CUDA 및 분할-정복 기반의 효율적인 다차원 척도법

(An Efficient Multidimensional Scaling Method based on CUDA and Divide-and-Conquer)

박 성 인 † 황 규 백 ††
(Sungin Park) (Kyu-Baek Hwang)

요 약 다차원 척도법(multidimensional scaling)은 고차원의 데이터를 낮은 차원의 공간에 매핑(mapping)하여 데이터 간의 유사성을 표현하는 방법이다. 이는 주로 자질 선정 및 데이터를 시각화하는 데 이용된다. 그러한 다차원 척도법 중, 전통 다차원 척도법(classical multidimensional scaling)은 긴 수행 시간과 큰 공간을 필요로 하기 때문에 객체의 수가 많은 경우에 대해 적용하기 어렵다. 이는 유클리드 거리(Euclidean distance)에 기반한 $n \times n$ 상이도 행렬(dissimilarity matrix)에 대해 고유쌍 문제(eigenpair problem)를 풀어야 하기 때문이다(단, n은 객체의 개수). 따라서, n이 커질수록 수행 시간이 길어지며, 메모리 사용량 증가로 인해 적용할 수 있는 데이터 크기에 한계가 있다. 본 논문에서는 이러한 문제를 완화하기 위해 GPGPU 기술 중 하나인 CUDA와 분할-정복(divide-and-conquer) 기법을 활용한 효율적인 다차원 척도법을 제안하며, 다양한 실험을 통해 제안하는 기법이 객체의 개수가 많은 경우에 매우 효율적일 수 있음을 보인다.

키워드 : 다차원 척도법, GPGPU, CUDA, 비감독 학습, 분할-정복

Abstract Multidimensional scaling (MDS) is a widely used method for dimensionality reduction, of which purpose is to represent high-dimensional data in a low-dimensional space while preserving distances among objects as much as possible. MDS has mainly been applied to data visualization and feature selection. Among various MDS methods, the classical MDS is not readily applicable to data which has large numbers of objects, on normal desktop computers due to its computational complexity. More precisely, it needs to solve eigenpair problems on dissimilarity matrices based on Euclidean distance. Thus, running time and required memory of the classical MDS highly increase as n (the number of objects) grows up, restricting its use in large-scale domains. In this paper, we propose an efficient approximation algorithm for the classical MDS based on divide-and-conquer and CUDA. Through a set of experiments, we show that our approach is highly efficient and effective for analysis and visualization of data consisting of several thousands of objects.

Key words : multidimensional scaling, GPGPU, CUDA, unsupervised learning, divide-and-conquer

1. 서 론

다차원 척도법(multidimensional scaling)은 고차원의 데이터를 각 객체의 유사성에 기반하여 낮은 차원의 공간에 표현해 객체들 간의 관계를 분석하고 시각화하기 위한 기법으로, 데이터의 직관적인 분석 및 자질 선정 등에 활용된다[1,2]. 이러한 다차원 척도법 중 널리 이용되고 있는 전통 다차원 척도법(classical multidimensional scaling)은 객체의 개수를 n이라고 할 때, 시간 복잡도가 $O(n^3)$이기 때문에 객체의 개수가 많은 경우에는 실제 적용하기 어렵다[3]. 이는 $n \times n$ 상이도 행렬(dissimilarity matrix)에 대해 고유쌍 문제(eigenpair problem)를 풀어야 하기 때문이다. 한편, 최근 대규모 데이터에 대한 연산을 빠르게 수행하기 위하여 GPU의 병렬 프로세서들을 사용하여 연산을 병렬로 처리하는 CUDA[4], OpenCL[5], 라라비(Larrabee)[6] 등의 병렬 프로그래밍 기법이 소개되고 있다.

본 논문에서는 다차원 척도법 중 전통 다차원 척도법을 다루며, GPU를 활용한 병렬처리 기법 중 CUDA를 이용한다. 일반적으로 GPU의 메모리 크기에는 한계가 있으며, 따라서 CUDA를 적용할 수 있는 최대 행렬 크기에도 제한이 있다.[1] 이의 해결을 위해 본 논문에서는 분할-정복(divide-and-conquer) 기법을 적용하였다.

논문의 구성은 다음과 같다. 2절에서는 전통 다차원

· 본 논문은 숭실대학교 교내연구비 및 2007년도 정부재원(교육인적자원부 학술연구조성사업비)으로 한국학술진흥재단의 지원(KRF-2007-331-D00414)을 받아 연구되었음
· 이 논문은 제36회 추계학술발표회에서 '대규모 데이터를 위한 CUDA 및 분할-정복 접근법 기반의 효율적인 다차원 척도법'의 제목으로 발표된 논문을 확장한 것임

† 학생회원 : 숭실대학교 컴퓨터학과
sipark@ml.ssu.ac.kr
†† 정 회 원 : 숭실대학교 컴퓨터학부 교수
kbhwang@ssu.ac.kr

논문접수 : 2009년 12월 23일
심사완료 : 2010년 1월 28일

정보과학회논문지: 컴퓨팅의 실제 및 레터 제16권 제4호(2010.4)

1) 일반적으로 그래픽 카드의 메모리 용량은 64~512MB 정도이다. 또한 그래픽 작업에도 메모리를 사용하기 때문에, 그래픽 작업 이외의 작업들을 위해 사용할 수 있는 메모리의 용량은 전체 용량보다 작다.

척도법 및 GPU를 이용한 병렬처리 기법인 CUDA에 대하여 기술한다. 3절에서는 본 논문에서 제안하는 기법에 대하여 설명하며, 4절에서 실제 데이터를 이용하여 실험한 결과에 대하여 기술한다. 마지막으로 결론 및 향후 연구 방향에 대해 5절에서 논의한다.

2. 관련 연구

2.1 전통 다차원 척도법

전통 다차원 척도법은 객체들의 상이도(dissimilarity)에 기반한다. 이 때 객체들 간의 상이도를 측정하기 위하여 객체 간 유클리드 거리(Euclidean distance)를 계산하며, 이에 기반한 상이도 행렬을 생성하여 사용한다.

전통 다차원 척도법은 반복 수행(iteration)이 필요한 방법이 아니며, 일반적인 선형대수 방법을 사용하여 계산할 수 있다는 장점이 있다. 전통 다차원 척도법은 몇 가지 선형대수 기법을 활용하며 알고리즘은 다음과 같은 단계를 거치게 된다[1].

1. 상이도 행렬의 각 원소들을 제곱한다. $P^{(2)} = [p^2]$
2. J라는 행렬을 $J = I - n^{-1}11'$ 식을 이용하여 생성한다. (n: 객체의 개수, I: 단위행렬, 1: 일행렬) 생성한 J 행렬을 이용하여 다음의 수식에 따라 B라는 행렬을 생성한다. $B = -1/2JP^{(2)}J$
3. 행렬 B에 대하여 m개의 가장 큰 고유값들 $\lambda_1, ..., \lambda_m$과 고유값들의 고유벡터 $e_1, ..., e_m$들을 구한다. (m: 표현하고자 하는 저차원의 차원 수)
4. n개의 객체들을 m차원의 공간에 표현하기 위한 좌표 행렬 X를 다음 수식을 이용하여 계산한다. $X = E_m \Lambda_m^{1/2}$ (E_m: m차원의 고유벡터 행렬, $\Lambda_m^{1/2}$: m 차원의 고유값의 제곱근 값으로 이루어진 대각행렬)

2.2 CUDA(Compute Unified Device Architecture)

CUDA는 NVIDIA 사에서 개발한 GPGPU[2] 기술이다. 이는 GPU가 보유하고 있는 대량의 연산장치를 병렬로 활용하여 프로그램의 성능을 향상시키는 병렬처리 기법이다. CUDA의 구조는 여러 개의 스레드와, 스레드들로 이루어지는 블록 등으로 구성되어 있으며 그 구조는 아래 그림 1과 같다.

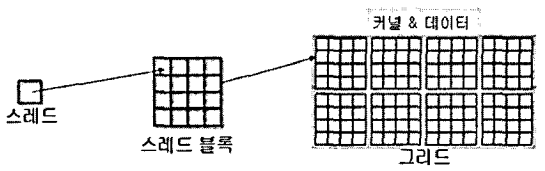

그림 1 CUDA의 구조 (커널은 함수를 의미)

CUDA 프로그래밍은 ANSI-C를 병렬처리 프로세서들을 위해 확장한 형태로 이루어져 있으며, CPU에서 수행되는 호스트 코드와 GPU에서 수행되는 디바이스 코드로 나뉜다[4]. GPU에서 수행되는 함수를 커널(kernel)이라고 부르며, 여러 개의 스레드에 대해 동일한 커널을 적용하는 SIMD(Single Instruction Multiple Data) 방식을 사용한다.

이러한 CUDA 프로그래밍을 통해 병렬처리를 구현하는 경우 고려해야 할 것이 있다. 이는 GPU를 사용하는 경우 일반적으로 고려해야 하는 것으로, 데이터를 그래픽 장치에 복사하고 그 연산결과를 다시 그래픽 장치로부터 복사해오는 과정의 비용(시간)이 높다는 것이다. 최근 그래픽 장치와 주기억장치 간의 데이터 전송속도를 증가시킨 새로운 인터페이스가 개발되어 사용되고 있지만 여전히 그 비용은 크다. 이러한 이유로, 데이터의 크기가 크거나 그래픽 장치와 주기억장치 간의 통신 횟수가 많아질수록 계산비용보다 데이터 통신비용이 더 커질 수 있다는 것을 고려하여 CUDA 프로그래밍을 적용해야 한다.

CUDA에서는 CUBLAS라는 라이브러리를 제공하는데, 이것은 LAPACK[3]의 함수들 중 일부를 CUDA로 미리 구현해 놓은 것이다[7]. 또한 EM Photonics 사에서 개발한 CULA라는 라이브러리도 사용할 수 있다[8]. CULA는 CUBLAS에서 제공하지 않는 LAPACK의 함수들을 CUDA로 구현하여 라이브러리로 제공한다.

3. 표본 추출 기반의 근사 다차원 척도법

일반적인 그래픽 카드가 장착된 PC에서는 객체의 개수가 약 3,000 이상인 경우, 메모리 문제로 인해 CUDA 환경에서 전통 다차원 척도법을 구현하는 것이 불가능하다. 본 논문에서는 이러한 문제를 처리하기 위해 아래와 같이 분할-정복 기법을 이용한다.

전체 기법의 개요는 다음과 같다. 우선, 전체 상이도 행렬을 여러 개의 부분 행렬로 분할하며 각 부분 행렬에서 임의로 표본을 추출한다. 표본 추출된 객체들을 이용하여 새로운 상이도 행렬을 생성하며, 분할된 부분 행렬들과 함께 각각 전통 다차원 척도법을 적용하고 이를 다시 결합하여 전체 데이터의 결과를 근사한다. 그 과정은 아래의 그림 2와 같다.

상세한 절차는 다음과 같다.

1. $n \times n$ 크기의 상이도 행렬 D의 대각(diagonal)을 따라 행렬을 p개로 분할한다. (n: 객체 개수, p: 분할 개수)
2. 각 부분 행렬 $D_1, D_2, ..., D_p$에서 s개만큼 임의로 객체

2) GPGPU: General Purpose computation on GPU의 약자로, GPU의 다중 코어를 그래픽 알고리즘이 아닌 일반적인 알고리즘을 위해 사용하고자 하는 병렬처리 기술

3) Fortran으로 구현된 공개 선형대수 라이브러리. http://www.netlib.org/lapack/

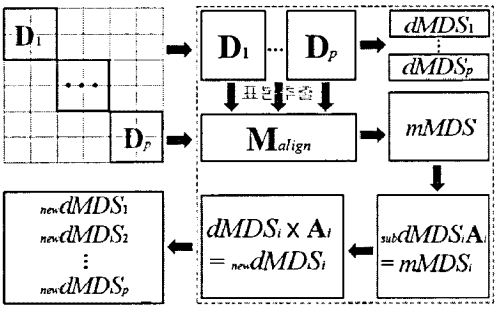

그림 2 CUDA와 분할-정복 기반의 근사 다차원 척도법 과정

를 표본 추출한다. (D_i: $(n/p) \times (n/p)$ 크기의 분할된 부분 행렬, s: 표본 추출 개수)

3. 각각의 부분 행렬 D_i에서 추출된 객체들을 이용하여 $sp \times sp$ 크기의 새로운 상이도 행렬 M_{align}을 생성한다.
4. M_{align} 행렬과 부분 행렬 D_i에 각각 전통 다차원 척도법을 적용하여 $dMDS_i$와 $mMDS$를 생성한다. ($dMDS_i$: 부분 행렬 D_i의 전통 다차원 척도법 결과, $mMDS$: M_{align} 행렬의 전통 다차원 척도법 결과)
5. D_i에서 이전에 표본 추출된 객체들의 전통 다차원 척도법 결과 좌표들을 $dMDS_i$와 $mMDS$에서 각각 추출하여 ${}_{sub}dMDS_i$와 $mMDS_i$를 생성한다.
6. 모든 ${}_{sub}dMDS_i$에 대한 선형변환 행렬 A_i를 선형 최소 자승법(linear least squares)에 의해 생성한다. 이는 다음의 수식으로 나타낼 수 있다.

$$ {}_{sub}dMDS_iA_i = mMDS_i $$

7. 계산된 모든 ${}_{sub}dMDS_i$의 A_i를 이용하여 D_i에서 표본 추출되지 않은 나머지 객체들에 대해 어파인 매핑(affine mapping)을 다음의 수식에 따라 적용한다.

$$ dMDS_iA_i = {}_{new}dMDS_i $$

8. 새롭게 생성된 ${}_{new}dMDS_i$ 결과들을 결합하여 근사된 형태의 전체 전통 다차원 척도법 결과를 생성한다.

위에서는 선형 최소 자승법을 통해 A_i를 계산하여 어파인 매핑을 적용한다. 이것은 ${}_{sub}dMDS_i$와 $mMDS_i$ 간의 동일한 객체의 결과 좌표를 서로 일치시키고, 이를 기준으로 D_i의 표본 추출하지 않은 나머지 객체들을 표본 추출된 객체들과 동일한 좌표공간에 매핑하기 위해서이다.

본 논문에서는 각 D_i와 M_{align} 행렬에 전통 다차원 척도법을 적용하는 과정과 어파인 매핑을 적용하기 위해 A_i를 계산하는 최소 선형 자승법 부분을 CUDA로 구현하여 적용하였다. 또한 계산한 A_i를 $dMDS_i$에 적용하는 부분도 CUDA로 구현하여 적용하였다.

위와 같은 분할-정복 기반의 다차원 척도법은 매우 일반적인 아이디어로 지금까지 다양한 변이를 가지며 적용되어 왔다[3,9,10]. 본 논문에서는 이러한 방법들 중 기본적이지만 정확도가 일반적인 경우에 평균적으로는 가장 높다고 알려진 임의 표본 추출 기반의 분할 정복 기법을 GPU의 메모리 한계를 극복하기 위해 적용하고 있다[11].

4. 실 험

본 논문에서 제안하는 방법에 대한 성능 평가를 위해 결과의 정확도와 프로그램의 수행 시간을 비교하였다. 비교 대상으로는 C# 및 Matlab으로 구현한 일반적인 전통 다차원 척도법을 경우에 따라 각각 활용하였다.

4.1 실험 환경

CUDA를 이용한 실험에 사용된 시스템은 Intel Quad Core Q6600과 4GB 메모리, 지포스 8600GT, CUDA 2.3, CULA Basic 1.0 및 윈도우 XP 프로페셔널 32-bit 환경이다. 그리고 CUDA가 아닌 CPU를 활용하는 경우에도 위와 동일한 환경에서 C#을 이용하여 실험을 하였으며, 위의 환경에 적용하기 어려운 매우 큰 크기의 데이터는 리눅스 기반의 32GB 메모리를 가지는 64-bit 서버에서 Matlab을 사용하여 실험하였다.

4.2 데이터 설명

본 논문에서 제안하는 방법의 검증을 위하여 3종류의 데이터를 사용하였으며, 표 1은 이를 설명하고 있다.

표 1 실험에 사용한 데이터

이 름	출 처	객체의 개수	차원	클래스
마이크로 어레이 1	Gene Expression Omnibus	3000	4000	N/A
마이크로 어레이 2	Gene Expression Omnibus	9300	1000	N/A
MNIST	The MNIST DB	10000	784	10

마이크로어레이[4] 1, 2는 GEO(Gene Expression Omnibus, http://www.ncbi.nlm.nih.gov/geo/)에서 제공하는 데이터를 가공한 것으로 각각 생쥐(*Mus musculus*)와 효모(*Saccharomyces cerevisiae*)의 유전자 정보를 다루고 있으며, 클래스 정보가 주어지지 않은 데이터이다. MNIST는 사람들이 손으로 직접 쓴 0~9 사이의 숫자를 스캔해서 생성한 28 × 28 크기의 이미지 데이터이다(http://yann.lecun.com/exdb/mnist/).

4.3 실험 설정

실험에서는 표 1의 데이터들을 모두 2 및 3차원으로 매핑하였다. 또한, CUDA와 분할-정복 기반의 근사기법

4) 마이크로어레이: 칩 위에 집적되어 있는 모든 유전자를 대상으로 하여 유전자 발현 차이를 한번의 실험으로 조사할 수 있는 대용량 유전자 발현 분석 시스템

표 2 데이터에 따른 실험 설정

데이터	상이도 행렬 크기	분할 개수 (p)	표본 크기 (s)
마이크로 어레이 1	3000 × 3000	10	30, 60, 90 120, 150
마이크로 어레이 2	9300 × 9300	10	100, 150
MNIST	10000 × 10000	10	100, 150

을 적용할 때, 전체 데이터의 분할 개수 p와 표본 추출 개수 s에 따른 수행 시간 및 정확도 변화를 관측하기 위해 설정을 변경하며, 각각의 설정에 대해 100번씩 실험을 반복하였다. 위의 표 2는 이를 정리하고 있다.

정확도 평가 기준으로는 피어슨 상관 계수(Pearson's correlation coefficient)를 계산하여 성능을 측정하였다. 구체적으로 설명하자면, 두 기법을 통해 나온 다차원 척도법 결과(저차원 상의 좌표 데이터)에 대해 각각 가능한 객체 사이의 모든 순서쌍에 대한 유클리드 거리를 계산하여 벡터를 생성하였다. 그리고 모든 순서쌍 사이의 거리로 구성된 두 개의 벡터(CPU에서의 결과 및 CUDA에서의 결과) 사이의 상관 계수를 계산하였다. 상관 계수의 값이 1에 가까우면, 두 결과가 객체들 사이의 거리를 선형적으로 유사하게 표현하고 있다는 의미가 된다.

4.4 결과 및 분석

이 절에서는 제안하는 기법의 속도의 향상 정도 및 정확도에서의 하락 정도를 기술한다. 실험 결과는 저차원의 차원 개수에 따라서 큰 차이를 보이지 않았으므로, 논문에서는 2차원에 대한 경우만을 제시한다. 또한, 표본 크기에 있어서는 표본의 크기가 커질수록 속도는 늦어지고, 정확도는 높아졌기 때문에 가장 큰 표본 크기 s = 150에 대한 결과만을 기술한다.

우선, 본 논문에서 제안하는 방법과 CPU를 이용한 일반적 방법간의 평균수행 시간을 비교한 결과는 아래의 그림 3과 같다.

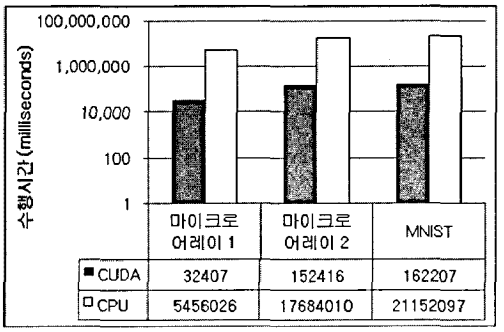

그림 3 제안 기법의 수행시간 향상 정도(세로축: 로그척도). 동일한 환경에서 100번 실험한 평균

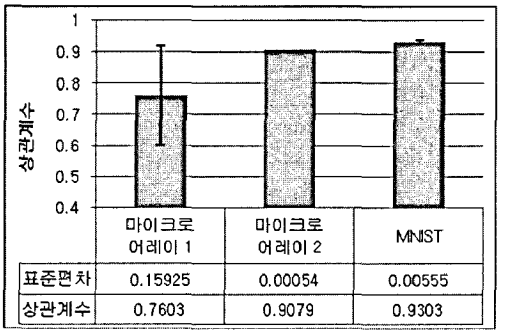

그림 4 제안 기법 결과의 정확도

그림 3에서 보면, CUDA와 분할-정복 기법을 사용한 제안 기법이 CPU만을 이용한 일반적인 기법에 비해 항상 100배 이상 빠른 것을 알 수 있다. 이는, 작업 수행 시간이 수 시간에서 수 분으로 단축됨을 의미한다. 물론, 이러한 속도에서의 향상은 필연적으로 결과의 정확도에 대한 훼손을 야기하며, 이는 본 논문에서 제안하는 기법이 CUDA에서의 메모리 제한을 다루기 위해 분할-정복 기반의 근사 기법을 적용하고 있기 때문이다.

그림 4에서는 CPU 만을 이용한 전통 다차원 척도법 결과와 제안하는 방법의 결과가 얼마나 유사한지 앞에서 설명한 상관 계수 기반 방법을 이용하여 측정한 결과를 보이고 있다.

그림 4에서 보면, 마이크로어레이 2 데이터 및 MNIST 데이터의 경우는 CPU만을 이용한 결과와의 유사한 정도가 상관 계수 0.9 이상임을 알 수 있다. 100번 실험에 대한 표준편차 역시 0.006 미만으로 임의 표본 추출에 기반하고 있지만, 그 결과는 안정적임을 알 수 있다.

반면, 마이크로어레이 1 데이터의 경우는 동일한 표본 추출 방법으로 많은 수의 표본을 추출했음에도 불구하고 편차가 다른 데이터에 비해 크고 평균 정확도 역시 떨어지고 있다. 이는 데이터의 분포 모양 등에 기인한 것으로 생각되며, 이에 대한 해결은 향후 연구 과제이다.

아래 그림 5~7은 본 논문의 제안 기법을 적용하여 얻은 전통 다차원 척도법 결과와, 일반 전통 다차원 척도법을 적용하여 얻은 결과를 2차원의 공간에 시각화한 것으로, 두 결과가 시각적으로는 상당히 유사하다는 것을 알 수 있다. 아래 그림에서 모든 결과는 100번의 실험 중 가장 정확하게 나온 경우이다.

5. 결 론

본 논문에서는 객체의 개수가 수 천 이상인 데이터에 전통 다차원 척도법을 효율적으로 적용하기 위해 CUDA와 분할-정복 기법을 제안하였다. 다수의 데이터에 대한 실험 결과에서 정확도는 약 75%~90%를 보였

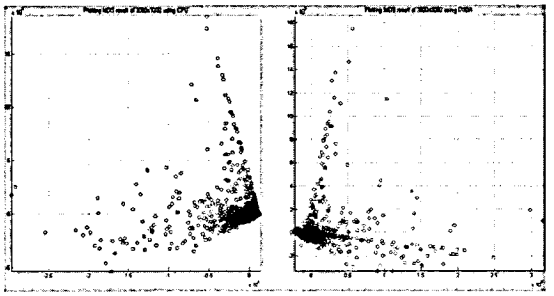

그림 5 마이크로어레이 1 데이터의 일반 전통 다차원 척도법과 CUDA와 분할-정복 기법 적용 결과 비교(왼쪽: 일반적 기법, 오른쪽: CUDA와 분할-정복 기법)

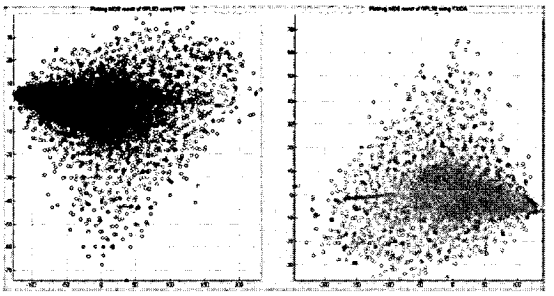

그림 6 마이크로어레이 2 데이터의 일반 전통 다차원 척도법과 CUDA와 분할-정복 기법 적용 결과 비교(왼쪽: 일반적 기법, 오른쪽: CUDA와 분할-정복 기법)

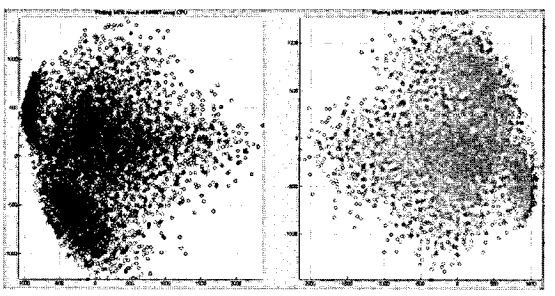

그림 7 MNIST 데이터의 일반 전통 다차원 척도법과 CUDA와 분할-정복 기법 적용 결과 비교(왼쪽: 일반적 기법, 오른쪽: CUDA와 분할-정복 기법)

으며, 평균 수행 시간은 약 100배 정도 단축되었다. 이는 기존의 PC에서 수 시간 걸리던 작업을 동일한 PC에서 그래픽 카드를 이용하여 수 분 이내에 할 수 있음을 의미한다. 물론, 정확도가 많이 훼손될 수도 있으나, 많은 경우 그 정도가 그리 크지 않을 수 있으며, 객체의 개수가 수 천 이상인 다양한 데이터를 신속하게 저차원 공간 상에서 시각화하려 할 때에 본 논문에서 제안하는 기법을 활용할 수 있다. 한편, 정확도가 많이 저하되거나 임의 표본 추출 결과에 따라 큰 편차를 보이는 경우는 데이터의 분포 모양과 관련이 있다고 추정되며, 이에 대한 보다 상세한 분석 및 해결은 향후 연구 과제이다.

참 고 문 헌

[1] Borg, I. and Groenen, P. J. F., Modern Multidimensional Scaling: Theory and Applications, Second Edition, Springer Science+Business Media, New York, NY, USA, 2005.

[2] Shashua, A. and Wolf, L., Kernel feature selection with side data using a spectral approach, *Proc. of ECCV 2004*, pp.39-53, 2004.

[3] Yang, T., Liu, J., McMillan, L., and Wang, W., A fast approximation to multidimensional scaling, Proc. of the ECCV 2006 Workshop on Computation Intensive Methods for Computer Vision, 2006.

[4] Kirk, D. and Hwu, W.-M., CUDA Textbook, Draft Version, 2009.

[5] OpenCL, Khronos group, http://www.khronos.org/opencl.

[6] Larrabee, Intel, http://www.intel.com/technology/visual/microarch.htm.

[7] CUDA CUBLAS Library Ver.2.3, NVIDIA Corporation, Santa Clara, CA, USA, 2009.

[8] CULA tools, EM Photonics, http://www.culatools.com.

[9] de Silva, V. and Tenenbaum, J.B., Sparse multidimensional scaling using landmark points, Technical Report, Stanford University, 2004.

[10] Faloutsos, C. and Lin, K.-I., FastMap: a fast algorithm for indexing, data-mining and visualization, *Proc. of ACM SIGMOD 1995*, pp.163-174, 1995.

[11] Pechenizkiy, M., Puuronen, S., and Tsymbal, A., The impact of sample reduction on PCA-based feature extraction for supervised learning, *Proc. of ACM SAC 2006 Data Mining Track*, pp.553-558, 2006.