KIPS Tr. Comp. and Comm. Sys.
Vol.3, No.8 pp.251~260 pISSN: 2287-5891

http://dx.doi.org/10.3745/KTCCS.2014.3.8.251

A Parallel Approach for Accurate and High Performance Gridding of 3D Point Data

Changseop Lee[†] · Permata Nur Miftahur Rizki[††] · Heezin Lee[†††] · Sangyoon Oh[††††]

ABSTRACT

3D point data is utilized in various industry domains for its high accuracy to the surface information of an object. It is substantially utilized in geography for terrain scanning and analysis. Generally, 3D point data need to be changed by Gridding which produces a regularly spaced array of z values from irregularly spaced xyz data. But it requires long processing time and high resource cost to interpolate grid coordination. Kriging interpolation in Gridding has attracted because Kriging interpolation has more accuracy than other methods. However it haven't been used frequently since a processing is complex and slow. In this paper, we presented a parallel Gridding algorithm which contains Kriging and an application of grid data structure to fit MapReduce paradigm to this algorithm. Experiment was conducted for 1.6 and 4.3 billions of points from Airborne LiDAR files using our proposed MapReduce structure and the results show that the total execution time is decreased more than three times to the convention sequential program on three heterogenous clusters.

Keywords : 3D Point Cloud Data, Gridding, Interpolation, Kriging, Parallel

3D 점 데이터 그리딩을 위한 고성능 병렬처리 기법

이 창 섭[†] · Permata Nur Miftahur Rizki[††] · 이 희 진[†††] · 오 상 윤[††††]

요 약

3D 점 데이터는 높은 정확성을 가진 사물의 표면 정보 데이터로 다양한 분야에서 사용되고 있으며, 특히 지리학에서 지형 파악과 분석에 많이 사용되고 있다. 일반적으로 3D 점 데이터의 Gridding 과정을 거치게 되는데 이는 불연속적인 점 데이터를 일정한 좌표 값으로 만드는 과정으로 긴 실행 시간과 높은 비용이 필요하다. 특히 Gridding 과정 중 보간 작업을 위해서 Kriging이 높은 정확성으로 주목받고 있지만 처리 과정이 복잡하고 연산이 많아 처리속도가 상대적으로 느리기 때문에 많이 사용되지 않고 있다. 본 논문에서는 Gridding을 고성능으로 처리하기 위해 Kriging 연산 과정을 병렬화했으며 격자 자료구조를 MapReduce 패러다임에 맞게 변형하여 Kriging에 적용하였다. 실험은 항공 LiDAR 데이터 약 1.6백만 개와 4.3백만 개의 점 데이터를 이용해서 제안한 MapReduce 구조에 적용하였고, 그 결과 3대의 이기종 클러스터에서 전체 실행시간이 순차적 프로그램에 비해 최대 3.4배 단축하였다.

키워드 : 3D 점 데이터, 그리딩, 보간법, 크리깅, 병렬

1. 서 론

3D 점 데이터(3D Point Data or Point Cloud)는 3D 센서를 통해서 얻어진 데이터로 3차원 X, Y, Z축의 점 데이터의 군집을 말한다. 최근에 3D 점 데이터는 CAD에서 3D 표면 모델링[1], 도시 모델링 및 건물 추출[2, 3], 실시간 사물 인지[4, 5] 등과 같이 다양한 분야에서 활용되고 있다. 또한 이 3D 점 데이터는 지리학과 생태학에서 해안 및 서식지 분석[6], 숲의 탐사와 관리[7] 등의 다양한 분야에 지형 파악 및 분석을 위해 사용되고 있는데 3D 점 데이터를 활용하는 경우 현실에 가까운 시뮬레이션을 할 수 있어서 예측 및 분석의 정확성이 높아지는 장점을 가지며, 이는 측정되는 데이터의 높은 정확성에서 기인한다. 3D 점 포인터 센서들 가운데 LiDAR센서는 Airborne LiDAR(Light Detection And Ranging), TLS(Terrestrial Laser Scanning)등이 존재하고 일반적으로 수 센티미터 이내의 정확도를 보인다[8]. 이러한

※ 본 연구는 미래창조과학부 및 정보통신산업진흥원의 대학 IT연구센터 지원사업의 연구결과로 수행되었음(NIPA-2014-(H0301-14-1011)).
※ LiDAR 데이터는 NSF-sponsored National Center for Airborne Lase Mapping(NCALM)에 의해 수집되고 제공받음.
† 준 회 원 : 아주대학교 컴퓨터공학과 석사과정
†† 준 회 원 : 아주대학교 컴퓨터공학과 석·박사통합과정
††† 비 회 원 : 미 캘리포니아대학교 책임연구원
†††† 정 회 원 : 아주대학교 컴퓨터공학과 부교수
Manuscript Received : June 9, 2014
First Revision : August 7, 2014
Accepted : August 8, 2014
* Corresponding Author : Sangyoon Oh(syoh@ajou.ac.kr)

3D 점 데이터를 사용하기 위해서는 일반적으로 Gridding을 거치게 되는데 Large-scale 데이터의 경우 분석 또는 보간(Interpolation)작업을 위해 긴 처리시간과 비용이 필요하다. Gridding은 격자 범위 내의 점 데이터들을 하나의 3차원 좌표로 보간하는 방법이고 이렇게 처리된 데이터는 필요한 해상도에 따라 다양한 분야에 사용할 수 있다.

IDW(Inverse Distance Weighting), Kriging, Spline 등의 다양한 보간 방법 중에서 IDW 과정이 단순하고 처리 시간이 빠른 이유로 상대적으로 많이 쓰이고 있다. 이에 대해 Kriging 알고리즘은 입력된 정보를 샘플링한 후에 그 값의 분산을 최소화 하는 경우의 예측 값을 구하는 방법[9]으로 정확성이 높지만 이 알고리즘은 상대적으로 처리 과정이 복잡하고, 시간이 오래 걸리는 단점때문에 많이 사용되지 못하였다[10, 11].

일반적으로 보간 작업의 병렬화는 현재 많이 적용되어 있지 않지만 상대적으로 복잡하고 연산 집중적인 Kriging 작업의 경우 병렬처리를 함으로써 실행시간을 단축시킬 수 있으며 만약 점 데이터 및 탐색구역이 넓어지면 연산 작업도 늘어나기 때문에 이를 효율적으로 처리하는 위해 병렬화가 필요하다. 최근 3D 센서의 발달로 데이터의 규모가 커짐에 따라 정확도가 높은 Kriging 방식을 직접 적용하기 위해서는 기존의 방식과 다른 High Performance 방식의 설계와 구현이 필요하며 이에 따른 자료구조 및 알고리즘에 대한 연구가 절실히 요구되고 있다. 본 논문에서는 Kriging을 적용한 Gridding을 병렬화하여 기존의 순차적인 Gridding 프로그램보다 처리 시간을 단축시키며, 전체 보간 결과의 정확도를 높이는 방법을 제안한다.

본 제안 방법에서는 격자 탐색을 병렬로 수행하고 Kriging의 단계 중 Semivariogram과 Prediction을 병렬화 하였다. 병렬화를 위한 방법으로는 최근 빅데이터 및 병렬 처리 분야에서 각광을 받고 있는 MapReduce를 이용하였으며, 본 논문에서 제안하는 구조의 성능을 검증하기 위해서 항공 LiDAR 센서로 스캔한 3D 점 데이터를 이용하여 실험을 진행하였다. Gridding의 순차적인 프로그램의 실행시간과 이기종의 노드에 분산 처리한 MapReduce 프로그램의 실행시간을 비교분석하였으며, 약 1.1백만 개의 점 데이터와 그 탐색범위가 5일 때 약 3.4배의 성능이 향상됨을 확인하였다.

본 논문의 구성은 다음과 같다. 2장에서는 3D 점 데이터에 관련된 연구 및 기존 병렬 처리 방식의 소개와 3장에서는 제안하는 Gridding 기법을 설명한다. 그리고 4장에서는 이를 병렬처리 하는 방법을 설명하였으며, 제안한 방법을 적용함에 필요한 세부사항을 5장에서 정리하고, 6장에서는 실험 및 분석을 통해 제안한 방법의 성능을 분석하여 제안 방법에 대한 평가를 진행하였다. 마지막으로 7장에서 결론을 맺는다.

2. 관련 연구

3D 점 데이터는 사용하고자 하는 분야 또는 편의를 위해 삼각불규칙망(TIN : Triangulated Irregular Network) 또는 Gridding을 거쳐 데이터를 재표현할 수 있다[8]. 원래 데이터를 기반으로 TIN 또는 Gridding을 해야 하는 이유는 3D 점 데이터의 분석을 용이하게 만들기 때문이다. 항공 LiDAR 센서의 정밀도는 일반적으로 센티미터 스케일이며, 만약 수 제곱킬로미터를 스캔하면 경우에 따라 수천만 개에서 수억 개의 3D 점 데이터가 생성된다. 이를 처리하는 과정은 연산 집중적인(Computation Intensive) 작업이며 그 실행시간이 매우 오래 걸린다.

그래서 적용하는 분야에 따라 다양하게 3D 점 데이터의 해상도를 조절하고 그 데이터를 재구성해야 한다. 이와 관련하여 Tomislav Hengl[12]은 다양한 점 데이터의 적합한 Grid 해상도를 찾기 위해 지도 제작 방식 및 데이터의 통계, 가용한 컴퓨팅 파워 등에 따른 분류를 하고 Nyquist 이론을 적용한 Grid 해상도 도출 방법을 연구하였다. 그리고 Gridding에서 다양한 보간법이 사용되는데, Voronoi 다이어그램을 이용해서 샘플 데이터를 추출한 후에 보간하는 방법인 NN(Natural Neighbor)과 거리를 기반으로 가중치를 정해 보간하는 IDW 방법과 Spline과 같이 점 데이터 간의 곡률을 최대한 부드럽게 만드는 방법이 있다[10]. 그리고 전체 점 데이터의 통계학적인 분석으로 각 점의 가중치를 정해 예측하는 Kriging[9] 방법이 있는데, 이 연구에서 이용되었으며, 3.2절에서 자세하게 소개한다.

모든 원본 3D 점 데이터는 원하는 해상도의 Gridding을 통해 재표현 되는데, Pankaj K. Agrwal[13]은 3D 점 데이터 전체가 메모리에 로드 될 수 없으면 이를 Quad-tree로 만든 후에 보간하는 Gridding 방법을 제안했다. 이외에도 Kd-tree를 이용해서 멀티코어에 TIN을 처리한 연구도 있었다[14].

3D 점 데이터가 저장되어 있는 클러스터에 특정 좌표 구간의 쿼리를 보내 그 구간에 포함하는 3D 점 데이터를 병렬로 찾는 시도가 있었다[15]. 사용한 자료구조는 Quad-tree와 Bucketing 방법이 사용되었으며, 검색 파티션 기법과 점 데이터 분배 방식을 다르게 해서 병렬 처리함으로써 검색 성능을 향상시켰다.

최근에는 여러 병렬 처리 기술을 활용해서 Gridding을 한 시도가 있었다. Xuefeng Guan[16]은 공유메모리 환경에서 멀티스레드를 구현하는 OpenMP를 활용해서 쿼드코어 환경에 1개에서 8개 스레드로 IDW 작업을 병렬처리 하였고, Sriram Krishnan[17]은 IDW를 적용한 Gridding을 MapReduce를 사용해서 각 노드에 분산처리 하였다. 또한 Katharina Henneböhl[18]은 GPGPU로 CUDA를 사용해 Gridding을 병렬화한 연구를 제안했다.

Kriging 보간 방법의 적용이 가능한 상용 소프트웨어인 Surfer[21]의 경우 전체 데이터를 활용하지 않고 가까운 일부 점 데이터만으로 연산하는 샘플링 방식 또는 사용자가 직접 중요 파라미터를 입력하는 방법이기 때문에 처리시간

은 단축되지만 보간 정확성이 낮아진다는 단점이 있다. 정확한 알고리즘은 알기 힘들지만 4.5백만 개의 데이터 경우 110초에 Kriging 보간 방법이 적용된 Gridding 작업이 완료된다. 이는 매우 작은 데이터만으로 빠르게 샘플링한 후 Kriging에서도 전체 데이터가 아닌 일부 데이터로만 보간 작업함을 알 수 있는데 이는 상대적으로 낮은 정확성을 지닌 결과값이 출력되기 때문에 이를 보완할 필요가 있다.

3. 3D 점 데이터 Gridding 기법

3D 점 데이터의 Gridding에서 높은 정확도가 요구되는 경우에 Kriging을 사용할 수 있도록 병렬화 설계를 하였다. 먼저 3D 점 데이터를 Bucketing 하고 탐색범위를 설정한 후 탐색범위 내에 있는 점 데이터만을 사용해서 특정 격자 좌표의 값을 예측하였다.

3.1 버켓팅(Bucketing)

우선 3D 점 데이터의 Gridding을 위해 어느 한 점, $P(x_i,y_i,z_i)$가 어느 격자에 포함되는지 파악하고 그 자료구조를 만들어야 한다. Bucketing[16]은 Figure 1과 같이 점 데이터를 정해진 격자 크기에 매핑하는 것을 의미하며 수식[19]은 아래와 같다.

$$X_{grid}=\frac{\lfloor x_i-x_{\min}\rfloor}{gridsize},i=1\ldots n$$
$$Y_{grid}=\frac{\lfloor y_i-y_{\min}\rfloor}{gridsize},i=1\ldots n \quad (1)$$

여기서 n은 3D 점 데이터의 총 포인트 개수를 나타내고, $x_{\min}$과 $y_{\min}$은 전체 포인트 중에서 가장 작은 x좌표 값과 y좌표 값이다. $gridsize$는 격자 하나의 길이를 나타낸 것으로 실험에서는 기본 값 1로 사용했다. (1)의 식을 사용해서 모든 좌표의 Bucketing이 끝나면 격자 간의 탐색범위를 지정해서 주위의 포인트로만 Kriging 보간법을 수행해서 격자 좌표의 값을 예측한다.

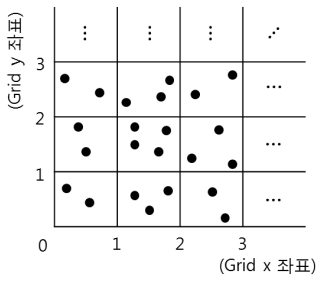


Fig. 1. Method of Bucketing

Bucketing을 통해 자료구조를 구성하면 이를 기반으로 Semivariogram, Fit, Prediction 과정을 수행하는데 이는 아래의 Kriging 보간법을 참고한다.

3.2 Kriging 보간법(Semivariogram, Fit, Prediction)

보간법이란 임의의 위치에서 모르는 값을 구하기 위해 다른 측정된 값을 이용해서 그 값을 예측하는 방법이다. 특정 위치의 값을 보간하기 위한 방법 중의 하나인 Kriging은 다른 보간법에 비해 상대적으로 복잡하고 그 연산처리량이 많은 특징을 가진다[10].

Kriging 보간법은 주위의 측정된 가중치의 선형조합으로 임의의 위치의 값을 예측하는 방법이다[9]. 이를 수식으로 정리하면 다음과 같다.

$$\hat{Z}(x_0)=\sum_{i=1}^{n}w_iZ(x_i) \quad (2)$$

$\hat{Z}(x_0)$는 임의의 위치 x_0에서 예측 값을 나타내고, 측정된 위치인 x_i의 값 $Z(x_i)$에 가중치 w_i를 곱해서 예측 값을 구하며 n은 측정된 자료의 총 개수를 나타낸다. 여기서 가중치를 구하는 작업이 다른 보간법에 비해서 복잡하고 시간이 오래 걸리는 작업이다. 가중치를 정하기 위해서 구한 예측값과 참값의 오차가 최소가 되도록 해야 하며 편향되지 않아야 한다.

$$\sigma^2=E\left[\left\{\hat{Z}(x_0)-Z(x_0)\right\}^2\right] \quad (3)$$

σ^2는 오차분산을 나타내며 이 값이 최소가 될 때의 가중치를 구하고자 한다. 이 식을 구체적으로 전개해서 정리하면 다음의 식에서 가중치를 얻을 수 있다.

$$\sum_{i=1}^{n}w_i\gamma_{ij}+\Phi=\gamma_{0j}\ ,\quad \forall j \quad (4)$$

j는 탐색범위 내의 점을 의미하며 γ_{ij}와 γ_{0j}의 성격은 자기공분산이며 Semivariogram이라고 불린다. w_i는 위와 마찬가지로 가중치를 나타내며 여기서 Φ는 라그랑주 승수로 $\sum_{i=1}^{n}w_i=1$의 제약사항에서 최솟값을 찾는다. 가중치를 찾기 위해 Semivariogram을 우선 결정해야 하는데 이는 어떤 주어진 거리 h에 있는 데이터들을 계산한 것으로 공간 데이터의 변화 및 유사성을 나타내는 값이다.

$$\gamma(h)=\frac{1}{2M(h)}\sum_{i=1}^{M(h)}\{Z(x_i)-Z(x_i+h)\}^2 \quad (5)$$

여기서 h는 분리거리(lag)라고 표현하며 지정된 분리거리에 따라 $\gamma(h)$를 계산한다. $M(h)$는 h만큼 떨어진 데이터 쌍의 개수를 나타낸다. 지정된 h들에 따라서 $\gamma(h)$가 계산되는데 이를 실험적 Semivariogram이라고 말한다. 일반적으로 3가지 파라미터를 이용해서 Semivariogram을 모델링하게 되는데 c_o는 nugget 값이고 c는 sill값, a값은 range 값이 되는데 이는 다음과 같이 설명할 수 있다.

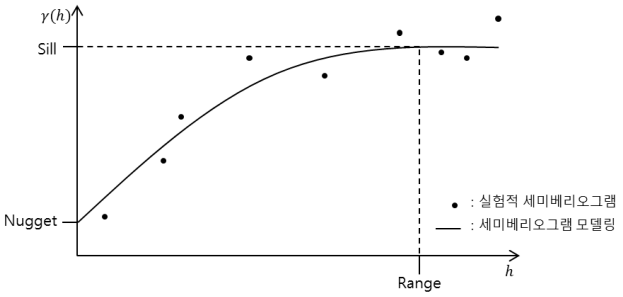


Fig. 2. Semivariogram Modeling

Semivariogram에서는 선형, 구형, 지수, 가우스 등의 다양한 모델링을 제시하고 있는데 우리는 구형모델(Spherical Model)을 적용해서 실험을 진행했다. 구형모델링은 아래와 같다.

$$\gamma(h)=\begin{cases} c_0+c\left(\frac{3h}{2a}-\frac{1}{2}\left(\frac{h}{a}\right)^3\right), & h\le a \\ c_0+c, & h>a \end{cases} \quad (6)$$

Figure 2에서 실험적 Semivariogram은 점을 나타내고 Least Squared 방법을 이용해서 실선 그래프로 모델링이 가능하다. 이 모델링을 통해 c_o, c, a의 값을 정할 수 있고 (6)의 식을 이용해서 다음의 계산을 할 수 있다.

$$\begin{bmatrix} \gamma_{11}(d_{11}) & \gamma_{12}(d_{12}) & \cdots & \gamma_{1n}(d_{1n}) & 1 \\ \gamma_{21}(d_{21}) & \gamma_{22}(d_{22}) & \cdots & \gamma_{2n}(d_{2n}) & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \gamma_{n1}(d_{n1}) & \gamma_{n2}(d_{n2}) & \cdots & \gamma_{nn}(d_{nn}) & 1 \\ 1 & 1 & \cdots & 1 & 0 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \\ \Phi \end{bmatrix} = \begin{bmatrix} \gamma_{10}(d_{10}) \\ \gamma_{20}(d_{20}) \\ \vdots \\ \gamma_{n0}(d_{n0}) \\ 1 \end{bmatrix} \quad (7)$$

위의 행렬은 Ordinary Kriging 방정식의 전개식이며 임의의 포인트 x_i와 x_j의 유클리디안 거리 d_{ij}을 구해서 이 값을 γ_{ij}에 대입한 결과로 계산한다. 미지수인 w_i는 AX=B의 선형방정식을 이용해 구하게 되며, 모든 가중치가 구해졌으면 (2)의 식에 대입해서 예측 값 $\hat{Z}(x_0)$을 구할 수 있다. 지금까지 제시된 공식 및 방법에 기반을 두어 다음 절에서 설명하는 병렬구조에서는 전체 Kriging 보간법을 Semivariogram, Fit, Prediction의 세 부분으로 나누어서 진행했다.

4. Gridding 병렬화 구조

Gridding을 고성능으로 적용하기 위해서 본 논문에서 Grid 좌표 데이터를 분산시켜 Kriging 보간법을 적용시켰다. 제안구조의 이해를 돕기 위해서 기존의 순차적 처리구조를 간단히 설명하고 이와 비교하여 본 제안 구조를 설명한다.

4.1 기존의 순차적 처리구조

순차적 처리구조에서는 3절에서 제시된 Bucketing, Semivariogram, Fit, Prediction 부분을 모두 하나의 노드에서 실행하는 구조

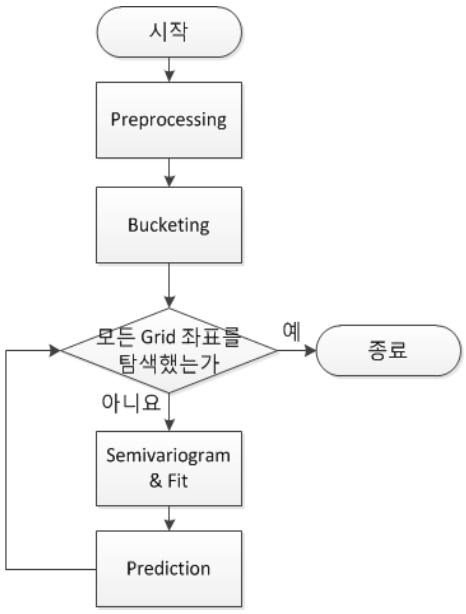


Fig. 3. Flow chart of sequential Gridding algorithm

를 취한다. Figure 3에서 Preprocessing은 전체 점 데이터의 경계설정 및 사용자 정의 입력 값이 설정되고, Bucketing, Semivariogram, Fit, Prediction 순으로 진행되며 각 격자에 대해 Semivariogram, Fit, Prediction 단계가 반복적으로 실행된다. 각 격자의 정보는 버켓이라고 부르는 2차원 Linked list 배열에 저장된다[16].

Semivariogram을 구하는 방법에서 우리는 각 격자마다 탐색범위 내의 점 데이터를 이용해서 모델링을 했는데 기존의 모든 점 데이터의 Semivariogram을 구하는 경우 처리과정이 매우 오래 걸려서 (1.1백만 개 데이터의 경우 약 20,000초) 정확도가 일부 높아지는 것에 비해 실제 3D 점 데이터의 처리 과정에 적용하는 것이 의미가 없다. 따라서 본 논문에서는 순차적 처리과정의 Semivariogram은 탐색 범위 내의 점 데이터만으로 구하며, 이를 기반으로 이후 Fit, Prediction을 진행한다. 순서도에서 볼 수 있듯 모든 격자를 탐색한 후 Prediction까지 완료하면 프로그램이 종료되며 그 결과 각 격자마다 그 예측 값이 구해지게 된다.

4.2 Gridding 병렬화 구조

4.1절에서 설명된 순차적 처리 방법을 효과적으로 병렬화하기 위해서 정보를 저장하는 새로운 구조, 이를 탐색하기 위한 방법과 이에 대한 처리 방안이 제시되어야 하며 각 단계는 효율적으로 제시되어야 병렬화를 통한 고성능화가 가능해진다.

Gridding에서 격자 정보를 저장 및 탐색하는 방법으로는 일반적인 공간 인덱싱에서 사용하는 Brute-force 방식의 선형검색, Bucketing[16], Quad-tree[13], Kd-tree[14] 등을 사용할 수 있다. 이들 중에서 Kd-tree를 사용해서 격자정보를 저장하려는 방법은 모든 점 데이터를 Kd-tree 구조로 저장한 후 사용자로부터 입력받은 격자 반경에 따라 그 데이터

를 검색하는 방식이다. 그러나 이 방법은 범위에 따라 탐색할 때 내부 노드들 간에 의존성이 존재하기 때문에 병렬구조, 특별히 본 논문의 제안 구조에서 사용하는 MapReduce 방식으로 처리하기에 효율적이지 않고, 점 데이터의 규모가 클수록 성능이 Bucketing 방식에 비해 떨어지는 특징이 있다. Kd-tree의 시간복잡도는 생성의 경우 $O(nlogn)$이고, 특정 점을 찾는 탐색의 경우는 $O(\log n)$인데 비해, Bucketing의 경우 생성의 경우 $O(n)$, 탐색의 경우 $O(1)$이 소요된다. 이와 같이 Kd-tree와 같은 중앙 집중적인 자료구조는 제안하는 고성능-병렬화 구조에 적합하지 않으며, 본 논문에서는 병렬화에 적합한 Bucketing을 사용한다.

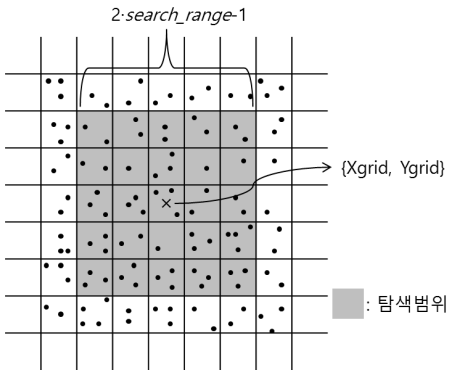


Fig. 4. Find 3D point data to Interpolate One Grid coordinate if *search_range* is 3

버켓에서 격자 탐색 범위에 따른 검색방법은 탐색 범위가 n이라고 한다면 현재 격자 좌표를 중심으로 한 변이 2n-1개의 정사각형 내부의 점 데이터들을 찾는 방법이다. 즉, Figure 4에서와 같이 탐색 범위가 3라고 한다면 현재 격자를 기준으로 한 겹의 격자를 정사각형으로 둘러싼 내부의 25개의 격자들이 검색 대상이다.

Algorithm 1 : Bucketing

```
1   MAP (in_key, in_value, out_key, out_value)
2     set search_range from user's defined value
3     extract in_value for Point
4     calculate grid point using Point and set GridPoint
5
6     FOR all grid points are found around GridPoint
7     by search_range
8       set out_key from each grid point
9       set out_value from Point
10      output <out_key, out_value> pair
11    ENDFOR
12  ENDMAP
```

Bucketing을 병렬화 구조에서 탐색하기 위해 우리는 격자 좌표의 자료구조를 변경했다. 본 제안 구조에서 사용하는 MapReduce에서는 서로 다른 Map 또는 Reduce 간의 통신이 없기 때문에 그래프처럼 의존성이 강하고 결합된 특성을 병렬처리 하는 데 불리하다. 그러나 위의 순차적인 알고리즘에서 버켓은 이차원 배열이지만 이를 <key, value>의 쌍으로 만들 수 있기 때문에 그 결과 노드의 각 Reduce에서 분산 처리가 가능하다. 그래서 클러스터 내의 사용 가능한 Reduce의 개수가 많아지면 전체 실행시간도 짧아진다. 위의 알고리즘을 포함한 전체적인 흐름을 Figure 5에 나타내었다. Map에서 탐색범위에 해당하는 각 격자 좌표와 실제 점 데이터를 출력하면 Reduce에서 그 격자 좌표를 기준으로 점 데이터를 모으고 정렬해서 Semivariogram과 Prediction 부분을 실행하게 된다.

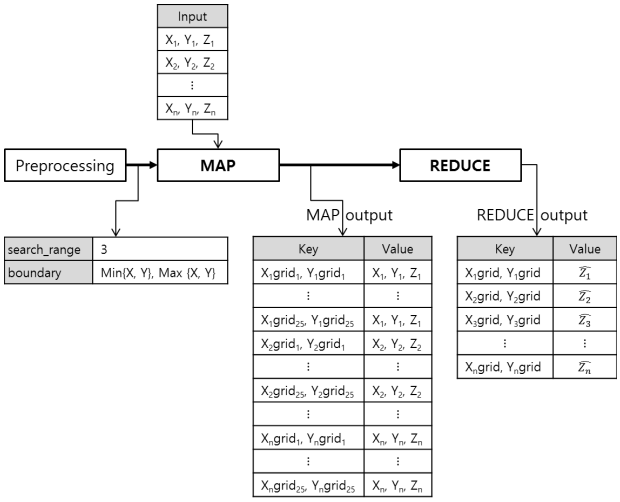


Fig. 5. MapReduce Parallel data Flow by the Proposed algorithm

순차적인 프로그램과 병렬화를 적용한 구조의 가장 큰 차이는 탐색범위 내에 있는 점 데이터를 분산시키는지 여부이다. 분산이 되기 때문에 데이터를 격자 단위로 쪼개서 연산을 할 수 있고 고성능의 병렬화를 제시할 수 있었다. 하지만 MapReduce는 <key, value>의 특수한 데이터 구조로 정보를 전달하기 때문에 자료구조를 변형하는 과정에서 병렬화의 추가적인 연산비용을 생각해야만 했으며 파일입출력 기반인 하둡 프레임워크의 오버헤드도 전체 성능에 많은 영향을 줄 수 있기 때문에 이를 고려하여 구현 및 실험을 진행했다.

5. 제안구조의 적용(Implementation)

5.1 순차적 처리구조의 구현 환경

우리는 병렬화 구조의 구현 프로그램과 성능을 비교하기 위해서 순차적 처리구조 또한 구현했다. 4.1절에서 설명된 내용을 기반으로 C 언어를 사용하여 Gridding 알고리즘을 구현하였고 Centos 6.4에서 gcc 4.4.7의 -O2 최적화 옵션을 사용해서 컴파일 하였다.

본 논문의 실험을 위해서 순차적 처리와 병렬처리 각각의 성능을 최적화하여 비교하는 것이 객관적인 검증을 위해 중

요한 사안이라 판단하여 각 처리 프로그램의 성능향상에 노력하였다. 본 순차처리의 Semivariogram과 Prediction 단계는 모두 각 점들 간의 거리를 구하는 과정을 수행하게 되며, 성능 향상을 위해 이 과정을 중복 수행하지 않도록 Semivariogram에서 구해진 거리정보가 저장된 배열을 Prediction 단계를 수행하는 함수에 파라미터로 넘겨주도록 설계하였으며, 따라서 Semivariogram 부분이 끝나면 기본적인 설정에 따라 실험적 Semivariogram 값이 도출되며 이를 Fit 과정으로 전달하여 Least Squared 방법으로 모델링을 수행한다. 모델링이 끝나면 Prediction 부분에서 (7)식을 통해 가중치를 구하는데 이때 선형방정식을 풀기 위해 가우스 소거법보다 성능이 우수한 LU 분해법 알고리즘을 사용해서 최적화시켰다. 격자의 예측 값이 도출되면 이를 출력 포맷에 맞도록 {Xgrid, Ygrid, Z}의 형식으로 파일로 출력한다.

5.2 MapReduce를 이용한 병렬화 구조의 구현 환경

1) 전처리 작업

4.2절에서 설명된 바와 같이 병렬화 구조에서는 버켓의 경계를 찾아야 하며, 이를 위해서 MapReduce를 이용한 처리의 전단계로 모든 점 데이터를 읽어서 X축으로 최솟값・최댓값과 Y축으로 최솟값・최댓값을 구한다. 이 과정을 MapReduce로 처리하는 것보다 전처리로 하는 것이 전체 성능을 높일 수 있다고 판단했다. 또한 격자 탐색범위(search_range)는 순차와 병렬 처리 성능에서 독립적이며 가장 중요한 변수이므로 이는 사용자로부터 입력을 받게 되며 (즉, 사용자 설정 변수)이 값과 X, Y축의 경계 값을 Job 전체 파라미터로 저장한다. 이는 본 전처리 작업 후 Map에서 사용된다.

2) Map phase

Map 과정에서는 전처리 과정에서 구한 X, Y축의 경계값과 탐색범위를 바탕으로 Gridding을 진행한다. 하둡 프레임워크는 HDFS(Hadoop Distributed File System)를 기반으로 동작하고 Map과 Reduce는 HDFS에 입출력한다. 그래서 입력되는 점 데이터는 하나의 파일로 HDFS의 지정된 경로에 위치하며 한 라인에 하나의 좌표정보({X, Y, Z})로 구성되어 있다. 일반적인 하둡 프레임워크에서는 입력 파일을 HDFS 기본 블록 크기인 64MB 단위로 쪼개서 Map에 할당하기 때문에 입력 파일의 크기가 기본 블록 크기보다 작으면 Map이 하나 이상 생성되지 않는다. 따라서 블록 기준이 아닌 입력된 파일의 라인 기준으로 Map을 생성하고 할당하는 NLineInputFormat 옵션을 취했으며, 이는 입력 파일을 블록 크기로 분할하는 것이 아니고 라인 수로 분할하기 때문에 입력 파일의 크기가 작아도 원하는 Map의 개수에 할당하는 작업이 가능했다. 이를 통해 HDFS의 블록 사이즈보다 작은 데이터로 고연산을 요하는 경우에 각 Map으로 분산시켜 병렬처리를 할 수 있다.

이 과정을 통해 Map 함수에서는 {X, Y, Z} 형태의 value를 입력받게 되며, 이를 버켓팅을 수행한 후에 탐색범위에 맞게 그 좌표 값을 출력한다. 예를 들어 탐색범위가 2이라고 한다면 현재 격자좌표를 중심으로 3×3 격자 범위 내에 있는 모든 격자 좌표의 {Xgrid, Ygrid}를 key로 하고, 각 value는 입력받은 좌표 값인 {X, Y, Z}로 해서 출력한다. 만약 탐색범위가 경계 밖으로 벗어난다면 그 부분을 제외한 나머지 격자좌표를 key로 설정한 후 출력한다.

3) Reduce Phase

본 단계에서는 Reduce 함수가 입력 key인 {Xgrid, Ygrid}를 기준으로 병합된 value {X, Y, Z}를 이용해서 순차적으로 Semivariogram, Fit, Prediction 처리를 한다. 입력 value, 즉 {X, Y, Z} 점들은 Semivariogram에 적용되기 전 점들 간의 X, Y축의 최솟값 및 최댓값을 구해야 하며, 작업 완료 후 Semivariogram 과정이 수행되어 5.1절에서 설명된 실험적 Semivariogram 값이 도출된다. 이 값들을 기반으로 Fit과 Prediction 부분이 실행되는데 결과 값으로 격자 중심 좌표의 Z값을 예측할 수 있다. 예측된 Z 값은 출력 포맷에 맞게 {Xgrid, Ygrid, Z} 형식으로 HDFS로 출력한다. Kriging 방정식의 해를 구하기 위해 Apache Common Math 3.2[20] 라이브러리의 LU 분해법을 활용했다. LU 분해법은 선형방정식 Ax=B에서 A를 L*U로 바꾼 뒤 가우스 소거법을 적용하는 것을 말한다. 행렬 L은 하삼각행렬이고, 행렬 U는 상삼각행렬이다. LU 분해법 외에도 자바 언어 기반의 다양한 수학, 통계적인 문제해결을 위한 라이브러리를 제공하고 있다.

탐색 범위에 포함된 격자좌표와 그 점 데이터를 각 Map에서 출력하고, 이를 격자좌표별로 정렬 및 병합한 후 Reduce에서 Kriging 보간법을 적용함하여 Gridding 알고리즘을 병렬화했다. 위의 세 단계로 MapReduce 패러다임에 맞는 3D 점 데이터 병렬 보간법을 제시할 수 있었으며 Cloudera에서 배포된 하둡 프레임워크 0.20.2을 사용해서 성능 평가를 통해 그 효율성을 입증했다.

6. 성능 평가

6.1 성능평가 환경

우리가 제시했던 3D 점 데이터의 Gridding을 MapReduce 시스템에 적용하고 성능평가를 했다. 성능평가는 C로 구현한 순차적인 Gridding 프로그램과 MapReduce로 구현한 프로그램의 실행속도를 비교했으며 실험환경은 다음과 같다.

Table 1. Experimental environment

	Turing	Jobs	Neumann
CPU	xeon X3430	xeon E5606	xeon E3-1220
	2.40GHz	2.13GHz	3.1GHz
	Quad-core	2×Quad-core	Quad-core
RAM	8GB	8GB	8GB
OS	Centos 6.4	Centos 6.4	Centos 6.4

위의 표에서 제시된 세 대의 노드(16개 코어)를 1Gbps ethernet과 내부 스위치로 네트워크를 설정한 후 클러스터를 만들어 MapReduce 프로그램을 실험하였다. 순차적인 프로그램은 Neumann 노드에서 실행해 그 시간을 측정했는데 Neumann을 선택한 이유는 세 대의 노드 중 하드웨어 성능이 가장 우수하기 때문이다.

6.2 성능평가를 위한 데이터 분석

성능 평가를 위해 사용한 LiDAR 데이터는 미국 시애틀 근처의 Cedar River Watershed (Data 1)와 캘리포니아의 Arroyo Seco (Data 2)이고 그라운드 추출을 거친 점 데이터이다. 그라운드 추출은 지구 지표면에 가까운 정보만 남기고 나머지 정보는 걸러진다. 아래의 Table 2에서는 각 실험 데이터의 세부 정보를 나타내고, Figure 6은 Data 1의 그라운드 추출된 이미지이다.

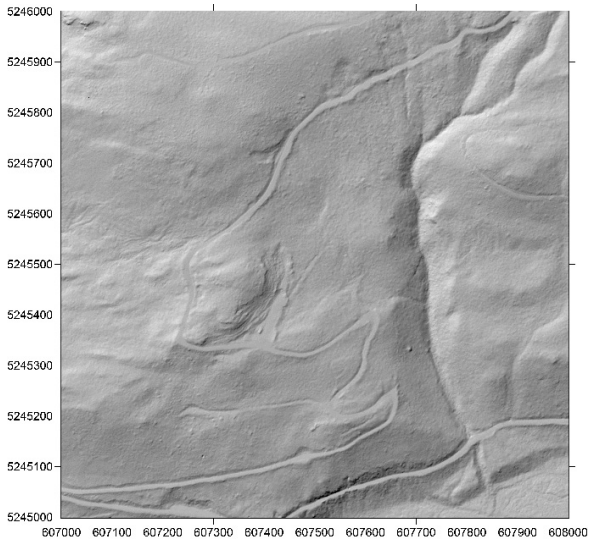


Fig. 6. Point data Image of Data 1

Table 2. Input LiDAR data information

	Data 1	Data 2
용 량	35MB	133MB
점 개수	1,186,845	4,517,569

특히 Table 3에서 같은 크기의 탐색반경에서 그 밀도가 차이가 많이 나는 것을 확인할 수 있다. 우리는 탐색범위를 3과 5로 설정해서 실험했는데, 표를 참고하면 범위 내에 점 데이터가 없는 부분이 있는 반면, 수백 개가 있는 범위도 존재한다. 우리가 실험한 LiDAR 데이터는 그라운드 추출된 데이터로 지면에 근접한 데이터만 남아 있기 때문에 일반적인 LiDAR 원본데이터와 차이가 있다.

Table 3. Analysis of two input files by search range

데이터분류	Data 1		Data 2	
탐색범위	3	5	3	5
최소개수	0	0	0	12
최대개수	188	572	293	879
평균개수	29.57	95.58	108.28	350.10
표준편차	34.29	103.86	50.50	147.95
(평균개수) × (표준편차)	1,013.95	9,926.93	5,468.14	5,1797.29

6.3 성능평가 결과

성능평가는 크게 위에서 나눈 네 부분으로 그 실행시간 그래프와 함께 비교 분석을 한다. Preprocessing 과정을 성능분석하지 않는 이유는 Data 1에서 평균적으로 1.9초, Data 2에서 5.8초로 성능에 큰 영향을 미치지 않기 때문이다. 실험한 데이터는 위에서 언급한 Data 1과 Data 2를 탐색범위에 따라 그 실행시간을 측정하였다. 아래의 그래프는 그 실행시간을 나타내고 x축의 Data 1-3은 입력데이터는 Data 1, 탐색반경(search_range)이 3이라는 의미이다.

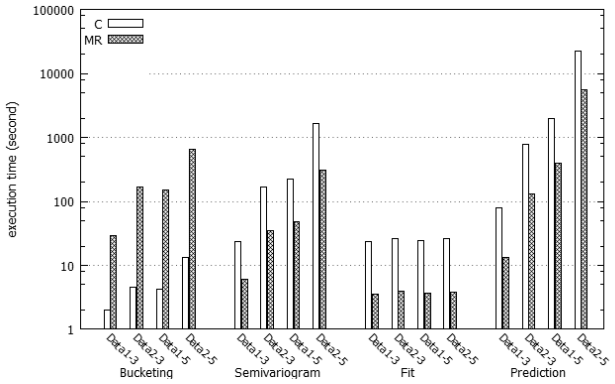


Fig. 7. Comparisons on execution times between Sequential and Our Proposed Parallel Approach

우선 MapReduce의 실행시간 측정을 위해 전체 클러스터에서 실행된 시간을 T_{all}라고 하고 Reduce의 개수를 N_r이라고 한다면, 각 Reduce에서의 평균 실행 시간은 $T_{aver} = T_{all}/N_r$이 된다. 여기서 N_r은 실행시간이 가장 짧았던 Reduce 개수로 설정했다. 버켓팅, Semivariogram, Fit, Prediction 각 부분의 T_{aver}와 C 프로그램 실행 시간을 비교한 그래프가 Figure 7이다. Bucketing을 MapReduce에서 실행된 구간은 Map과 copy, sort 과정이 포함되었기 때문에 그 구간의 시간도 합쳤다. 그래서 Figure 7을 보면 Bucketing에서 MapReduce의 시간이 느린데 그 이유는 copy와 sort 같은 구조적인 측면에서의 오버헤드가 있기 때문이다. 그리고 Data 2-3보다 Data 1-5가 더 빠른데 그 이유는 탐색 반경을 적용한 점 데이터의 개수가 Data 2-3이 더 작아서 그 자료구조를 만드는데 적은 시간이 걸리기 때문이다. Table 4는 Figure 7에서 비교된 각 입력 데이터와 탐색 범위에 따른 C와 MapReduce의 실행 시간 비교를 나타낸다.

Table 4. Comparison on total execution times

데이터 분류	실행시간(second)	
	C	MR
Data 1-3	144.68	94.87
Data 2-3	1,012.69	335.74
Data 1-5	2,264.15	658.41
Data 2-5	23,931.15	7,489.04

Semivariogram 부분부터는 Reduce 구간에서 실행되는 것으로 최대 시간 복잡도는 $O(n^2)$이다. 그래서 순차적 프로그램과 MapReduce 프로그램에서 탐색범위와 입력 데이터가 달라지면 그 실행시간도 크게 변화함을 그래프를 통해 알 수 있는데, Figure 7을 기준으로 Data 1-3의 3.9배부터 Data 2-5의 5.8배까지 MapReduce를 적용한 실행 시간이 단축되었고, 평균적으로 5.1배 실행시간이 단축되었다. Fit 부분은 정해진 실험적 Semivariogram 데이터로만 연산이 되는 작업으로 최대 시간 복잡도는 $O(n^2)$이다. 하지만 비교적 작은 입력값과 그 개수가 정해져 있는 부분이기 때문에 탐색 범위나 점 데이터의 밀도와 상관없이 거의 일정한 실행시간을 보이고 있다. Prediction 부분의 최대 시간 복잡도는 LU 분해법 구간으로 $O(\frac{2}{3}n^3)$인데, 이는 좌표 예측 내의 알고리즘 중에서도 가장 높은 복잡도를 가지며 위의 그래프를 기준으로 Data 2-5의 4.3배부터 Data 2-3의 6.9배까지 MapReduce를 적용한 실행시간이 단축되었고, 평균적으로 5.7배 실행시간이 단축되었다.

그래프에서 MapReduce와 C에서 모두 Semivariogram 부분과 Prediction 부분은 그래프 x축의 순서대로 그 실행시간이 커지는데 이는 입력 데이터의 특성과 밀접한 관련이 있다. Table 3을 참조하면 평균개수가 크고 표준편차가 클수록 그 실행속도는 급격히 느려지는 현상을 보이고 있다. 아래의 Figure 8은 (평균개수) × (표준편차)와 실행시간을 비교한 그래프이다. 아래의 그래프를 참조하면 (평균개수) × (표준편차)와 실행시간은 대체로 비례하는 양상을 보이고 있는데 이는 점 데이터의 평균개수가 크고 그 분포가 평균에서 멀어질수록 실행시간은 길어짐을 의미한다.

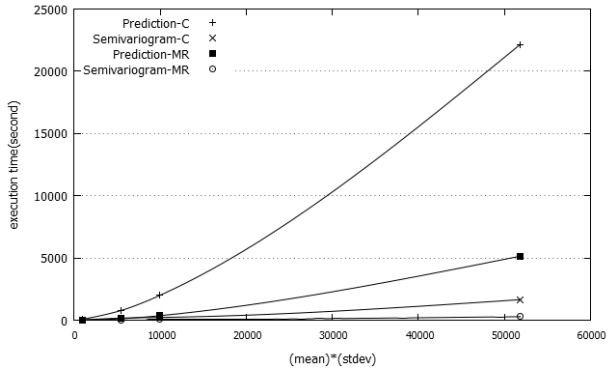


Fig. 8. Relation between Execution time with (mean) × (stdev) on Semivariogram and Prediction

우리가 제안하는 알고리즘의 핵심적인 네 부분인 Bucketing, Semivariogram, Fit, Prediction의 각각에 대한 실행시간을 비교했지만 지금부터 Reduce 개수에 따른 실행시간의 비교분석을 했다. MapReduce의 전체 실행 시간은 각 요소들의 초기화되는 시간, 클린업 등과 같은 하둡 프레임워크의 필수적인 실행 시간을 포함하고 있다. Figure 9에서 speedup은 다음과 같이 구한다. T_c는 순차적 프로그램의 전체 실행시간이고 T_{MR}은 MapReduce 프로그램의 전체 실행 시간이라고 한다면 $speedup = T_c / T_{MR}$이다. 위의 그래프에서 Data 1-5의 경우 순차적인 프로그램보다 최대 3.4배 정도 실행시간의 빠름을 나타낸다.

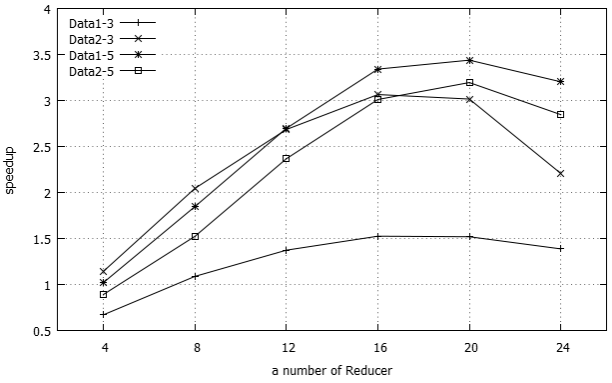


Fig. 9. Speedup of MapReduce approach by a number of Reduces

또한 Data 1-3은 다른 실험에 비해 speedup이 낮은데 이는 하둡 프레임워크의 특성상 작은 Task로 수행할 경우, 즉 실행시간이 짧은 Task일 경우 하둡 프레임워크의 오버헤드로 그 성능이 저하되는 현상을 보이고 있다.

7. 결 론

기존의 Kriging 보간법을 적용한 Gridding 알고리즘은 정확도가 필요한 작업에 유리한 특성이 있지만 그 연산이 복잡하고 실행시간이 오래 걸려서 크게 주목받지 못했다. 우리는 Kriging 보간법을 바탕으로 3D 점 데이터의 Gridding 알고리즘을 제시하고, 하둡 프레임워크를 활용해서 고성능-병렬화 구조의 제안과 실험을 했다. 본 실험에서는 순차적 프로그램만 비교분석 했는데 차후 연구에서는 다양한 병렬 플랫폼(CUDA, OpenMP, MPI 등)과 함께 성능을 분석할 예정이고, MapReduce 실험도 노드를 점진적으로 늘려가면서 그 성능과 병렬 오버헤드를 구체적으로 파악하고자 한다.

제안한 병렬 구조는 버켓을 분산시키고 각 Reduce에서 보간 작업을 병렬로 처리하기 때문에 Reduce의 개수도 성능에 큰 영향을 미친다. 대체로 총 프로세서 개수보다 하나 더 많이 Reduce를 할당하는 것이 최적에 근사한 실행시간을 얻을 수 있음을 확인했다. 또한 위의 실험에서 실행시간

이 짧은 Task는 병렬 효율성이 저하되는 구조적 문제를 가지고 있지만 보다 실행시간이 긴 Task에서는 효과적으로 병렬화가 됨을 확인하였다. 그리고 3D 점 데이터의 다양한 점 분포에 따른 실험에서도 비슷한 병렬 효율성을 보였으며, Bucketing 부분은 하둡 프레임워크의 오버헤드로 순차적 프로그램에 비해 많이 느렸지만, 버켓을 분산시킴으로써 전체 Gridding 알고리즘 실행시간을 순차적 프로그램에 비해 Data 1-5일 때 약 3.4배 향상시켰고, 실험한 모든 데이터의 평균적인 실행시간을 약 2.8배 단축시켰다.

따라서 본 논문에서 제안하는 MapReduce를 활용한 병렬 Gridding 알고리즘은 Kriging 같은 연산 집중적인 보간법을 분산 처리하는 데 유용할 것으로 판단된다.

Reference

[1] H. Woo, E. Kang, S. Wang, and K. H. Lee, "A new segmentation method for point cloud data", *International Journal of Machine Tools and Manufacture*, Vol.42, Issue.2, pp.167-178, 2002.

[2] V. Verma, R. Kumar, and S. Hsu, "3d building detection and modeling from aerial lidar data", *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Vol.2, pp.2213-2220, 2006.

[3] G. Sohn and I. Dowman, "Data fusion of high-resolution satellite imagery and LiDAR data for automatic building extraction", *ISPRS Journal of Photogrammetry and Remote Sensing*, Vol.62, Issue.1, pp.43-63, 2007.

[4] M. Himmelsbach, T. Luettel, and H. Wuensche, "Real-time object classification in 3D point clouds using point feature histograms", *IROS 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp.994-1000, 2009.

[5] M. Szarvas, U. Sakai, and J. Ogata, "Real-time pedestrian detection using LIDAR and convolutional neural networks", *Intelligent Vehicles Symposium*, IEEE, pp.213-218, 2006.

[6] G. Chust, I. Galparsoro, Á. Borja, J. Franco, and A. Uriarte, "Coastal and estuarine habitat mapping, using LIDAR height and intensity and multi-spectral imagery", *Estuarine, Coastal and Shelf Science*, Vol.78, Issue.4, pp.633-643, 2008.

[7] H. Lee, K. C. Slatton, B. E. Roth, and W. P. Cropper Jr, "Adaptive clustering of airborne LiDAR data to segment individual tree crowns in managed pine forests", *International Journal of Remote Sensing*, Vol.31, Issue.1, pp.117-139, 2010.

[8] N. El-Sheimy, C. Valeo, and A. Habib, "Digital terrain modeling: acquisition, manipulation and applications", Artech House, Boston, 2005.

[9] M. Oliver, R. Webster, and J. Gerrard, "Geostatistics in physical geography. Part I: theory", *Transactions of the Institute of British Geographers*, pp.259-269, 1989.

[10] Q. Guo, W. Li, H. Yu, and O. Alvarez, "Effects of topographic variability and lidar sampling density on several DEM interpolation methods", *Photogrammetric Engineering and Remote Sensing*, Vol.76, Issue.6, pp.701-712, 2010.

[11] C. D. Lloyd and P. M. Atkinson, "Deriving DSMs from LiDAR data with kriging", *International Journal of Remote Sensing*, Vol.23, Issue.12, pp.2519-2524, 2002.

[12] T. Hengl, "Finding the right pixel size", *Computers & Geosciences*, Vol.32, Issue.9, pp.1283-1298, 2006.

[13] P. K. Agarwal, L. Arge, and A. Danner, "From point cloud to grid DEM: A scalable approach", in *Progress in Spatial Data Handling*, Springer Berlin Heidelberg, pp.771-788, 2006.

[14] H. Wu, X. Guan, and J. Gong, "ParaStream: A parallel streaming Delaunay triangulation algorithm for LiDAR points on multicore architectures", *Computers & Geosciences*, Vol.37, Issue.9, pp.1355-1363, 2011.

[15] M. Hongchao and Z. Wang, "Distributed data organization and parallel data retrieval methods for huge laser scanner point clouds", *Computers & Geosciences*, Vol.37, Issue.2, pp.193-201, 2011.

[16] X. Guan and H. Wu, "Leveraging the power of multi-core platforms for large-scale geospatial data processing: Exemplified by generating DEM from massive LiDAR point clouds", *Computers & Geosciences*, Vol.36, Issue.10, pp.1276-1282, 2010.

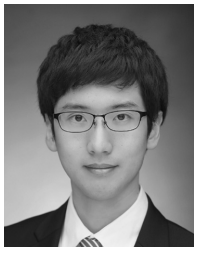
[17] S. Krishnan, C. Baru, and C. Crosby, "Evaluation of MapReduce for gridding LIDAR data", *2010 IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom)*, pp.33-40, 2010.

[18] K. Henneböhl, M. Appel, and E. Pebesma, "Spatial interpolation in massively parallel computing environments", *in Proceedings of the 14th AGILE International Conference on Geographic Information Science*, 2011.

[19] S. H. Han, J. Heo, H. G. Sohn, and K. Yu, "Parallel processing method for airborne laser scanning data using a pc cluster and a virtual grid", *Sensors*, Vol.9, Issue.4, pp.2555-2573, 2009.

[20] The Apache Common Mathematics Library [Internet] http://commons.apache.org/proper/commons-math/

[21] Surfer [Internet] http://www.goldensoftware.com/products/surfer



이 창 섭

e-mail : ckd3745@ajou.ac.kr

2005년~2012년 아주대학교 정보 및 컴퓨터공학(학사)

2012년~현 재 아주대학교 컴퓨터공학과 석사과정

관심분야 : 병렬 프로그래밍, 클라우드 컴퓨팅, LiDar 데이터 처리

Permata Nur Miftahur Rizki

e-mail : permatarizki@gmail.com

2003년～2007년 Electrical Engineering, Institute Teknologi Bandung(ITB) Indonesia(Bachelor degree)

2011년～현 재 Computer Engineering, Ajou University Master & Doctoral

관심분야 : Parallel Programming (Map Reduce, CUDA, & MPI), Cloud Computing, Big Data Analytics

이 희 진

e-mail : heezin.lee@berkeley.edu

2008년 미 플로리다대학교 전기컴퓨터공학(박사)

2008년～2011년 미 플로리다대학교(박사후 연구원)

2011년～현 재 미 캘리포니아대학교 책임 연구원

관심분야 : 신호처리, LiDar Remote Sensing, 영상처리, 패턴인식

오 상 윤

e-mail : syoh@ajou.ac.kr

2006년 미 인디애나대학교 전산학(박사)

2006년～2007년 SK 텔레콤

2008년～현 재 아주대학교 컴퓨터공학과 부교수

관심분야 : 고성능컴퓨팅(HPC), Large Scale Software, RDF, 클라우드 컴퓨팅