3차원 공간정보 시스템을 위한 병렬 알고리즘

조 정 우† · 김 진 석††

요 약

3D 공간정보를 이용하여 3D 이미지를 처리하는 시스템이 많이 상용화되어 있다. 기존에 3D 이미지를 처리하기 위한 방법으로 고성능의 시스템을 이용하거나 이미지 압축 기술을 사용하였다. 하지만 고성능의 시스템을 사용하여 GIS 시스템을 구현할 경우 가격의 부담이 크다는 문제점이 있고 이미지 압축 기술을 사용하여 GIS 시스템을 구현할 경우 원 이미지에 손실이 크다는 문제점이 있다. 또한 일반 시스템에서 3D 이미지를 처리하려면 3D 이미지의 파일의 크기가 크기 때문에 공간 이미지를 처리하는데 시간이 오래 걸린다는 단점이 있다. 따라서 본 논문에서는 3D 이미지를 병렬로 처리하여 디스플레이 시간을 단축하는 병렬 알고리즘을 제안한다. 본 논문에서 제시된 병렬 알고리즘은 3D 이미지를 다수의 노드로 분할하여 각 노드에서 이미지를 화면에 디스플레이 하는 방법을 사용한다. 병렬컴퓨터의 노드의 수가 증가함에 따라 제안된 알고리즘의 성능이 증가함을 실험을 통해 보였다.

A Parallel Algorithm for 3D Geographic Information System

Jeong Woo Jo† · Jin Suk Kim††

ABSTRACT

Many systems handle 3D-image were used. High-performance computer systems and techniques of compressing images to handle 3D-image were used. But there will be cost problems, if GIS system is implemented, using the high-performance system. And if GIS system is implemented, using the techniques of compressing images, there will be some loss of a image. It will take a long processing time to handle 3D images using a general PC because the size of 3D-image files are very huge. The parallel algorithm presented in the paper can improve speed to handle 3D-image using parallel computer system. The system uses the method of displaying images from nodes to screens, dividing a 3D-image into multiple sub images on multiple nodes. The performance of the presented algorithm showes improving speed by experiments.

키워드 : 병렬 컴퓨터 알고리즘(Parallel Computer Algorithm), 래스터 이미지(raster image), 3D GIS, 컴퓨터 시스템(computer system)

1. 서 론

GIS(Geographic Information System)를 이용하여 3D(Dimension) 이미지를 처리하는 시스템이 많이 상용화되어 있다[12-15]. 하지만 이러한 시스템들에서 3D 이미지를 빠르게 처리하는데는 한계가 있다. 기존에 3D 이미지를 처리하기 위한 방법으로 고성능의 시스템을 이용하거나 이미지압축 기술을 사용하였다. 고성능의 시스템을 사용하여 GIS 시스템을 구현할 경우 가격의 부담이 크다는 문제점이 있으며 3D 이미지를 처리하는 시간을 줄이기에는 한계가 있다. 이미지압축 기술을 사용하면 원 3D 이미지의 손실이 발생한다는 문제점이 있다. 또한 이러한 3D 이미지를 일반 시스템에서 처리하려면 3D 이미지의 파일의 크기가 크기 때문에 공간이미지를 처리하는데 시간이 오래 걸린다는 문제점이 있다. 따라서 본 논문에서는 가격 대비 성능에서 큰 효과를 볼 수 있는 3D GIS를 위한 병렬화 방법을 제안한다. GIS 시스템을 병렬화하면 3D 이미지를 디스플레이할 때 이미지의 손실이 없고 3D 이미지를 빠르게 디스플레이할 수 있다.

3D 이미지를 처리하는 방법은 크게 GIS의 연산을 병렬화 하는 방법과, 3D 이미지의 이미지 처리 과정을 병렬화 하는 방법으로 나눌 수 있다. GIS의 연산을 병렬화 하는 방법은 여러 연구들에서 다루어 왔다[1, 11]. 또한 이미지 처리 과정을 병렬화 하는 방법에 대해서도 연구가 이루어졌다[2]. 3D 이미지를 처리할 때 GIS의 연산 과정 못지 않게 디스플레이 과정의 수행시간이 오래 걸린다. 하지만 기존 연구는 GIS의 연산만을 병렬화 하거나 이미지를 다루더라도 평면 이미지만을 다루었기 때문에 3D GIS에 적용이 쉽지 않다. 따라서 본 논문에서는 디스플레이 과정의 수행시간을 줄이기 위한 병렬화 방법론을 제안한다.

본 논문에서는 3D 이미지를 병렬화할 때 이미지의 손실

※ 이 논문은 2001년도 서울시립대학교 학술연구조성비에 의하여 연구되었으며, 이에 감사드립니다.
† 준 회 원 : 서울시립대학교 대학원 컴퓨터 · 통계학과
†† 정 회 원 : 서울시립대학교 컴퓨터 · 통계학과 교수
논문접수 : 2002년 2월 2일, 심사완료 : 2002년 5월 10일

을 줄이기 위한 이미지 분할 방법 3가지를 제안하고 이 방법들을 비교 평가하여 가장 좋은 이미지 분할 방법을 선택한다. 또한 이미지 분할방법을 확장하여 고정 분할방법을 사용한 알고리즘과 가변 분할방법을 사용한 알고리즘으로 구분하여 실험하였다.

본 논문의 구성은 다음과 같다. 제 2장에서는 3D 이미지를 처리하는 기존의 GIS 시스템과 병렬 GIS 시스템 및 관련 연구들에 대해 살펴보고, 이러한 방법들의 특징과 문제점을 검토한다. 제 3장에서는 3D 이미지 분할 방법에 대해 설명하고, 각 방법들의 특징을 살펴본다. 제 4장에서는 실험을 통하여 고정 이미지 분할 방법과 가변 이미지 분할방법의 수행시간을 비교 분석한다. 마지막으로 제 5장에서는 결론 및 향후 연구방향을 고찰한다.

2. 관련 연구

2.1 GIS의 병렬화 방법

병렬 GIS 응용을 효율적으로 처리할 수 있는 분산 공유 메모리(DSM : Distributed Shared Memory) 기반 병렬처리 시스템이 제안되었다[1]. 이 시스템에서는 GIS에서의 공간 객체를 공유의 기본단위로 설정하였으며 네트워크의 오버헤드를 줄이기 위하여 복수의 객체를 한번에 읽어오는 bulk access가 가능하도록 하였다. 또한 GIS를 병렬화 하기 위해 GIS 연산중 상대적으로 연산비용이 많이 드는 spatial join 연산을 병렬화의 대상으로 하였다. spatial join의 예는 "미국에서 도시 내에 위치한 숲을 모두 찾아라" 와 같은 형태가 있다. 이 예에서는 '숲'과 '도시'에 해당되는 두 가지 종류의 공간객체들에 대해 join 작업을 수행한다.

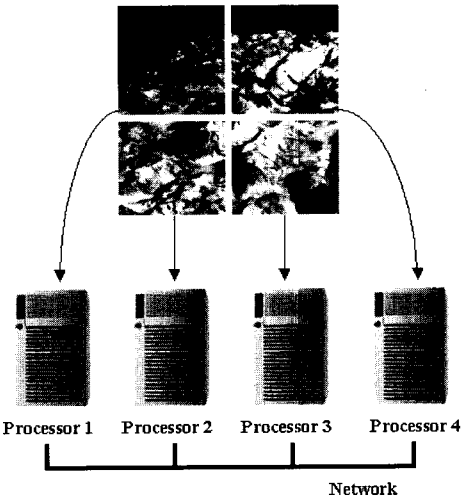
이 연구에서는 DSM 기반 병렬처리 시스템과 MPI(Message-Passing Interface)[17]를 사용한 병렬처리 시스템 두 가지를 놓고 실험을 하였고 전반적으로 이 연구에서 구현한 방법이 MPI를 사용한 시스템보다 빠른 성능을 보인다. 하지만 이 연구는 GIS의 3D 이미지를 병렬로 처리하는 것이 아니라 GIS의 연산만을 다루었다. 따라서 디스플레이 하는데 시간이 오래 걸리는 3D 이미지 같은 작업에 이 연구를 적용하기에는 적당하지 않다.

또 다른 연구로는 볼륨렌더링을 병렬화 하는 방법에 대한 것이 있다[2]. 이 연구에서는 병렬 연산, 병렬 입출력, 병렬 연산 및 병렬 입출력, 순차 볼륨렌더링 이렇게 네 가지 경우를 직접 구현하고 성능비교를 하였다.

이 연구에서 각 노드에 작업을 할당하는 방법은 이미지 분할 방법을 사용하는데, 고정크기의 정방형 타일 모양으로 분할한다. 하지만 이 연구는 평면 이미지일 경우에 고정크기의 정방형 타일 모양으로 분할 할 수 있다. 3D 이미지 같은 경우에는 평면 이미지가 아니기 때문에 위의 방법을 적용하면 이미지의 손실이 생길 수 있으므로 3D GIS에 직접응용할 수 없다.

2.2 이미지 병렬화 방법

병렬 GIS 시스템에서 3D 이미지를 처리하는 경우를 살펴보면 (그림 1)과 같다. (그림 1)에서 보듯이 병렬 GIS에서는 하나의 3D 이미지를 분할하여 여러 프로세서에서 실행을 시킨다. 이렇게 여러 프로세서에서 3D 이미지를 처리하면 하나의 프로세서를 이용하는 기존의 GIS 시스템보다 빠른 시간에 3D 이미지를 처리할 수 있다. 이 병렬 GIS는 다수의 프로세서를 이용하기 때문에 꼭 고성능의 시스템을 사용하지 않더라도 고속으로 3D 이미지를 처리 할 수 있다는 장점이 있다. 하지만 병렬 GIS를 사용하여 3D 이미지를 디스플레이 하려면 추가로 이미지 분할 방법에 관한 연구가 있어야 한다. 그렇지 않으면 3D 이미지를 병렬로 디스플레이 할 때 이미지의 손실이 발생할 수 있다. 또한 분할된 이미지들을 다시 하나로 합치는 과정도 필요할 수 있다.



(그림 1) 병렬 시스템을 이용한 3D 이미지처리

3. 병렬 컴퓨터 시스템에서의 이미지 분할 방법

3.1 GIS에서의 래스터 형식

GIS에는 3D 이미지를 표시하기 위한 여러 가지의 디스플레이 형식이 있다. 예를 들어 래스터(raster), 벡터(vector), 이미지(image), 사이트(sites)등의 형식이 있다. 본 논문에서는 이런 디스플레이 형식 중에서 래스터 형식을 사용하기로 한다. 래스터는 화면공간상에 x, y, z 좌표정보를 가진 이미지이며 픽셀(pixel)이라고 하는 작은 사각형들로 이루어진다.

래스터 이미지를 3D GIS를 사용하여 디스플레이 하는 방법은, 이미지를 한 줄씩 왼쪽에서 오른쪽으로 선을 그리는 것이다. 이때 그려지는 방법은 다음과 같이 두 점의 좌표값을 연결한 선을 그리는 것이다.

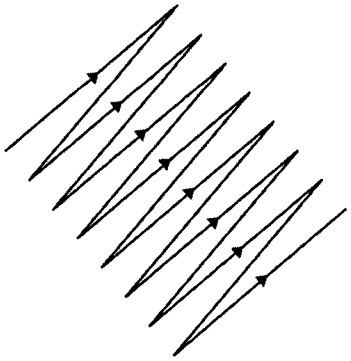
draw_line(x1, y1, x2, y2)

본 논문에서 제안하는 병렬 알고리즘은 이 래스터 이미지를 분할하여 디스플레이 부분을 병렬화 하는 것이다.

3.2 이미지 분할 방법

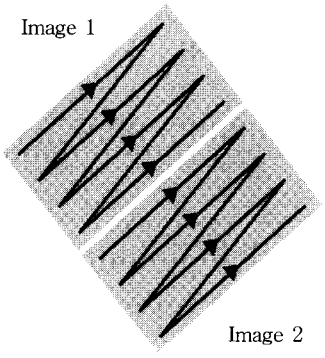
3D 이미지를 손실없이 병렬로 디스플레이 하기 위해서는 이미지 분할 방법이 필요하다. 따라서 본 논문에서는 래스터 이미지를 분할하는 방법을 크게 3가지로 나누었다. 수평선으로 나누는 방법과, 수직선으로 나누는 방법, 그리고 마지막으로 대각선으로 나누는 방법이 그것이다. (그림 3), (그림 4), (그림 5)에서 3D 이미지를 분할하는 방법을 보여주고 있다.

(그림 2)에서는 실제 GIS에서 래스터 3D 이미지를 디스플레이 하는 방법을 보여준다. 한 줄 단위로 왼쪽에서 오른쪽으로 디스플레이하고 원 이미지를 3D로 보여주기 위해 약간의 각도를 가지고 이미지가 기울어져 있다. (그림 3)은 래스터 이미지를 디스플레이 할 때 위와 아래로 분할하여 각각을 디스플레이 하는 방법이다. 예를 들면 (그림 3)의 1번 이미지를 프로세서 1에 할당하고 2번 이미지를 프로세서 2에 할당하여 디스플레이하게 된다. (그림 4)는 그리는 행(줄)을 기준으로 수직으로 분할하여 각 프로세서에 할당하는 방법이고, (그림 5)는 사람의 시선으로 봤을 경우 수직으로 3D 이미지를 분할하는 방법을 사용한다.

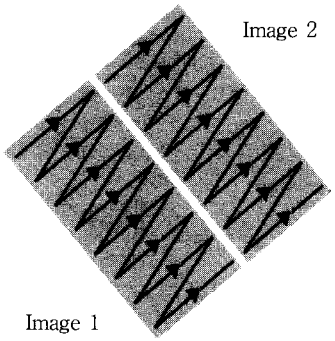


(그림 2) 분할 전 래스터 이미지

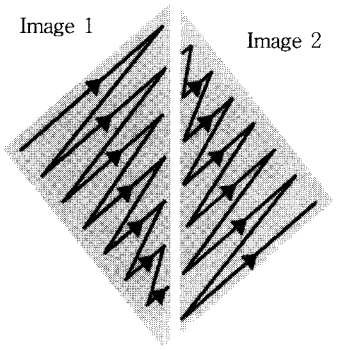
3.3 3D 이미지 분할 방법의 문제점



(그림 3) 수평선 분할



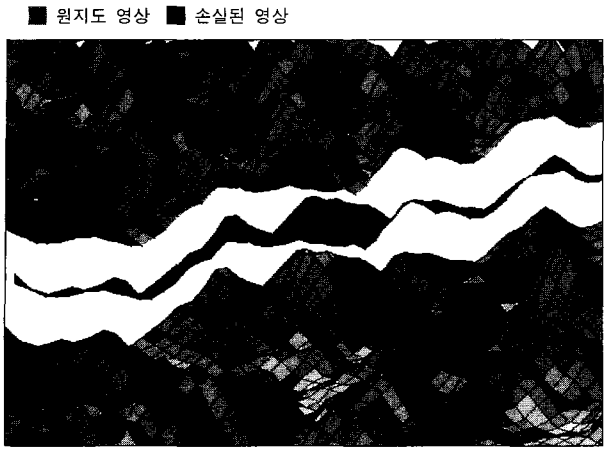
(그림 4) 수직선 분할



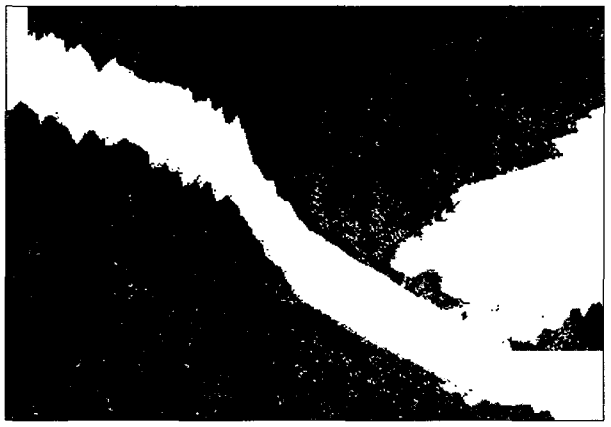
(그림 5) 대각선 분할

(그림 3)에서 처럼 이미지를 수평선으로 분할 할 경우 3D 이미지를 디스플레이 하는데 문제가 발생한다. 예를 들어, 2개의 이미지로 분할 한 경우를 살펴보면, 3D 이미지를 두개의 프로세서가 거의 동시에 제일 위쪽에서 시작해서 왼쪽에서 오른쪽으로 이미지를 디스플레이 하게 된다. 그런데 이미지의 디스플레이가 완료되었을 경우에 위의 이미지 1이 아래의 이미지 2를 덮어쓰면서 디스플레이 하게 된다. 즉 뒤에 있어야 할 이미지가 앞으로 나오는 경우가 발생한다. 이 현상은 이미지가 3D이기 때문에 발생하는 것이다. 이렇게 되면 원 3D 이미지와 다른 결과를 디스플레이 하게 된다. (그림 6)에 3D 이미지를 수평선으로 분할하여 디스플레이 할 경우에 문제가 되는 것을 보여주고 있다. 그림 가운데 파란 부분이 수평선으로 분할하여 디스플레이 할 때 위의 이미지가 아래의 이미지를 덮어쓴 부분이다. 이렇게 되면 원래의 래스터 이미지에 손실을 가져오게 된다.

이러한 문제점은 (그림 4)와 (그림 5) 같은 이미지 분할 방법을 사용하면 제거할 수 있다. (그림 4)와 (그림 5) 둘 다 뒤의 이미지가 앞의 이미지에 영향을 주지 않는다. 하지만 디스플레이 하는 방법을 보면 (그림 4)와 같이 단순히 행으로 분할하는 것에 비해 (그림 5)와 같이 이미지를 분할할 때 더 많은 연산이 필요하다. (그림 4)는 행을 프로세서의 수에 따라 나누면 되지만, (그림 5)는 행과 열을 모두 줄여가면서 화면을 분할하는 연산이 추가된다. 즉 대각선분할 방법이 수직선 분할 방법보다 더 많은 디스플레이 시간을 소



(그림 6) 수평선 분할의 이미지 손실



(그림 7) 수직선 분할 이미지

비하게 된다. 따라서 본 논문에서 구현한 3D GIS를 위한 병렬 알고리즘에서는 (그림 4)의 수직선 분할 방법을 이용한다.

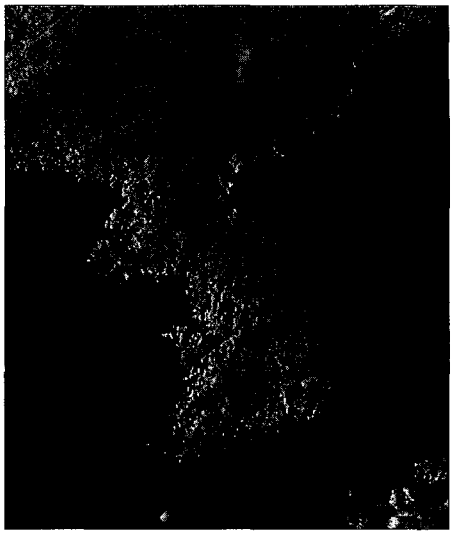
(그림 4)는 래스터 이미지가 두 점의 선을 연결하기 때문에 한 줄에서는 겹쳐지지 않는다는 특징을 이용한 것이다. (그림 7)에서 수직선으로 분할한 3D 이미지를 보여주고 있다.

이와 같이 3D 이미지를 분할하면 뒤의 이미지가 영향을 받지 않기 때문에 여러 프로세서가 이미지를 그려도 원 3D 이미지를 정확히 만들어 낼 수 있다.

4. 실험 및 분석

본 논문에서의 실험을 위해 10개의 노드를 갖는 병렬 클러스터링 시스템을 사용하였다. 또한 각 노드는 리눅스(Linux)를 기반으로 100Mbps Ethernet으로 연결된다. GIS 프로그램은 리눅스에서 동작하는 GRASS라는 프로그램을 사용하였다[16]. GRASS는 USA-CERL(미국 공병대)에서 개발되었고 소스 코드가 개방된 GIS 프로그램이다.

본 실험에 사용된 래스터 이미지는 (그림 8)에 나와있는 KDEM이라는 한반도 이미지와 (그림 9)에 있는 Aspect라는 지형 이미지이다. (그림 8)의 KDEM 이미지는 약 2Mbyte의 크기를 가진 3D 컬러 이미지고 Aspect 이미지는 약 1Mbyte의 크기를 가진 3D 흑백 이미지이다.



(그림 8) KDEM 이미지

(그림 9) Aspect 이미지

<표 1>과 <표 2>는 KDEM과 Aspect 이미지를 수직선 분할 방법을 사용하여 병렬로 디스플레이 한 수행시간을 측정한 결과이다. 이 실험에서는 3D 이미지를 분할할 때, 행을 기준으로 노드의 수에 따라 고르게 분할하였다. 즉 행을 노드의 개수를 이용하여 나누는 방법을 사용하였다.

다음의 <표 1>, <표 2>와 같이 노드의 수가 많아질수록 각 이미지를 디스플레이 하는 시간이 감소하는 것을 알 수 있다. 표에 나와있는 t_i는 i번째 노드에서의 디스플레이 시간이다. 또한 단일노드에서의 수행시간 대 p개의 노드에서의 수행시간의 비율인 Sp(Speedup)는 다음과 같이 정의한다.

$$Sp = \frac{T_1}{T_p}$$

T_1 : 1개 프로세서에서 이미지를 디스플레이 하는데 걸리는 시간

Tp : p개의 프로세서에서 이미지를 디스플레이 하는데 걸리는 시간

<표 1>, <표 2>에서 알 수 있듯이 각 노드들마다 시간의 차이가 생기게 되는데 이때 가장 오랜 시간이 걸리는 노드의 시간에 최종 디스플레이 시간이 결정되기 때문에 T_p는 가장 시간이 오래 걸리는 노드의 시간에 의하여 결정된다.

〈표 1〉 KDEM 이미지의 고정분할

노드수	각 노드에서의 수행시간(초)	Sp
1	t_1 : 45.4	1
2	t_1 : 24.3 t_2 : 22.0	1.86
3	t_1 : 8.9 t_2 : 30.9 t_3 : 8.3	1.46
4	t_1 : 4.3 t_2 : 21.4 t_3 : 19.6 t_4 : 4.0	2.12
5	t_1 : 2.6 t_2 : 12.7 t_3 : 20.4 t_4 : 11.5 t_5 : 3.0	2.22
6	t_1 : 2.1 t_2 : 8.4 t_3 : 16.6 t_4 : 14.6 t_5 : 6.9 t_6 : 2.6	2.73
7	t_1 : 1.9 t_2 : 5.3 t_3 : 11.7 t_4 : 14.8 t_5 : 11.5 t_6 : 4.6 t_7 : 2.4	3.05
8	t_1 : 1.8 t_2 : 4.0 t_3 : 8.4 t_4 : 13.4 t_5 : 11.9 t_6 : 8.8 t_7 : 3.4 t_8 : 2.2	3.38
9	t_1 : 1.7 t_2 : 3.1 t_3 : 7.1 t_4 : 10.5 t_5 : 11.9 t_6 : 9.8 t_7 : 6.5 t_8 : 2.8 t_9 : 2.1	3.79
10	t_1 : 1.7 t_2 : 2.5 t_3 : 5.7 t_4 : 8.2 t_5 : 11.1 t_6 : 9.6 t_7 : 8.2 t_8 : 4.7 t_9 : 2.3 t_{10} : 1.9	4.06

t_i : i 번째 노드에서의 수행시간

〈표 2〉 Aspect 이미지의 고정분할

노드수	각 노드에서의 수행시간(초)	Sp
1	t_1 : 69.5	1
2	t_1 : 34.0 t_2 : 35.9	1.94
3	t_1 : 20.4 t_2 : 27.1 t_3 : 22.5	2.56
4	t_1 : 14.3 t_2 : 20.2 t_3 : 19.7 t_4 : 16.6	3.43
5	t_1 : 11.1 t_2 : 14.9 t_3 : 16.5 t_4 : 15.1 t_5 : 13.0	4.19
6	t_1 : 8.9 t_2 : 12.0 t_3 : 14.1 t_4 : 13.5 t_5 : 12.1 t_6 : 10.9	4.90
7	t_1 : 7.5 t_2 : 9.7 t_3 : 11.8 t_4 : 11.7 t_5 : 11.6 t_6 : 10.5 t_7 : 9.6	5.89
8	t_1 : 6.8 t_2 : 8.7 t_3 : 10.6 t_4 : 10.6 t_5 : 10.4 t_6 : 9.6 t_7 : 9.0 t_8 : 8.1	6.54
9	t_1 : 5.9 t_2 : 7.3 t_3 : 8.6 t_4 : 9.5 t_5 : 9.1 t_6 : 9.2 t_7 : 8.2 t_8 : 7.8 t_9 : 7.1	7.31
10	t_1 : 5.2 t_2 : 6.2 t_3 : 7.3 t_4 : 8.3 t_5 : 8.4 t_6 : 8.2 t_7 : 8.0 t_8 : 7.3 t_9 : 6.9 t_{10} : 6.4	8.21

t_i : i 번째 노드에서의 수행시간

그런데 <표 1>을 살펴보면 노드를 여러개로 나누었을 때, 각 노드마다 수행시간의 차이가 크다는 것을 알 수 있다. 이것은 각 노드마다 할당된 3D 이미지에 차이가 있기 때문이다. 즉, (그림 8)과 같은 3D 이미지를 분할하였을 때 그려야 할 좌표가 많은 이미지를 가진 노드는 그려야 할 좌표가 적은 이미지를 가진 노드보다 총 수행시간이 오래 걸리게 된다. 그래서 전체적인 병렬 수행시간을 줄이려면 각 노드에 이미지를 할당할 때, 노드의 수에 따라 고정으로 이미지를 분할하는 것이 아니라 이미지를 그려야할 좌표에 따라 고르게 분할하여야 한다. 따라서 고정분할이 아닌 가변분할 방법을 제안한다.

3D 이미지 분할을 고정으로 하지 않고 가변으로 했을 경우의 실험결과는 (그림 12), (그림 13)에 그래프를 이용하여 나타내었다. 가변길이 분할은 래스터 이미지의 좌표값이 NULL인 것은 그 좌표에 그림을 그리지 않는다는 특성을 이용하여 분할하였다. 즉 모든 이미지는 행과 열으로 구성되는데 열 단위로 묶어서 좌표값이 NULL이 아닌 것들을 모두 합하여 전체좌표들의 총합에서 뺀다. 그러면 각 열마다 일정값을 갖게 되는데 이 값들을 모두 합하여 분할하고자하는 노드의 수로 나눈다. 그러면 한 노드에서 그려야하는 좌표의 개수를 알 수 있다. 하지만 본 논문에서는 이미지를 열 단위로 나누었기 때문에 위에서 나눈 수에 근접한 열으로 이미지를 분할하였다. 분할 식은 다음과 같다. 아래의 분할식에서 P(N)은 N개의 노드 중에서 하나의 노드에서 그려야 하는 좌표의 개수이다. 본 논문에서 제안하는 알고리즘에서는 이미지를 열 단위로 분할하기 때문에 실제로 하나의 노드에서 그려야 하는 좌표의 개수는 P(N)과 근사한 값이다.

$$p(N) = (P - P_0) / N$$

P : 전체 좌표들의 총 개수
P_0 : (0, 0)인 좌표들의 총 개수
N : 총 노드의 수
p(N)에 근접한 열(column)으로 이미지를 분할

본 논문에서 제안하는 병렬 알고리즘 중 각 노드에 분할될 열의 갯수를 계산하는 알고리즘 Partition_Image는 다음 (그림 10)과 같다. 알고리즘의 변수중에서 tot[ ]는 하나의 열에서 그려야할 좌표의 개수이고, total1은 전체 3D 이미지에서 그려야할 좌표의 개수이다. 또한 total2는 전체 3D 이미지에서 NULL값을 갖는 좌표의 개수이다. 또한 get_corners는 행값을 가지고 각 열을 읽어오는 함수이다.

(그림 11)은 (그림 10)에서 분할한 행을 이용하여 각 노드가 병렬로 디스플레이하는 알고리즘이다. 이 알고리즘은 각 노드마다 Partition_Image에서 구한 br[ ] 값의 열만큼 화면에 디스플레이 한다.

Algorithm Partition_Image(N, cols, rows, br)

Input : ***N(노드의 수), cols(이미지의 열 값), rows(이미지의 행 값)***
Output : ***br[ ] (각 노드에서 그려야하는 열 값)***

```
for ( i = 0 ; i <= rows ; i++ ) {
    get_corners( i, &x1, &y1 ) ;
    for ( j = 0 ; j <= cols ; j++ ) {
        tot[ j ] = tot[ j ] + 1 ;
        total1 = total1 + 1 ;
        if ( x1[ j ] = 0 && y1[ j ] = 0 ) {
            tot[ j ] = tot[ j ] - 1 ;
            total2 = total2 + 1 ;
        }
    }
}
j = 0 ; k = 0 ;

for ( i = 0 ; i <= cols ; i++ ) {
    j = j + tot[ i ] ;
```

```
        if ( j > (( total1 - total2 ) / N )) {
            br[ k ] = i ;
        }
        j = 0 ; k++ ;
    }
    return br ;
```

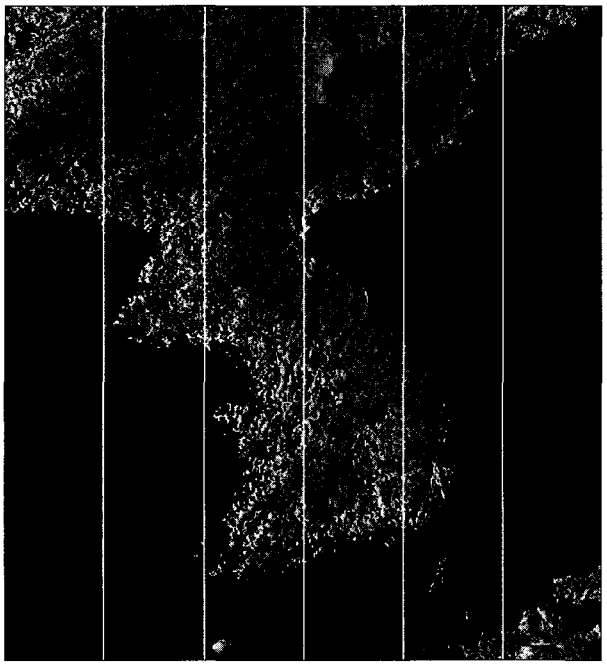
(그림 10) 분할할 열 계산 알고리즘

```
Algorithm Parallel_Display ( br[ ], x10[ ], y1[ ] )
Input : br[ ](각 노드에서 그려야하는 열 값),
        x1[ ](좌표의 X 값), y1[ ](좌표의 Y 값)

    if ( MY_ID = 0 )                // node 0
    {
     for ( m = 0 ; m < br [ MY_ID ] ; m++ )
          draw_line(x1 [ m ], y1 [ m ], x1 [ m+1 ], y1 [ m+1 ]) ;
    }
    else                            // node1에서 node N
    {
     for ( m = br [ MY_ID-1 ] ; m < br [ MY_ID ] ; m++ )
          draw_line( x1 [ m ], y1 [ m ], x1 [m+1], y1 [ m+1 ] ) ;
    }
```

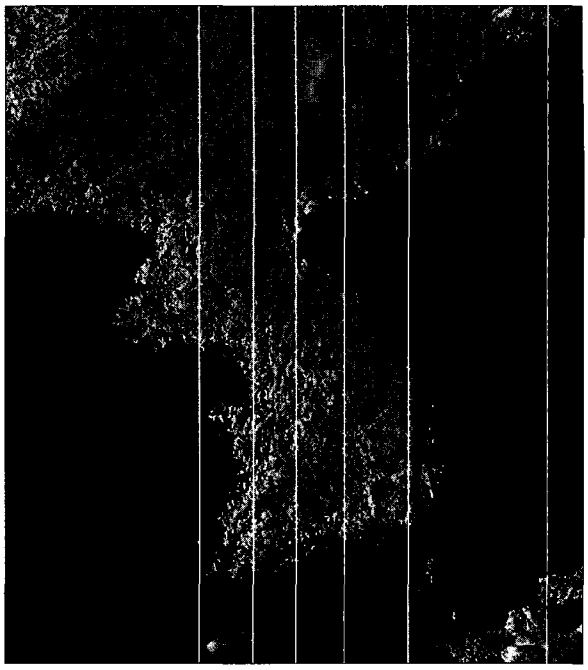
(그림 11) 병렬화 된 디스플레이 알고리즘

(그림 12)와 (그림 13)이 고정분할을 사용하여 이미지를 분할한 경우와 위의 가변 분할 알고리즘을 사용하여 분할한 이미지이다.



(그림 12) KDEM 이미지의 고정분할 영역

KDEM 이미지 (그림 8)를 살펴보면 디스플레이 할 이미지가 좌·우보다는 가운데에 집중되어 있다는 것을 알 수 있다. 따라서 고정분할 방법으로 이미지를 분할했을 경우, 즉 한 행을 정확히 n등분 (n : 노드의 수) 했을 경우 노드



(그림 13) KDEM 이미지의 가변분할 영역

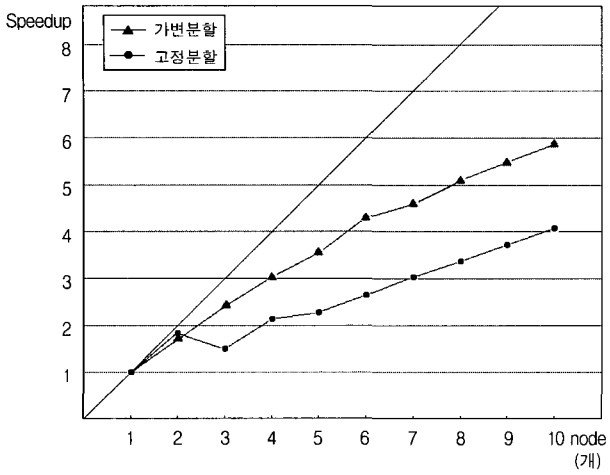
가 많아질 수록 처음과 끝 쪽의 노드에서 수행시간이 빠르게 나타나게 된다. 예를 들어 6개의 노드로 분할했을 경우 고정길이 분할에서는 1번과 6번의 노드에 비해 3번 4번의 노드는 그려야할 이미지가 많아지게 되어 느린 디스플레이 시간을 보인다.

이 방법을 수정하여 가변 길이 알고리즘으로 나누었을 경우 속도는 더욱 향상되었다. 아래의 (그림 14)와 (그림 15)는 고정 길이와 가변 길이로 이미지를 분할했을 때의 결과이다. 결과적으로 4개의 노드인 경우 KDEM 이미지에서 고정 길이 분할으로 이미지를 분리했을 경우 최대 21.4의 속도로서 약 2.12배의 속도향상을 보이지만 가변 길이 분할을 사용하여 이미지를 분할했을 경우 최대 14.8의 속도로서 약 3.06배의 속도향상을 보인다. 6개의 노드에서는 2.73배에서 4.22배의 성능향상을 보인다.

하지만 Aspect 이미지 (그림 9)에서는 고정보다 가변 분할의 Speedup이 조금 낮다는 것을 알 수 있다. 이것은 (그림 9)에서도 알 수 있듯이 이미지가 좌·우와는 상관없이 고르게 분포되어 있기 때문이다. 따라서 디스플레이 시간에 앞의 가변 분할 알고리즘을 계산하는 오버헤드가 추가되어 고정 길이 분할보다 Speedup이 낮게 된다. 하지만 그래프에서 볼 수 있듯이 Speedup에서 큰 차이는 나지 않는다.

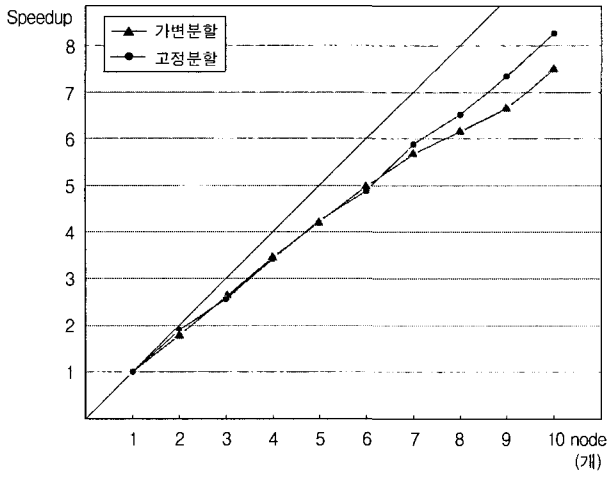
다음의 (그림 14), (그림 15)는 가변 분할과 고정 분할을 했을 경우 노드의 수에 따른 Speedup을 보여주고 있다. 그래프를 보면 좀더 확실히 노드의 수가 증가할수록 3D 이미지를 디스플레이 하는 속도가 빨라진다는 것을 알 수 있다.

(그림 14)는 KDEM 이미지를 병렬화 할 때 노드의 수에 따른 Speedup을 보여준다. 가운데 대각선으로 있는 실선이



(그림 14) KDEM 이미지의 Speedup

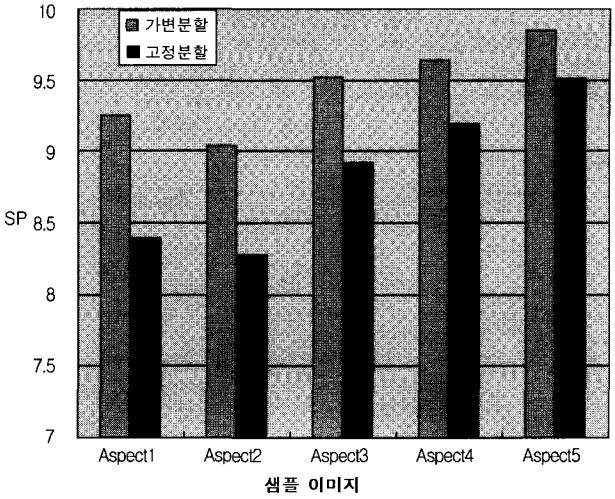
병렬화를 했을 때 이상적으로 기대되는 Speedup이다. 하지만 이미지를 분할하는 오버헤드와 마스터 노드가 각 슬레이브 노드에 정보를 보내는 시간 때문에 이상적인 Speedup보다는 낮은 Speedup을 보여준다. 가운데 원형으로 되어 있는 선이 고정 분할 방법을 적용한 병렬화의 Speedup이고, 삼각형으로 되어 있는 선이 가변 분할 방법을 적용한 병렬화의 Speedup이다.



(그림 15) Aspect 이미지의 Speedup

(그림 15)는 Aspect 이미지를 병렬화 했을 때의 Speedup 그래프로서 고정 분할과 가변분할 방법을 적용하였다.

가변 분할이 Aspect 이미지에서는 고정 분할보다 속도가 늦다는 것을 알 수 있었다. 그래서 Aspect 이미지를 변형하여 5개의 샘플 이미지(Aspect1, Aspect2, Aspect3, Aspect4, Aspect5)를 생성하였다. 이 이미지들은 Aspect 이미지에서 랜덤하게 공백을 만들어 넣어서 만들었다. Aspect1이 좌표값이 NULL인 것을 하나도 가지지 않은 이미지이고, Aspect5로 갈수록 좌표값이 NULL인 좌표를 랜덤으로 많이 삽입한 이미지이다. 테스트 결과는 (그림 16)에 나와있다.



(그림 16) 샘플이미지의 고정분할과 가변분할

결과적으로 NULL인 좌표값이 많이 있는 이미지일수록 가변분할의 Sp가 고정분할의 Sp에 근접하게 된다는 것을 알 수 있다.

5. 결론 및 향후 연구 방향

본 논문에서는 3D 이미지를 고속으로 처리하기 위해 3D 이미지를 병렬화 시스템을 이용하여 디스플레이 하는 병렬 알고리즘을 제안하였다. 이미지를 병렬로 처리하기 위해 이미지 분할 방법 3가지를 제안하였고, 원 이미지의 손실이 적고 빠른 디스플레이 시간을 보이는 수직선 분할 방법을 적용하였다. 또한 수직선 분할 방법을 확장하여 고정 분할과 가변 분할 방법을 적용하였다.

기존의 GIS에서처럼 하나의 프로세서를 가지고 3D 이미지를 처리할 때 생길 수 있는 처리시간 저하와 이미지 손실을 처리하였으며 또한 고성능의 시스템을 사용하지 않고 이미지를 분할하여 병렬로 처리함으로써 디스플레이 시간을 단축시킬 수 있었다.

(그림 17) 이미지 분할 적용분야

본 논문에서 구현한 병렬 알고리즘은 (그림 17)과 같이 하나의 이미지을 여러 모니터에 분할하여 디스플레이 하는 시스템 [3]에도 적용할 수 있을 것이다.

향후 연구로는 병렬화를 통해 디스플레이 한 이미지를 하나의 이미지로 병합하는 연구와 제안한 이미지 병렬화 분할 방법을 GIS의 다른 이미지 형식에 적용하는 것이다.

참 고 문 헌

[1] 정상화, 류광렬, 고윤영, 곽민석, "병렬 GIS를 위한 효율적인 분산공유메모리 시스템", 정보과학회논문지(c), 제 5권 제6호, 1999.

[2] 김진호, 김남규, 김지인, 정갑주, "볼륨렌더링을 위한 병렬화 방법들의 성능평가", 99 가을 학술발표논문집(I), 제 26권 제2호, pp.641-643, 1999.

[3] 성운재, 허성준, 손호준, 원광연, "병렬 디스플레이 시스템과 이를 이용한 분산 이미지 모자이크 기법", HCI 2000 학술대회 발표논문집, pp.909-913, 2000.

[4] M. Mirmehdi and M. Petrou, "Segmentation of color textures," IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol.22, pp.142-159, Feb., 2000.

[5] W. S. Lin, W. H. Lau, K. Hwang, Fellow, X. Lin, and Y. S. Cheung, "Adaptive Parallel Rendering on Multiprocessors and Workstation Clusters," IEEE Trans. on Paralle and Distributed systems, Vol.12, pp.241-258, Mar., 2001.

[6] W. Li, D. Zhang, Z. Liu, and X. Qiao, "A parallel algorithm for Image information restoration," Proc. of High Performance Computing in the Asia-Pacific Region, Vol.2, pp.790-793, 2000.

[7] M. Ishii, "Cluster technologies for high performance computing," Proc. of Parallel Architectures, Algorithms, and Networks, pp.168-170, 1999.

[8] J. L. Martin, G. Aranguren, J. Ezquerra, and P. Ibanez, "Parallel Raster Image Processor for PCB Manufacturing," Conf. on 20th International IECON 1994, Vol.2, pp.1184-1189, 1994.

[9] S. Barsamian, "Uses of Parallel Processors in a Photo Based Image Generator," IEEE Proc. of Aerospace and Electronics Conference NAECON 1990, Vol.2, pp.688-693, 1990.

[10] A. L. DeCegama, "PARALLEL PROCESSING ARCHITECTURES AND VLSI HARDWARE VOLUME 1," PRENTICE HALL, pp.106, 1989.

[11] S. Shekhar, S. Ravada, V. Kuma, D. Chubb, and G. Turner, "Declustering and load_balancing methods for parallelizing geographic information systems," IEEE Trans on. Knowledge and Data Engineering, Vol.10, pp.632-655, 1998.

[12] ESRI, "http : //www.esri.com".

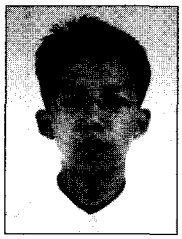
[13] Intergraph, "http : //www.intergraph.com".

[14] Mapinfo, "http : //www.mapinfo.com".

[15] KSIC(한국공간정보통신), "http : //www.ksic.net".

[16] GRASS, "http : //grass.itc.it".

[17] MPI, "http : //www-unix.mcs.anl.gov/mpi".



조 정 우

e-mail : ilsin95@hitel.net

1999년 서울시립대학교 전산통계학과 졸업 (학사)

2002년 서울시립대학교 전산통계학과 졸업 (석사)

2002년~현재 서울시립대학교 컴퓨터·통계학과 박사과정

관심분야 : 병렬처리, GRID, 클러스터링 시스템

김 진 석

e-mail : jskim@venus.uos.ac.kr

1990년 과학기술대학 전산학과 졸업 (학사)

1992년 KAIST 전산학과 졸업(석사)

1997년 KAIST 전산학과 졸업(박사)

1997년~1999년 KAIST 인공지능연구 센터 Postdoc 연구원

1997년~1998년 미국 M.I.T. Laboratory for Computer Science Postdoc Fellow

1998년~1999년 전자통신연구원(ETRI) 슈퍼컴퓨터센터 초빙연구원

1999년~현재 서울시립대학교 컴퓨터·통계학과 조교수

관심분야 : 병렬처리시스템, 멀티미디어시스템, 인터넷과 정보검색