 같다. Ⅱ장에서는 GPU를 사용한 가속컴퓨팅에 대한 특성에 대해 기술한다. Ⅲ장에서는 GPU 가속컴퓨팅을 사용한 성능 측정 실험 구성을 설명하고 Ⅳ장에서는 결론 및 향후 연구 방향에 대해 기술한다.

Ⅱ. GPU를 사용한 가속 컴퓨팅

2.1 CUDA API

CPU 코어는 복잡한 제어 로직과 순차적 프로그램 실행에 최적화 되어있으며, GPU 코어는 병렬 프로그램의 처리량에 중점을 둔 간단한 제어 로직을 가지며 병렬 작업에 최적화 되어 있다.[2] 이러한 하드웨어적 특성을 가진 GPU 코어를 사용하여 CPU에서 전통적으로 관리했던 응용 프로그램들의 계산을 수행하는 기술을 그래픽 처리 장치를 통한 일반 목적의 컴퓨팅이라 하여 GPGPU (General-Purpose computing on Graphics Processing Units)라고 부르고 있다.[3] CUDA는 NVIDIA GPU의 병렬 컴퓨팅 엔진을 활용하여 여러 복잡한 계산 문제를 보다 효율적으로 해결하는 범용 병렬 컴퓨팅 플랫폼으로서 NVIDIA 아키텍처를 사용한 GPGPU 프로그래밍 모델이다. CUDA를 사용하면 CPU에서 전통적으로 수행 된 것처럼 계산을 위해 GPU에 액세스 할 수 있다. CUDA를 통한 병렬 컴퓨팅에서 GPU는 독립형 플랫폼이 아닌 CPU의 보조 프로세서로서 그림 1과 같이 PCI-Express 버스를 통해 CPU 기반의 Host와 GPU 기반의 Device가 함께 구성되어 작동된다.[2]

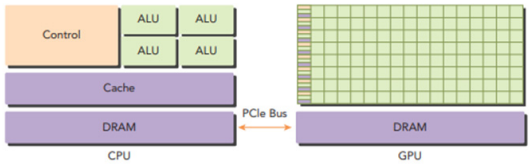


그림 1. Host-Device 구조
Fig. 1. Host-Device Configuration

GPU에서 데이터 연산을 위해서는 GPU메모리의 할당과 CPU 메모리에 있는 데이터를 GPU 메모리로의 복사가 수행되어야 하며 연산 이후 GPU에 할당된 메모리를 삭제하는 과정이 필요하다. 해당 과정에서 메모리 오버헤드에 따른 성능 손실이 발생하기 때문에 플랫폼 사용에 따른 성능 손실을 최소화하기 위해 본 논문에서는 2D 렌더링을 사용한 시각 데이터의 좌표 계산과 신호 분석 처리를 위한 FFT(Fast Fourier Transform), Data Type Casting, 신호 분석 결과의 Log Scale 변환과 같은 대용량 데이터 처리가 필요한 부분에서 병렬 컴퓨팅 기법을 활용한 연산을 수행할 수 있도록 하여 성능 손실을 최소화한 로직을 구성한다.

2.2 OpenGL API

신호 분석 결과를 확인하는 여러 기법 중에서 스펙트럼 분석은 시각 자료를 통해 시퀀싱 된 데이터를 특성화하고 분석하는 통계 기법이다.[4] 스펙트럼 분석 결과를 PC 기반의 응용 프로그램에서 출력 할 때, 2D 렌더링 기반의 Chart를 구성함에 있어 기존의 GDI API를 사용한 렌더링 방식은 하드웨어 가속을 사용하지 않고 CPU를 사용해 모든 렌더링을 수행하여 속도 향상에 제한이 있는 것이 단점이었다.[5]

본 논문에서는 렌더링에 하드웨어 가속을 지원하는 OpenGL API를 사용하였다. OpenGL은 렌더링을 위한 속성 정보가 Buffer를 통해 GPU의 그래픽 파이프라인에서 실행되는 특징을 가지는 출력 지향적인 렌더링 API로 hardware-accelerated rendering을 통한 고속 렌더링이 가능하다는 장점이 있다.[6]

Ⅲ. 성능 측정 실험 구성

3.1 알고리즘 설계

CPU를 사용한 순차적인 프로그래밍 구성과 GPU 병렬 컴퓨팅 플랫폼을 사용하여 가속 효과를 얻을 수 있도록 구획화를 진행한 방식의 두 가지 구성을 선택

적으로 적용하여 성능 비교가 가능하도록 시험 소프트웨어를 설계하였다. NVIDIA GPU 계산을 위한 CUDA 환경은 할당된 여러 스레드가 동시에 동일한 명령을 병렬로 처리함으로서 데이터 처리를 빠르게 할 수 있는 특징이 있으며, 또한 입력한 명령이 동시에 처리되기 때문에 전처리 과정이 필요한 데이터는 병렬 연산에 부적합하다는 단점이 있다.

즉, 데이터가 재귀적으로 호출되지 않으며 반복적인 데이터 처리과정이 수행되는 부분을 CUDA API을 사용하여 성능을 개선할 수 있다는 가정으로부터 CPU 연산 프로세스를 재구성하였다.

신호 분석 컴퓨팅에서 CUDA API을 사용한 병렬

```
Main(){
#Step 1 : Signal Data Sampling
          SigData = GetSigSampleData();
          //DSSS 신호 데이터 샘플 생성
#Step 2 : Ready CUDA Data
          SetDataForCUDA();
          //GPU에 데이터 전달
#Step 3: Convert I/Q Data
          KernelConvertIQData<<<>>>();
          //Idata와 Qdata로 나누어진
          Sample 을 하나의 Complex Type
          으로 병합
#Step 4 : Execute cuFFT
          cufftExecC2C();
          //CUDA Library의 cuFFT를
          사용하여 계산.
#Step 5 : Swap Result Data
          KernalSwapData<<<>>>();
          //cuFFT 계산 결과값의 중간을
          기준으로 전/후 데이터의 위치를
          변경하고 I데이터와 Q데이터로
          분리
#Step 6 : Convert Result Data to Log Scale
          KernelLogScale<<<>>>();
          //분리한 데이터를 LogScale 또는
          Linear Scale로 변환
#Step 7 : Transfer Rendering Data
          RenderData = SetCalcData();
          //GPU에서 CPU로 데이터 복사
}
```

그림 2. 신호분석 cuFFT 알고리즘에 대한 의사 코드
Fig. 2. pseudocode for Signal Analysis cuFFT Algorithm

컴퓨팅을 적용할 수 있는 구간에는 FFT 연산을 위해 필요한 형변환(Data Type Casting)과 I/Q Data Format으로 재구성하는 전처리 과정, FFT 연산을 CUDA Library의 cuFFT로 변환하는 과정, 그리고 FFT 연산 결과를 Log Scale로 변환하는 과정으로 나누었다.

또한 분석 과정 외에도 대용량의 데이터를 사용하여 연산 처리가 수행되는 렌더링 좌표 계산에도 CUDA API을 사용하여 병렬 컴퓨팅 처리를 수행하는 알고리즘을 추가하였다. 그림2는 DSSS 신호 데이터의 샘플로부터 FFT 계산과 Scale 변환을 수행하는 알고리즘에 대한 의사 코드로서 CPU와 CPU에서 각각 독립적인 형태로 구현하여 테스트 및 수행시간 측정이 용이하도록 하였다.

CUDA Library는 FFT 계산과 관련된 API를 제공하고 있으며 cuFFT 연산에 필요한 cufftComplex Data Type 형식으로 변환하는 함수에서 I/Q Data를

```
FUNCTIONS KernelLogScale
(IData, QData, DataLength, ScaleType)

Input tid, epsil, ICheck, QChek,
 DataLength, BlockDim, BlockId,
 ThreadId, GridDim

tid ← BlockDim * BlockId + ThreadId

WHILE tid < DataLength
  IF ScaleType = LogScale THEN
    ICheck ← AbsoluteValue(IData)
    QCheck ← AbsoluteValue(QData)
    IF ICheck<epsil and
    QCheck<epsil THEN
      IData ← 0
    ELSE
      IData ← 10 * Log10(IData^2 + QData^2)
    ENDIF
  ELSE
    IData ← SquareRoot(IData^2 + QData^2)
  ENDIF
  tid ← BlockDim * GridDim
ENDWHILE
```

그림 3. I/Q Data Scaling의 GPU 데이터 처리 슈도코드
Fig. 3. GPU data processing pseudocode in I / Q Data Scaling

하나의 복소 형식(Complex Type) Data로 변환한다.

이후 cufftExecC2C()와 이후 Kernel 함수에서 GPU에 생성된 데이터를 다수의 스레드를 한꺼번에 처리하며, SetCalcData()함수를 통해 GPU에서 CPU로 데이터를 보내는 작업을 거쳐 Chart 형식의 시각자료로 데이터를 출력하게 된다. 이 과정에서 GPU-CPU 간 메모리 할당/복사에 대한 시간이 추가로 소요되며 알고리즘의 처리 시간은 GPU로의 메모리 할당 및 복사시간을 더하여 구현하였다.

그림 3의 슈도코드를 보면 스레드와 블록을 참고하여 GPU에서 데이터를 할당하여 처리하도록 설계되었다. GPU에서 데이터 연산 처리 이벤트가 발생하는 과정으로는 KernelConvertIQToComplex(), KernelSwapData(), KernelLogScale(), cufftExecC2C()가 있다. DSSS 신호데이터로부터 읽어 들이는 샘플의 수가 적으면 단일 블록에서 샘플 수만큼 스레드를 사용하나 샘플 수가 많아질수록 여러 블록을 할당하여 블록의 스레드를 전부 사용하여 데이터를 처리한다. 블록 당 스레드 수와 블록의 수는 사용되는 GPU에 따라 차이가 있다.

3.2 시험 소프트웨어 및 장치 구성

본 논문에서는 대용량 신호 데이터로부터 사용자의 임의에 따라 데이터 샘플을 구성하고 분석 결과를 스펙트럼 형식의 시각자료를 통해 확인한다는 가정을 기반으로 분석 시험 소프트웨어를 구성하였다. 성능 평가에 사용될 시험 소프트웨어를 그림 4와 같이 구성하였으며 연산 과정별로 구간을 나누어 신호 분석 알고리즘 처리 과정에서 발생한 소요 시간을 전시하여 확인할 수 있도록 하였다.

신호 분석 모델은 I/Q 형변환 및 FFT, LogScale 연산 과정을 수식 기반의 인스턴스(Instance) 구조로 작성된 CPU 직렬 컴퓨팅 연산 방식과 동일 연산 과정을 커널 함수와 CUDA Library의 cuFFT 연산을 cudaStream으로 최적화한 구조로 작성된 GPU 병렬 컴퓨팅 연산 방식을 선택적으로 적용하여 실행시간을 확인 할 수 있도록 하였다.

신호 분석 결과를 나타내는 스펙트럼 형식의 시각자료는 2D렌더링을 통한 차트 전시 형식으로 하였으며 렌더링 API는 GUI를 사용한 렌더링 방식과 OpenGL을 사용한 렌더링 방식을 선택적으로 적용하여 확인 할 수 있도록 하였다.

테스트 환경은 그림 5과 같이 2개의 장치를 구성하여 진행하였다. 단위 샘플 수를 변경하며 4회씩 분석을 수행하여 소요 시간의 평균값을 기준으로 측정하였다. 신호 분석에 사용된 데이터는 BPS 30초 데이터로, short형 데이터 타입을 가지는 787,500,000개의 샘플로 이루어져 있으며 샘플 데이터를 구성할 때 2^n로 나누어 사용하였다.

본 논문에서는 전체 데이터에서 샘플로 추출되는 데이터 수를 213 ~ 220로 제한하였으며 동일한 하드웨어 조건에서 직렬 컴퓨팅과 병렬 컴퓨팅의 신호 분석 시간과 동일 컴퓨팅 조건에서 Device 1과 Device 2의 하드웨어의 신호 분석 시간을 비교하여 성능 분석을 진행하였다.

	Device 1(노트북)	Device 2(데스크톱)
CPU	Inter® Core i7-7700HQ 2.80GHz	Inter® Core i3-7100 3.90GHz
RAM	8.00GB	16.00GB
GPU	GeForce GTX 950M	GeForce GTX 1060
OS	Windows 10 Home	Windows 10 Home

그림 5. 시험 소프트웨어 운용 하드웨어 구성
Fig. 5. Test software operation hardware configuration

3.3 성능 평가

그림 6은 신호 분석에 사용되는 알고리즘인 FFT 연산을 CUDA 라이브러리 함수인 cuFFT를 사용하여 GPU를 통한 하드웨어 가속과 기존의 분석 방식을 비교한 성능 평가 결과이다. 그림 7는 그림 6의 cuFFT가 사용된 프로세스에 I/O 데이터 타입과 LogScale 변환식을 병렬화하여 GPU를 사용한 하드웨어 가속

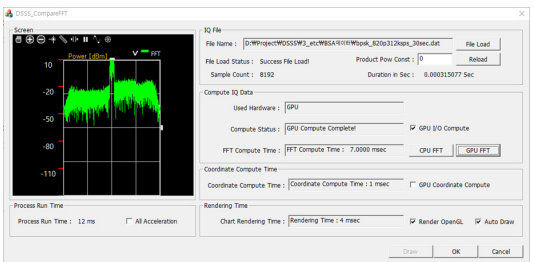


그림 4. 성능 비교를 위한 시험 소프트웨어
Fig. 4. Test software for performance comparison

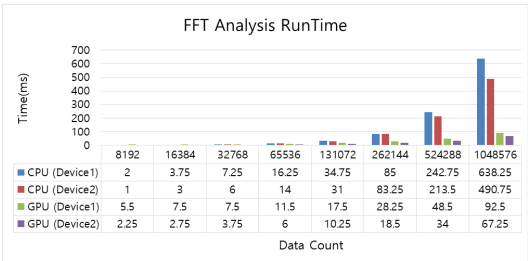


그림 6. FFT Analysis Runtime Check
Fig. 6. FFT Analysis Runtime Check

분석에 대한 성능 평가 결과이다. 두 결과는 모두 데이터 전처리 구간부터 Log Scale 변환까지 처리 시간을 기준으로 측정하였다. 그림 6에서는 연산에 사용되는 샘플 수가 증가함에 따라 cuFFT를 통한 가속 효과를 비교하였을 때, 하드웨어 가속을 사용함으로서 Device 1은 최대 6.9배, Device 2는 최대 7.2배의 가속 효과를 얻었으며, 그림 6에서 전처리 및 결과데이터 변환의 가속 효과는 Device 1은 최대 17.8배, Device 2는 최대 33.8배의 가속 효과를 얻을 수 있었다.

한편, 그림 8과 그림 9는 GDI와 OpenGL을 사용한 스펙트럼 렌더링 처리 방식을 비교한 성능 평가 결과이다. 렌더링을 수행하기 위해 OpenGL의 좌표계로 결과 데이터를 변환하는 구간에서는 Device 1은 최대 1.2배, Device 2은 최대 2.5배의 가속 효과를 얻을 수

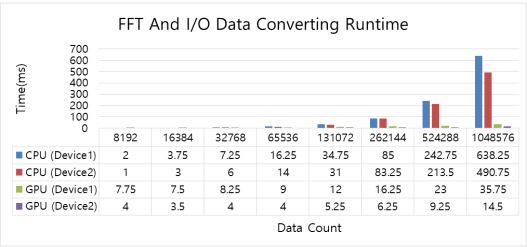


그림 7. FFT And I/O Data Converting Runtime Check
Fig. 7. FFT And I/O Data Converting Runtime Check

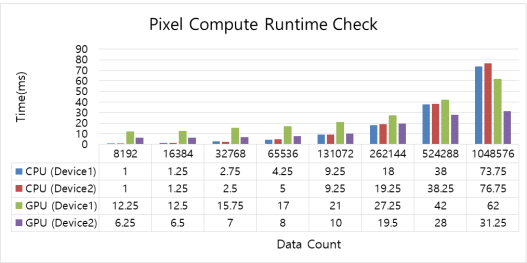


그림 8. Pixel Compute Runtime Check
Fig. 8. Pixel Compute Runtime Check

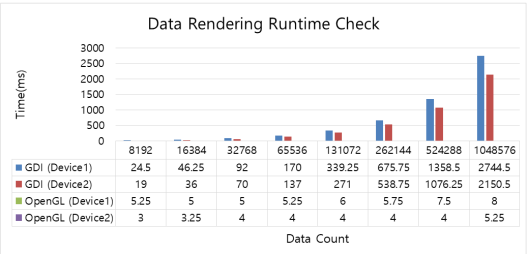


그림 9. Data Rendering Runtime Check
Fig. 9. Data Rendering Runtime Check

있었으나, 적은 수의 샘플로 연산을 수행하였을 경우에는 CPU를 사용한 순차적 프로그래밍의 연산 시간이 Device 1은 최대 12배, Device 2은 최대 6배 더 적게 걸렸다. 이는 샘플로 읽은 신호 데이터가 신호 분석을 거치면서 확장된 대역폭이 제거되고 좌표 변환을 수행하는 알고리즘이 단순 계산 위주로 구성되어 있기 때문에 가속 효과가 미비한 것을 확인되었다.

렌더링 처리 구간에서는 GDI를 사용했을 때보다 OpenGL을 사용하여 렌더링을 수행하였을 때, 샘플 수가 증가함에 따라 Device 1은 최대 340배, Device 2는 최대 410배의 처리 속도를 얻을 수 있었다.

전체적으로 CPU를 사용한 직렬 컴퓨팅보다 GPU를 사용한 병렬 컴퓨팅이 더 빠른 처리 속도를 얻을 수 있었으나, 신호 분석 과정의 FFT 연산에서 65536개의 샘플 수보다 적은 수를 GPU를 사용한 하드웨어 가속을 이용하여 연산을 수행하였을 때에는 CPU를 사용한 순차적 프로그래밍에 비해 낮은 연산 속도를 얻을 수 있었는데, 전체 프로세스 과정에서 CPU와 GPU 사이 메모리를 할당 및 복사하는 과정에서 발생하는 지연시간에서 얻어지는 결과임을 확인하였다.

Ⅳ. 결론 및 향후 연구 방향

본 논문에서는 GPGPU를 활용하여 대용량 DSSS 신호 스펙트럼 분석을 고속으로 수행하는 가속 컴퓨팅 기법을 제안하였다. 시험 소프트웨어에서 CPU와 GPU를 선택적으로 적용하는 것으로 두 장치 간 성능을 비교하였으며 대용량 신호 분석 수행 시 처리 속도가 크게 향상됨을 성능 평가 결과를 통해 확인하였다. 그러나 적은 수의 데이터로 병렬 컴퓨팅을 구성하는 경우 GPU 메모리를 할당하고 CPU와 GPU간 메모리를 복사하는 과정에서 발생하는 손실이 발생한다는 단점이 크게 작용하고 병렬화로 얻을 수 있는 성능 개선효과의 적용이 미비하여 CPU의 순차적인 연산 처리가 더 빠르다는 것을 확인하였다.

좌표를 변환하는 과정과 FFT 및 I/O데이터의 변환 과정은 동일한 반복 횟수를 가지며 이를 병렬처리 하였을 때 동일한 성능 개선효과를 예상하였으나, 실제 결과 데이터를 보았을 때 좌표를 변환하는 과정은 FFT를 수행하는 과정에 비해 병렬화로 얻을 수 있는 성능적인 이점이 낮게 측정되었다. 이러한 시험 결과와 기존의 CPU 직렬 컴퓨팅 방식을 비교하였을 때, GPU를 사용한 병렬 컴퓨팅으로 얻는 개선 효과는 미비하며 또한 신호분석을 위해 CPU-GPU 사이의 메모리복사가 이루어 진 후 추가적으로 OpenGL API를

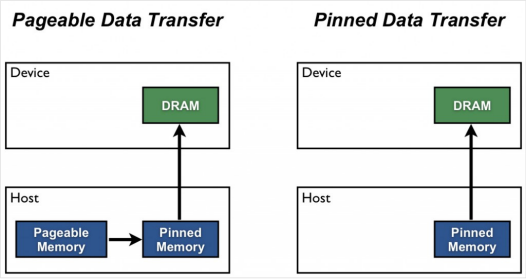


그림 10. Pageale Data 와 Pinned Data 비교[7]
Fig. 10. Pageale Data vs Pinned Data[7]

사용한 렌더링에서 GPU와 CPU 사이 메모리 복사가 한번 더 이루어지면서 전체 프로세스 수행 시간이 지연됨을 확인하였다.

따라서, 제안한 GPU 병렬 컴퓨팅을 적용한 DSSS 신호 스펙트럼 분석 기법을 활용하면 기존의 CPU 직렬 컴퓨팅을 사용한 신호분석을 수행할 때보다 분석에 소요되는 시간을 단축함으로서 실시간 신호 분석에 직접 활용 가능할 것으로 판단된다. 또한 CPU-GPU간 데이터 I/O에서 소요되는 시간을 줄여 추가적인 성능 향상을 얻는 방안을 고려할 수 있다.

관련 연구 방안으로 CPU에서 대용량의 데이터를 반복적으로 처리하는 대부분의 연산과정을 GPU로 변환할 때 직접 Kernel 소스를 구성하는 방식을 사용하였으며 CUDA Library에서 제공하는 여러 API 중 cuFFT만을 사용한 가속 컴퓨팅을 수행하였다. 그러나 CUDA Library에는 GPGPU를 활용한 연산 과정에서의 가속 지원 외에도 CPU-CPU간 메모리 복사 과정에서 가속화 기능을 제공하고 있다. 그림 10과 같이 기존의 Pageable 데이터 방식이 아닌 Pinned 데이터 방식을 사용하여 메모리 복사 과정을 간소화하는 데이터 이동 관련 가속 기법이 있으며 Stream 옵션을 사용하여 데이터의 복사와 연산을 병행 처리하는 가속화 기법이 있다.[7,8]

이 두 기법을 통해 PCIe bus의 하드웨어적 구조에서 발생하는 손실을 보완할 수 있을 것으로 판단되며 CUDA Library에서 제공하는 API를 활용하는 것으로 추가적으로 성능을 개선하는 향후 연구가 가능하다. 또한, 분석 과정 외에 렌더링을 수행하는 과정에서 GPGPU를 사용한 좌표 계산을 위해 CPU-GPU간 메모리 복사가 진행되는 과정에서 발생하는 손실의 개선이 필요함을 확인하였다. 대안으로 OpenGL에서 제공하는 GPGPU와 비슷한 연산 처리 수행을 위한 GLSL(GL Shader Language)의 Compute Shader를 사용하여 좌표를 계산함으로서 렌더링 속도를 올릴 수 있을 것으로 사료되며 해당 기술을 적용한 성능 개선을 목표로 추가적인 연구가 가능하다.

References

[1] J. Kim, H. Yu, D. Kwon, and U. Park, “Performance of a direct sequence spread spectrum (DS-SS) data link system according to jamming types,” in *Proc. KSAS Fall Conf.*, pp. 857-860, Nov. 2014.

[2] J. Cheng, M. Grossman, and Ty McKercher, *Professional CUDA C Programming*, pp. 8-26, John Wiley & Sons, Inc. Indianapolis, Indiana, 2014.

[3] Y. Choi, “GPGPU based real time collision detection,” *IEIE Mag.*, vol. 36, no. 5, pp. 68-78, May 2009.

[4] J. N. Rayner, *Spectral Analysis*(2001), Retrieved Oct. 18, 2019, from https://www.sciencedirect.com/topics/biochemistry-genetics-and-molecular-biology/spectral-analysis

[5] J. R. Miller, *Shader-Based OpenGL Architecture* (2019), Retrieved Oct. 17, 2019, from https://people.eecs.ku.edu/~jrmiller/Courses/OpenGL/Architecture.html

[6] J. N. Jacobson, *Assessing OpenGL for 2D rendering of geospatial data*(2014), Retrieved Oct. 17, 2019, from https://pdfs.semanticscholar.org/7b47/a61583f93f26099909e2589314e965723878.pdf.

[7] M. Harris, *How to Optimize Data Transfers in CUDA C/C++*(2012), Retrieved Oct. 17, 2019, from https://devblogs.nvidia.com/how-optimize-data-transfers-cuda-cc/

[8] M. Harris, *GPU Pro Tip: CUDA 7 Streams Simplify Concurrency*(2015), Retrieved Oct. 17, 2019, from https://devblogs.nvidia.com/gpu-pro-tip-cuda-7-streams-simplify-concurrency/

[9] 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.

우 성 원 (Woo Seong Woo)

2005년 2월 : 충청대학 SW 공학과 전문학사 졸업
2015년 5월~현재 : 피플아이 연구소 차장
<관심분야> 전자전신호처리, 통신응용, GPGPU
[ORCID:0000-0002-2920-1864]

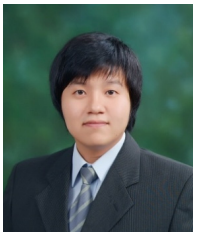
정 진 아 (Jeong Jin Ah)

2010년 2월 : 충남대학교 컴퓨터 공학과 학사 졸업
2017년 5월~현재 : 피플아이 연구소 과장
<관심분야> 전자전신호처리, 통신응용, GPGPU
[ORCID:0000-0002-7318-2939]

강 현 석 (Kang Hyeun Seok)

2017년 8월 : 충남대학교 전기공학과 학사 졸업
2017년 8월~현재 : 피플아이 연구소 대리
<관심분야> 딥러닝, 디지털통신, GPGPU
[ORCID:0000-0002-2012-9624]

윤 현 철 (Yoon Hyun Chul)



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]

박 철 순 (Park Cheol Sun)

1989년 2월 : 경기대학교 전자계산학과 학사
1991년 2월 : 인하대학교 전자계산공학과 석사
1997년 5월 : 전자계산조직응용기술사
2007년 2월 : 충남대학교 정보통신공학과 박사
1991년 2월~현재 : 국방과학연구소 책임연구원
<관심분야> 신호처리, 통신응용
[ORCID:0000-0002-3678-4041]