Parallel Clustering for Visualizing Large Scientific Line Data

Jishang Wei* University of California, Davis
Hongfeng Yu† Sandia National Laboratories
Jacqueline H. Chen‡ Sandia National Laboratories
Kwan-Liu Ma§ University of California, Davis

ABSTRACT

Scientists often need to extract, visualize and analyze lines from vast amounts of data to understand dynamic structures and interactions. The effectiveness of such a visual validation and analysis process mainly relies on a good strategy to categorize and visualize the lines. However, the sheer size of line data produced by state-of-the-art scientific simulations poses great challenges to preparing the data for visualization. In this paper, we present a parallelization design of regression model-based clustering to categorize large line data derived from detailed scientific simulations by leveraging the power of heterogeneous computers. This parallel clustering method employs the Expectation Maximization algorithm to iteratively approximate the optimal data partitioning. First, we use a sorted-balance algorithm to partition and distribute the lines with various lengths among multiple compute nodes. During the following iterative clustering process, regression model parameters are recovered based on the local lines on each individual node, with only a few inter-node message exchanges involved. Meanwhile, the workload of regression model computing is well balanced across the nodes. The experimental results demonstrate that our approach can effectively categorize large line data in a scalable manner to concisely convey dynamic structures and interactions, leading to a visualization that captures salient features and suppresses visual clutter to facilitate scientific exploration of large line data.

1 INTRODUCTION

Advanced computing and imaging techniques enables scientists to study problems of unprecedented complexity at high fidelity. Especially, an advanced simulation can generate vast amounts of data from hundreds to thousands of time steps with tens of variables. From such large complex data, scientists often need to extract or derive line data to help understand dynamic structures and interactions hidden in the data. Typical examples of line data include white matter fibers, time series curves, and vector field lines, which are of great interest in many areas of study from diffusion-tensor imaging (DTI), groundwater simulations, and design of particle accelerators, to any studies generating vector fields.

To ensure all essential aspects of features of interest are captured, a large number of lines are often generated. However, it is challenging to cope with large line data. First, very dense lines are typically intertwined, and introduce high visual complexity. Visual clutter can be easily occurred in a visualization, which hinders users from perceiving structural information contained in the line data. Second, the sheer number of lines can easily overwhelm the computing and memory capacity of a single PC, making it difficult to perform analysis tasks efficiently.

To address these issues, our solution is based on parallel cluster analysis. Cluster analysis is an intelligent data analysis method, which categorizes data items with similar properties into clusters. This method has been widely employed to assist users to study dense line data. In our case, by applying clustering, a large line data set is partitioned into many small subsets, each of which represents a characteristic line type. The clustering results allow us to examine selected line clusters independent of others, or to derive a higher-level view of the data by using representative lines from each cluster. The resulting visualization is thus free of visual clutter.

However, cluster analysis of large data is computationally expensive. In order to accelerate the calculations, we turn to heterogeneous computers with multiple CPUs and GPUs for high-performance clustering of large line data. Our design distributes the line data among the cluster nodes based on a sorted-balance algorithm to ensure a well-balanced workload assignment. Then, the lines on each GPU are first smoothed with a B-Spline model and then sampled to obtain their vector descriptors. Next, the lines are partitioned with a parallel regression model-based clustering process into a user-specified number of categories. The resulting clusters of lines can then be visualized individually or together in any combination. In addition, visualizing line data in conjunction with volume or surface rendering of the field data can help scientists better validate their data, understand temporal correlations between different features, and possibly discover previously unknown interactions.

We present our work on extracting and classifying large line data derived from data generated by detailed scientific simulations, such as solar plume and turbulent combustion, for uncovering complex structures and correlations in the data. We note that the requirements and challenges of visualizing and analyzing large line data are representative, and our approach can be extended and benefit other fields that involve large line data analysis.

2 RELATED WORK

Cluster analysis has proved to be a pivotal technique to assist in line data analysis and visualization. Tremendous research has been done for clustering and visualizing lines in the literature [1, 14, 20]. Researchers have presented various approaches for clustering different types of lines. Examples include white matter fibers, time series curves, and vector field lines, to name a few.

To segment and visualize tractography fibers produced from DTI data, most approaches generally share a common procedure of first defining a similarity metric, and then employing clustering algorithms. For instance, Shimony et al. [22] tested several distance metrics that include functions of the distance between tracks and shape information. They used the fuzzy c-means algorithm for clustering. Brun et al. [3] compared pairwise fiber traces in a dimension-reduced Euclidean feature space to create a weighted and undirected graph that is partitioned into the coherent sets using the normalized cut. O'Donnell et al. [17] utilized the symmetrized Hausdorff distance as the similarity measurement among trajectories and achieved spectral clustering using the Nyström method and the k-means algorithm in an embedding space. Tsai et al. [24] constructed tract distances between fiber tracts from dual-rooted graphs where both local and global dissimilarities are taken into account. The considered distance is then incorporated in a locally linear embedding framework and clustering is performed using the k-means algorithm. Curve modeling has also been utilized in clustering white matter fibers. For example, Maddah et al. [13] defined a spatial similarity measure between curves for a supervised cluster-

*e-mail:jswei@ucdavis.edu
†e-mail:hyu@sandia.gov
‡e-mail:jhchen@sandia.gov
§e-mail:ma@cs.ucdavis.edu

ing algorithm, and the Expectation-maximization (EM) algorithm is used to cluster the trajectories in the context of a gamma mixture model.

In time series curves analysis and visualization, Van Wijk and Selow [25] proposed a cluster and calendar based analytical tool to explore and visualize univariate time series data. Schreck et al. [21] introduced a user-supervised self-organizing map (SOM) clustering algorithm that enables users to watch and control the computation process visually. Anderson et al. [2] presented a segmentation framework for analysis and meaningful visualization of function field data.

Regarding vector field visualization, it is critical to extract distinct clusters to deliver important information and avoid clutter. Yu et al. [28] presented hierarchical streamline bundles, a new approach to simplifying and visualizing 2D flow fields. Wei et al. [26] advocated a user-centric approach to cluster and visualize field lines in the vector field. The method allows users to sketch curves for trajectory pattern matching and classification. Rössl et al. [19] developed a method that maps 3D streamlines to points in 3D based on the preservation of the Hausdorff metric in streamline space. Then they applied standard clustering methods to the point sets to construct a segmentation of the original 3D vector field.

The overwhelming data generated in scientific experiments and simulations present a great challenge to data clustering and visualization. A main route to handle the large data issue is to adapt traditional clustering approaches in a parallel and distributed computing environment, such as CLARANS [15], Fractionization [4] and BIRCH [29]. In our work, we extend and parallelize the regression model based clustering to categorize and visualize large lines data.

3 Cluster Analysis of Lines

Conventional clustering techniques roughly fall into five families: partitioning methods, hierarchical methods, model-based methods, density-based methods, and grid-based methods [9]. Note that these algorithms are based on either pairwise similarities or vector descriptors of data objects. The algorithms based on pairwise similarities, such as the hierarchical methods, have computational complexities that usually are quadratic in the number of objects n, or worse. Such high orders can incur high run-time memory requirements for applications with large data. Moreover, regarding line data, specially when the lengths of which vary, it is nontrivial to design an appropriate pairwise similarity metric. On the other hand, the algorithms based on vector descriptors, such as the model-based methods, have the complexity of order n and are comparatively efficient. Furthermore, the model-based methods can incorporate prior knowledge naturally. They provide a principled approach for clustering lines with different lengths if a proper mixture model is chosen. This is a favorable characteristic in line data clustering. Therefore, we utilize and improve a polynomial regression model-based clustering method [6, 7, 8] to categorize large line data.

Before introducing our parallel implementation for large line data in Section 4, we first give an overview of our polynomial regression model-based clustering method on B-Spline modeled lines.

3.1 Line Representation

In our work, we use vectors to describe lines. Without loss of generality, a line l in D dimensional space is represented as:

$$l = (\mathbf{p}_1, \mathbf{p}_2, ...\mathbf{p}_C) \tag{1}$$

where C, the number of points along l, is defined as the line length, and $\mathbf{p}_i$ is a point of dimension D.

3.2 Line Preprocessing

As with most problems in line data analysis, a suitable choice of line description and representation would lead to the ease and efficiency of clustering. In our application, we first fit lines with a uniform B-spline model to smooth data while preserving their shape and location information. We then sample a sequence of points at an equal arc length along a modeled line, and use this set of sample points to represent the line for the successive clustering process.

3.2.1 Smoothing Lines with Uniform B-Spline Model

Uniform B-Spline is a convenient form to represent complex, smooth curves. It is in general chosen because of the ease of manipulation. By being fitted to the uniform B-Spline model, a line l can be defined as a function of parameter t in the range of $[t_{\min}, t_{\max}]$, where $t_{\min}$ and $t_{\max}$ correspond to the beginning and end of the line:

$$l = \mathbf{p}(t) \tag{2}$$

3.2.2 Sampling on B-Spline Modeled Lines

In our application, we represent a line by a set of sample points at an equal arc length. It is desirable to evaluate a parametric B-Spline line at points based on its arc length instead of the line's original parameter t. Thus, the line l needs to be represented as a function of parameter s in the range of $[0,\ L]$, where L is the total length of the line, i.e.,

$$l = \mathbf{p}(s) \tag{3}$$

Let $\mathbf{p}(s)$ and $\mathbf{p}(t)$ denote the same line, in which $\mathbf{p}(s) = \mathbf{p}(t)$ implies a relationship between t and s. We apply the chain rule to obtain:

$$\frac{d\mathbf{p}(t)}{dt} = \frac{d\mathbf{p}(s)}{ds}\frac{ds}{dt} \tag{4}$$

Then we have:

$$\left|\frac{d\mathbf{p}(t)}{dt}\right| = \left|\frac{d\mathbf{p}(s)}{ds}\right|\left|\frac{ds}{dt}\right| = \frac{ds}{dt} \tag{5}$$

where $\left|\frac{d\mathbf{p}(s)}{ds}\right| = \frac{\left|d\mathbf{p}(s)\right|}{|ds|} = 1$, and $\frac{ds}{dt}$ is nonnegative as the arc length s increases with the increase of t. From Equation 5, we can get the relationship between s and t by integration:

$$s = \int_{t_{min}}^{t} \left|\frac{d\mathbf{p}(\tau)}{dt}\right| d\tau \tag{6}$$

where $s = 0$ for $t = t_{min}$, and $s = L$ for $t = t_{max}$. It means that given a parameter t, we can determine the corresponding arc length s from the integration. However, what is needed is to solve an inverse problem: Given an arc length s, we want to know the corresponding t. To solve this, we employ numerical methods to compute $t \in [t_{min},\ t_{max}]$, given the value of $s \in [0,\ L]$:

$$\frac{dt}{ds} = \frac{1}{\left|\frac{d\mathbf{p}(t)}{dt}\right|} \tag{7}$$

Equation 7 can be solved numerically with any standard differential equation solver such as the Runge-Kutta method.

With $l = \mathbf{p}(s)$, we then can obtain R sample points at a specified equal arc length:

$$l = (\mathbf{p}(s_1), \mathbf{p}(s_2), ...\mathbf{p}(s_R)) \tag{8}$$

3.3 Regression Model-based Clustering

3.3.1 Linear Regression Models

Regression is a method for fitting a line through a set of points using some goodness-of-fit criterion. One of the most common types of regression is linear regression. Let x be an independent variable, and let $\mathbf{p}(x)$ denote an unknown function of x that we want to approximate. Assume there are R observations, i.e., the values of $\mathbf{p}(x)$ measured at the specified values of x_r are given as:

$$\mathbf{p}(x_r) = \mathbf{p}_r, \;\; r = 1, \cdots, R. \tag{9}$$

Regarding each dimension of one line $p(x)$, the idea behind linear regression is to model $p(x)$ by a linear combination of Q basis functions:

$$p(x) \approx \beta_1 \psi_1(x) + \cdots + \beta_Q \psi_Q(x) \tag{10}$$

In our case, we use polynomial basis functions, namely,

$$p(x) \approx \beta_0 + \beta_1 x + \cdots + \beta_Q x^Q \tag{11}$$

If we consider a line $l = \big(\mathbf{p}(x_1),\ \mathbf{p}(x_2),\ \ldots,\ \mathbf{p}(x_R)\big)$, Equation 11 can be written as,

$$l = \mathbf{X}\beta + \varepsilon \tag{12}$$

where l is a matrix of $R \times D$, representing a line of length R in D dimensional space. β is a $Q \times D$ dimensional matrix of regression coefficients, and ε is an $R \times D$ noise matrix. $\mathbf{X}$ is the usual $R \times Q$ Vandermonde regression matrix:

$$\mathbf{X} = \begin{bmatrix} {x_1}^0 & {x_1}^1 & {x_1}^2 & \ldots & {x_1}^Q \\ {x_2}^0 & {x_2}^1 & {x_2}^2 & \ldots & {x_2}^Q \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ {x_R}^0 & {x_R}^1 & {x_R}^2 & \ldots & {x_R}^Q \end{bmatrix} \tag{13}$$

3.3.2 Model-based Clustering

Model-based clustering can be regarded as the generalization of the K-means algorithm [10, 12]. In the context of model-based clustering, the whole set of lines is assumed to be derived from a mixture model of K components that correspond to K clusters. Each component model is associated with a probabilistic density function. With regard to our case, each line is represented by a mixture model of K component polynomial regression models with Gaussian error terms, as shown in Equation 12. Let p_k denote the probability at which a line is generated by the cluster k, and then the mixture density for generating one line l is:

$$p(l|\Theta) = \sum_k \alpha_k p_k(l|\theta_k) \tag{14}$$

where α_k denotes the probability of cluster k, which is nonnegative and all component probabilities (for $k = 1 \ldots K$) sum to one. θ_k indicates the parameters of component model k. Each component θ_k contains regression coefficients β_k and Gaussian covariance parameter δ_k. Thus, the mixture model is represented as $\Theta = \{\theta_1,\ \cdots,\ \theta_K\}$.

$p_k(l|\theta_k)$ means the probability of component model k generating line l. In our work, the model component takes a form as Equation 12. As a result, the regression model leads to a cluster-specific probabilistic density function for l:

$$\begin{aligned} p_k(l|\theta_k) &= \mathcal{N}(l|\mathbf{X}\beta_k,\ {\sigma_k}^2\mathbf{I}) \\ &= \prod_{r=1}^{R} \mathcal{N}(\mathbf{p}_r|\mathbf{X}\beta_k,\ {\sigma_k}^2\mathbf{I}) \end{aligned} \tag{15}$$

where $\mathcal{N}(.)$ is a D dimensional Gaussian probability density function; $\mathbf{X}\beta_k$ and ${\sigma_k}^2\mathbf{I}$ are the mean vector and covariance matrix of the kth Gaussian density function, respectively.

As a flavor of K-means method, the objective function of model-based clustering maximizes the likelihood, $\mathscr{L}(\Theta|\mathbf{L})$, of generating the line data set $\mathbf{L}(\mathbf{L} = \{l_1, l_2, \ldots, l_N\})$, given the mixture model Θ. In practice, the likelihood can be represented by any function of Θ that is proportional to the probability $p(\mathbf{L}|\Theta)$. In our application, the log of the likelihood of $\mathbf{L}$ is applied:

$$\mathscr{L}(\Theta|\mathbf{L}) = log\, p(\mathbf{L}|\Theta) = \sum_n log \sum_k^K \alpha_k p_k(l_n|\theta_k) \tag{16}$$

To this end, conducting model-based clustering is to estimate the parameters of the K component models given a set of lines, and then to assign each line to a cluster with the highest probability among all K clusters. The Expectation-Maximization (EM) algorithm provides an efficient framework for parameter estimation in the mixture model context [5], and we choose it for our model-based clustering. In the polynomial regression model-based clustering, the EM algorithm is executed as follows.

- E-Step: We assume that y_n, associated with each l_n, indicates the line membership in one of the K clusters. The posterior $p(y_n|l_n)$ is calculated to give the probability that the nth line is generated by cluster y_n. The probability of l_n generated by cluster k takes the form [7]:

$$w_{nk} = p(y_n = k|l_n) \propto \alpha_k p_k(l_n) = \alpha_k \mathcal{N}(l_n|\mathbf{X}\beta_k,\ {\sigma_k}^2\mathbf{I}) \tag{17}$$

- M-Step: The likelihood in Equation 16 is maximized with respect to the parameters $\{\beta_k,\ {\sigma_k}^2,\ \alpha_k\}$. The solutions [7] are given as:

$$\beta_k = \big[\sum_n w_{nk}\mathbf{X}_n'\mathbf{X}_n\big]^{-1} \sum_n w_{nk}\mathbf{X}_n' l_n \tag{18}$$

$$\sigma_k^2 = \frac{1}{\sum_n w_{nk}} \sum_n w_{nk} \|l_n - \mathbf{X}_n\beta_k\|^2 \tag{19}$$

$$\alpha_k = \frac{1}{N} \sum_n w_{nk} \tag{20}$$

After obtaining these mixture model parameters, we can use them to compute the probability value by Equation 15, and then refer each line to a cluster with the highest probability value.

4 Parallel Implementation

To process large line data, we implement our regression model-based clustering using a hybrid approach with MPI and CUDA by leveraging the power of CPUs and GPUs on multiple nodes in a heterogeneous environment. To ensure the feasibility and scalability of our approach, we consider the complete course of our clustering approach, design the parallel implementation of each step and assign them to the CPUs and/or GPUs based on their characteristics and associated constraints.

First, for the line preprocessing step (Section 3.2), we could treat it as a one-time preprocessing step, save the resulted B-Spline modeled lines, and repeatedly load them for later different clustering runs to generate clusters with different inputs. However, given the sheer size of line data, it is generally desired to minimize storage overhead for particular analysis tasks, thus requiring us to perform line preprocessing on-the-fly. We note that the operations of smoothing and sampling on different lines are independent of each

Algorithm 1 Parallel Regression Model-based Clustering

Input: N lines; M compute nodes.
Output: K clusters.

// Partitioning and distributing N lines to M nodes
for each compute node **in parallel do**
    Performs the sorted balancing algorithm independently to decide its own line assignment
    Performs MPI collective I/O to fetch its own line data
end for
// Preprocessing
for each compute node **in parallel do**
    Transfers its lines from CPU to GPU
    Smooths its lines with uniform B-Spline model on GPU
    Samples its lines to obtain their vector descriptors on GPU
    Copies the line vector descriptors from GPU to CPU
end for
// Initialization
for each compute node **in parallel do**
    Randomly initializes the probability p_{ik} of each local line l_i belonging to a cluster k, constrained by $\sum_{k=1}^{K} p_{ik} = 1$
end for
All nodes calculate $\beta_k, \sigma_k^2, \alpha_k$ according to Equations 18, 19 and 20 using the **psgels** routine of SCALAPACK
// Model-based clustering
while true **do**
    // E-Step
    for each compute node **in parallel do**
        Calculates the probability p_{ik} of each local line l_i belonging to a cluster k by Equation 17, constrained by $\sum_{k=1}^{K} p_{ik} = 1$
    end for
    // Likelihood calculation
    All nodes **all gather** the total likelihood $\mathscr{L}$ in Equation 16
    if the increment of $\mathscr{L}$ is less than a specified threshold **or** the iteration number is greater than a specified limit **then**
        break
    end if
    // M-Step
    All nodes calculate $\beta_k, \sigma_k^2, \alpha_k$ according to Equations 18, 19 and 20 using the **psgels** routine of SCALAPACK
end while
// Generating final membership
for each compute node **in parallel do**
    Puts each local line l_i to a cluster k where the line has the highest probability p_{ik}
end for

other, which allows us to carry out the preprocessing step efficiently on the GPUs.

Second, for the clustering step (Section 3.3), researchers [11, 18] have been developing different implementations of the EM algorithm to estimate mixture Gaussian models in parallel. We extend the parallel EM algorithm with respect to regression mixture models. Different from Gaussian mixture models, regression mixture models describe a group of lines with a regression line and the associated deviation. One key step is to obtain $\{\beta_k,\ {\sigma_k}^2,\ \alpha_k\}$ by solving Equation 12 as a linear least square problem. Such a solving step can be performed more efficiently on the CPU than on the GPU, because the solving step is bandwidth limited and the current bandwidth between the CPU and the GPU is lower than that of the CPU's bus [23]. Therefore, in our design, we choose the SCALAPACK package with the optimized CPU BLAS to solve the linear least square problem on large data using multiple CPUs.

Third, we need to design a data partitioning and distribution scheme to favor both the preprocessing step and the clustering step,

Algorithm 2 Sorted Balancing

Input: list of lengths and indexes of N lines; number of compute nodes M.
Output: assigned line list A_m, where $m \in [1, M]$.

For each node m, set its assigned line list $A_m \leftarrow \varnothing$, and set its assigned accumulated line length $L_m \leftarrow 0$
Sort the input line list in a decreasing order of their lengths
for each line l in the sorted line list **do**
    Let d be the node whose L_d is minimal
    // Assign l to the node d
    $A_d \leftarrow l$'s index
    $L_d \leftarrow L_d + l$'s length
end for

and minimize the overhead of data transformation and transfer between different steps. The challenge comes from the disparity of data partition requirements between different operations to achieve a balanced workload. For the operations of smoothing and sampling in the preprocessing step, and for the operation of E-Step, each atomic operand is an individual line, implying a partition of the lines with respect to their associated operation costs. For the operation of M-Step, the operands are the matrix presentations of all lines, and the corresponding solver provided by SCALAPACK requires an even partition of the matrices. In our design, we use a sorted balancing algorithm to partition and distribute the lines to achieve a well-balanced workload for each step and minimize the inter-node communication cost for data exchanges.

Algorithm 1 lists the detailed procedure of our parallel regression model-based clustering. We first partition and distribute the lines to multiple compute nodes based a sorted balancing algorithm. After obtaining its line assignment, each node first smooths lines with the B-Spline model and then samples the lines to obtain their vector descriptors. This step is performed on the GPUs by exploiting the high levels of concurrency enabled by CUDA. Next, the parallel model-based clustering algorithm is launched to collectively find a number of clusters based on the line vector descriptors. Our line partition and distribution scheme makes it possible to leverage the scalability of SCALAPACK and perform this step efficiently across the CPUs. Thus, by considering the characteristics of different steps and the interplay between them, we carefully distribute the workloads of different steps among the different processing units and can maximize the utilization of the heterogeneous computers.

4.1 Lines Partitioning and Distribution

We need partition and distribute the lines with different lengths to multiple compute nodes and ensure the balanced workload of each step assigned to the nodes. First, in the preprocessing step and the E-Step, we note that each line is processed independently of each other. Moreover, as we sample a line at an equal arc length, the workload associated with a line is proportional to its length. Thus, the total length of lines assigned to each node should be approximately equal.

Second, given the vector descriptors of lines, we construct the matrices l, $\mathbf{X}$, β, and ε in Equation 12, where the row number of l, $\mathbf{X}$ and ε is equal to the total sampling point number of all lines. To solve such a large system in the M-Step, we use the **psgels** routine of SCALAPACK, which requires to evenly divide the matrices along rows to produce a Cartesian distribution of each matrix [1]. We can implicitly divide these large matrices by constructing the local matrices from the local vector descriptors at each node. However, as the local total sampling point number at each node is not neces-

[1] Compared with the total number of sampling points, the dimension D is relatively small (such as two or three) in our current study. Thus we did not divide the matrices along columns.

case	data set	# lines	# avg points per line	# avg samples per line	# compute nodes 1	2	3	4	5	6	7	8
1 (small case)	solar plume	10,000	501	71	x	x	x	x	x	x	x	x
2 (small case)	combustion	10,000	101	35	x	x	x	x	x	x	x	x
3 (medium case)	combustion	100,000	101	35	x	x	x	x	x	x	x	x
4 (large case)	combustion	1,000,000	101	35				x	x	x	x	x

Table 1: Setup of scalability study. Entries marked with "x" represent experiment runs.

sarily the same, the constructed local matrices can have the different numbers of rows. One possible solution is to exchange partial vector descriptors among the nodes to make all the local matrices have a same or nearly same number of rows. But this method requires the message exchanges of partial vector descriptors possibly among all nodes at each iteration of the Model-based clustering. On the other hand, we note that if the total length of lines assigned to each node is nearly equal, the differences of total length of the local vector descriptors among the nodes are marginal. It allows us to append a minimal number of rows of zeros to the local matrices to make them have the same number of rows. This method achieves the same solving results without involving communication cost, and the computational overhead due to the extra zeros is negligible in practice.

To this end, we use a sorted balancing algorithm in our work to divide and assign the lines among the nodes. Algorithm 2 sketches our sorted balancing algorithm. The only inputs to the algorithm are the list of lengths and indexes of all lines and the number of compute nodes. Each node executes the sorted balancing algorithm independently to calculate the line assignment that makes the total line length assigned to each node is nearly equal.

By the sorted balancing algorithm, each node can obtain the index list of its assigned lines. For each node, the assigned line data may not be contiguously stored in the original line data file. To minimal the I/O overhead for each node fetching its own line data from the storage, we first use MPI_Type_create_struct and MPI_File_set_view to set a file view for each node with respect to its own list of assigned line indexes and the list of line offsets. And then all nodes use the MPI collective I/O routine MPI_File_read_all to collaboratively fetch its own line data in parallel.

4.2 Line Preprocessing

On each compute node, the preprocessing stage takes advantage of the parallel capability of GPUs to smooth and sample the bunch of lines. The operations of smoothing and sampling performed on one line is independent of other lines. This allows us to use the typical CUDA program model to process each line using one CUDA thread. After fetching its own line data from the storage, each node transfers the line data from the main memory to the GPU memory given the line representation as described in Section 3.1. Then, a kernel SMOOTH-SAMPLE is called on a grid of $\frac{N}{B}$ thread blocks, where N is the number of lines assigned to one processor and B is the thread number of each block. The appropriate value of B depends on the available GPU resources, such as on-chip shared memory and registers. Each thread in the SMOOTH-SAMPLE kernel first smooths a line using the uniform B-Spline Model, and then generates a set of sample points along the line at a specified equal arc length. Then, the set of sample points of each line are transferred back from the GPU memory to the main memory, which are used as the vector descriptors for the successive clustering step.

4.3 Model-based Clustering

After preprocessing, each node obtains a vector descriptor for each local line. During the initialization stage of the clustering, given a specific target cluster number K, each node first randomly assigns a probability value p_{ik} of each local line l_i belonging to a cluster k. Each node constructs its local matrices l, $\mathbf{X}$, β, and ε in Equation 12. A minimal number of rows of zeros are appended to the end of the matrices to make them have the same number of rows among all nodes. Then the linear least square solver **psgels** of SCALAPACK is called to solve β_k, σ_k^2, and α_k according to Equations 18, 19 and 20 with the initial guess of probability values. After the initialization stage, the clustering then carries out the iterative EM algorithm, until the overall likelihood has converged at an optimal value or the iteration number is greater than a specified limit. Finally, each node puts each local line l_i to a cluster k where the line has the highest probability value p_{ik} to generate the final membership.

4.3.1 E-Step

The E-Step is responsible for calculating the probability of assigning lines into each cluster, which can be executed independently for each line. Although we could use CUDA to implement a kernel for this step, such computing is bandwidth limited, and the data transferring cost between the CPU and the GPU can exceed the computational saving obtained on the GPU. Therefore, we advocate a simple CPU implementation that lets each node iterate through all local lines and compute the probability densities of each line belonging to each of the mixture components according to Equation 17.

4.3.2 M-Step

The M-Step estimates the parameters of each component regression model, β_k, σ_k^2 and α_k. We use a linear least square solver to estimate β_k according to Equation 18. Since the whole line data set is distributed across different nodes, we choose the linear least square solver **psgels** of SCALAPACK using multiple CPUs. Then we employ the matrix multiplication routine **psgemm** of SCALAPACK to calculate σ_k^2 according to Equation 19. The size of α_k is $1 \times K$, where K is the number of clusters. α_k is obtained to all processors using the MPI_Allreduce routine. Note that the local matrices of all nodes have been made to have the same number of rows by appending a minimal number of rows of zeros, and such appending only needs to be done once during the initialization stage. In this way, we can satisfy the data partition specifications posed by SCALAPACK and ensure the balanced workloads of the solvers on multiple CPUs.

5 Results and Discussion

We have experimented our parallel polynomial regression model-based clustering on a heterogeneous system containing 8 nodes connected by the Gigabit Ethernet. Each node contains one Intel quad-core 3.00GHz CPU with 4GB of memory, and one NVIDIA GeForce GTX 285 GPU.

The line data sets from two large scientific simulations have been used in our experimental study. The first data set contains 10,000 streamlines extracted from the vector field of a solar plume simulation provided by the scientists at the National Center for Atmospheric Research. The domain grid size of the vector field is $504 \times 504 \times 2048$ and the average streamline length is proportional to the domain diagonal. The second data set contains the time series curves correlating multiple variables, which are generated from

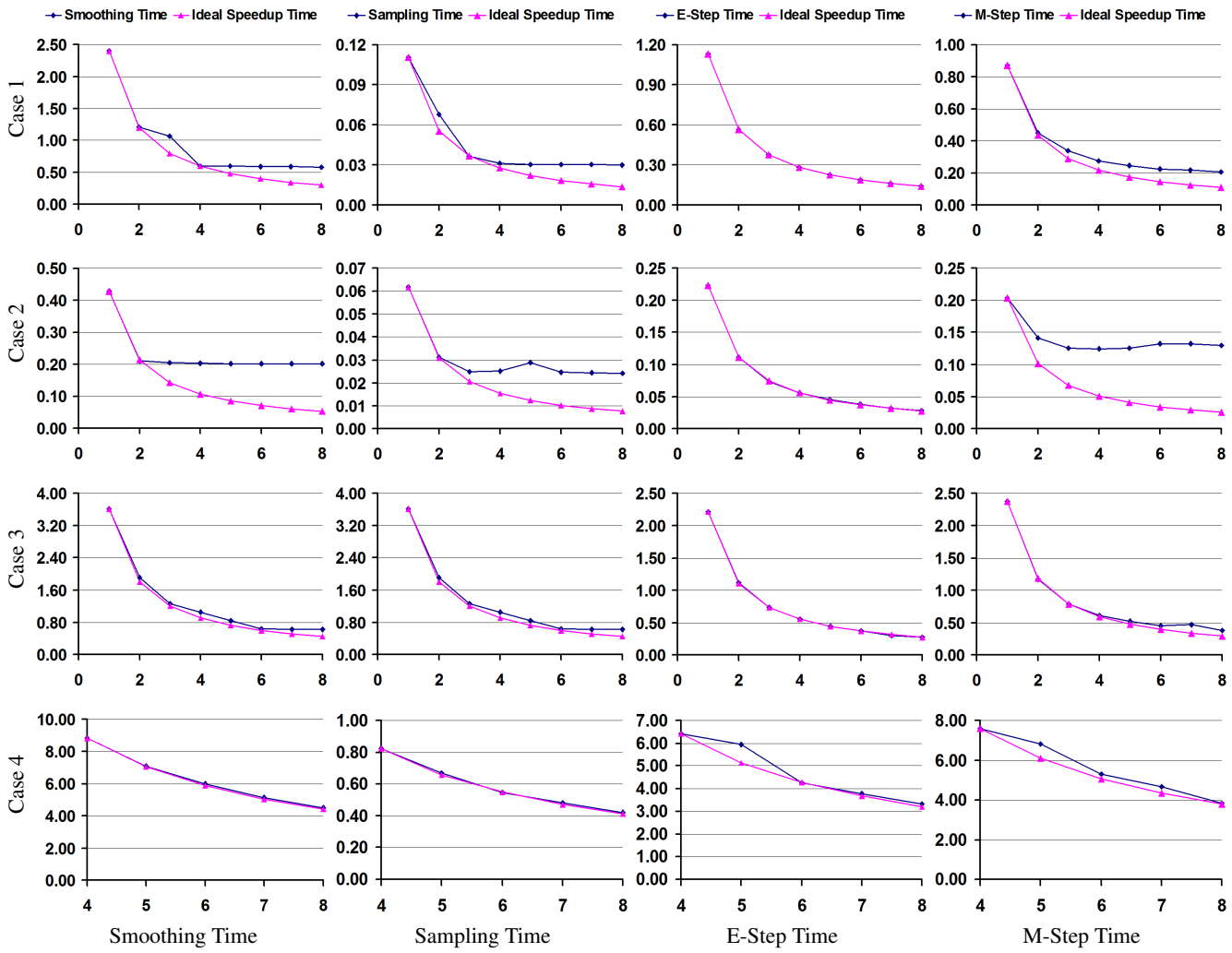

Figure 1: Speedups of scalability study. In each plot, the horizontal axis represents the number of nodes, and the vertical axis represents the running time in second. The left to the right columns show the timing results of smoothing, sampling, E-Step, and M-Step, respectively, for Cases 1 to 4. The timing results corresponding to each case are plotted on the rows.

large-scale combustion simulations conducted by Sandia National Laboratories. This data set contains 1,000,000 time series curves and each curve correlates, over 808 time steps, two key parameters in the phase space: mixture fraction and temperature.

We used these two data sets to design and conduct four sets of scalability experiments with respect to the different problem sizes. Table 1 shows the experimental setup. We used the 10,000 streamlines of the solar plume data set in Case 1, and the 1,000,000 time series curves of the combustion data set in Case 4. The curves of Cases 2 and 3 were obtained by randomly sampling the curves of Case 4. We tested any number of compute nodes from 1 to 8 in Cases 1 to 3, and any number of compute nodes from 4 to 8 in Case 4. Figure 1 shows the performance results. Each row represents the results for a set of scalability experiments. Each column represents the results of the same operation for the different problem sizes.

The first column of Figure 1 shows the performance of the smoothing operation performed on the GPUs. In general, the smoothing time decreases accordingly as we use more compute nodes. However, we notice that the timing roughly remains about the same beyond 4 nodes for Case 1, and beyond 2 nodes for Case 2. The main possible reason is that when the node count increases to a certain number, the workload associated with the lines assigned to each GPU may not invoke sufficient number of threads to fully utilize the parallelism provided by CUDA, and the performance gain obtained by the GPU can be suppressed by the data transfer overhead between the CPU and the GPU [16]. Consequently, the parallel efficiency is only 51.3% (8 nodes vs. 1 node) for Case 1. The line number used in Case 2 is same as Case 1, but the average point number per line in Case 2 is much smaller so that the parallel efficiency is even worse for Case 2, which is only 26.7% (8 nodes vs. 1 node). In Cases 3 and 4, when much large line sets are considered, we can better exploit the GPU parallelism and improve the scalability performance. The parallel efficiencies are 72.5% and 98.4% for Case 3 (8 nodes vs. 1 node) and Case 4 (8 nodes vs. 4 nodes), respectively.

The second column of Figure 1 shows the performance of the sampling operation performed on the GPUs. Similar to the smoothing operation, the speedup of the sampling operation improves accordingly when we increase the number of lines to saturate the GPU. The parallel efficiencies of sampling are 45.9%, 32.0%, 75.3%, and 98.4% for Cases 1 to 4, respectively. Moreover, the high parallelism provided by the GPU allows us to perform the computing intensive processing task efficiently. For instance, in our experimental study, the time to smooth and sample 100,000 lines using

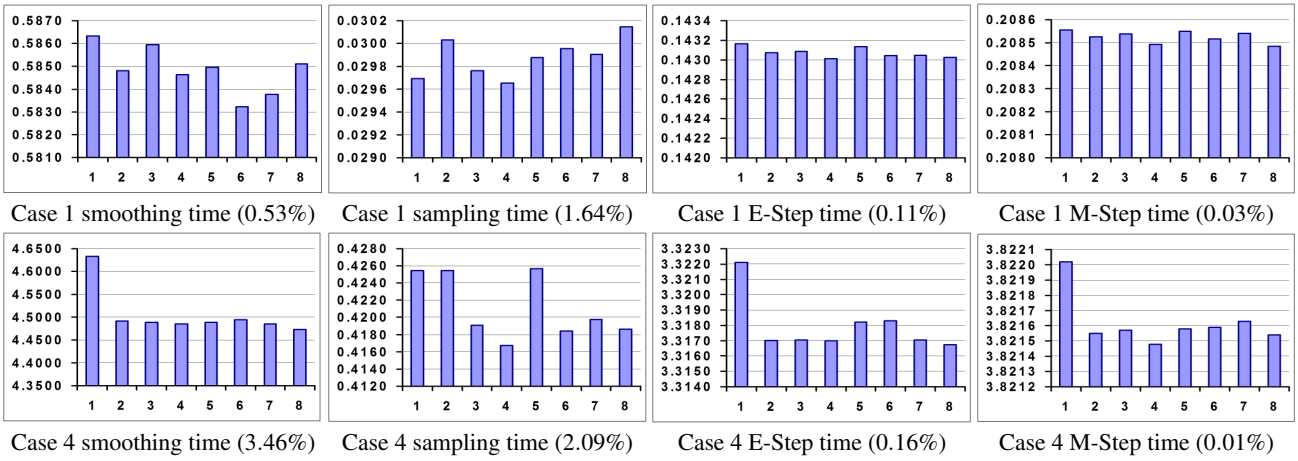

Figure 2: Workloads among 8 nodes for Cases 1 and 4. In each plot, the horizontal axis represents the node ID, and the vertical axis represents the running time in second. The left to the right columns show the timing results of smoothing, sampling, E-Step, and M-Step, respectively. The percentage number associated with each plot is the difference ratio between the maximum and minimum times among the nodes.

one GPU is about two orders of magnitude less than the one using one CPU core.

The third column of Figure 1 shows the performance of the E-Step operation performed on the CPUs. For this operation, each node independently calculates the probability value of each local line belonging to a cluster, and no communications are required. Thanks to our sorted balancing algorithm, the E-Step achieves almost ideal speedup, and the parallel efficiencies of E-Step are 99%, 98%, 99.5%, and 96.7% for Cases 1 to 4, respectively.

The fourth column of Figure 1 shows the performance of the M-Step operation performed on the CPUs. The parallel efficiencies of M-Step are only 52.3% and 19.5% for Cases 1 and 2, respectively, which are less satisfactory. The main possible reason is that the **psgels** and **psgemm** routines of SCALAPACK are used to estimate the parameters of each regression model in parallel, and inter-node communications are required in these routines. When the number of lines is relatively small, such as in Case 1 or 2, the communication overhead can dominate the overall time of operation and incur performance degradation with more nodes. When we increase the number of lines, the computing time of solving local linear systems tends to dominate the overall time. With our sorted balancing algorithm, the solving workload can be well-balanced among the nodes, and the parallel efficiencies of M-Step are 77.7% and 99.5% for Cases 3 and 4, respectively.

Figure 2 shows the workload of each operation among 8 nodes for Cases 1 and 4 with respect to the small and large problem sizes. We define the difference ratio, dr, of the workloads as:

$$dr = (max\_time - min\_time)/max\_time,$$

where max_time (min_time) is the maximum (minimum) time among all nodes for the same operation. The dr values for the operations of Cases 1 and 4 are shown in Figure 2. For instance, in terms of the smoothing time, the dr values are 0.53% and 3.46% for Cases 1 and 4, respectively. In terms of the M-Step time, the dr values are 0.03% and 0.01% for Cases 1 and 4, respectively. We can observe that based on our sorted balancing algorithm, we can achieve well balanced workloads for all operations in both the preprocessing and clustering stages. We note that each stage is performed on a different type of processor, and has a different data representation and a different data access pattern.

Figure 3 shows the clustering result of the 10,000 streamlines of the solar plume data set in Case 1, where eight clusters are generated. Figure 3 (a) shows the overview of all streamlines, and Figure 3 (b)-(i) show the individual clusters of the streamlines. By using our clustering method, the bundle of lines are partitioned to a number of clusters which facilitate recognition and understanding. Effective visualization results are generated to depict the different streamline features, by observing which people can perceive the complex structures and correlations inherent in the turbulent vector field more clearly.

Figure 4 shows the clustering result of the 100,000 time series curves of the combustion data set in Case 3, where fourteen clusters are generated. Figure 4 (a) shows the overview of all curves where the visual clutter is occurred, making it difficult for a user to perceive structure information. Figure 4 (b)-(j) show the individual cluster of the curves after applying our clustering method. We can see that after clustering the data set is well partitioned, and each cluster consists of similar curves, which allows scientists to observe the correlations between two variables more easily. Especially, the patterns shown in Figure 4 (j) are the outliers that do not match the profile curves from the hypothesis, and the scientists want to capture and isolate them for further study. For a detailed application based on the model-based clustering method, please refer to our recent work on visual analysis of turbulent combustion particle data [27].

6 Conclusions and Future Work

We have demonstrated how clustering for visualization of large line data can be done efficiently with a combination of multiple GPUs and CPUs. The key aspects of our work include how we prepare and distribute the line data to facilitate the clustering calculations and how we devise and implement the model-based clustering algorithm in CUDA and MPI. The scientists we have worked with are eager to integrate this new visualization capability into their routine scientific data analysis and discovery process. Presently, the best scenario for using our design is with in-situ construction of lines followed by interactive visualization of the lines using a GPU cluster. Later on, as we move to the petascale and exascale computing, we will likely have to also conduct clustering in situ and compress the line data as much as possible to reduce storage and transfer costs. Other future research opportunities include metric design for categorizing other types of line data, line data packing and streaming, and visualization of higher dimensional line data.

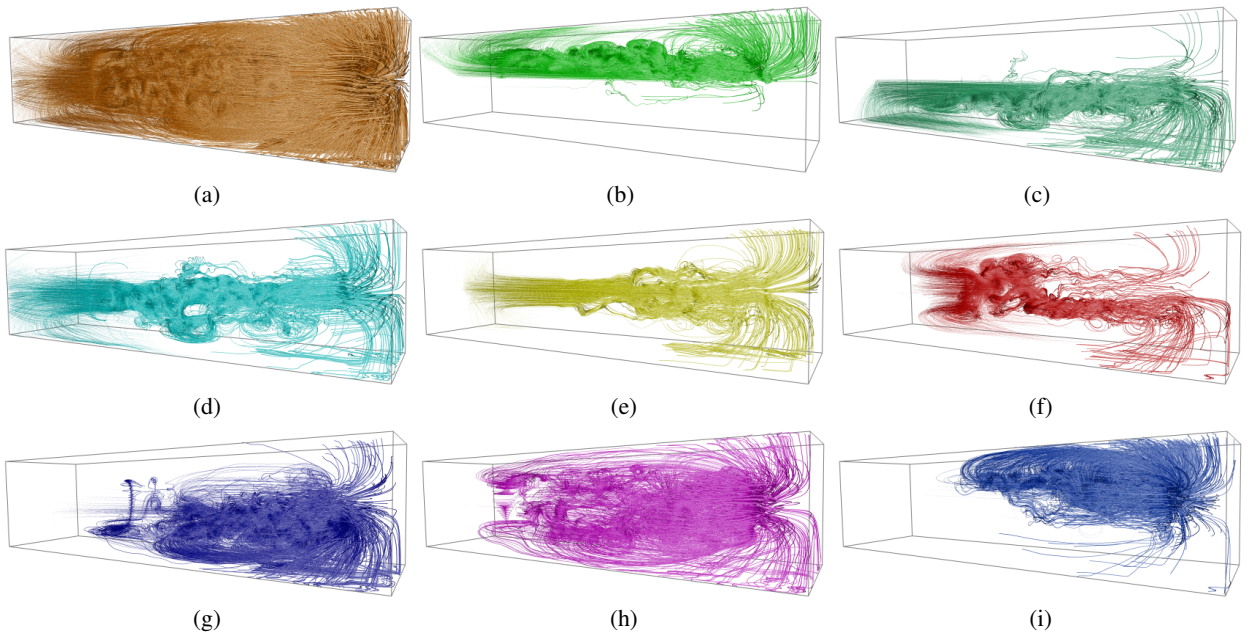

Figure 3: This figure shows the clustering result of the streamlines generated from the solar plume velocity vector field. (a) shows the overview of all 10,000 streamlines. (b)-(i) show the eight different groups of streamlines.

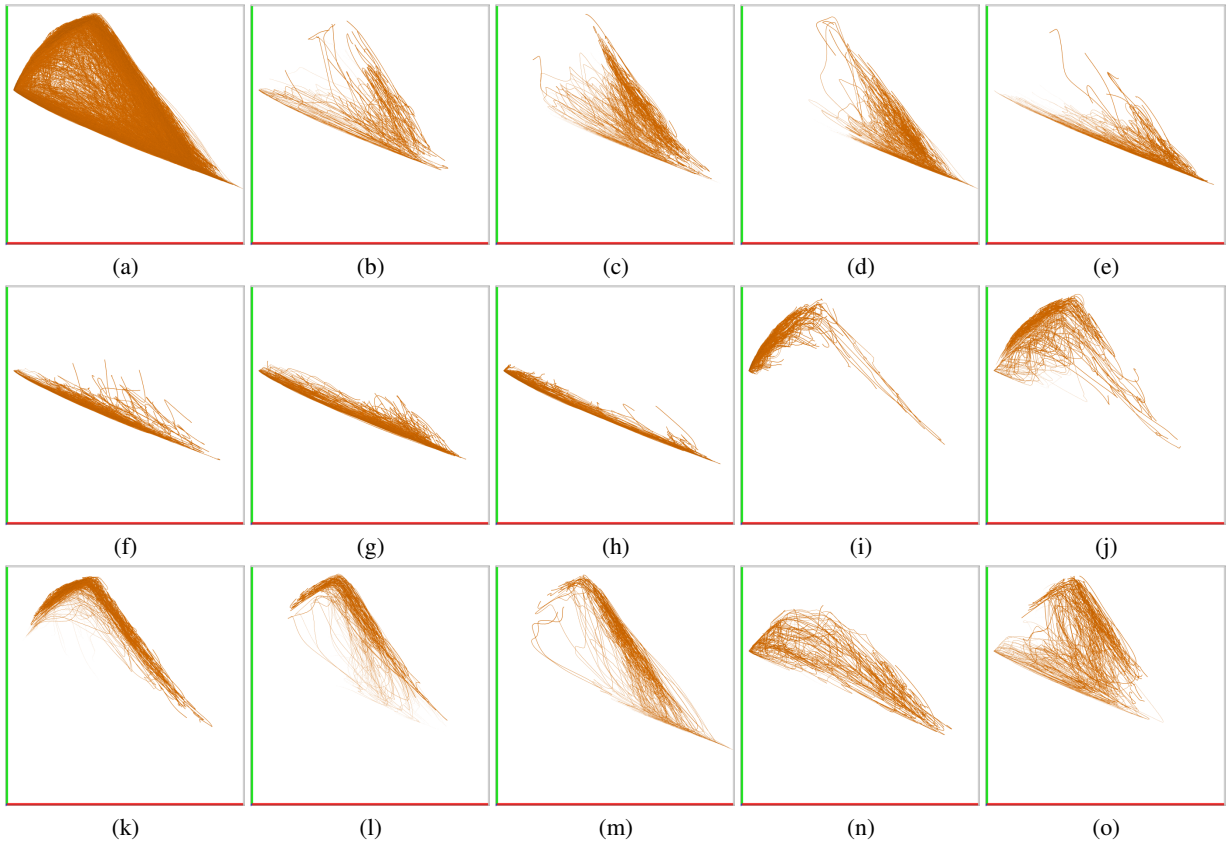

Figure 4: This figure shows the clustering result of the time series curves relating two variables, mixture fraction (the red axis) and temperature (the green axis), in the combustion simulation. (a) shows the overview of all 100,000 time series curves. (b)-(j) show the fourteen different groups of time series curves.

ACKNOWLEDGEMENTS

This work has been sponsored in part by the U.S. Department of Energy through the SciDAC program with Agreement No. DE-FC02-06ER25777, and by the U.S. National Science Foundation through grants OCI-0749227, CCF-0811422, OCI-0749217, OCI-0950008, and OCI-0850566. Sandia National Laboratories is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the U.S. Department of Energy under contract DE-AC04-94-AL85000. The solar plume data set is provided by John Clyne at the National Center for Atmospheric Research.

REFERENCES

[1] W. Aigner, S. Miksch, W. Müller, H. Schumann, and C. Tominski. Visual methods for analyzing time-oriented data. *IEEE Transactions on Visualization and Computer Graphics*, 14(1):47–60, 2008.

[2] J. C. Anderson, L. Gosink, M. A. Duchaineau, and K. Joy. Interactive visualization of function fields by range-space segmentation. *Computer Graphics Forum*, 28(3):727–734, 2009.

[3] A. Brun, H. Knutsson, H. J. Park, M. E. Shenton, and C.-F. Westin. Clustering fiber tracts using normalized cuts. In *Proceedings of International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 368–375, 2004.

[4] D. R. Cutting, D. R. Karger, J. O. Pedersen, and J. W. Tukey. Scatter/gather: a cluster-based approach to browsing large document collections. In *Proceedings of ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 318–329, 1992.

[5] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society, Series B*, 39(1):1–38, 1977.

[6] S. Gaffney and P. Smyth. Trajectory clustering with mixtures of regression models. In *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 63–72, 1999.

[7] S. Gaffney and P. Smyth. Joint probabilistic curve clustering and alignment. In *Advances in Neural Information Processing Systems*, pages 473–480, 2004.

[8] S. J. Gaffney, A. W. Robertson, P. Smyth, S. J. Camargo, and M. Ghil. Probabilistic clustering of extratropical cyclones using regression mixture models. *Climate Dynamics*, 29(4):423–440, 2007.

[9] J. Han. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.

[10] J. A. Hartigan and M. A. Wong. A k-means clustering algorithm. *JSTOR: Applied Statistics*, 28(1):100–108, 1979.

[11] N. P. Kumar, S. Satoor, and I. Buck. Fast parallel expectation maximization for gaussian mixture models on GPUs using CUDA. In *Proceedings of IEEE International Conference on High Performance Computing and Communications*, pages 103–109, 2009.

[12] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297, 1967.

[13] M. Maddah, W. E. L. Grimson, S. K. Warfield, and W. M. Wells. A unified framework for clustering and quantitative analysis of white matter fiber tracts. *Medical Image Analysis*, 12(2):191–202, 2008.

[14] B. Moberts, A. Vilanova, and J. J. van Wijk. Evaluation of fiber clustering methods for diffusion tensor imaging. In *Proceedings of IEEE Visualization Conference*, pages 65–72, 2005.

[15] R. T. Ng and J. Han. Clarans: A method for clustering objects for spatial data mining. *IEEE Transactions on Knowledge and Data Engineering*, 14:1003–1016, 2002.

[16] NVIDIA. *CUDA C Best Practices Guide (version 4.0)*, May 2011.

[17] L. O'Donnell and C.-F. Westin. White matter tract clustering and correspondence in populations. In *Proceedings of International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 140–147, 2005.

[18] A. D. Pangborn. Scalable data clustering using GPUs. Master's thesis, Rochester Institute of Technology, 2010.

[19] C. Rössl and H. Theisel. Streamline embedding for 3D vector field exploration. *IEEE Transactions on Visualization and Computer Graphics*, 99(PrePrints), 2011.

[20] T. Salzbrunn, H. Jänicke, T. Wischgoll, and G. Scheuermann. The state of the art in flow visualization: Partition-based techniques. In *SimVis*, pages 75–92, 2008.

[21] T. Schreck, J. Bernard, T. Von Landesberger, and J. Kohlhammer. Visual cluster analysis of trajectory data with interactive kohonen maps. *Information Visualization*, 8(1):14–29, 2009.

[22] J. S. Shimony, A. Z. Snyder, N. Lori, and T. E. Conturo. Automated fuzzy clustering of neuronal pathways in diffusion tensor tracking. In *Proceedings of International Society for Magnetic Resonance in Medicine*, pages 453–456, 2003.

[23] S. Tomov, R. Nath, P. Du, and J. Dongarra. *MAGMA Users Guide (version 0.2)*, November 2009.

[24] A. Tsai, C.-F. Westin, A. O. Hero, and A. S. Willsky. Fiber tract clustering on manifolds with dual rooted-graphs. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–6, 2007.

[25] J. J. van Wijk and E. R. van Selow. Cluster and calendar based visualization of time series data. In *Proceedings of IEEE Symposium on Information Visualization*, pages 4–9, 1999.

[26] J. Wei, C. Wang, H. Yu, and K.-L. Ma. A sketch-based interface for classifying and visualizing vector fields. In *Proceedings of IEEE Pacific Visualization Symposium*, pages 129–136, 2010.

[27] J. Wei, H. Yu, R. W. Grout, J. H. Chen, and K.-L. Ma. Dual space analysis of turbulent combustion particle data. In *Proceedings of IEEE Pacific Visualization Symposium*, pages 91–98, 2011.

[28] H. Yu, C. Wang, C.-K. Shene, and J. H. Chen. Hierarchical streamline bundles for visualizing 2D flow fields. In *IEEE VisWeek 2010 Posters*, 2010.

[29] T. Zhang, R. Ramakrishnan, and M. Livny. Birch: an efficient data clustering method for very large databases. In *Proceedings of ACM SIGMOD international conference on Management of data*, pages 103–114, 1996.