Journal of the Korean Society of Surveying, Geodesy, Photogrammetry and Cartography
Vol. 37, No. 6, 445-452, 2019
https://doi.org/10.7848/ksgpc.2019.37.6.445

ISSN 1598-4850(Print)
ISSN 2288-260X(Online)
Original article

대용량 위성영상의 무감독 분류를 위한 k-Means Clustering 알고리즘의 병렬처리: 다중코어와 PC-Cluster를 이용한 Hybrid 방식

Parallel Processing of k-Means Clustering Algorithm for Unsupervised Classification of Large Satellite Images: A Hybrid Method Using Multicores and a PC-Cluster

한수희[1] · 송정헌[2]
Han, Soohee · Song, Jeong Heon

Abstract

In this study, parallel processing codes of k-means clustering algorithm were developed and implemented in a PC-cluster for unsupervised classification of large satellite images. We implemented intra-node code using multicores of CPU (Central Processing Unit) based on OpenMP (Open Multi-Processing), inter-nodes code using a PC-cluster based on message passing interface, and hybrid code using both. The PC-cluster consists of one master node and eight slave nodes, and each node is equipped with eight multicores. Two operating systems, Microsoft Windows and Canonical Ubuntu, were installed in the PC-cluster in turn and tested to compare parallel processing performance. Two multispectral satellite images were tested, which are a medium-capacity LANDSAT 8 OLI (Operational Land Imager) image and a high-capacity Sentinel 2A image. To evaluate the performance of parallel processing, speedup and efficiency were measured. Overall, the speedup was over N / 2 and the efficiency was over 0.5. From the comparison of the two operating systems, the Ubuntu system showed two to three times faster performance. To confirm that the results of the sequential and parallel processing coincide with the other, the center value of each band and the number of classified pixels were compared, and result images were examined by pixel to pixel comparison. It was found that care should be taken to avoid false sharing of OpenMP in intra-node implementation. To process large satellite images in a PC-cluster, code and hardware should be designed to reduce performance degradation caused by file I / O. Also, it was found that performance can differ depending on the operating system installed in a PC-cluster.

Keywords : k-Means Clustering, Parallel Processing, PC-Cluster, Multicores

초 록

본 연구에서는 대용량 위성영상의 무감독분류를 위해 k-means clustering 알고리즘의 병렬처리 코드를 개발하여 PC-cluster에서 구현하였다. 이를 위해 OpenMP (Open Multi-Processing)를 기반으로 CPU (Central Processing Unit)의 다중코어를 이용하는 intra-node 코드와 message passing interface를 기반으로 PC-cluster를 이용하는 inter-nodes 코드, 그리고 이 둘을 병용하는 hybrid 코드를 구현하였다. 본 연구에 사용한 PC-cluster는 한 대의 마스터 노드와 여덟 대의 슬래이브 노드로 구성되어 있고 각 노드에는 여덟 개의 다중코어가 장착되어 있다. PC-cluster에는 Microsoft Windows와 Canonical Ubuntu의 두 가지 운영체제를 설치하여 병렬처리 성능을 비교하였다. 실험에 사용한 자료는 두 가지 다중분광 위성영상으로서 중용량인 LANDSAT 8 OLI (Operational Land Imager) 영상과 대용량인 Sentinel 2A 영상이다. 병렬처리의 성능을 평가하기 위하여 speedup과 efficiency를 측정한 결과 전반적으로 speedup은 N/2 이상, efficiency는 0.5 이상으로 나타났다. Microsoft Windows와 Canonical Ubuntu를 비교한 결과 Ubuntu가 2-3배의 빠른 결과를 나타내었다. 순차처리와 병렬처리 결과가 일치하는지 확인하기 위해 각 클래스의 밴드별 중심값과 분류된 화소의 수를 비교하고 결과 영상간 화소대 화소 비교도 수행하였다. Intra-node 코드를 구현할 때에는 OpenMP에 의한 false sharing이 발생하지 않도록 주의해야 하고, PC-cluster에서 대용량 위성영상을 처리하기 위해서는 파일 I/O에 의한 성능저하를 줄일 수 있도록 코드 및 하드웨어를 설계해야 함을 알 수 있었다. 또한 PC-cluster에 설치된 운영체제에 따라서도 성능 차이가 발생함을 알 수 있었다.

핵심어 : k-Means Clustering, 병렬처리, PC-Cluster, 다중코어

Received 2019. 10. 31, Revised 2019. 11. 19, Accepted 2019. 11. 28

1) Corresponding Author, Member, Dept. of Geoinformatics Engineering, Kyungil University (E-mail: scivile@kiu.kr)
2) HYPERSENSING (E-mail: newssong@hypersensing.net)

1. 서 론

위성에 탑재된 센서의 성능이 발전함에 따라 위성으로부터 취득할 수 있는 영상의 해상도가 수 십 센티미터 급으로 향상되고 다수의 고해상도 위성을 이용한 지구 관측 빈도가 높아짐에 따라 취득되는 영상의 용량은 실제로 처리 가능한 수준을 넘어서고 있다. 병렬처리는 대용량 위성영상을 효율적으로 처리할 수 있는 중요한 방법으로서 다양한 접근이 시도되고 있다. 소규모의 병렬처리는 SMP (Symmetric Multi-Processing)라 하여 공유 메모리를 기반으로 두 개 이상의 프로세서를 사용하는 방식으로, CPU (Central Processing Unit)에 집적된 다중코어(multicores)를 이용하거나 GPU (Graphics Processing Unit)에 집적된 절대 다수의 코어를 이용한다. 중·대규모의 병렬처리는 MPP (Massive Parallel Processing)라 하여 독립된 메모리를 가진 프로세서들을 고성능 네트워크로 연결하여 연산을 수행하는 것으로, PC를 수 대에서 수 천 대 연결하여 이용하는 PC-cluster나, 특수한 하드웨어와 소프트웨어를 고안하여 제작한 슈퍼컴퓨터를 이용한다(IBM, 2019). 원격탐사 분야에서 병렬처리를 활용하는 연구는 대체로 초분광 영상 처리 분야에서 활발하게 수행하고 있으며, 특히 PPI (Pixel Purity Index) 산출 또는 endmember 분석과 관련된 연구가 다수 존재한다(González *et al*., 2010; Plaza *et al*., 2006; Sánchez and Plaza, 2010). 하드웨어 또는 플랫폼을 기준으로 분류하면, GPU를 이용한 연구가 다수이고(Koo, 2012; Sun *et al*., 2009; Lu *et al*., 2017; Fredj *et al*., 2017), 최근에는 클라우드 컴퓨팅을 활용하려는 움직임이 나타나고 있다(Wang *et al*., 2013; Sugumaran *et al*., 2018).

현재까지 대용량 위성영상을 효율적으로 처리하기 위한 다양한 알고리즘들이 개발되었지만 이들을 병렬화할 수 있는 보편적인 기술은 존재하지 않는다. 따라서 각 알고리즘들의 연산 부하와 구조적 특성을 고려하여 개별적으로 병렬처리 코드를 구현해야 한다. Han(2017)은 대용량 위성영상의 처리를 위해 CPU의 다중코어를 이용하는 병렬 k-means clustering 알고리즘을 소개하였다. 대표적인 무감독분류 방식 중 하나인 k-means clustering 알고리즘은 비교적 코드 구현이 쉽지만 높은 분류 정확도를 기대하기는 어려우므로 주로 감독분류의 전처리 과정으로 활용한다. 그럼에도 불구하고 k-means clustering 알고리즘은 연산 집약적이고 사용자의 개입이 적기 때문에 병렬처리의 성능을 객관적으로 평가하기에 적합하다. 아울러 클래스 정보가 모든 화소의 영향을 받으며 수시로 업데이트되는 global optimization 알고리즘으로서, 단순히 영상을 분할하여 여러 컴퓨터에 처리하는 분산처리 방식으로는 구현하기 어렵다(Han, 2017).

본 연구는, 대용량 위성영상의 무감독 분류를 위한 k-means clustering 알고리즘의 병렬처리(Han, 2017)의 후속연구로서, PC-cluster를 이용하여 처리 속도를 기존 연구보다 향상시키는 것을 목표로 한다. PC-cluster는 PC 또는 웍스테이션을 네트워크로 연결하여 MPP를 수행하도록 고안된 고가용성 하드웨어이다. 그러나 SMP에 비하여 높은 수준의 운영 기술을 필요로 하여 저변 확대가 어려운 것으로 판단된다. 따라서 대용량 위성영상 처리에도 적극적으로 적용하지 못하는 것으로 판단된다. 본 연구에서 사용하는 하드웨어는 아홉 대의 일반 PC로 구성된 PC-cluster이며 각 PC의 CPU에 장착된 다중코어를 활용하는 hybrid 방식이다. PC-cluster에는 두 가지 운영체제, 즉 Microsoft Windows와 Canonical Ubuntu를 설치하고 병렬처리 코드를 실행하여 성능을 비교하였다. 실험에 사용한 자료는 다중분광 위성영상인 LANDSAT 8 OLI (Operational Land Imager) 영상과 Sentinel 2A 영상이다. 본 논문은 2절에서 Han(2017)의 연구를 요약적으로 소개한 후 PC-cluster를 이용한 병렬처리 알고리즘을 설명한다. 3절에서는 개발된 병렬처리 코드의 적용 결과를 제시하고, 4절에서는 결론을 제시한다.

2. 본 론

2.1 k-Means clustering 알고리즘

위성영상의 무감독 분류를 위한 k-means clustering 알고리즘은 다음과 같은 과정으로 구성된다.

① k개 클래스의 밴드별 초기 중심 설정
② 모든 화소를 k개의 중심 중 가장 가까운 중심의 클래스로 분류
③ 모든 화소의 분류가 끝나면 각 클래스의 중심 재설정
④ ②,③의 과정을 반복하다가 클래스가 변경되는 화소의 비율이 임계치 이하일 때 종료

```
for (iter = 0 to nMaxIter) // iteration
    for (i = 0 to nRow * nColumn) // through all pixels
        for (j = 0 to nClass) // through all classes
            for (k = 0 to nBand) // through all bands
                ...
            Dist[j] = Distance from Pixel[i] to Center[j]
                // calculate the Euclidean distance from pixel i to class center j
        Class[i] = argmin(Dist[j]) // classify pixel i to class j which minimizes the distance
        if (isChanged(Class[i])) // if Pixel[i] is classified to a different class
            nChanged++ // count no. of reclassified pixels
            nPixel[Class[i]]++ // count no. of pixels of class j
            update Center[Class[i]] // update the center of class j
        if (nChanged < nMaxChanged) // ending condition
            break
```

Fig. 1. Pseudo codes of k-means clustering (modified from Han(2017))

과정 ②의 pseudo 코드는 4중 for loop으로 구성되어 있는데 두 번째 loop가 가장 긴 시간을 사용하므로 병렬처리를 이용하여 성능을 향상시킬 가능성이 가장 높다(Fig. 1).

2.2 k-Means clustering 알고리즘의 병렬처리

본 연구에서 k-Means clustering 알고리즘의 병렬처리는 intra-node 병렬처리와 inter-nodes 병렬처리로 구분한다. Intra-node 병렬처리란 공유 메모리를 기반으로 CPU의 다중코어를 이용하여 병렬처리를 수행하는 것을 말한다. 즉 OpenMP(OpenMP ARB, 2016)를 이용하여 PC 내에서 CPU에 장착된 다중코어를 제어하고 RAM (Random Access Memory)과 HDD (Hard Disk Drive)에 저장된 자료를 공유하며 병렬처리 하는 것이다. Intra-node 병렬처리는 2.1의 과정 ②를 병렬처리 하는 것이다. Inter-nodes 병렬처리란 분산 메모리 기반의 PC-cluster를 이용하여 병렬처리를 수행하는 것을 말한다. 즉 MPI(Message Passing Interface)(Argonne National Laboratory, 2012)를 이용하여 독립적인 CPU와 메모리를 가진 노드(PC 또는 workstation)를 제어하고 네트워크(예, Ethernet)를 통해 자료를 전송함으로써 병렬처리를 하는 것을 말한다. Inter-nodes 병렬처리는 2.1 단원의 과정 ①과 ③을 병렬처리 하는 것이다. 이와 같이 PC-cluster와 각 PC의 다중코어를 이용하여 inter-nodes 병렬처리와 intra-node 병렬처리를 함께 구현하는 방식을 hybrid 병렬처리라고 말한다.

2.2.1 Intra-node 병렬처리

본 연구에서 intra-node 병렬처리는, 각 슬레이브 노드(slave node)에서 *n*개의 다중코어로 병렬처리 하는 것을 말한다. 이는 Fig. 1의 두 번째 for loop 앞에 OpenMP 구문인 #pragma omp parallel for 구분을 추가하여 구현할 수 있다(Fig. 2).

```
for (iter = 0 to nMaxIter) // iteration
#pragma omp parallel for private(i, j, k, c) reduction(+:nChanged)
    //implementation of multi-thread
    for (i = 0 to nRow * nColumn) // through all pixels
        ...
        Class[i] = argmin(Dist[j])
                // classify pixel i to class j which minimizes the distance
        if (isChanged(Class[i])) // if Pixel[i] is classified to a different class
            ...
            nPixel[Class[i]]++ // count no. of pixels of class j
            update Center[Class[i]] // update the center of class j
        ...
```

Fig. 2. Implementation of intra-node parallel processing using OpenMP (modified from Han(2017))

OpenMP는 false sharing라는 문제점을 안고 있는데, 이는 서로 다른 코어가 동일한 캐시 라인에 있는 변수를 수정할 때 캐시의 일관성을 유지하도록 메모리가 강제 업데이트되기 때문에 발생하는 현상이다. False sharing을 해결하지 않으면 병렬처리를 하지 않는 경우보다 더욱 많은 시간이 소요된다. False sharing을 방지하기 위해서는 loop 내에서 값이 변하는 변수(예, i, j, k, c)를 parallel for 구분 이후 private() 구문에 넣음으로써 각 코어에서 독립적으로 제어하도록 한다(Fig. 2). 또한, 클래스가 바뀐 화소의 수를 단순 합산하기 위한 변수인 nChanged은 모든 코어에서 동시에 접근할 필요가 있으므로 reduction(+:nChanged) 구문을 사용하여 false sharing을 방지한다. 그러나 각 클래스에 속한 화소의 수와 밴드별 중심값을 저장하는 포인터 배열인 nPixel[]와 Center[]의 값은 여러 코어에서 동시에 변경할 수 있으므로 여전히 false sharing을 발생시킬 수 있다. 이를 해결하기 위하여 Fig. 3과 같이 nPixel[]와 Center[]를 global 변수로 선언하고 nPixel_local[]과 Center_local[]을 각 코어마다 local 변수로 선언한다. 각 코어에서는 global 변수를 참조용으로만 사용하고 값을 변경시킬 필요가 있을 때에는 local 변수를 사용한다. 한 번의 분류가 끝나면 Fig. 4처럼 모든 코어의 local 변수를 global 변수에 합산하고 평균값을 계산함으로써 각 클래스의 중심값을 갱신한다(Han, 2017).

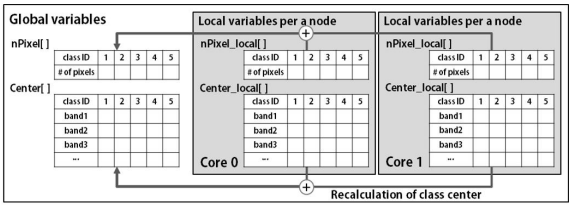

Fig. 4. Global and local declaration of nPixel[] and Center[] variables (modified from Han(2017))

```
for (i=0 to nClass) // through all classes
    for (j=0 to nThread) // through all cores
        for (k = 0 to nBand) // for all bands
            ...
        Center[i] += Center_local[i] // update global center using local ones
        nPixel[i] += nPixel_local[i] // update global no. of pixels using local ones
    Center[i] = Center[i]/nPixel[i] // recalculate class center
```

Fig. 4. Recalculation of class center (modified from Han(2017))

2.2.2 Inter-nodes 병렬처리

본 연구에서 inter-nodes 병렬처리는, ① 원영상을 1/*N* 크기로 분할하여 *N*개의 슬레이브 노드로 전송, ② 슬레이브 노드에서 계산된 밴드별 중심값을 취합, 새로운 클래스 중심값을 계산하여 슬레이브 노드에 전송, ③ 분류 iteration 수행, ④ 최종 분류 결과를 취합하여 결과 영상을 만드는 과정으로 구성한다(Fig. 5).

단계 ①에서는, 마스터 노드(master node)에 있는 전체 영상

을 N 분할하여 MPI를 통해 슬래이브 노드로 전송한다. 단계 ②에서는, 각 슬래이브 노드에서 영상의 최소값과 최대값을 탐색하여 마스터 노드로 전송한다. 마스터 노드에서는 전송된 최소값과 최대값들을 비교하여 전체 최소값과 최대값을 결정하고 초기 클래스 중심을 결정한 후 각 슬래이브 노드로 전송한다. 단계 ③에서는, 마스터 노드로부터 전송된 클래스 중심 정보를 이용하여 각 슬래이브 노드에서 분류를 수행한다. 모든 화소에 대해서 분류를 수행한 후 클래스 별 화소의 갯수($\#_{x,m}$)와 각 클래스의 밴드별 중심값($C^b_{x,m}$)을 마스터 노드로 전송한다. 마스터 노드에서는 클래스 별 중심값의 합을 구하고 밴드별 화소수의 합으로 나누어 새로운 클래스 중심($C^b_{x,new}$)을 결정한 후 슬래이브 노드로 전송한다(Eq. (1)).

$$C^b_{x,new} = \sum_{m=1}^{N} \left(C^b_{x,m} \cdot \#_{x,m} \right) / \sum_{m=1}^{N} \#_{x,m} \tag{1}$$

Where $C^b_{x,new}$ denotes a new center of band b of class x, $C^b_{x,m}$ denotes the center of band b of class x in the slave node m, $\#_{x,m}$ denotes the number of pixels classified to class x in the slave node m.

이와 같이 슬래이브 노드의 분류수행과 마스터 노드의 클래스 중심 재계산을 반복하여 총 클래스 이동 화소수가 임계값 이하일 경우 과정을 종료한다. 마지막으로 단계 ④에서는, 각 슬래이브 노드로부터 분류 결과를 전송 받아 마스터 노드에서 전체 결과 영상을 생성한다.

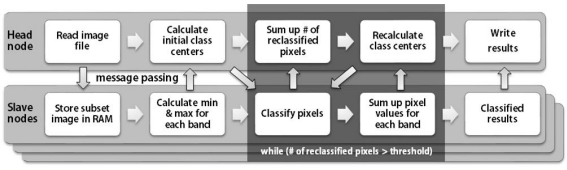

Fig. 5. Implementation of inter-nodes parallel processing

k-Means clustering은 사용자의 개입의 적은 자동화된 방식으로서 여러 번의 재실행을 통해 최적의 결과를 도출하는 것이 일반적이다. 즉, 클래스의 수(k)와 종료 조건을 다르게 하여 여러 번 실행할 수 있다. Fig. 5에서 영상 전송은 각 슬래이브 노드에 서로 다른 1/N 영상을 순차적으로 전송하는 unicating 방식을 사용하므로 네트워크 장치의 성능에 따라 많은 시간이 소요될 수 있다. 따라서 k-means clustering을 여러 번 실행하면 동일한 영상 전송 과정이 반복됨으로써 필요 이상의 많은 시간이 소요될 수 있다.

이러한 단점을 극복하기 위해서, 영상 입력 단계에서 마스터 노드의 전체 영상을 모든 슬래이브 노드에 전송하고 각 슬래이브 노드에서는 local 파일로 저장하는 과정을 전처리 과정으로 일회만 수행한다(Fig. 6). 전체 영상 전송은 broadcasting 방식인 MPI_Bcast 명령을 사용하므로 앞에서 1/N 영상을 순차적으로 전송하는 방식보다 긴 시간을 소요하지는 않는다. 따라서 Fig. 6에서는 k-means clustering을 여러 번 수행하더라도 추가적인 영상 전송은 발생하지 않는다. 각 슬래이브 노드에서는 1/N 크기의 영상을 local 파일로부터 읽어 분류를 수행하며 이후의 과정은 Fig. 5와 동일하다.

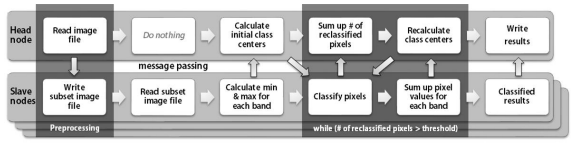

Fig. 6. Revised version of inter-nodes parallel processing

3. 적용 및 평가

3.1 적용 방법

본 연구에서는 병렬처리를 하지 않는 sequential 코드, intra-node 병렬처리만 하는 intra-node 코드, 그리고 intra-node와 inter-nodes 병렬처리를 함께 수행하는 hybrid 코드의 성능을 비교하였다. 실험에 사용한 PC-cluster는 한 대의 마스터 노드와 여덟 대의 슬래이브 노드로 구성되어 있으며 각 슬래이브 노드의 CPU에는 여덟 개의 다중코어가 장착되어 있다. 운영체제에 따른 성능을 평가하기 위하여 동일한 PC-cluster에 Microsoft Windows와 Canonical Ubuntu를 설치하여 동일한 실험을 하였다. Sequential 코드와 intra-node 코드는 마스터 노드에서 실험하였으며 hybrid 코드는 2, 4, 8개의 슬래이브 노드를 투입하여 실험하였다. 실험에 사용한 자료는 다중분광 위성영상인 LANDSAT 8 OLI 영상과 Sentinel 2A 영상이다. PC-cluster의 제원 및 k-means clustering 알고리즘의 요소는 각각 Tables 1 and 2와 같다. 영상의 제원 및 형태는 Table 3과 Figs. 7 and 8과 같다.

Table 1. Specifications of systems

CPU		AMD FX - 8150 eight cores @ 3.60GHz	
RAM	**Master node**	32 GB (DDR3)	
	Slave node	16 GB (DDR3)	
HDD		512 GB (7200 rpm)	
OS	**Master node**	Windows server 2008	Ubuntu desktop 18.04
	Slave node	Windows 7	Ubuntu server 18.04
Compiler		Visual studio 2010	g++ (mpicc)

Table 2. Parameters of k-means clustering

Number of classes (= k)	10
Max. number of iterations	100
Termination condition	1.5% of pixels moved to different classes

Table 3. Specifications of satellite images

Satellite sensor	LANDSAT 8 OLI	Sentinel 2A
Image size	868.38 MB	18.92 GB
Row	7841	30978
Column	7691	40980
Number of applied bands	7 (1,2,3,4,5,6,7)	8 (2,3,4,5,6,7,8,8A)
Ground sampling distance	30 m	10 m (5,6,7,8A resampled)
Radiometric resolution	16-bit	16-bit

Fig. 7. LANDSAT 8 OLI image

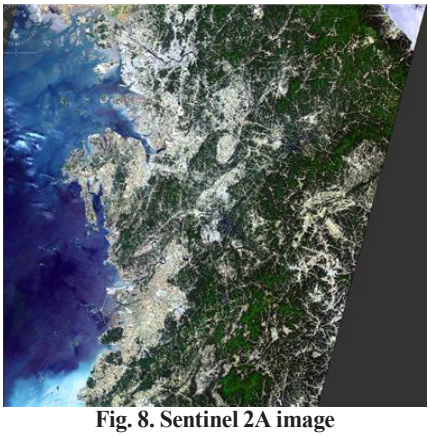

Fig. 8. Sentinel 2A image

3.2 평가 방법

일반적으로 병렬처리의 성능은 투입된 코어 또는 슬래이브 노드의 수와 처리 시간의 비율인 speedup과 efficiency로 평가한다(Eq. (2))(Eager *et al*., 1989). 이상적인 경우 speedup은 투입된 코어 또는 슬래이브 노드의 수와 같고 efficiency는 1이 되는데, 이는 투입된 코어 또는 슬래이브 노드의 수에 정비례하여 작업 속도가 증가했다는 것을 의미한다. 그러나 이러한 수치는 Amdahl's law(Gustafson, 2011)에서 알 수 있듯이 실제로는 달성하기 어렵다. 그 원인은 슬래이브 노드간 작업량 불평등과 네트워크 지연, 기타 운영체제에 의한 지연 등이다. 한편 efficiency가 1을 초과할 때가 있는데 이를 super-linearity라고 하며, 이는 한 개의 코어 또는 node로 처리하기 어려운 용량 또는 복잡도의 자료를 처리할 때 발생할 수 있다.

$$S_p = \frac{T_s}{T_p},\ E_p = \frac{T_s}{p \cdot T_p} \tag{2}$$

Where S_p and E_p denote speedup and efficiency using p cores or p slave nodes, T_s and T_p denote running times for sequential and parallel processing.

Sequential 코드, intra-node 코드, 그리고 hybrid 코드를 사용하여 영상 입력부터 결과 파일 출력까지의 시간을 측정하였다. Intra-node 코드의 speedup과 efficiency를 계산할 때에는 sequential 코드의 실행 시간을 T_s로 설정하였다. Hybrid 코드의 speedup과 efficiency를 계산할 때에는 sequential 코드의 실행 시간과 intra-node 코드의 실행 시간을 T_s로 설정하였다. 단, hybrid 코드의 speedup과 efficiency를 계산할 때에는 마스터 노드에서 슬래이브 노드로 영상을 전송하는 과정을 전처리로 여기고 그 시간은 포함시키지 않았다.

각 코드에 의한 결과의 무결성을 검증하기 위하여 분류 결과들이 서로 일치하는가를 비교· 평가하였으며, Windows 시스템의 결과와 Ubuntu 시스템의 결과도 마찬가지로 평가하였다. 이를 위하여 sequential 코드의 결과 영상과 병렬처리 결과 영상들을 차분하여 화소 단위로 비교하였다. 또한 분류 결과에서 각 클래스의 밴드별 중심값과 분류된 화소의 개수도 같은지 확인하였다.

3.3 평가 결과

세 가지 코드를 두 가지 위성영상에 적용한 결과는 Tables 4 and 5와 같다. 코어 또는 노드의 수가 증가함에 따라 speedup이 감소하여 코어 기준 speedup은 약 N/2-N/1.4, 노드 기준 speedup(괄호 안의 숫자)는 약 N/1.4-N으로 나타났다. 마찬가

Table 4. Running time, speedup, and efficiency for LANDSAT 8 OLI (number of iterations = 9)

Codes	Cores (nodes)	Windows system			Ubuntu system		
		Running time	Speedup	Efficiency	Running time	Speedup	Efficiency
Sequential	1	277.37	N/A	N/A	94.60	N/A	N/A
Intra-node	8 (1)	51.39	5.40	0.67	19.07	4.96	0.62
Hybrid	16 (2)	27.07	10.25 (1.90)	0.64 (0.95)	9.95	9.51 (1.92)	0.59 (0.96)
	32 (4)	14.03	19.77 (3.66)	0.62 (0.92)	5.42	17.45 (3.52)	0.55 (0.88)
	64 (8)	8.81	31.48 (5.83)	0.49 (0.73)	3.05	31.02 (6.25)	0.48 (0.78)

Table 5. Running time, speedup and efficiency for Sentinel 2A (number of iterations = 15)

Codes	Cores (nodes)	Windows system			Ubuntu system		
		Running time	Speedup	Efficiency	Running time	Speedup	Efficiency
Sequential	1	12210.51	N/A	N/A	3739.62	N/A	N/A
Intra-node	8 (1)	2053.53	5.95	0.74	874.30	4.28	0.53
Hybrid	16 (2)	1363.53	8.96 (1.51)	0.56 (0.75)	454.15	8.23 (1.93)	0.51 (0.96)
	32 (4)	684.56	17.84 (3.00)	0.56 (0.75)	232.39	16.09 (3.76)	0.50 (0.94)
	64 (8)	388.91	31.40 (5.28)	0.49 (0.66)	127.65	29.30 (6.85)	0.46 (0.86)

지로 코어 기준 efficiency는 약 0.5-0.7, 노드 기준 efficiency (괄호 안의 숫자)는 약 0.7-1.0으로 나타났다. 따라서 병렬처리에 의한 처리성능 향상이 분명히 있었다고 판단할 수 있다. Windows 시스템과 Ubuntu 시스템의 비교에서는 Ubuntu 시스템이 Windows 시스템에 비하여 전반적으로 2-3배의 빠른 속도를 나타내었다. LANDSAT 8 OLI 영상에서는 Windows 시스템과 Ubuntu 시스템 모두 efficiency가 잘 유지되는 반면, Sentinel 2A 영상에서는 Ubuntu 시스템에서 비교적 잘 유지되는 것으로 나타났다.

Tables 6 and 7은 Windows 시스템에서 sequential 코드와 hybrid 코드(여덟 개의 슬래이브 노드 투입)의 처리 시간을 파일 I/O와 분류 loop로 구분하여 비교한 것이다. 분류 loop는 hybrid 코드가 sequential 코드에 비해 38-50배의 속도를 나타내나 파일 I/O는 2-3배의 속도를 나타내고 있다. Sequential 코드에서는 파일 I/O가 2% 내외의 시간을 차지하고 있으나 hybrid 코드에서는 20-40%의 시간을 차지하고 있다. 결국 hybrid 코드에서는 속도 향상 효과가 적은 파일 I/O가 많은 비중을 차지함으로써 전체적인 속도 향상 효과가 줄어든다. 따라서 PC-cluster에서 파일 I/O 성능을 향상시킨다면 대용량 위성영상의 처리 성능을 보다 크게 향상시킬 수 있을 것으로 판단된다. 이를 위해서는 HDD대신 고속 SSD (Solid State Drive)를 사용하는 방법, 노드간 네트워크를 10Gbps 이상의 초고속 네트워크로 개선하는 방법을 고려할 수 있다.

Table 6. Running time comparison of sequential and hybrid codes for LANDSAT 8 OLI

	Sequential code (A)	Hybrid code (B)	A/B (speedup)
File I/O	5.30 (= 1.91%)	1.78 (= 20.20%)	2.75
Classification loop	272.07 (= 98.09%)	7.03 (= 79.80%)	38.70
Total	277.37	8.81	31.48

Table 7. Running time comparison of sequential and hybrid codes for Sentinel 2A

	Sequential code (A)	Hybrid code (B)	A/B (speedup)
File I/O	331.50 (= 2.71%)	152.31 (= 39.16%)	2.18
Classification loop	11879.01 (= 97.29%)	236.60 (= 60.84%)	50.21
Total	12210.51	388.91	31.40

마지막으로, sequential 코드, intra-node 코드, 그리고 hybrid 코드의 결과가 일치하는지 확인하였다. 각 클래스의 밴드별 중심값과 분류된 화소의 수를 비교한 결과 모든 정보가 일치하였고 분류 결과 영상간 차분 영상에서도 이상점을 발견할 수 없었다. 따라서 각 코드의 결과가 완벽히 일치함을 알 수 있었다. 세

영상의 분류 결과 영상은 Figs. 9 과 10과 같다.

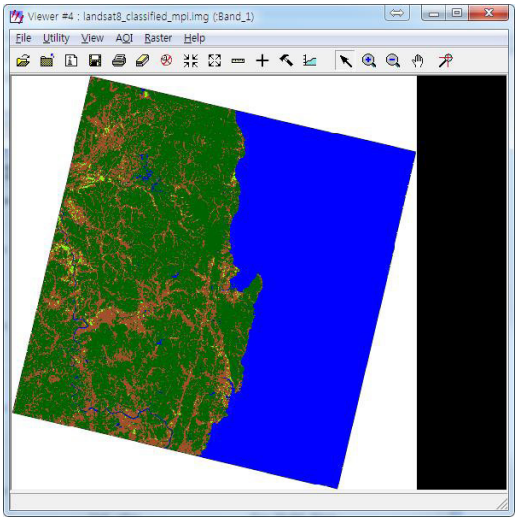

Fig. 9. Classification result of LANDSAT 8 OLI

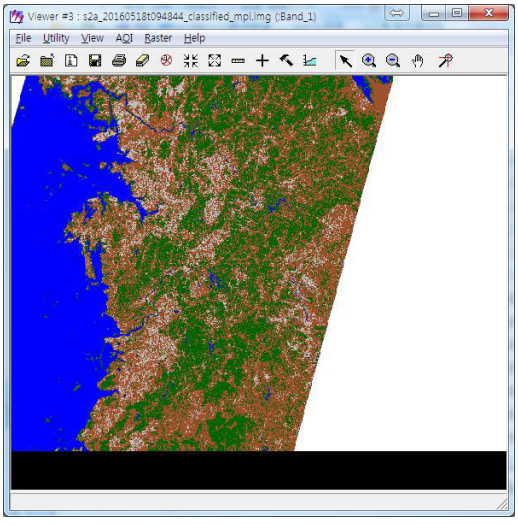

Fig. 10. Classification result of Sentinel 2A

4. 결 론

본 연구에서는 대용량 위성영상의 무감독분류를 위해 k-means clustering 알고리즘의 병렬처리 코드를 개발하여 PC-Cluster에서 구현하였다. 본 연구에서 제안한 병렬처리 방식은 CPU의 다중코어를 이용하는 intra-node 방식과 PC-cluster를 이용하는 inter-nodes 방식을 혼합한 hybrid 방식이다. 병렬처리의 성능을 평가하기 위하여 speedup과 efficiency를 비교·평가하였으며 결과의 무결성도 확인하였다. 한 대의 마스터 노드와 여덟 대의 슬레이브 노드로 구성된 PC-cluster에서 LANDSAT 8 OLI 영상과 Sentinel 2A 영상을 처리한 결과, *N*/2이상의 speedup과 0.5 이상의 efficiency를 나타내었다. PC-cluster의 운영체제로는 Microsoft Windows를 사용하는 것보다 Canonical Ubuntu을 사용하는 것이 2-3배 빠른 속도를 나타내었다. OpenMP를 이용하면 intra-node 병렬처리를 비교적 쉽게 구현할 수 있지만 false sharing을 억제하도록 코드를 설계해야 함을 확인할 수 있었다. Inter-nodes 병렬처리로 대용량의 위성영상을 처리할 때에는 파일 I/O의 부담이 커서 성능 향상에 한계가 있으므로 PC-cluster의 파일 I/O를 향상시키기 위한 방안을 강구해야 함을 알 수 있었다.

앞에서 언급하였듯이 현재까지 모든 알고리즘들을 병렬화할 수 있는 보편적인 기술은 존재하지 않는다. 따라서 개별 알고리즘들의 연산 부하와 구조적 특성을 고려하여 병렬처리 하드웨어에 적합한 병렬 코드를 구현해야 한다. 본 연구는 대용량 위성영상의 병렬처리를 위해 PC-cluster를 이용하였고 이를 위해 고려할 사항들을 제시한 것에서 의미를 찾을 수 있다고 판단된다. 향후, 다양한 위성영상 처리 알고리즘들의 병렬처리 코드를 구현하고 평가하는 연구를 수행하고자 한다. 또한 SSD와 10Gbps 이상의 네트워크 장비를 도입하여 PC-cluster 장비의 성능을 향상시키고자 한다.

감사의 글

본 과제는 행정안전부 재난안전 산업육성지원 사업의 지원을 받아 수행된 연구임(2019-MOIS32-015).

References

Argonne National Laboratory (2012), The message passing interface (MPI) standard, *Argonne National Laboratory*, https://www.mcs.anl.gov/research/projects/mpi/ (last date accessed: 30 August 2019).

Eager, D.L., Zahorjan, J., and Lazowska, E.D. (1989), Speedup versus efficiency in parallel systems, IEEE Transactions on Computers, Vol. 38, No. 3, pp. 408-423.

Fredj, H.B., Ltaif, M., Ammar, A., and Souani, C. (2017), Parallel implementation of Sobel filter using CUDA, *2017 International Conference on Control, Automation and Diagnosis (ICCAD)*, 19-21 January, Hammamet, Tunisia, pp. 209-212.

González, C., Resano, J., Mozos, D., Plaza, A., and Valencia, D. (2010), FPGA implementation of the pixel purity index algorithm for remotely sensed hyperspectral image analysis, EURASIP Journal on Advances in Signal Processing 2010, Vol. 2010:969806.

Gustafson, J.L. (2011), Amdahl's Law, In: Padua, D. (eds.), *Encyclopedia of Parallel Computing*, Springer, Boston, MA, USA, pp. 53-60.

Han, S. (2017), Parallel processing of k-means clustering algorithm for unsupervised classification of large satellite imagery, *Journal of the Korean Society of Survey, Geodesy, Photogrammetry, and Cartography*, Vol. 35, No. 3, pp. 187-194. (in Korean with English abstract)

IBM (2019), Parallel processing environments, *IBM Knowledge Center*, https://www.ibm.com/support/knowledgecenter/en/SSZJPZ_11.7.0/com.ibm.swg.im.iis.ds.parjob.dev.doc/topics/parallelprocessingenvironments.html (last date accessed: 19 November 2019).

Koo, I.H (2012), *High-speed processing of satellite image using GPU*, Master's thesis, Chungnam National University, Daejeon, Republic of Korea, 69p.

Lu, Y., Gao, Q., Chen, S., Sun, D., Xia, Y., and Peng, X. (2017), Fast implementation of image mosaicing on GPU, *2017 10th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)*, 14-16 October, Shanghai, China, pp. 1-5.

OpenMP ARB (2016), The OpenMP API specification for parallel programming, *OpenMP ARB*, http://www.openmp.org (last date accessed: 31 October 2019).

Plaza, A., Valencia, D., Plaza, J., and Martinez, P. (2006), Commodity cluster-based parallel processing of hyperspectral imagery, *Journal of Parallel and Distributed Computing*, Vol. 66, No. 3, pp. 345-358.

Sánchez, S. and Plaza, A. (2010), GPU implementation of the pixel purity index algorithm for hyperspectral image analysis, *2010 IEEE International Conference on Cluster Computing Workshops and Posters (CLUSTER WORKSHOPS)*, 20-24 September, Heraklion, Crete, Greece, pp. 1-7.

Sugumaran, R., Hegeman, J.W., Sardeshmukh, V.B., Armstrong, M.P., Hegeman, J.W., Sardeshmukh, V.B., and Armstrong, M.P. (2018), *Processing remote-sensing data in cloud computing environments, In Remote Sensing Handbook - Three Volume Set*, CRC Press, Boca Raton.

Sun, X., Li, M., Liu, Y., Tan, L., and Liu, W. (2000), Accelerated segmentation approach with CUDA for high spatial resolution remotely sensed imagery based on improved Mean Shift, *2009 Joint Urban Remote Sensing Event*, 20-22 May, Shanghai, China, pp. 1-6.

Wang, P., Wang, J., Chen, Y., and Ni, G. (2013), Rapid processing of remote sensing images based on cloud computing, *Future Generation Computer Systems*, Vol. 29, No. 8, pp. 1963-1968.