Journal of the Korean Association of Geographic Information Studies, 19(3) 2016, pp. 29-42 ISSN 1226-9719(Print)
http://dx.doi.org/10.11108/kagis.2016.19.3.029 ISSN 2287-6952(Online)

CUDA 라이브러리를 이용한 위성영상 병렬처리 : NDVI 연산을 중심으로

이강훈[1] · 조명희[1] · 이원희[1※]

Parallel Processing of Satellite Images using CUDA Library: Focused on NDVI Calculation

Kang-Hun LEE[1] · Myung-Hee JO[1] · Won-Hee LEE[1※]

요 약

원격탐사는 넓은 지역을 직접 접촉하지 않고 정보를 취득할 수 있고 다양한 분야에 적용할 수 있음으로써 급속히 발전하게 되었다. 이에 따라 위성의 제원 또한 원격탐사의 발전과 함께 급속한 발전을 이루게 되었다. 이러한 이유로 여러 분야에서 활용에 관한 연구가 활발히 이루어지고 있다. 현재 활용에 관한 연구는 활발히 이루어지고 있지만, 자료처리에 관련된 연구가 부족한 실정이다. 예전보다 인공위성의 제원이 발전하면서 많은 양의 정보 획득이 가능해진 것과 동시에 데이터 크기 또한 매우 커졌다. 이는 과거에 비해 자료의 처리속도가 저하된다는 단점이 존재한다. 따라서 본 논문에서는 병렬 처리의 한 가지 기법인 NVIDIA에서 제공하고 있는 CUDA (Compute Unified Device Architecture) 라이브러리를 활용하여 위성영상 자료처리 성능의 최적화를 목적으로 하고 있다. 본 연구의 순서는 다음과 같다. 다목적실용위성(Korea Multi-Purpose Satellite, KOMPSAT)의 영상을 크기를 기준으로 5가지 Type으로 나눈다. 이렇게 나누어진 영상을 원격탐사 분야의 한 가지 방법인 NDVI (Normalized Difference Vegetation Index)로 구현한다. 이때 CPU (Central Processing Unit, 중앙처리장치) 기반과 GPU (Graphic Processing Unit, 그래픽처리장치) 기반의 두 가지 방법과 상용 소프트웨어인 ArcMap을 이용하여 NDVI를 구현한다. 그리고 동일한 영상 유무를 판단하기 위해 구현된 결과 영상들을 히스토그램과 시각적으로 비교하고 CPU 버전과 GPU 버전의 처리속도를 비교 분석하였다. 연구결과 CPU 버전과 GPU 버전의 결과 영상은 ArcMap으로 구현한 영상과 시각적 그리고 히스토그램 비교를 통해 같은 결과를 나타내어 NDVI 코드는 올바르게 구현되었으며, 처리속도는 CPU보다 GPU가 약 5배 정도 빠른 것으로 확인하였다. 본 연구에서 병렬 처리의 한 기법인 CUDA 라이브러리를 활용하여 위성영상 자료처리 성능을 향상시킬 수 있었으며, 향후 NDVI와 같은 단순한 픽셀 연산 이외에도 다양한 원격탐사 기법의 적용이 필요할 것으로 사료된다.

주요어 : CUDA 라이브러리, 정규화식생지수, 병렬 처리, 그래픽처리장치, 중앙처리장치

2016년 5월 17일 접수 Received on May 17, 2016 / 2016년 6월 21일 수정 Revised on June 21, 2016 / 2016년 7월 12일 심사완료 Accepted on July 12, 2016

1 경북대학교 융복합시스템공학부 School of Convergence & Fusion System Engineering, Kyungpook National University
※ Corresponding Author E-mail : wlee33@knu.ac.kr

ABSTRACT

Remote sensing allows acquisition of information across a large area without contacting objects, and has thus been rapidly developed by application to different areas. Thus, with the development of remote sensing, satellites are able to rapidly advance in terms of their image resolution. As a result, satellites that use remote sensing have been applied to conduct research across many areas of the world. However, while research on remote sensing is being implemented across various areas, research on data processing is presently insufficient; that is, as satellite resources are further developed, data processing continues to lag behind. Accordingly, this paper discusses plans to maximize the performance of satellite image processing by utilizing the CUDA(Compute Unified Device Architecture) Library of NVIDIA, a parallel processing technique. The discussion in this paper proceeds as follows. First, standard KOMPSAT(Korea Multi-Purpose Satellite) images of various sizes are subdivided into five types. NDVI(Normalized Difference Vegetation Index) is implemented to the subdivided images. Next, ArcMap and the two techniques, each based on CPU or GPU, are used to implement NDVI. The histograms of each image are then compared after each implementation to analyze the different processing speeds when using CPU and GPU. The results indicate that both the CPU version and GPU version images are equal with the ArcMap images, and after the histogram comparison, the NDVI code was correctly implemented. In terms of the processing speed, GPU showed 5 times faster results than CPU. Accordingly, this research shows that a parallel processing technique using CUDA Library can enhance the data processing speed of satellites images, and that this data processing benefits from multiple advanced remote sensing techniques as compared to a simple pixel computation like NDVI.

KEYWORDS : Compute Unified Device Architecture Library, Normalized Difference Vegetation Index , Parallel Processing, Graphic Processing Unit, Central Processing Unit

서 론

90년대 제 1차 국가 공간정보정책(1995~2000년)으로 시작하여 제 5차 국가 공간정보정책(2013~2017년)까지 거쳐 오면서 우리나라 원격탐사 기술은 많은 발전을 이루어 왔다. 이렇게 원격탐사 기술이 발전함에 따라 위성의 제원 또한 발전하게 되었으며, 현재 IKONOS, QuickBird, OrbView, GeoEye-1, WorldView 등 공간해상도(Pan)가 0.41m~1m 사이로 발전하였다. 실제 미국에서 무료로 제공하고 있는 지구관측위성인 Landsat 영상을 비교해보면 Landsat-1의 MS밴드는 80m의 해상도를 가졌지만 Landsat-7의 MS밴드는 30m, PAN밴드는 15m까지 향상되었다. 우리나라의 대표적인 국토관측위성인 다목적실용위성(Korea Multi-Purpose Satellite, KOMPSAT) 1호의 MS 밴드는 6.6m의 해상도를 가졌으며, 최근 발사된 다목적실용위성 3호의 MS 밴드는 0.55m 급으로 향상되었다. 이처럼 Landsat의 경우 해상도가 약 2배 이상 발전하였으며, 우리나라의 경우

해상도가 10배 가까이 발전하였다.

국토교통부에 따르면 실제 2012년을 기준으로 10개 국가기관 중 공간 정보와 관련된 사업이 51개가 있으며, 원격탐사 기술과 매우 밀접한 사업이 대부분이다. 환경부에서는 인공위성 영상자료를 이용하여 토지피복도 제작서비스를 제공하고 있으며, 농림수산식품부는 농지전용 현황도 DB구축 및 농 · 식품부 농지 종합정보시스템을 구축하여 정보를 제공하고 있다. 이외에도 임상도 제작사업, 종합 해양정보시스템 구축사업 등 원격탐사 기술들을 활용하여 많은 정보를 제공하고 있는 중이다. 대부분의 원격탐사 기술을 이용한 사업들은 위성영상을 제공하거나 자료를 가공하여 서비스를 제공하고 있다. 하지만 대부분의 위성영상자료들은 고해상도 영상을 서비스하고 있기 때문에 제공과 자료처리에 많은 시간이 소요되고 있다.

이러한 이유로 최근 처리속도에 대한 관심이 높아졌으며, 영상처리 분야에서 딥러닝(deep leaning)에 대한 기술이 발전하고 있다. GPU(Graphic Processing Unit, 그래픽처리장치)는 CPU(Central Processing Unit, 중앙처리장치)에 비해 처리속도가 빠르다는 것이 장점이다. 이러한 GPU를 활용할 수 있는 방법 중 하나가 NVIDIA사의 CUDA(Compute Unified Device Architecture)이다. CUDA는 병렬 처리를 제공하는 소프트웨어 개발도구이며, CUDA의 장점은 개발자가 보다 쉽게 프로그래밍할 수 있도록 C 언어의 형태와 비슷하게 지원한다는 점이다.

CUDA를 활용한 연구사례로는 의료용 영상을 대상으로 볼륨 렌더링기법을 CPU와 GPU의 기반으로 자료 처리속도를 비교하였고(Kye and Nam, 2013), CUDA라이브러리를 이용하여 블록암호기법을 구현하였으며, 난수발생기나 스트림 암호에도 응용이 가능하다는 결론을 내렸다(Yeom and Cho, 2008). CUDA 프로그래밍에서 메모리 사용을 기반으로 프로그램 유형을 분류하고 각각의 분류에 따라 커널의 최적화된 알고리즘을 제시하였다(Kim *et al.*, 2010). 위성영상과 관련된 연구로는 Landsat-7 ETM 영상을 활용하여 분광휘도보정 알고리즘을 CUDA로 구현하여 처리시간을 단축시켰다(Jeong *et al.*, 2012). 대부분 CUDA 연구는 다양한 분야에서 활용되고 있으며 주로 자료처리 속도에 중점에 맞춘 연구가 주를 이루었다.

본 논문에서는 점차 발전하는 고해상도 영상의 처리속도를 높이고자 NVIDIA사의 CUDA를 활용하였다. 실험영상은 KOMPSAT 영상을 이용하였으며 원격탐사의 Band별 특징을 활용한 Band math알고리즘을 적용하여 영상처리를 수행하고, CPU 기반과 GPU 기반의 자료처리의 속도를 비교 분석 하였다.

이론적 고찰

1. GPU(Graphic Processing Unit, 그래픽 처리장치)

Intel Pentium과 같은 CPU에 기반을 둔 컴퓨터가 시장에서 주를 이뤘으며 빠른 성장을 하였다. CPU의 장점은 빠른 처리속도에 있다. 그래서 컴퓨터의 하드웨어적 성능이 발전할 때마다 CPU의 처리속도 또한 소프트웨어를 더욱 빠르게 구동할 수 있도록 발전해왔다. 하지만 2000년 초기에 CPU의 성능이 높아짐에 따라 에너지 소비와 열 방출 문제 등 하드웨어적 문제에 부딪혀 성장 속도가 늦춰지게 되었다. 이러한 “단일 프로세싱 코어”의 단점을 보완하기 위해 “다중 프로세싱 코어”로 전환하게 되었다. 다중 프로세싱 코어의 목적은 단순히 소프트웨어의 처리 속도를 빠르게 하는 것이 아닌 많은 소프트웨어를 동시에 사용할 때 매끄럽게 수행되는 것에 목적이 있다. 다음 그림 1은 CPU와 GPU의 발전 속도를 나타낸 것이다.

컴퓨터에서 그래픽 연산을 담당하는 기존의 GPU는 3차원 그래픽에 사용되는 부동 소수점 데이터를 빠르게 처리하기 위해 동시에 수행 가능한 수백 개의 코어로 구성되어 있다(Kim *et al.*, 2013). GPU의 발전은 급속도로 성장하는 비디오 게임 산업에 의해 이루어졌으며, 비디오 게임의 특성상 고화질의 영상이 빠르게 변화하는 장면을 처리하기 위하여 GPU가 설계

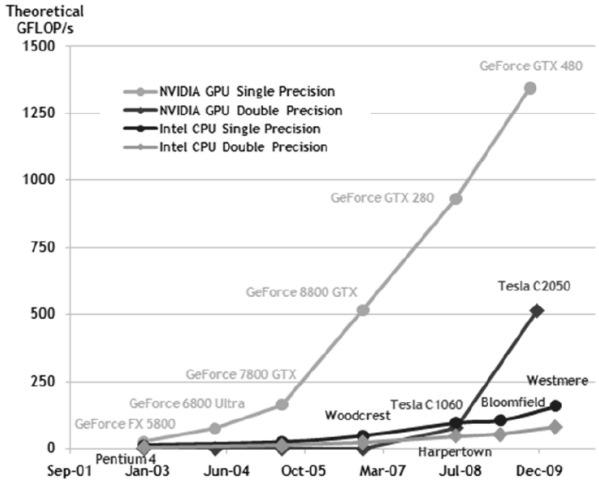

FIGURE 1. CPU and GPU development speed
(Source: CUDA toolkit documentation, 2015)

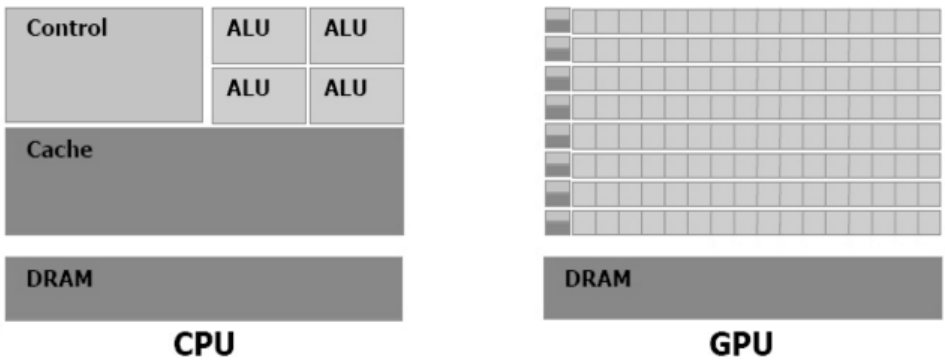

FIGURE 2. CPU and GPU architecture (Source: CUDA toolkit documentation, 2015)

되었다. 이러한 이유로 CPU와 GPU 간에 다른 설계구조를 가지게 되었다. 자료처리 속도를 높이기 위해 칩 영역과 파워 비용을 최대화하는 방안을 강구하였고 그 결과 수많은 스레드를 자료처리에 이용하였다. CPU와 GPU의 구조는 다음 그림 2와 같다.

2. CUDA(Compute Unified Device Architecture)

CUDA는 C프로그래밍 언어를 기반으로 라이브러리를 추가하여 작성할 수 있는 병렬처리 기술이다. 2006년 11월에 처음 발표되었으며, 초기에는 C/C++만 지원했지만 현재는 포트란, C#, MATLAB, PYTHON 등 다양한 언어에서 사용가능하다(Jeong *et al.*, 2012).

CUDA는 CPU와 달리 그래픽카드의 연산능력을 이용하여 처리하는 방법이다. 따라서 기존의 CPU의 처리방식에 몇 가지 처리 방법을 추가해야 한다. 추가된 과정은 CPU로부터 입력 자료를 그래픽카드 메모리에 복사하고 복사된 그래픽메모리에 있는 자료를 GPU가 처리한 후, 처리된 자료를 다시 그래픽 메모리에서 PC의 메모리로 복사하는 과정이 추가로 이루어져야 된다. 다음 그림 3에서 화살표로 표시된 부분이 바로 PC와

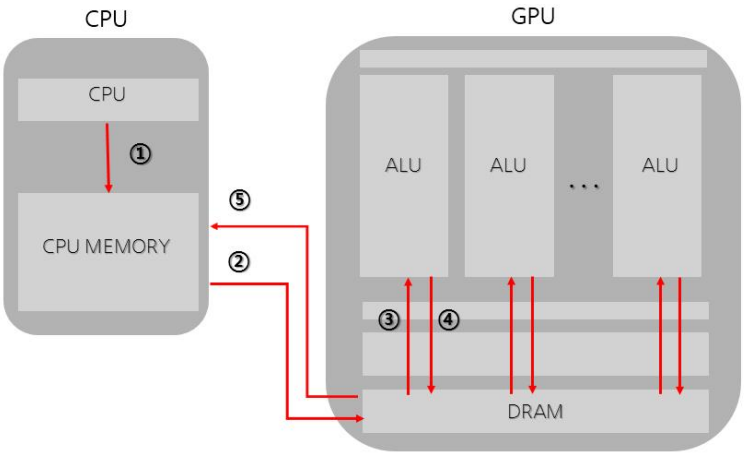

FIGURE 3. GPU processing flow

그래픽 메모리 간의 이동을 나타낸 것이다.

CUDA를 이용하여 자료를 병렬 처리의 과정은 다음과 같다.

① 입력과 출력에 사용할 데이터를 PC메모리에 할당
② 입력과 출력에 사용할 데이터를 그래픽 메모리에 할당
③ 처리하고자 하는 값을 PC메모리에 입력
④ PC메모리에 있는 입력 데이터를 그래픽 메모리로 복사
⑤ 데이터를 분할하여 GPU로 이동
⑥ 스레드를 활용하여 커널 함수로 병렬 처리 수행
⑦ 처리된 결과를 병합
⑧ PC메모리에 결과를 전송
⑨ 그래픽 메모리를 해제
⑩ PC메모리를 해제

알고리즘 구현

1. 연구 활용 위성영상

연구에 사용된 위성영상은 다목적실용위성 3호의 영상이며, 영상의 크기를 기준으로 총 5가지 크기로 분할하여 각각에 대한 자료처리 속도를 기록하였다. 영상의 크기는 다음 표 1과 같다.

2. 연구 알고리즘

일반영상과 달리 위성영상의 센서는 고해상도, 스펙트럼 등의 차이점이 있다(Kurte and Durbha, 2013). 연구에 활용된 알고리즘은 위성영상 스펙트럼의 특징을 활용한 NDVI 알고리즘을 적용하였다. NDVI는 식생의 분포량, 밀집도 및 활동성 등의 각종 정보를 나타내며(Jo, 2012), 널리 이용되는 식생지수(vegetation index)

TABLE 1. Image size and file size

	KOMPSAT-3	
	Image size(pixel)	File size(GB)
Type 1	1,000x1,000	0.002
Type 2	5,000x5,000	0.2
Type 3	10,000x10,000	0.8
Type 4	15,000x15,000	1.8
Type 5	20,000x20,000	3.2

이다(Shin and An, 2004). NDVI는 영상의 가시광선 적색밴드와 근적외밴드를 이용하여 나타낼 수 있다(Justice *et al.*, 1985, Benedetti *et al.,* 1994). NDVI는 다음 식(1)을 이용하여 구할 수 있다.

$$NDVI = \frac{NIR - RED}{NIR + RED} \quad (1)$$

NDVI는 일반영상과 달리 위성영상 밴드의 특성을 이용한 알고리즘이므로 타 분야에서의 활용이 드물며 원격탐사 분야에서 주로 활용되고 있는 알고리즘이다. NDVI의 알고리즘은 원격탐사분야에서 널리 이용되는 지수로써 영상 밴드간의 1:1 매칭을 통해 연산이 이루어지는데, 이는 각 화소에 대해서 동일한 작업을 수행하는 것으로 병렬처리 구조에서 SIMD (Single Instruction Multiple Data) 구조와 어울리며 (Kim *et al.*, 2011), 본 연구에서 구현하였다.

3. 연구 환경

본 연구에서 구현된 알고리즘의 성능 측정을 위한 시험 환경은 다음 표 2와 표 3과 같다.

TABLE 2. Hardware performance

OS	Window7 Ultimate K 64-bit
CPU	intel Core i5-3210M Processor
GPU	GeForce GT 630M
Memory	DDR2-8GB
IDE	MS Visual Studio 2010, C++
CUDA version	5.5

TABLE 3. GeForce GT 630M performance

Model name	GeForce GT 630M
Number of cores	96
Core speed	660MHz
Architecture	Fermi GF108
Memory	1024MB
Memory speed	900MHz
Memory type	DDR3
Memory bus	128Bit

Window7 운용체제에서 연구를 진행하였으며, 연구에 사용된 CPU는 Intel core i5-3210m이다. Intel core i5-3210m은 듀얼코어로 2.5 GHz의 클럭 주파수이며, 그래픽 클럭 주파수는 650MHz이다. GeForce GT 630M은 96개의 코어로 구성되며 프로세서 클럭은 800MHz이다. 지원 CUDA 버전은 ver. 5.5를 지원한다. 운영체제는 Window7 64 bit, 병렬화 환경은 CUDA Ver. 5.5가 사용되었다.

4. CPU 기반 NDVI 구현

NDVI는 영상의 Rband와 Nband를 이용하여 구현할 수 있다. 먼저 영상을 불러오기 위해서 LIBtiff 라이브러리를 이용하였으며 LIBtiff라이브러리의 함수인 TIFFreadRawStrip함수를 이용하여 영상을 한 행씩 픽셀 데이터로 불러올 수 있다. 불러온 각 행의 픽셀 데이터를 변수 데이터에 저장하였으며, Rband와 Nband에 해당하는 픽셀 데이터를 변수 데이터로부터 추출하였다. 변수 데이터에 저장되어 있는 픽셀 데이터는 Bband, Gband, Rband, Nband 순으로 저장되어 있기 때문에 for문을 활용하여 3번째와 4번째 있는 픽셀 데이터를 추출하여 각각의 변수 Rbnad와 Nband에 저장하였다. 코드는 다음 그림 4와 같다.

이렇게 저장되어진 Rband와 Nband를 이용하여 NDVI 구현을 수행하였다. NDVI 영상은 -1, 0, 1의 값만 가진다. 따라서 NDVI 식에 의해 Rband가 Nband보다 큰 값을 가지면 해당 위치에 1의 값으로 저장하고 반대로 Rband가 Nband보다 작으면 해당 위치에 -1 값으로 저장한다. 그리고 Rband와 Nband가 같은 값을 가지면 해당 위치에 0의 값을 저장하도록 설계하였다. 코드는 다음 그림 5와 같다.

5. GPU 기반 NDVI 구현

GPU 기반의 NDVI 알고리즘은 몇 가지 동작이 추가되어야 한다. 먼저 그래픽 메모리카드로 데이터를 전송할 수 있는 공간을 만들어

```
//위성영상의 pixel data를 접근하는 방법
for(int count=0;count<imageheight/height;count++){
                    uint32 col =0;
                    for(int h=count*height;h<(count+1)*height;h++){
                                strip = h;
                                TIFFReadRawStrip(image,strip, buff, bc[strip]);
                                data = (uint16*)buff;
                                gettime = clock();
                                for(int w=0;w<width*4;w+=4){
                                            *(Rband + ((col*width)+w/4)) = data[w+1];
                                            *(Nband + ((col*width)+w/4)) = data[w+3];
                                }
                                col++;
                    }
```

FIGURE 4. Approach pixel data method based on CPU

```
//Rband,Nband,ndvi 각각의 3가지 버퍼를 이용하여 NDVI구현
void cpundwi(uint16 *R,uint16 *N,uint16 *ndvi){
//각각의 버퍼 크기만큼 for문을활용하여 ndvi연산을 수행
          for(int i=0;i<width*height;i++){
                                if(R[i] == N[i]){ ndvi[i] =-1;}
                                else if(R[i] > N[i]){ ndvi[i] =0;}
                                else if(R[i] < N[i]){ ndvi[i] =1;}
                    }
}
```

FIGURE 5. Code realization based on CPU

준다. 그리고 CPU 기반의 NDVI 알고리즘과 동일하게 변수 데이터에 픽셀 데이터를 읽어오고 각각의 픽셀 데이터를 Rband와 Nband로 저장하였다. Rband와 Nband는 CPU에 저장되어 있기 때문에 이 데이터들을 그래픽메모리로 전송하는 작업이 필요하다. 그래픽메모리로 전송하는 함수는 CudaMemcpy를 이용하여 변수 Rband와 Nband의 데이터를 그래픽카드 메모리로 전송한다. GPU를 통해서 실행될 함수를 커널이라 부르는데 CPU공간에서 GPU공간으로 데이터를 전송 후 커널을 통하여 NDVI 연산을 수행하였다. 코드는 다음 그림 6과 같다.

GPU를 활용한 NDVI 구현은 CPU의 NDVI 구현과 같은 원리로 구현하였다. CPU 함수에 비해 추가적인 부분은 BlockIdx.x * BlockDim.x * threaIdx.x 명령어이다. 이 명령어는 프로그램의 최소 실행 단위인 스레드의 전체 개수를 알 수 있으며 BlockDim.x * threaDim.x 명령어를 통해서 증가 값을 설정함으로써 실행되어진 스레드의 개수 내에서 프로그램이 수행하게 된다. 코드는 다음 그림 7과 같다.

```
//GPU공간에 메모리 생성
        uint16 *dev_G, *dev_N, *dev_ndwi;
//GPU공간에 메모리 크기 설정
        cudaMalloc((void**)&dev_G,width*height*sizeof(uint16));
        cudaMalloc((void**)&dev_N,width*height*sizeof(uint16));
        cudaMalloc((void**)&dev_ndwi,width*height*sizeof(uint16));
//CPU 버퍼를 GPU버퍼로 COPY
cudaMemcpy(dev_G,
                Gband,
                width*height*sizeof(uint16),
                cudaMemcpyHostToDevice);
cudaMemcpy(dev_N,Nband,
                width*height*sizeof(uint16),
                cudaMemcpyHostToDevice);
//GPU상에서 NDWI연산 함수
add<<<grids,threads>>>(dev_G,dev_N,dev_ndwi);
//GPU상에서 연산된 결과값을 CPU공간으로 COPY
cudaMemcpy(ndwi,
                dev_ndwi,width*height*sizeof(uint16),
                cudaMemcpyDeviceToHost);
```

FIGURE 6. Approach pixel data method based on CUDA

```
//커널을 이용한 NDVI구현
__global__ void ndvi(int *g,int *n,int *ndvi){
             int tid = blockIdx.x * blockDimx.x + threadIdx.x;

            while(tid<(width * height)){
                    if(g[tid] == n[tid]){ndvi[tid] = -1;}
                   else if(g[tid] < n[tid]) ndvi[tid] = 0;}
                   else if(g[tid[ > n[tid]] ndvi[tid] = 1;]

                    tid += blockDim.x * gridDim.x;
}
}
```

FIGURE 7. Code realization based on CUDA

구현 결과 비교 및 분석

1. 결과 이미지 비교

다목적실용위성 3호 영상을 영상의 크기 기준으로 하여 총 5가지로 분류하였다. 분류되어진 영상들은 ArcMap 10.1 버전과 CPU 기반, GPU 기반의 3가지 방법을 이용하여 NDVI를 구현하였으며, 구현된 NDVI 영상을 시각적 비

교와 히스토그램 비교를 수행하여 동일한 영상이 획득되었는지를 판단하였다. 다음 그림 8은 ArcMap과 CPU, GPU 기반의 방법으로 구현한 NDVI 영상을 크기별로 분류한 이미지이다. 그 결과 시각적으로는 차이점을 발견하지 못하였고, 모두 동일한 영상을 획득한 것으로 판단되어진다. 다음 그림 9는 각 영상의 히스토그램을 비교하기 위해 나열하였다. Type 1의 경우 1,000,000개의 픽셀 중 -1 값을 가진 픽셀이 3가지 방법 모두 동일하게 660,324개로 일치

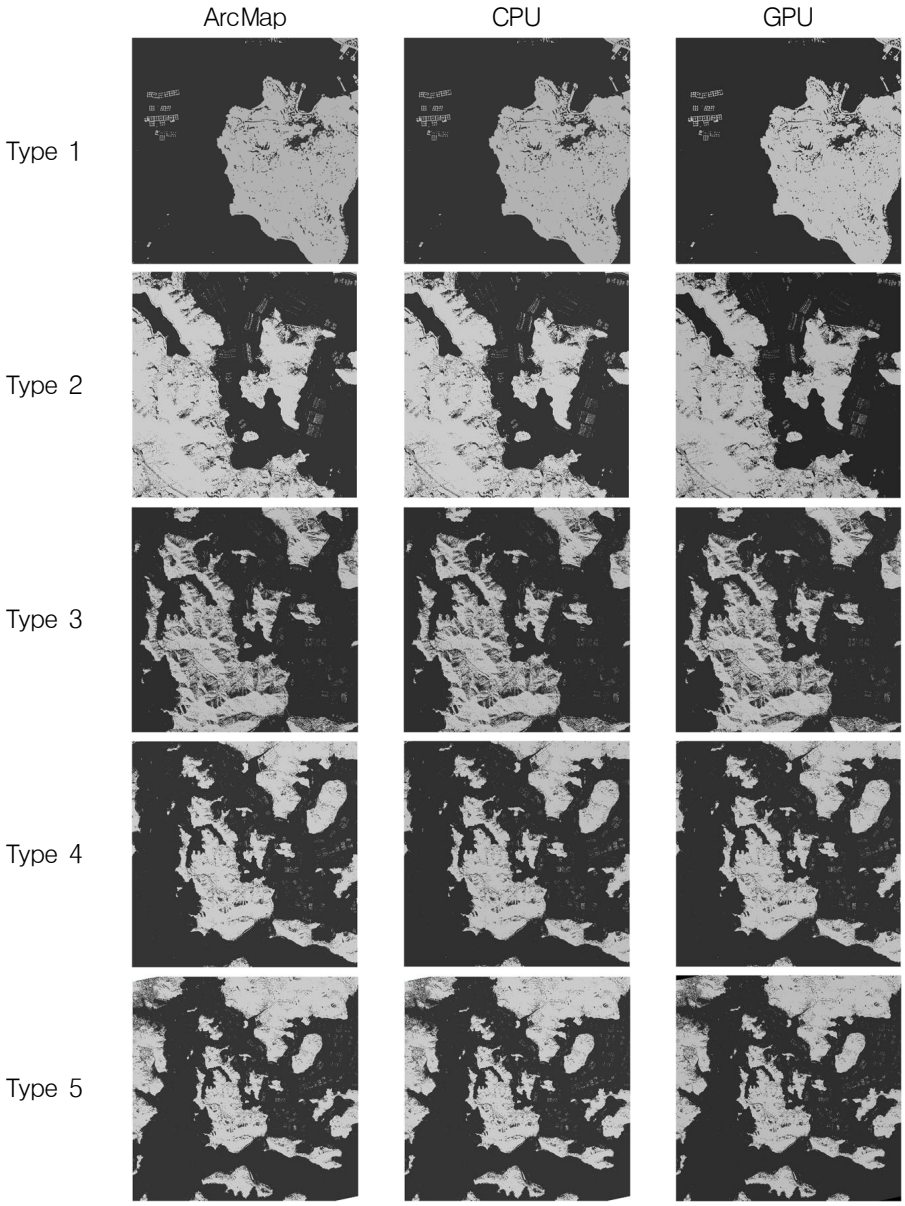

FIGURE 8. NDVI images by ArcMap, CPU and GPU in order of size

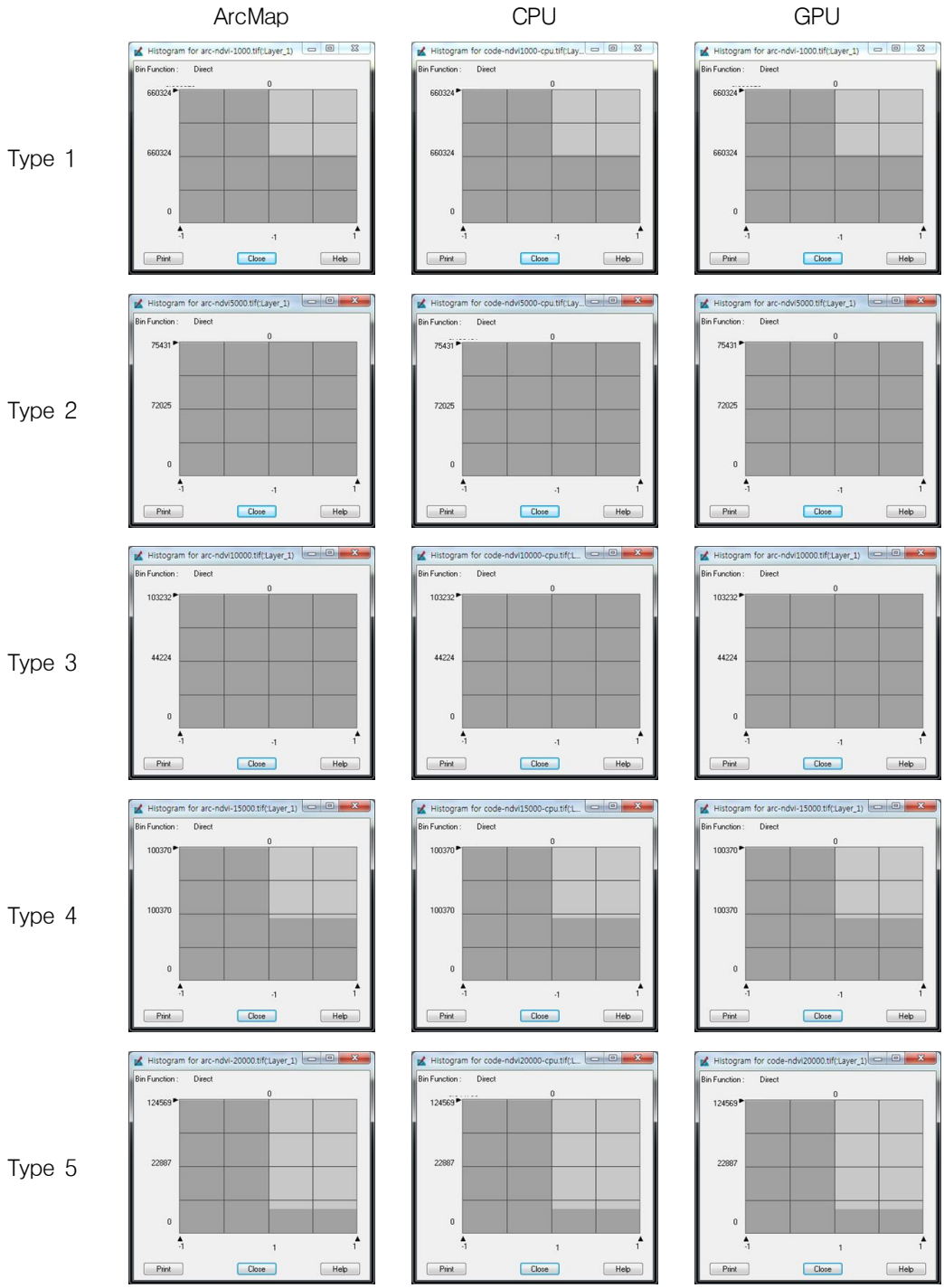

FIGURE 9. Histograms of images by ArcMap, CPU and GPU in order of size

하였으며, Type 5의 경우 400,000,000 픽셀 중 -1 값을 가진 픽셀은 124,569개로 3가지 방법 모두 일치하였다. 이처럼 ArcMap, CPU, GPU 기반으로 구현된 5가지의 크기별 NDVI 영상은 시각적 및 히스토그램이 모두 일치하는 것으로 확인되었다.

2. 속도 측정 및 결과 분석

다목적실용위성 3호 영상을 크기 기준으로 5가지 Type으로 만들고, 각각에 CPU 기반과 GPU 기반으로 10번을 반복하여 평균 처리 시간을 측정하였다.

표 4에서 CPU 기반의 평균처리시간은 Type 1은 0.092초, Type 2는 2.129초, Type 3은 8.214초, Type 4는 13.774초, Type 5는 23.681초로 Type 5는 기존 Type 1에 비해 약 230배로 처리시간이 늘어난 것을 확인할 수 있다.

표 5에서 GPU 기반의 평균처리시간은 Type 1은 0.304초, Type 2는 0.635초, Type 3은 1.596초, Type 4는 3.307초, Type 5는 4.868초로 Type 5는 기존 Type1에 비해 약 15배로 처리시간이 늘어난 것을 확인할 수 있었다.

각각의 Type별 평균으로 시간 차이를 확인해보면 Type 1의 결과만 CPU가 GPU보다 약 0.2초 빨랐으며, 그 외 모든 Type은 GPU가 월등히 처리속도가 빠른 것을 확인할 수 있었다. Type 2의 경우 CPU의 평균 처리속도는

TABLE 4. CPU average time

	Type 1 1,000X1,000(pixel)	Type 2 5,000X5,000(pixel)	Type 3 10,000X10,000(pixel)	Type 4 15,000X15,000(pixel)	Type 5 20,000X20,000(pixel)
Try 1	0.097	2.153	8.648	14.134	23.431
Try 2	0.094	2.153	8.439	13.713	23.291
Try 3	0.093	2.168	8.923	13.634	23.447
Try 4	0.094	2.137	8.877	13.665	23.649
Try 5	0.094	2.059	8.892	13.947	23.697
Try 6	0.094	2.059	8.595	13.852	24.083
Try 7	0.093	2.184	8.736	13.588	23.556
Try 8	0.078	2.043	8.502	13.837	23.149
Try 9	0.093	2.168	8.518	13.743	24.487
Try 10	0.093	2.169	9.084	13.635	24.018
Average time(sec)	0.092	2.129	8.214	13.774	23.681

TABLE 5. GPU average time

	Type 1 1,000X1,000(pixel)	Type 2 5,000X5,000(pixel)	Type 3 10,000X10,000(pixel)	Type 4 15,000X15,000(pixel)	Type 5 20,000X20,000(pixel)
Try 1	0.312	0.639	1.549	3.262	4.821
Try 2	0.312	0.641	1.579	3.264	4.805
Try 3	0.297	0.624	1.648	3.435	4.802
Try 4	0.312	0.639	1.583	3.279	4.898
Try 5	0.271	0.626	1.586	3.341	4.915
Try 6	0.309	0.624	1.575	3.262	4.914
Try 7	0.308	0.639	1.603	3.289	4.882
Try 8	0.305	0.641	1.598	3.371	4.882
Try 9	0.310	0.639	1.669	3.291	4.914
Try 10	0.311	0.639	1.577	3.284	4.853
Average time(sec)	0.304	0.635	1.596	3.307	4.868

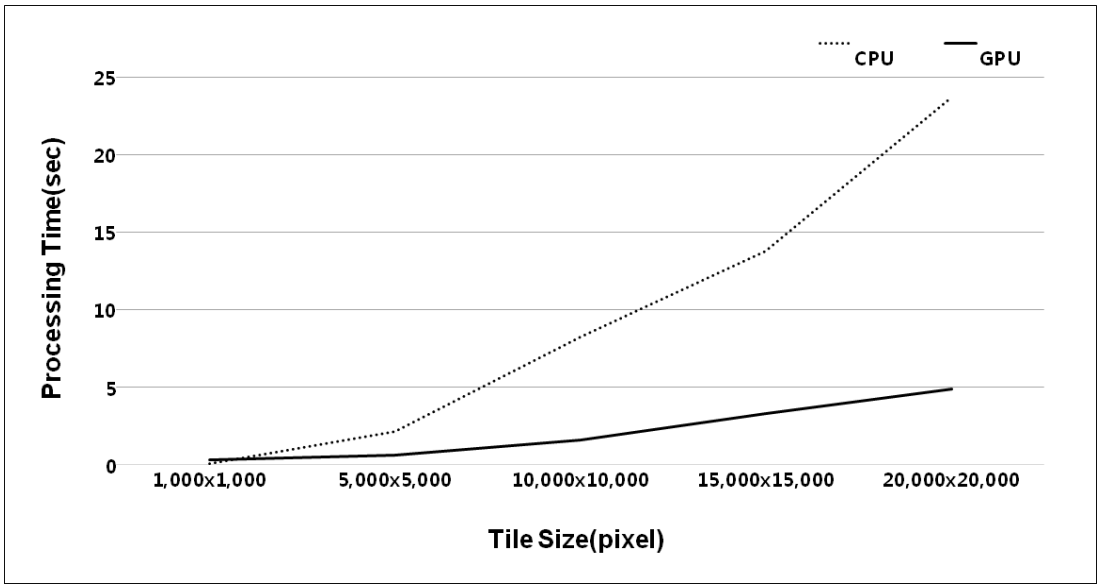

FIGURE 10. Data processing time of CPU and GPU

2.129초, GPU의 경우 0.635초, Type 3의 경우 CPU의 평균처리속도는 8.214초, GPU의 경우 1.596초이며, Type 4의 경우 CPU는 13.774초 GPU는 3.307초로 나타났다. Type 5의 경우 CPU는 23.681초, GPU는 4.868초로 Type1을 제외한 모든 Type의 GPU 처리속도가 빠른 것으로 나타났다. Type 2, 3, 4, 5 순으로 GPU의 처리속도가 CPU의 처리속도보다 약 3배, 약 4배, 약 5배, 약 5배 빠른 것을 확인할 수 있다. 그림 10은 처리시간을 그래프로 나타낸 것이다.

결 론

본 논문에서는 원격탐사의 밴드 연산의 한 방법인 NDVI 방법을 CUDA 라이브러리를 이용하여 병렬 처리를 수행하였다. 현재 원격탐사에 관련된 연구는 활용분야에 집중되어 있으며 자료처리 효율에 관련된 연구는 미흡한 상태이다. 초기 원격탐사에 사용되던 인공위성의 제원은 현재 많이 발전하였으며 발전된 위성영상은 데이터양이 점차 많아질 수밖에 없다. 따라서 본 논문에서는 점차 증가하는 원격탐사 데이터를 빠른 속도로 처리하기 위하여 CUDA 라이브러리를 이용한 GPU 기반의 병렬 처리를 제시하였으며, 이를 확인하기 위하여 CPU 기반 및 GPU 기반의 데이터처리를 비교·분석하였다. 먼저 CPU 기반과 CUDA를 이용한 GPU 기반으로 NDVI를 구현하였으며, 이렇게 구현된 영상을 상용 소프트웨어인 ArcMap으로 구현한 NDVI 영상과 비교하였다. 시각적 비교와 히스토그램을 통하여 비교한 결과 ArcMap, CPU, GPU로 구현된 NDVI 영상은 모두 동일한 영상으로 확인되었다. 마지막으로 CPU를 기반으로 수행된 데이터 처리시간과 GPU를 기반으로 수행된 데이터 처리 시간을 비교·분석하였다. 그 결과 GPU 기반이 CPU 기반보다 빨랐으며, 데이터의 양이 늘어날수록 데이터 처리속도는 GPU 기반의 방법이 빠른 것을 확인할 수 있었다.

본 논문에서 제안한 NDVI 연산은 병렬처리에 적합한 SIMD구조와 비슷하기에 데이터 처리속도를 효과적으로 줄일 수 있었다. 하지만 원격탐사 기법의 한 방법인 감독분류(supervised classification)는 화소단위 분류 기법으로 분류항목의 트레이닝샘

플을 사용자가 지정하고 지정된 샘플을 기준으로 토지피복을 분류하는 방법으로써(Choung, 2015), 밴드 간의 연산이 아닌 데이터의 모양과 값을 매칭시키는 원리로 본 연구에 적용시킨 NDVI와 차이를 나타내고 있다. 추후 연구로써는 감독분류와 같이 SIMD구조와 유사하지 않은 알고리즘에 적용이 필요하다고 사료된다. KAGIS

REFERENCES

Benedetti, R., P. Rossini and R. Taddei. 1994. Vegetation classification in the middle mediterranean area by satellite data. International Journal of Remote Sensing 15(3):583-596.

Choung, Y.J. 2015. Land cover change detection in the Nakdong river basin using LiDAR data and multi-temporal Landsat imagery. Journal of the Korean Association of Geographic Information Studies 18(2):135-148. (정윤재. 2015. LiDAR DEM과 다중시기에 촬영된 Landsat 영상을 이용한 낙동강 유역 내 토지피복 변화 탐지. 한국지리정보학회지 18(2):135-148).

CUDA toolkit documentation. 2015. http://docs.nvidia.com/cuda/cuda-c-programming-guide (Accessed September 1, 2015).

Jeong, I.K., M.G. Hong, K.S. Hahn, J.S. Choi and C. Kim. 2012. Performance study of satellite image processing on graphics processors unit using CUDA. Korea Journal of Remote Sensing 28(6):683-691.

Jo, M.H. 2012. A study on the extraction of a river from the RapidEye image using ISODATA algorithm. Journal of the Korean Association Geographic Information Studies 15(4):1-14 (조명희. 2012. ISODATA 기법을 이용한 RapidEye 영상으로부터 하천의 추출에 관한 연구. 한국지리정보학회지 15(4):1-14).

Justice, C.O., J.R.G. Townshend., B.N. Holben and C.J. Tucker. 1985. Analysis of the phenology of global vegetation using meteorological satellite data. International Journal of Remote Sensing 6(8):1271-1318.

Kim, J.H., H.C. Shin, W.S. Cheong and G. Bang. 2011. The performance of fast view synthesis using GPU. Proceedings of the 2011 Summer Conference of Journal of Broadcast Engineering. pp.22-24 (김재한, 신홍창, 정원식, 방건. 2011. GPU를 이용한 고속 영상 합성기법의 성능. 한국방송공학회 하계 학술대회발표집. 22-24쪽).

Kim, K.W., H.W. Kim, H.J. Kim, T.Y. Huh, S.H. Jung and Y.H. Song. 2013. An analytical model for performance prediction of AES on GPU architecture. Journal of The Institute of Electronics Engineers of Korea 50(4): 849-856 (김규운, 김현우, 김희정, 허태영, 정상혁, 송용호. 2013. GPU 아키텍처의 AES 암호화 성능 예측 분석 모델. 전자공학회논문지 50(4):849-856).

Kim, S.S., D.H. Kim, S.K. Woo and J.S. Ihm. 2010. Analysis of programming techniques for creating optimized CUDA software. Korean Institute of Information Scientists and Engineers 16(7):775-787. (김성수, 김동헌, 우상규, 임인성. 2010. 최적화된 CUDA소프트웨어 제작을 위한 프로그래밍 기법 분석. 정보과학회논문지 16(7):775-787).

Kurte, K.R. and S.S. Durbha. 2013. High resolution disaster data clustering using graphics processing units. Proceeding of the 2013 IEEE International Geoscience and Remote Sensing Symposium. pp.1696-1699.

Kye, H.W. and J.H. Nam. 2013. Speed

comparison of accelerated volume rendering based on CPU and GPU. Proceedings of the HCI Society of Korea. 2013(1):51-53 (계희원, 남진현. 2013. 가속화된 볼륨 렌더링의 CPU와 GPU기반 속도 비교, 한국HCI학회 학술대회 2013(1):51-53).

Shin, S.C. and T.Y. An. 2004. Estimation of areal evapotranspiration using NDVI and temperature data. Journal of the Korean Association of Geographic Information Studies 7(3):79-89. (신사철, 안태용. 2004. NDVI와 기온자료를 이용한 광역증발산량의 추정. 한국지리정보학회지 7(3):79-89).

Yeom, Y.J. and Y.K. Cho. 2008. High-speed implementations of block ciphers on graphics processing units using CUDA library. Korea institute of information security & cryptology 18(3): 23-32. (염용진, 조용국. 2008. GPU용 연산 라이브러리 CUDA를 이용한 블록암호고속 구현. 정보보호학회논문지 18(3):23-32).