High Performance Dimension Reduction and Visualization for Large High-dimensional Data Analysis

Jong Youl Choi*†, Seung-Hee Bae*†, Xiaohong Qiu* and Geoffrey Fox*†
*Pervasive Technology Institute
†School of Informatics and Computing
Indiana University, Bloomington IN, USA
{*jychoi, sebae, xqiu, gcf@indiana.edu*}

***Abstract*—Large high dimension datasets are of growing importance in many fields and it is important to be able to visualize them for understanding the results of data mining approaches or just for browsing them in a way that distance between points in visualization (2D or 3D) space tracks that in original high dimensional space. Dimension reduction is a well understood approach but can be very time and memory intensive for large problems. Here we report on parallel algorithms for Scaling by MAjorizing a COmplicated Function (SMACOF) to solve Multidimensional Scaling problem and Generative Topographic Mapping (GTM). The former is particularly time consuming with complexity that grows as square of data set size but has advantage that it does not require explicit vectors for dataset points but just measurement of inter-point dissimilarities. We compare SMACOF and GTM on a subset of the NIH PubChem database which has binary vectors of length 166 bits. We find good parallel performance for both GTM and SMACOF and strong correlation between the dimension-reduced PubChem data from these two methods.**

I. Introduction

Thanks to the most recent advancement in science and technologies, the amount of data to be processed or analyzed is rapidly growing and it is already beyond the capacity of the most commodity hardware we are using nowadays. To keep up with such fast development, study for data-intensive scientific data analyses [1] has been already emerging in recent years. Including dimension reduction algorithms which produce lower dimensional representation of high-dimensional data, which we are focusing in this paper, various machine learning algorithms and data mining techniques are also the main tasks challenged by such large and high dimensional data problem in this data deluge era. Unless developed and implemented carefully to overcome such limits, techniques will face soon the limit of usability.

Visualization of high-dimensional data in low-dimension is the core of exploratory data analysis in which users want to discover meaningful information obscured by the intrinsic complexity of data, usually caused by its high dimensionality. This task is also getting more difficult and challenged in recent days by the fast increasing amount of data to be analyzed. In most data analysis with such large and high-dimensional dataset, we have observed that such task is no more CPU bounded but rather memory bounded in that any single process or machine cannot hold the whole data in memory any more.

In this paper, we tackle this problem for developing high performance visualization for large and high-dimensional data analysis by using distributed resources with parallel computation. For this purpose, we will show how we developed two well-known dimension-reduction-based visualization algorithm Multidimensional Scaling (MDS) and Generative Topographic Mapping (GTM) in the distributed fashion so that one can utilize distributed memories and enable to process large and high dimensional dataset.

In this paper, we start with brief introduction of Multidimensional Scaling (MDS) and Generative Topographic Mapping (GTM), which are the dimension reduction algorithms to generate a configuration of the given high-dimensional data into target-dimensional space, and the correlation method we applied to compare outputs from different data visualization algorithms in Section II. Details of our parallelized version of MDS and GTM will be discussed in Section III. In the next, we show our performance results of our parallel version of MDS and GTM in various compute cluster settings and present the result processing up to 100,000 data points in Section IV followed by our closing discussion and future works in Section V.

II. Background

There are several kinds of dimension reduction algorithms, such as Principle Component Analysis (PCA), Generative Topographic Mapping (GTM) [2], [3], Self-Organizing Map (SOM) [4], Multidimensional Scaling (MDS) [5], [6], to name a few. Among those algorithms, we focus on MDS and GTM in our paper due to their popularity and theoretical strong backgrounds.

Although both GTM and SOM share the same object to find a map in low-dimensional user-defined space out of the data in high-dimensional space, GTM, however, finds a mapping based probability density model, which SOM lacks of. On the other hand, MDS tries to construct a mapping in target dimension with respect to the pairwise proximity information, mostly dissimilarity or distance.

A. Multidimensional Scaling (MDS)

Multidimensional scaling (MDS) [5], [6] is a general term for techniques of constructing a mapping for generally high-dimensional data into a target dimension (typically low dimension) with respect to the given pairwise proximity information. Mostly, MDS is used for achieving dimension reduction to visualize high-dimensional data into Euclidean low-dimensional space, i.e. two-dimension or three-dimension.

Generally, the proximity information, which is represented as an $N \times N$ dissimilarity matrix ($\boldsymbol{\Delta} = [\delta_{ij}]$), where N is the number of points (objects) and δ_{ij} is the dissimilarity between point i and j, is given for the MDS problem, and the dissimilarity matrix ($\boldsymbol{\Delta}$) should agree with the following constraints: (1) symmetricity ($\delta_{ij} = \delta_{ji}$), (2) nonnegativity ($\delta_{ij} \geq 0$), and (3) zero diagonal elements ($\delta_{ii} = 0$). The objective of MDS techniques is to construct a configuration of the given high-dimensional data into low-dimensional Euclidean space, while each distance between a pair of points in the configuration is approximated to the corresponding dissimilarity value as much as possible. The output of MDS algorithms could be represented as an $N \times L$ configuration matrix $\boldsymbol{X}$, whose rows represent each data points x_i ($i = 1, \ldots, N$) in L-dimensional space. It is quite straightforward to compute Euclidean distance between x_i and x_j in the configuration matrix $\boldsymbol{X}$, i.e. $d_{ij}(\boldsymbol{X}) = \|x_i - x_j\|$, and we are able to evaluate how well the given points are configured in the L-dimensional space by using suggested objective functions of MDS, called STRESS [7] or SSTRESS [8]. Definitions of STRESS (1) and SSTRESS (2) are following:

$$\sigma(\boldsymbol{X}) = \sum_{i<j\leq N} w_{ij}(d_{ij}(\boldsymbol{X}) - \delta_{ij})^2 \quad (1)$$

$$\sigma^2(\boldsymbol{X}) = \sum_{i<j\leq N} w_{ij}[(d_{ij}(\boldsymbol{X}))^2 - (\delta_{ij})^2]^2 \quad (2)$$

where $1 \leq i < j \leq N$ and w_{ij} is a weight value, so $w_{ij} \geq 0$.

As shown in the STRESS and SSTRESS functions, the MDS problems could be considered as a non-linear optimization problem, which minimizes STRESS or SSTRESS function in the process of configuring L-dimensional mapping of the high-dimensional data.

B. SMACOF & its Complexity

There are a lot of different algorithms to solve MDS problem, and Scaling by MAjorizing a COmplicated Function (SMACOF) [9], [10] is one of them. SMACOF is an iterative majorization algorithm to solve MDS problem with STRESS criterion. The iterative majorization procedure of the SMACOF could be thought of as Expectation-Maximization (EM) [11] approach. Although SMACOF has a tendency to find local minima due to its hill-climbing attribute, it is still a powerful method since it is guaranteed to decrease STRESS (σ) criterion monotonically. Instead of mathematical detail explanation of SMACOF algorithm, we illustrate the SMACOF procedure in this paper. For the mathematical details of SMACOF algorithm, please refer to [6].

Algorithm 1 SMACOF algorithm

1: $\boldsymbol{Z} \Leftarrow \boldsymbol{X}^{[0]}$;
2: $k \Leftarrow 0$;
3: $\varepsilon \Leftarrow$ small positive number;
4: $MAX \Leftarrow$ maximum iteration;
5: Compute $\sigma^{[0]} = \sigma(\boldsymbol{X}^{[0]})$;
6: **while** $k = 0$ or ($\Delta\sigma > \varepsilon$ and $k \leq MAX$) **do**
7: $\quad k \Leftarrow k + 1$;
8: $\quad \boldsymbol{X}^{[k]} = \boldsymbol{V}^{\dagger}\boldsymbol{B}(\boldsymbol{X}^{[k-1]})\boldsymbol{X}^{[k-1]}$
9: $\quad$ Compute $\sigma^{[k]} = \sigma(\boldsymbol{X}^{[k]})$
10: $\quad \boldsymbol{Z} \Leftarrow \boldsymbol{X}^{[k]}$;
11: **end while**
12: **return** $\boldsymbol{Z}$;

Alg. 1 illustrates the SMACOF algorithm for MDS solution. The main procedure of SMACOF is iterative matrix multiplications, called Guttman transform, as shown at Line 8 in Alg. 1, where $\boldsymbol{V}^{\dagger}$ is the Moore-Penrose inverse [12], [13] (or pseudo-inverse) of matrix $\boldsymbol{V}$. The $N \times N$ matrices $\boldsymbol{V}$ and $\boldsymbol{B}(\boldsymbol{Z})$ are defined as follows:

$$\boldsymbol{V} = [v_{ij}] \quad (3)$$

$$v_{ij} = \begin{cases} -w_{ij} & \text{if } i \neq j \\ \sum_{i\neq j} w_{ij} & \text{if } i = j \end{cases} \quad (4)$$

$$\boldsymbol{B}(\boldsymbol{Z}) = [b_{ij}] \quad (5)$$

$$b_{ij} = \begin{cases} -w_{ij}\delta_{ij}/d_{ij}(\boldsymbol{Z}) & \text{if } i \neq j \\ 0 & \text{if } d_{ij}(\boldsymbol{Z}) = 0, i \neq j \\ -\sum_{i\neq j} b_{ij} & \text{if } i = j \end{cases} \quad (6)$$

If the weights are equal to one ($w_{ij} = 1$) for all pairwise dissimilarity, then

$$\boldsymbol{V} = N\left(I - \frac{\boldsymbol{e}\boldsymbol{e}^t}{N}\right) \quad (7)$$

$$\boldsymbol{V}^{\dagger} = \frac{1}{N}\left(I - \frac{\boldsymbol{e}\boldsymbol{e}^t}{N}\right) \quad (8)$$

where $\boldsymbol{e} = (1, \ldots, 1)^t$ is one vector whose length is N. In this paper, equal weights is assumed and we use (8) for $\boldsymbol{V}^{\dagger}$.

As in Alg. 1, the computational complexity of SMACOF algorithm is $\mathcal{O}(N^2)$, since Guttman transform performs multiplication of $N \times N$ matrix and $N \times L$ matrix twice, typically $N \gg L$, and computing STRESS value, $\boldsymbol{B}(\boldsymbol{X}^{[k]})$, and $\boldsymbol{D}(\boldsymbol{X}^{[k]})$ also take $\mathcal{O}(N^2)$. In addition, the SMACOF algorithm requires $\mathcal{O}(N^2)$ memory because it needs several $N \times N$ matrices as in Table I. Due to the trends of digitization, data size increases enormously so it is critical to be able to investigate large data set. However, it is impossible to run SMACOF for large data set under a typical single node computer due to the memory requirement increases in $\mathcal{O}(N^2)$. In order to remedy the shortage of memory in a single node, the authors illustrate how to parallelize the SMACOF

algorithm via message passing interface (MPI) for utilizing distributed-memory cluster systems in Section III-A .

C. Generative Topographic Mapping (GTM)

Generative Topographic Mapping (GTM) is an unsupervised learning algorithm for modeling the probability density of data and finding a non-linear mapping of high-dimensional data in a low-dimension space. Contrast to the Self-Organizing Map (SOM) [4] which does not have any density model [2], GTM defines an explicit probability density model based on Gaussian distribution. For this reason, GTM is also known as a principled alternative to SOM [2]. The problem challenged by the GTM is to find the best set of parameters associated with Gaussian mixtures by using an optimization method, notably the Expectation-Maximization (EM) algorithm [3].

More specifically, The GTM algorithm is to find a non-linear manifold embedding of user-defined K latent discrete variables $\boldsymbol{z}_k$, usually form a rectangular grid, in low L-dimension space, called *latent space*, such that $\boldsymbol{z}_k \in \mathbb{R}^L (k = 1, ..., K)$, which can optimally represent the given N data points $\boldsymbol{x}_n \in \mathbb{R}^D (n = 1, ..., N)$ in the higher D-dimension space, called *data space* (usually $L \ll D$). This is achieved by the three steps: First, mapping the latent variables $\boldsymbol{z}_k$ in the latent space to the data space with respect to a non-linear mapping $f : \mathbb{R}^L \rightarrow \mathbb{R}^D$, such that map the point $\boldsymbol{y}_k \mapsto f(\boldsymbol{z}_k, \boldsymbol{W})$ with non-linear function f and its parameter set $\boldsymbol{W}$ to the data space. Secondly, estimating probability density between the mapped points $\boldsymbol{y}_k$ and the data points $\boldsymbol{x}_n$ by using the radially-symmetric Gaussian model in which the distribution is defined as an isotropic Gaussian probability centered on $\boldsymbol{y}_k$ with variance β^{-1}, such that

$$\mathcal{N}(\boldsymbol{x}_n | \boldsymbol{y}_k, \beta) = \left(\frac{\beta}{2\pi}\right)^{D/2} \exp\left\{-\frac{\beta}{2}\|\boldsymbol{x}_n - \boldsymbol{y}_k\|^2\right\}. \quad (9)$$

Thirdly, finding an optimal parameter set $\{\boldsymbol{W}, \beta\}$ which makes the following log-likelihood maximized:

$$\mathcal{L}(\boldsymbol{W}, \beta) = \underset{\boldsymbol{W}, \beta}{\operatorname{argmax}} \sum_{n=1}^{N} \ln\left\{\frac{1}{K}\sum_{k=1}^{K} \mathcal{N}(\boldsymbol{x}_n | \boldsymbol{y}_k, \beta)\right\}. \quad (10)$$

Since the last step is intractable, the GTM uses the EM method [11] to find an optimized solution as follows: In the E-step, compute the posterior probabilities, known as *responsibilities*, of each mapped point $\boldsymbol{y}_k$ for every data point $\boldsymbol{x}_n$ in the following form:

$$r_{kn} = \frac{\mathcal{N}(\boldsymbol{x}_n | \boldsymbol{y}_k, \beta)}{\sum_{k'=1}^{K} \mathcal{N}(\boldsymbol{x}_n | \boldsymbol{y}_{k'}, \beta)}. \quad (11)$$

In the M-step, maximize the expectation of log-likelihood (10) with respect to the parameter $\boldsymbol{W}$. As a result, the optimal $\boldsymbol{W}$ can be obtained by using the following matrix equation:

$$\boldsymbol{\Phi}^t \boldsymbol{G} \boldsymbol{\Phi} \boldsymbol{W} = \boldsymbol{\Phi}^t \boldsymbol{R} \boldsymbol{X}, \quad (12)$$

where $\boldsymbol{\Phi}$ is the $K \times M$ design matrix which holds $\boldsymbol{Y} = \boldsymbol{\Phi}\boldsymbol{W}$ for the $K \times D$ matrix $\boldsymbol{Y}$ containing mapped points, $\boldsymbol{X}$ is the $N \times D$ matrix containing the data points, $\boldsymbol{R}$ is the $K \times N$ responsibility whose (k, n)-th element is r_{kn} as in (11), and $\boldsymbol{G}$ is an $K \times K$ diagonal matrix whose k-th diagonal element $g_k = \sum_{n=1}^{N} r_{kn}$. Main matrices used in GTM are summarized in Table II.

Since the details of GTM algorithm is out of this paper's scope, we recommend readers to refer to the original GTM papers [2], [3] for more details. In Section III, we will discuss how we develop parallel GTM implementation based on the above algorithm.

D. Correlation measurement for comparison

In the fields of data analysis and machine learning area, we have various types of visualization algorithms available to use and each of them has its own purposes and characteristics which will differentiate its output from others even with using the same dataset as an input. To compare such different outputs, we need to quantify similarities (or dissimilarities) of outputs generated from different algorithms.

In our paper, we have measured correlations between two outputs by using so-called Canonical Correlations Analysis (CCA). CCA is a classical statistical method to measure correlations between two sets of variables in their linear relationships [14]. Different from ordinary correlation measurement methods, CCA has the ability to measure correlations of multidimensional datasets by finding an optimal projection to maximize the correlation in the subspace spanned by features and it has been successfully used in various areas [15]–[17].

Given two column vectors $X = (x_1, ..., x_m)^t$ and $Y = (y_1, ..., y_n)^t$ of random variables with finite second moments, one may define the cross-covariance $\Sigma_{XY} = \operatorname{cov}(x, y)$ to be the $n \times m$ matrix whose (i, j)-th entry is the covariance $\operatorname{cov}(x_i, y_j)$. CCA seeks two coefficient vectors a and b, known as *canonical variables*, such that the random variables $a^t X$ and $b^t Y$ maximize the correlation such that

$$\rho = \underset{a,b}{\operatorname{argmax}} \frac{\operatorname{cov}(a^t X, b^t Y)}{||a^t X||\,||b^t Y||} \quad (13)$$

We call the random variables $u = a^t X$ and $v = b^t Y$ are the first pair of canonical variables. Then, one seeks vectors uncorrelated with the first pair of canonical variables; this gives the second pair of canonical variables. This procedure continues $\min(m, n)$. In a nutshell, canonical variables U and V are can be obtained by

$$a = \operatorname{eig}(\Sigma_{11}^{-1}\Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21}) \quad (14)$$

$$b = \operatorname{eig}(\Sigma_{22}^{-1}\Sigma_{21}\Sigma_{11}^{-1}\Sigma_{12}) \quad (15)$$

where $\Sigma_{12} = \operatorname{cov}(X, Y)$, $\Sigma_{11} = \operatorname{cov}(X, X)$, $\Sigma_{22} = \operatorname{cov}(Y, Y)$, and $\operatorname{eig}(A)$ computes eigenvectors of matrix A. More detailed steps for derivation can be found from [15], [18], [19]

III. High Performance Visualization

We have observed that processing very large dataset is no more *cpu-bounded* computation but rather it is *memory-bounded* in that memory consumption is beyond the ability

TABLE I
MAIN MATRICES USED IN SMACOF

Matrix	Size	Description
$\boldsymbol{\Delta}$	$N \times N$	Matrix for the given pairwise dissimilarity $[\delta_{ij}]$
$\boldsymbol{D}(\boldsymbol{X})$	$N \times N$	Matrix for the pairwise Euclidean distance of mapped in target dimension $[d_{ij}]$
$\boldsymbol{V}$	$N \times N$	Matrix defined the value v_{ij} in (3)
$\boldsymbol{V}^\dagger$	$N \times N$	Matrix for pseudo-inverse of $\boldsymbol{V}$
$\boldsymbol{B}(\boldsymbol{Z})$	$N \times N$	Matrix defined the value b_{ij} in (5)
$\boldsymbol{W}$	$N \times N$	Matrix for the weight of the dissimilarity $[w_{ij}]$
$\boldsymbol{X}^{[k]}$	$N \times L$	Matrix for current L-dimensional configuration of N data points $\boldsymbol{x}_i^{[k]}(i = 1, \ldots, N)$
$\boldsymbol{X}^{[k-1]}$	$N \times L$	Matrix for previous L-dimensional configuration of N data points $\boldsymbol{x}_i^{[k-1]}(i = 1, \ldots, N)$

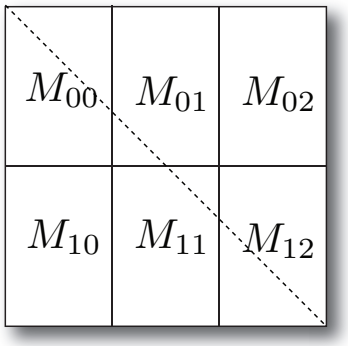


Fig. 1. $N \times N$ matrix decomposition of parallel SMACOF with 6 processes and 2×3 block decomposition. Dashed line represents where diagonal elements are.

of a single process or even a single machine. Thus, running machine learning algorithms to process large dataset, including MDS and GTM discussed in this paper, in a distributed fashion is crucial so that we can utilize multiple processes and distributed resources to handle very large data which usually not fit in the memory of a single process or a compute node. The problem becomes more obvious if the running OS is 32-bit which can handle at most 4GB virtual memory per process. To process large data with efficiency, we have developed parallel version of MDS and GTM by using Message Passing Interface (MPI) fashion. In the following we will discuss more details how we decompose the MDS and GTM algorithm to fit in a memory limit in a single process or machine and implemented them by using MPI primitives.

A. Parallel SMACOF

Table I describes frequently used matrices in SMACOF algorithm, and memory requirement of SMACOF algorithm increases quadratically as N increases. For the small dataset, memory would not be any problem. However, it turns out to be critical problem when we deal with large data set, such as thousands or even millions. For instance, if $N = 10,000$, then one $N \times N$ matrix of 8-byte double-precision numbers consumes 800 MB of main memory, and if $N = 100,000$, then one $N \times N$ matrix uses 80 GB of main memory. To make matters worse, SMACOF algorithm generally needs six $N \times N$ matrices, so at least 480 GB of memory is required to run SMACOF with 100,000 data points without considering two $N \times L$ configuration matrices in Table I.

If the weight is uniform ($w_{ij} = 1$, $\forall i, j$), we can use only four constants for representing $N \times N$ $\boldsymbol{V}$ and $\boldsymbol{V}^\dagger$ matrices in order to saving memory space. We, however, still need at least three $N \times N$ matrices, i.e. $\boldsymbol{D}(\boldsymbol{X})$, $\boldsymbol{\Delta}$, and $\boldsymbol{B}(\boldsymbol{X})$, which requires 240 GB memory for the above case, which is still infeasible amount of memory for a typical computer. That is why we have to implement parallel version of SMACOF with MPI.

To parallelize SMACOF, it is essential to ensure load balanced data decomposition as much as possible. Load balance is important not only for memory distribution but also for computation distribution, since parallelization makes implicit benefit to computation as well as memory distribution, due to less computing per process. One simple approach of data decomposition is that we assume $p = n^2$, where p is the number of processes and n is an integer. Though it is relatively less complicated decomposition than others, one major problem of this approach is that it is a quite strict constraint to utilize available computing processors (or cores). In order to release that constraint, we decompose an $N \times N$ matrix to $m \times n$ block decomposition, where m is the number of block rows and n is the number of block columns, and the only constraint of the decomposition is $m \times n = p$, where $1 \leq m, n \leq p$. Thus, each process requires only approximately $1/p$ of full memory requirements of SMACOF algorithm. Fig. 1 illustrates how we decompose each $N \times N$ matrices with 6 processes and $m = 2, n = 3$. Without loss of generality, we assume $N\%m = N\%n = 0$ in Fig. 1.

A process $P_k, 0 \leq k < p$ (sometimes, we will use P_{ij} for matching M_{ij}) is assigned to one rectangular block M_{ij} with respect to simple block assignment equation in (16):

$$k = i \times n + j \tag{16}$$

where $0 \leq i < m, 0 \leq j < n$. For $N \times N$ matrices, such as $\boldsymbol{\Delta}, \boldsymbol{V}^\dagger, \boldsymbol{B}(\boldsymbol{X}^{[k]})$, and so on, each block M_{ij} is assigned to the corresponding process P_{ij}, and for $\boldsymbol{X}^{[\boldsymbol{k}]}$ and $\boldsymbol{X}^{[\boldsymbol{k-1}]}$ matrices, $N \times L$ matrices, each process has full $N \times L$ matrices because these matrices are relatively much small size and it results in reducing a number of additional message passing. By scattering decomposed blocks to distributed memory, now we are able to run SMACOF with huge data set as much as distributed memory allows in the cost of message passing overheads and complicated implementation.

At the iteration k in Alg. 1, the application should be possible to acquire following information to do Line 8 and Line 9 in Alg. 1: $\boldsymbol{\Delta}$, $\boldsymbol{V}^\dagger$, $\boldsymbol{B}(\boldsymbol{X}^{[k-1]})$, $\boldsymbol{X}^{[k-1]}$, and $\sigma^{[k]}$. One good feature of SMACOF algorithm is that some of matrices are invariable, i.e. $\boldsymbol{\Delta}$ and $\boldsymbol{V}^\dagger$, through the iteration. On the other hand, $\boldsymbol{B}(\boldsymbol{X}^{[k-1]})$ and STRESS ($\sigma^{[k]}$) value keep changing at each iteration, since $d_{ij}(\boldsymbol{X}^{[k]})$ varies every

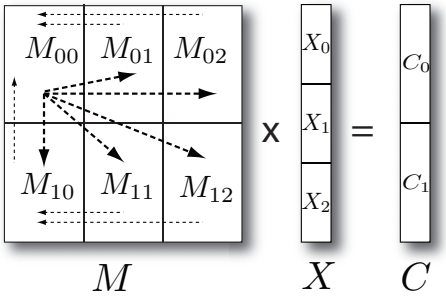


Fig. 2. Parallel matrix multiplication of $N \times N$ matrix and $N \times L$ matrix based on the decomposition of Fig. 1

iteration. In addition, in order to update $\boldsymbol{B}(\boldsymbol{X}^{[k-1]})$ and STRESS ($\sigma^{[k]}$) value in each iteration, we have to take $N \times N$ matrices information into account, so related processes should communicate via MPI primitives to obtain necessary information. Therefore, it is necessary to design message passing schemes to do parallelization for calculating $\boldsymbol{B}(\boldsymbol{X}^{[k-1]})$ and STRESS ($\sigma^{[k]}$) value as well as parallel matrix multiplication in Line 8 in Alg. 1.

Computing STRESS in (1) can be implemented simply through `MPI_Allreduce`. On the other hand, calculation of $\boldsymbol{B}(\boldsymbol{X}^{[k-1]})$ and parallel matrix multiplication is not simple, specially for the case of $m \neq n$. Fig. 2 depicts how parallel matrix multiplication applies between an $N \times N$ matrix $\boldsymbol{M}$ and an $N \times L$ matrix $\boldsymbol{X}$. Parallel matrix multiplication for SMACOF algorithm is implemented in three-step of message communication via MPI primitives. Block matrix multiplication of Fig. 2 for acquiring C_i $(i = 0, 1)$ can be written as follows:

$$C_i = \sum_{0 \leq j < 3} M_{ij} \cdot X_j \tag{17}$$

Since M_{ij} of $N \times N$ matrix is accessed only by the corresponding process P_{ij}, computing $M_{ij} \cdot X_j$ part is done by P_{ij}, and the each computed sub-matrix, which is $\frac{N}{2} \times L$ matrix for Fig. 2, is sent to the process assigned M_{i0} by MPI primitives, such as `MPI_Send` and `MPI_Receive`. Then the process assigned M_{i0}, say P_{i0}, sums the received sub-matrices to generate C_i, and send C_i block to P_{00}. Finally, P_{00} combines sub-matrix block C_i $0 \leq i < m$ to construct $N \times L$ matrix $\boldsymbol{C}$, and broadcast it to all other processes by `MPI_Broadcast`. Each arrows in Fig. 2 represents message passing direction. Thin dashed arrow lines describes message passing of $\frac{N}{2} \times L$ sub-matrices by `MPI_Send` and `MPI_Receive`, and message passing of matrix $\boldsymbol{C}$ by `MPI_Broadcast` is represented by thick dashed arrow lines. The pseudo code for parallel matrix multiplication in SMACOF algorithm is in Alg. 2

For the purpose of parallel computing $\boldsymbol{B}(\boldsymbol{X}^{[k-1]})$, whose elements b_{ij} is defined in (6), message passing mechanism in Fig. 3 should be applied under 2×3 block decomposition as in Fig. 1. Since $b_{ss} = -\sum_{s \neq j} b_{sj}$, a process P_{ij}, which is assigned to $\boldsymbol{B}_{ij}$, should communicate a vector $\boldsymbol{s}_{ij}$, whose element is the sum of corresponding rows, with

Algorithm 2 Pseudo-code for distributed parallel matrix multiplication in SMACOF algorithm

Input: $\boldsymbol{M}_{ij}, \boldsymbol{X}$
1: /* $m =$ `Row Blocks`, $n =$ `Column Blocks` */
2: /* $i =$ `Rank-In-Row`, $j =$ `Rank-In-Column` */
3: $\boldsymbol{T}_{ij} = \boldsymbol{M}_{ij} \cdot \boldsymbol{X}_j$
4: **if** $j \neq 0$ **then**
5: Send $\boldsymbol{T}_{ij}$ to P_{i0}
6: **else**
7: **for** $j = 1$ to $n - 1$ **do**
8: Receive $\boldsymbol{T}_{ij}$ from P_{ij}
9: **end for**
10: Generate $\boldsymbol{C}_i$
11: **end if**
12: **if** $i == 0$ and $j == 0$ **then**
13: **for** $i = 1$ to $m - 1$ **do**
14: Receive $\boldsymbol{C}_i$ from P_{i0}
15: **end for**
16: Combine $\boldsymbol{C}$ with $\boldsymbol{C}_i$ where $i = 0, \ldots, m - 1$
17: Broadcast $\boldsymbol{C}$ to all processes
18: **else if** $j == 0$ **then**
19: Send $\boldsymbol{C}_i$ to P_{00}
20: Receive Broadcasted $\boldsymbol{C}$
21: **else**
22: Receive Broadcasted $\boldsymbol{C}$
23: **end if**

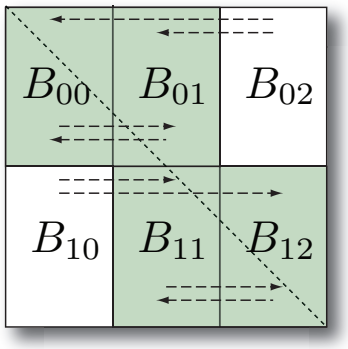


Fig. 3. Calculation of $\boldsymbol{B}(\boldsymbol{X}^{[k-1]})$ matrix with regard to the decomposition of Fig. 1.

processes assigned sub-matrix of the same block-row P_{ik}, where $k = 0, \ldots, n - 1$, unless the number of column blocks is 1 ($n == 1$). In Fig. 3, the diagonal dashed line indicates the diagonal elements, and the green colored blocks are diagonal blocks for each block-row. Note that the definition of *diagonal blocks* is a block which contains at least one diagonal element of the matrix $\boldsymbol{B}(\boldsymbol{X}^{[k]})$. Also, dashed arrow lines illustrate message passing direction. Alg. 3 shows the pseudo-code of computing sub-block B_{ij} in process P_{ij} with MPI primitives.

Algorithm 3 Pseudo-code for calculating assigned sub-matrix $\boldsymbol{B}_{ij}$ defined in (6) for distributed-memory decomposition in SMACOF algorithm

Input: $\boldsymbol{M}_{ij}, \boldsymbol{X}$

/* $m =$ `Row Blocks`, $n =$ `Column Blocks` */
/* $i =$ `Rank-In-Row`, $j =$ `Rank-In-Column` */
/* We assume that subblock B_{ij} is assigned to process P_{ij} */
Find diagonal blocks in the same row (row i)
if $\boldsymbol{B}_{ij} \notin$ diagonal blocks **then**
    compute elements b_{st} of $\boldsymbol{B}_{ij}$
    Send a vector $\boldsymbol{s}_{ij}$, whose element is the sum of corresponding rows, to P_{ik}, where $\boldsymbol{B}_{ik} \in$ diagonal blocks
else
    compute elements b_{st} of $\boldsymbol{B}_{ij}$, where $s \neq t$
    Receive a vector $\boldsymbol{s}_{ik}$, whose element is the sum of corresponding rows, where $k = 1, \ldots, n$ from other processes in the same block-row
    Send $\boldsymbol{s}_{ij}$ to other processes in the same block-row $\in$ diagonal blocks
    Compute $\boldsymbol{b}_{ss}$ elements based on the row sums.
end if

TABLE II
MAIN MATRICES USED IN GTM FOR N DATA POINTS IN D-DIMENSION WITH K LATENT POINTS IN L-DIMENSION.

Matrix	Size	Description
$\boldsymbol{Z}$	$K \times L$	Matrix for K latent points $\boldsymbol{z}_k (k = 1, ..., K)$
$\boldsymbol{\Phi}$	$K \times M$	Design matrix with M-dimension
$\boldsymbol{W}$	$M \times D$	Matrix for parameters
$\boldsymbol{Y}$	$K \times D$	Matrix for K mapped points $\boldsymbol{y}_k (k = 1, ..., K)$
$\boldsymbol{X}$	$N \times D$	Matrix for N data points $\boldsymbol{x}_n (n = 1, ..., N)$
$\boldsymbol{R}$	$K \times N$	Matrix for K responsibilities for each N data points

B. Parallel GTM

Among many matrices allocated in memory for processing in GTM as summarized in Table II, the responsibility matrix $\boldsymbol{R}$ is the most biggest one. For example, the matrix $\boldsymbol{R}$ for 8,000 latent points, corresponding to 20x20x20 3D grid, with 100,000 data points needs at least 6.4GB memory space saving 8-byte double precision numbers and even without considering additional memory requirements for other matrices, this easily prevents us from processing large data set in GTM by using a single process or machine. Thus, we have focused on the decomposition of responsibility matrix $\boldsymbol{R}$ in developing parallel GTM.

As shown in Fig. 4, in our parallel GTM we decompose the design matrix $\boldsymbol{\Phi} \in \mathbb{R}^{K \times M}$ into m row-based sub-blocks denoted by $\{\boldsymbol{\Phi}_i\}_{i=0}^{m-1}$ so that each sub-block $\boldsymbol{\Phi}_i$ has approximately K/m rows of $\boldsymbol{\Phi}$. In the same way, we also decompose the data matrix $\boldsymbol{X}$ into n row-based sub-blocks $\{\boldsymbol{X}_j\}_{j=0}^{n-1}$, each of which having approximately N/n rows of $\boldsymbol{X}$. Then, we can compute the sub-matrix $\boldsymbol{R}_{ij}$ $(i = 0, ..., m-1, j = 0, ..., n-1)$ for the latent points $\boldsymbol{Y}_i = \boldsymbol{\Phi}_i \boldsymbol{W}$ ($\boldsymbol{Y}_i$ is also i-th row-based sub-block of $\boldsymbol{Y}$) and data points $\boldsymbol{X}_j$ on the m-by-n mesh of logical compute grid where (i, j)-th node computes $\boldsymbol{R}_{ij}$ which consumes only $1/mn$ of memory space for the full matrix $\boldsymbol{R}$. Without loss of generality, we assume that $K\%m = N\%n = 0$ and denote $\bar{K} = K/m$ and $\bar{N} = N/n$.

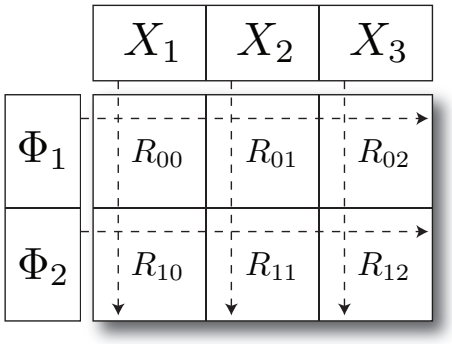


Fig. 4. Data decomposition of parallel GTM for computing responsibility matrix $\boldsymbol{R}$ by using 2-by-3 mesh of computing nodes.

Since our parallel GTM algorithm is not a pleasingly parallel application in which no dependency is needed between compute nodes but rather a typical parallel problem which can be solved by using a general map-reduce approach. To have systematic communication model, we can use MPI's cartesian grid topology in our m-by-n compute grid so that each node belongs to both row communications and column communications, denoted by `ROW-COMM` and `COL-COMM` respectively hereafter.

More details of our parallel GTM algorithm is as follows.

1) **Initialization** Prepare subblock data $\{\boldsymbol{\Phi}_i\}_{i=0}^{m-1}$ and $\{\boldsymbol{X}_j\}_{j=0}^{n-1}$ and distribute them to i-th row members and j-th column members respectively in the m-by-n compute grid.
2) **Responsibility** Initialize the responsibility matrix $\boldsymbol{R}_{ij}$ by setting (a, b)-th element $r_{ab} = \mathcal{N}(x_b | y_a)$, as defined in (9), for $a = 0, ..., \bar{K}-1$ and $b = 0, ..., \bar{N}-1$. Compute the column sum $\boldsymbol{c}_{ij} \in \mathbb{R}^{\bar{N}}$ of $\boldsymbol{R}_{ij}$ and exchange it with row members to get $\boldsymbol{c}_i = \sum_{j=0}^{n-1} c_{ij}$ and compute
$$\boldsymbol{R}_{ij} = \boldsymbol{R}_{ij} \oslash (\boldsymbol{e}\boldsymbol{c}_j^t) \tag{18}$$
where $\boldsymbol{e}$ is a vector of $(1, ..., 1)^t \in \mathbb{R}^{\bar{K}}$ and $\oslash$ represents element-wise division.
3) **Optimization** Compute a row-sum vector $\boldsymbol{g}_{ij} = \boldsymbol{R}_{ij}\boldsymbol{e}$ and exchange it with row members to get $\boldsymbol{g}_i = \sum_{j=0}^{n-1} \boldsymbol{g}_{ij}$. Compute a matrix $\boldsymbol{A}_i = \boldsymbol{\Phi}_i^t \boldsymbol{G}_i \boldsymbol{\Phi}_i$ where $\boldsymbol{G}_i$ is a diagonal matrix whose diagonal elements are $\boldsymbol{g}_i$ and exchange with column members to compute $\boldsymbol{A} = \sum_{i=0}^{m-1} \boldsymbol{A}_i$. Prepare another matrix $\boldsymbol{B}_{ij} = \boldsymbol{\Phi}_i^t \boldsymbol{R}_{ij} \boldsymbol{X}_j$ and exchange it with row-members to get $\boldsymbol{B}_i = \sum_{j=0}^{n-1} \boldsymbol{B}_{ij}$, followed by exchanging with column members to compute $\boldsymbol{B} = \sum_{i=0}^{m-1} \boldsymbol{B}_i$. Finally, solve $\boldsymbol{A}\boldsymbol{W} = \boldsymbol{B}$ with respect to the parameter matrix $\boldsymbol{W}$ and update latent points $\boldsymbol{Y}_i = \boldsymbol{\Phi}_i \boldsymbol{W}$. The last two steps will

Algorithm 4 Pseudo-code for distributed parallel GTM computing running on (i, j)-th compute node.

Input: $\boldsymbol{\Phi}_i, \boldsymbol{X}_j$
1: Prepare $\boldsymbol{R}_{ij} \in \mathbb{R}^{\bar{K} \times \bar{N}}$ by setting its (a, b)-th by (9)
2: $\boldsymbol{c}_{ij} \leftarrow \boldsymbol{R}_{ij}^t \boldsymbol{e}$ where $\boldsymbol{e} = (1, ..., 1)^t$
3: $\boldsymbol{c}_j \leftarrow$ `MPI_Allreduce`$(\boldsymbol{c}_{ij}, \texttt{MPI\_SUM}, \texttt{COL\_COMM})$
4: $\boldsymbol{R}_{ij} \leftarrow \boldsymbol{R}_{ij} \oslash (\boldsymbol{e}\boldsymbol{c}_j^t)$
5: $\boldsymbol{g}_{ij} \leftarrow \boldsymbol{R}_{ij}\boldsymbol{e}$
6: $\boldsymbol{g}_i \leftarrow$ `MPI_Allreduce`$(\boldsymbol{g}_{ij}, \texttt{MPI\_SUM}, \texttt{ROW\_COMM})$
7: $\boldsymbol{A}_i = \boldsymbol{\Phi}_i^t \boldsymbol{G}_i \boldsymbol{\Phi}_i$ where $\boldsymbol{G}_i = \mathrm{diag}(\boldsymbol{g}_i)$
8: $\boldsymbol{A} \leftarrow$ `MPI_Allreduce`$(\boldsymbol{A}_i, \texttt{MPI\_SUM}, \texttt{COL\_COMM})$
9: $\boldsymbol{B}_{ij} = \boldsymbol{\Phi}_i^t \boldsymbol{R}_{ij} \boldsymbol{X}_j$
10: $\boldsymbol{B}_i \leftarrow$ `MPI_Allreduce`$(\boldsymbol{B}_{ij}, \texttt{MPI\_SUM}, \texttt{ROW\_COMM})$
11: $\boldsymbol{B} \leftarrow$ `MPI_Allreduce`$(\boldsymbol{B}_i, \texttt{MPI\_SUM}, \texttt{COL\_COMM})$
12: Solve $\boldsymbol{W} = \boldsymbol{A}^{-1}\boldsymbol{B}$
13: Update $\boldsymbol{Y}_i \leftarrow \boldsymbol{\Phi}_i \boldsymbol{W}$

continue to run until we find the parameter matrix $\boldsymbol{W}$ converged.

Exchanging data with row (or column) members of the grid and collecting them, we can use a MPI primitive function `MPI_Allreduce` with `MPI_SUM` collective opeartion. A pseudo code with MPI functions is shown in Alg. 4.

IV. Performance and Correlation Measurement

For the performance analysis of both parallel SMACOF and parallel GTM discussed in this paper, we have applied our parallel algorithms for high-dimensional data visualization in low-dimension to the dataset obtained from PubChem database[1], which is a NIH-funded repository for over 60 million chemical molecules and provides their chemical structure fingerprints and biological activities, for the purpose of chemical information mining and exploration. Among 60 Million PubChem dataset, in this paper we have used randomly selected up to 100,000 chemical subsets and all of them have a 166-long binary value as a fingerprint, which corresponds to maximum input of 100,000 data points having 166 dimensions. With those data as inputs, we have performed our experiments on our two decent compute clusters as summarized in Table III.

In the following, we will show the performance results of our parallel SMACOF and GTM implementation with respect to 10,000, 20,000, 50,000 and 100,000 data points having 166 dimensions, represented as 10K, 20K, 50K, and 100K dataset respectively and discuss the correlation measurement between SMACOF and GTM results by using CCA.

A. Performance of Parallel SMACOF

Fig. 5 shows the performance comparisons for 10K and 20K PubChem data with respect to how to decompose the given $N \times N$ matrices with 32, 64, and 128 cores in Cluster-I and Cluster-II. A significant characteristic of those plots in Fig. 5 is that skewed data decompositions, such as $p \times 1$ or $1 \times p$, which decompose by row-base or column-base, are always worse in performance than balanced data decompositions, such as $m \times n$ block decomposition which m and n are similar as much as possible. The reason of the above results is cache line effect that affects cache reusability, and generally balanced block decomposition shows better cache reusability so that it occurs less cache misses than the skewed decompositions [20], [21]. As in Fig. 5, Difference of data decomposition almost doubled the elapsed time of 1×128 decomposition compared to 8×16 decomposition with 10K PubChem data. From the above investigation, the balanced data decomposition is generally good choice. Furthermore, Cluster-II performs better than Cluster-I in Fig. 5, although the clock speed of cores is similar to each other. There are two different factors between Cluster-I and Cluster-II in Table III which we believe that those factors result in Cluster-II outperforms than Cluster-I, i.e. L2 cache size and Networks, and the L2 cache size per core is 4 times bigger in Cluster-II than Cluster-I. Since SMACOF with large data is memory-bound application, it is natural that the bigger cache size results in the faster running time.

In addition to data decomposition experiments, we measured the parallel performance of parallel SMACOF in terms of the number of processes p. The authors investigate the scalability of parallel SMACOF by running with different number of processes, e.g. $p = 64$, 128, 256, and 384. On the basis of the above data decomposition experimental result, the balanced decomposition has been applied to this process scaling experiments. As p increases, the elapsed time should be decreased, but linear performance improvement could not be achieved due to the parallel overhead. In Fig. 6, both 50k and 100k data sets show the performance gain as p increases. However, performance enhancement ratio is reduced, because the ratio of message passing overhead over the assigned computation per each node increases due to more messaging and less computing per node as p increases. Note that we used 16 computing nodes in Cluster-II (total number of cores in 16 computing nodes is 384 cores) to perform the scaling experiment with large data set, i.e. 50k and 100k PubChem data, since SMACOF algorithm requires 480 GB memory for dealing with 100,000 data points, as we disscussed in Section III-A, and Cluster-II is only feasible to perform that with more than 10 nodes.

B. Performance of Parallel GTM

We have measured performance of parallel GTM with respect to each possible m-by-n decomposition of responsibility matrix R to use $p = 32$, 64, and 128 cores in Cluster-I and Cluster-II for 10k and 20k PubChem dataset. In the following experiments, we have fixed other parameters such as $K = 8,000$, $D = 166$, and $M = 9$.

As shown in Fig. 7, the performance of parallel GTM shows the similar pattern of the parallel SMACOF performance in which the balanced decomposition performs better than the skewed decomposition due to the cache line effect.

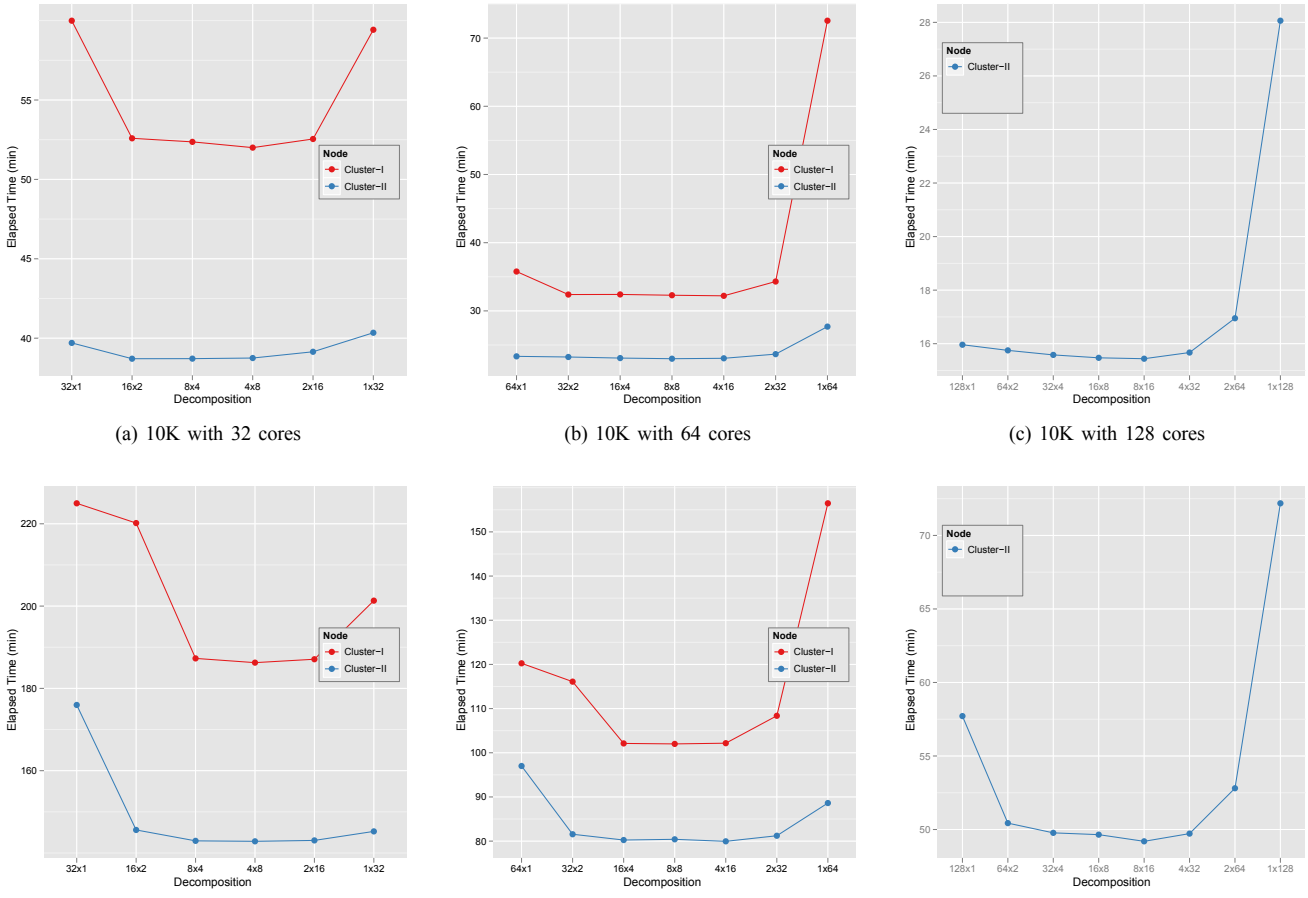
C. Correlation measurement by CCA

By using the CCA algorithm, we have measured similarity between MDS and GTM results for using 100K PubChem dataset . As a result shown in Fig. 8, the maximum correlation

[1] PubChem, http://pubchem.ncbi.nlm.nih.gov/

TABLE III
CLUSTER SYSTEMS USED FOR THE PERFORMANCE ANALYSIS

Features	Cluster-I	Cluster-II
# Nodes	8	32
CPU	AMD Opteron 8356 2.3GHz	Intel Xeon E7450 2.4 GHz
# CPU / # Cores per node	4 / 16	4 / 24
Total Cores	128	768
L2 Cache per core	512 KB	2 MB
Memory per node	16 GB	48 GB
Network	Giga bit Ethernet	20 Gbps Infiniband
Operating System	Windows Server 2008 HPC Edition (Service Pack 2) - 64 bit	Windows Server 2008 HPC Edition (Service Pack 2) - 64 bit



(a) 10K with 32 cores (b) 10K with 64 cores (c) 10K with 128 cores

(d) 20K with 32 cores (e) 20K with 64 cores (f) 20K with 128 cores

Fig. 5. Performance of Parallel SMACOF for 10K and 20K PubChem data with 32,64, and 128 cores in Cluster-I and Cluster-II w.r.t. data decomposition of $N \times N$ matrices.

of both results is about 0.90 which is close to the maximum (1.0) and so we can conclude that both MDS and GTM algorithms produces very similar output for the 100k dataset. Also, we have used the colors in Fig. 8 to identify two clusters as an output of k-mean ($k = 2$) clustering in the original 166-dimensional space. The output shows both MDS and GTM successfully preserved the cluster information in low dimension.

V. CONCLUSIONS AND FUTURE WORKS

In this paper, we have described two different dimension reduction algorithms, called MDS (SMACOF) and GTM, and how to utilize those algorithms for the huge data set. Main issues to deal with a large amount of data points are not only lots of computation but also huge memory requirements. As we described in Section III-A, it takes 480 GB of memory to

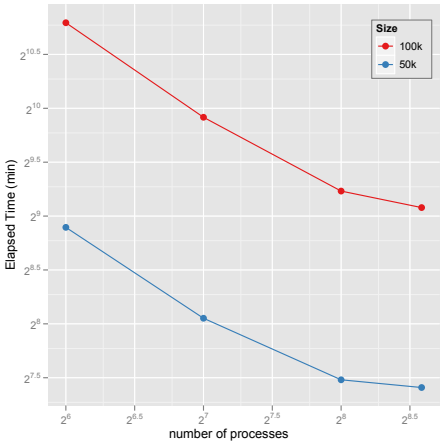


Fig. 6. Performance of parallel SMACOF for 50K and 100K PubChem data in Cluster-II w.r.t. the number of processes. Based on the data decomposition experiment, we choose balanced decomposition as much as possible, i.e. 8×8 for 64 processes. Note that both x and y axes are log-scaled.

run SMACOF algorithm with 100,000 data points. Parallelization via traditional MPI approach in order to utilize distributed memory computing system, which can extend the accessible memory size, is proposed as a solution for the amendment of memory shortage to treat large data with SMACOF and GTM algorithms.

As we discussed in the performance analysis, the data decomposition structure is important to maximize the performance of parallelized algorithm since it highly affects to message passing routines and message passing overhead as well as cache-line effect. Balanced data decomposition ($m \times n$) is generally better than skewed decomposition ($p \times 1$ or $1 \times p$) for both algorithms, specially for the MDS algorithm.

Another interesting aspect we found here is that the MDS and GTM results of the same data are highly correlated with each other as in Fig. 8 (c) even though the detailed mappings to low dimension are visually distinct as shown in Fig. 8(a) and (b).

There are important problems for which the data set size is too large for even our parallel algorithms to be practical. Because of this, we are now developing interpolation approaches for both algorithms. Here we run MDS or GTMs with a (random) subset of the dataset, and the dimension reduction of the remaining points are interpolated. We will report on this extension in a later paper where we will test on the full 60 million PubChem dataset. We will also present results elsewhere on cases such as gene sequences where only dissimilarities and not vectors are involved. We will compare different choices (as suggested by Sammon's algorithm [22]) for the weight function w_{ij} in (1) and (2).

ACKNOWLEDGEMENT

We would like to thank to Professor David Wild and Dr. Qian Zhu in the School of Informatics and Computing, Indiana University, for their valuable advices and feedbacks on PubChem data and analysis. We would also like to thank Microsoft for their collaboration and support.

REFERENCES

[1] G. Fox, S. Bae, J. Ekanayake, X. Qiu, and H. Yuan, "Parallel data mining from multicore to cloudy grids," in *Proceedings of HPC 2008 High Performance Computing and Grids workshop*, Cetraro, Italy, July 2008.

[2] C. Bishop, M. Svensén, and C. Williams, "GTM: A principled alternative to the self-organizing map," *Advances in neural information processing systems*, pp. 354–360, 1997.

[3] ——, "GTM: The generative topographic mapping," *Neural computation*, vol. 10, no. 1, pp. 215–234, 1998.

[4] T. Kohonen, "The self-organizing map," *Neurocomputing*, vol. 21, no. 1-3, pp. 1–6, 1998.

[5] J. B. Kruskal and M. Wish, *Multidimensional Scaling*. Beverly Hills, CA, U.S.A.: Sage Publications Inc., 1978.

[6] I. Borg and P. J. Groenen, *Modern Multidimensional Scaling: Theory and Applications*. New York, NY, U.S.A.: Springer, 2005.

[7] J. B. Kruskal, "Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis," *Psychometrika*, vol. 29, no. 1, pp. 1–27, 1964.

[8] Y. Takane, F. W. Young, and J. de Leeuw, "Nonmetric individual differences multidimensional scaling: an alternating least squares method with optimal scaling features," *Psychometrika*, vol. 42, no. 1, pp. 7–67, 1977.

[9] J. de Leeuw, "Applications of convex analysis to multidimensional scaling," *Recent Developments in Statistics*, pp. 133–145, 1977.

[10] ——, "Convergence of the majorization method for multidimensional scaling," *Journal of Classification*, vol. 5, no. 2, pp. 163–180, 1988.

[11] A. Dempster, N. Laird, and D. Rubin, "Maximum likelihood from incomplete data via the em algorithm," *Journal of the Royal Statistical Society. Series B*, pp. 1–38, 1977.

[12] E. H. Moore, "On the reciprocal of the general algebraic matrix," *Bulletin of American Mathematical Society*, vol. 26, pp. 394–395, 1920.

[13] R. Penrose, "A generalized inverse for matrices," *Proceedings of the Cambridge Philosophical Society*, vol. 51, pp. 406–413, 1955.

[14] H. Hotelling, "Relations between two sets of variates," *Biometrika*, vol. 28, no. 3, pp. 321–377, 1936.

[15] D. Hardoon, S. Szedmak, and J. Shawe-Taylor, "Canonical correlation analysis: an overview with application to learning methods," *Neural Computation*, vol. 16, no. 12, pp. 2639–2664, 2004.

[16] H. Glahn, "Canonical correlation and its relationship to discriminant analysis and multiple regression," *Journal of the Atmospheric Sciences*, vol. 25, no. 1, pp. 23–31, 1968.

[17] O. Friman, J. Cedefamn, P. Lundberg, M. Borga, and H. Knutsson, "Detection of neural activity in functional MRI using canonical correlation analysis," *Magnetic Resonance in Medicine*, vol. 45, no. 2, pp. 323–330, 2001.

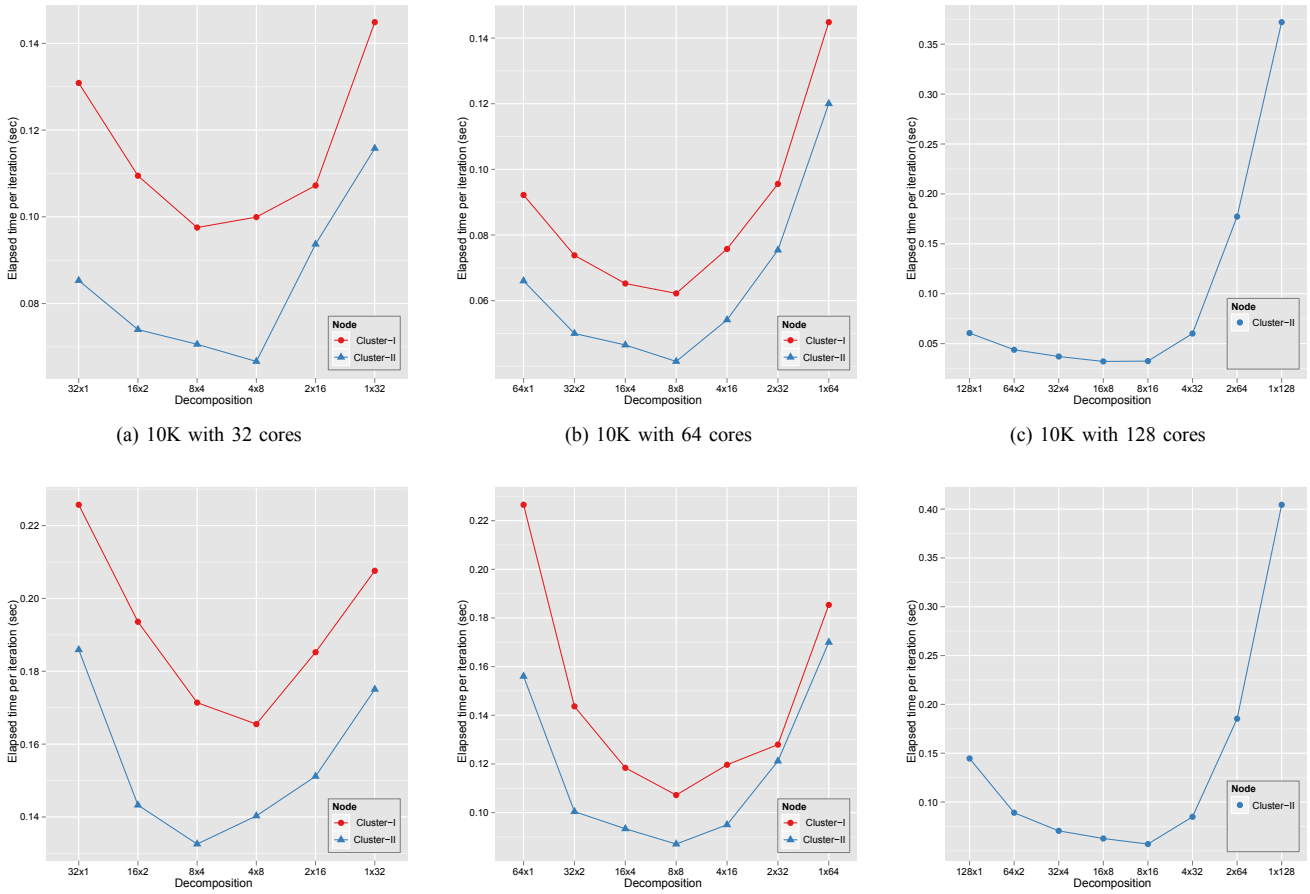
[18] N. Campbell and W. Atchley, "The geometry of canonical variate analysis," *Systematic Zoology*, pp. 268–280, 1981.

[19] B. Thompson, *Canonical correlation analysis uses and interpretation*. Sage, 1984.

[20] X. Qiu, G. C. Fox, H. Yuan, S.-H. Bae, G. Chrysanthakopoulos, and H. F. Nielsen, "Data mining on multicore clusters," in *Proceedings of 7th International Conference on Grid and Cooperative Computing GCC2008*. Shenzhen, China: IEEE Computer Society, Oct. 2008, pp. 41–49.

[21] S.-H. Bae, "Parallel multidimensional scaling performance on multicore systems," in *Proceedings of the Advances in High-Performance E-Science Middleware and Applications workshop (AHEMA) of Fourth IEEE International Conference on eScience*. Indianapolis, Indiana: IEEE Computer Society, Dec. 2008, pp. 695–702.

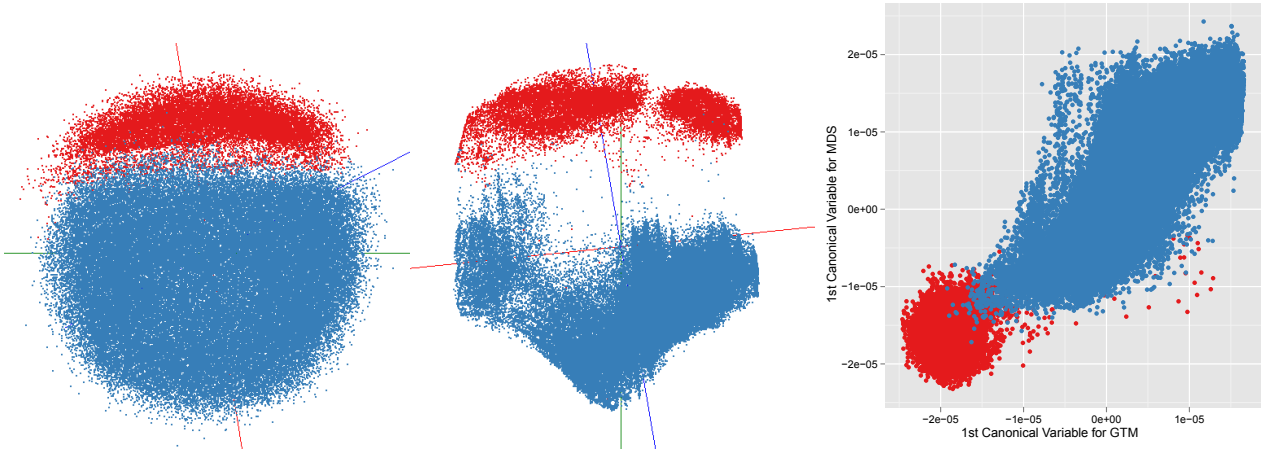
[22] J. Sammon, "A nonlinear mapping for data structure analysis," *IEEE Transactions on computers*, vol. 18, no. 5, pp. 401–409, 1969.



(a) 10K with 32 cores (b) 10K with 64 cores (c) 10K with 128 cores

(d) 20K with 32 cores (e) 20K with 64 cores (f) 20K with 128 cores

Fig. 7. Performance of Parallel GTM for 10K and 20K PubChem data with 32, 64, and 128 cores running on Cluster-I and Cluster-II w.r.t. the m-by-$n$data decomposition running on compute grids. The elapsed time is an average running time per iteration.



(a) MDS for 100K PubChem (b) GTM for 100K PubChem (c) Canonical variables plot for MDS and GTM

Fig. 8. MDS and GTM results for 100K PubChem dataset are shown in (a) and (b). MDS and GTM correlation computed by CCA is shown in (c) as a plot with canonical variables. The optimal correlation, so-called canonical correlation coefficient, is 0.90 (maximum is 1.00) which shows strong correlation between MDS and GTM.