Parallel Feature Extraction and Tracking for 3D Vortical Data (TBD)

Authors Name/s per 1st Affiliation (Author)
line 1 (of Affiliation): dept. name of organization
line 2: name of organization, acronyms acceptable
line 3: City, Country
line 4: Email: name@xyz.com

Authors Name/s per 2nd Affiliation (Author)
line 1 (of Affiliation): dept. name of organization
line 2: name of organization, acronyms acceptable
line 3: City, Country
line 4: Email: name@xyz.com

***Abstract*—A large-scale time-varying flow data set can take terabytes to petabytes of storage space. One promising approach is to extract features of interest and store only those features. This requires storage space that is several orders of magnitude smaller than the raw data would take. However, extracting and tracking 3D features is a non-trivial task since features from the global perspective are separated within different processing nodes. In this paper, we present a approach to contract the connectivity information of features reside in separated nodes, efficiently. Utilizing the connectivity information, feature extraction and tracking can be done in parallel as they were done in a single node. We demonstrate the effectiveness of this method with three vortical flow datasets and the scalability of our system on parallel and distributed systems.**

***Keywords*-component; formatting; style; styling;**

I. Introduction

The increasing computational power and accessibility to supercomputers have brought scientists the capability to simulate physical phenomena of unprecedented complexity at high spatial and temporal resolution. However, these large-scale time-varying data sets can take gigabyte or even terabytes of space to preserve. One promising solution to the problem is to introduce feature extraction and tracking techniques. Instead of duplicating raw data along the exploring process, dealing with features of interest requires memory space that is several orders of magnitude smaller than the raw data would take. However, extracting and tracking features in distributed vortical datasets is a non-trivial task. Existing researches on feature-based data visualization have been done mostly focusing on decomposing features using quantitative measures such as size, location, shape or topology information, etc. These measures cannot be applied to distribute volume data directly since vortical features are consist of certain amount of voxels and are very likely to span over blocks as they evolve over time. Therefore existing quantitative measures of partial data scattered in different data blocks cannot be used to describe an integrated feature, unless the distribution of partial features can be obtained beforehand.

To obtain the feature distribution, a connectivity map of each feature should be generated and maintained. In this paper, we proposed an approach for creating and maintaining feature residual, as well as connectivity information utilizing parallel graphs. Comparing to previous approaches, which generates and maintains the global features information in a single host node, our approach can be done locally that only involves residual data blocks of target features. This requires least communication overhead and avoids the potential bottleneck on the host node. We demonstrate the effectiveness of this method with three vortical flow datasets and the scalability of our system in a distributed environment.

II. Related Work

Extraction and tracking are two closely related problems in feature-based visualization. Although many feature tracking algorithms have been introduced, most of them extract features from single time steps and then try to associate them between consecutive time steps. Silver and Wang [26] considered threshold connected components as their features, and tracked overlapped features between successive time steps by calculating their differences. Octree was employed in their method to speed up the performance and the criteria they used were domain dependent. Reinders et al. [24] introduced a prediction verification tracking technique that calculates a prediction by linear extrapolation based on the previous feature path, an candidate will be added to the path if it corresponds to that prediction. Ji and Shen [17] introduced a method to track local features from time-varying data by using higher-dimensional isosurfacing. They also used a global optimization correspondence algorithm to improve the robustness of feature tracking [16]. Caban et al. [3] estimated a tracking window and then tried to find the best match between that window and different subvolumes of subsequence frames by comparing their distance of textural properties. Also, Bremer et al. [1] described two topological feature tracking methods, one employs Jacobi sets to track critical points while another uses distance measures on graphs to track channel structures.

Most of the above methods extract features from each time step independently and then apply the correspondence calculation. This could be very slow especially when the size of dataset becomes large. Muelder and Ma [19] introduced a prediction-correction approach that first predicts feature region based on the centroid location of previous time steps,

then correcting the predicted region by adjusting the surface boundaries via region growing and shrinking. This approach is appealing because of its computing efficiency and the reliability in an interactive system.

From the parallel processing side of perspective, most of the existing feature extraction and tracking approaches rely on some form of global feature information, while in distributed environment must be communicated among processors to obtain. Chen and Silver [] introduced a two stage partial-merge strategy, exchanging local connectivity information using Binary-tree merge, then a visualization server correlate local data to create a global data. This approach is not scalable since half of the processors will become idle after each merge. It is also unclear that how the visualization server can efficiently collect local connectivity information from non-server processor, since gathering operation is typically very expensive given a large number of processors.

The approach proposed in this paper follows a different paradigm. Instead of sending local connective information back to the host, the local connectivity information are computed and preserved only in nodes where correspondent feature resides in. There is no global connectivity information preserved in the host node and it only acts as the interface from where the criterion of feature of interest is broadcasted. By doing this, the computation of merging local connectivity information is distributed to those non-host nodes and hence effectively reduces the potential communication bottleneck on the host node. What's more, there's no need to set a barrier and wait before all connectivity information was send back to the host and thus if there exists features that span over a large number of nodes but was not selected by the user, the potentially long computation time for these features will not block the whole process. This makes it ideal for an interactive system, where users can select the feature of interest and instantly receive the visual feedback as the feature evolves.

III. Algorithm Design

Though features can be extracted using the aforementioned methods within individual processors, some of them may span over multiple data blocks, which is unavoidable as the number of processors increases.

To perform feature extraction and tracking over a distributed volume dataset, connectivity graphs from either global or local perspective should be constructed. Since meta-info such as feature size, curvature or velocity of partial features are not shared among processors, the connectivity information makes it possible to gather from each part of the feature that resides in different nodes these meta-info so that feature tracking and similarity test could be achieved.

However, constructing connectivity graphs requires exchanging of local feature information and hence will introduce extra communication cost. To design a proper communication schema for better performance and scalability, the following three factors should be carefully considered: 1. How many communications are required to complete the connectivity graph; 2. How many processors will be involved in one communication; 3. The amount of data being sent and received in one communication. In the following sections, we will give a detailed explanation on how to create and maintain the connectivity graph with minimum cost over the above three factors.

A. Feature Extraction

Feature extraction is a process that first detects features, then calculates quantitative attributes describing its characteristics. In general, features can be any interesting pattern, structure, or object that are considered relevant for the investigation of an application and they can be extracted by common technique such as region growing, geometry or topology based clustering method, or other domain specific algorithms [TODO].

In our system, a standard region-growing algorithm [14] is used to partition the original volume data into an initial set of features. This can be done by first spreading a set of seeding points inside the volume in a breath-first fashion, and then categorizing voxels into separate regions, each regarded as a single feature. Usually, scientists prefer to focus on specific features that are meaningful as dictated by their underlying research goal. Potential features detected by the region-growing algorithm, however, are often cluttered and relatively coarse, which might be far from expected. Further, reducing the number of tracked features makes sense from the perspective of computational expense. Therefore, after the potential features set was generated, an iterative process (TODO) of refinement could be applied to refine the result.

Existing research [27] has detailed the applicability and performance of context-based 3D shape descriptors and their corresponding retrieval methods. Based on the discussion in this work, we can characterize our interactive refinement process as primarily concerned with those computationally efficient yet with significant discriminative power and deformation robust descriptors, such as size, location, or skeletal curvature.

After specific features have been extracted in a single time step, their evolution can be tracked over time using a prediction-correction approach, which was proved effective and efficient [32]. The prediction of feature's region in space can be made based on the location of features reside in consecutive time steps. Once prediction is made, the actual region could be refined adjusting the surface of the predicted region by first shrink the edge surface points to obtain the mutual region between consecutive time steps, then use of a technique based on region-growing to obtain the actual region.

However, as the size of the data grows such that the original volume data cannot fit into one single processor,

a cluster of multiple processors need to be used to process the data in parallel. One challenge of the parallelization of the aforementioned approach lies in that, since volumetric features may span over multiple processors, the global feature descriptions cannot be obtained unless they can be shared and merged in a efficient way, as partial features reside in different nodes will be treated independently. An intuitive way of sharing such feature information is to find the connectivity of features span over multiple nodes, and then integrate them per feature.

B. Creating Partial connectivity graph

If we partition a large volumetric dataset using a regular grid, it is very likely that some of the feature blobs will be cut off on the boundary surface of its residing block. And for two data blocks that are physically adjacent to each other, their boundary surface should match. Leveraging this property, we can connect a feature resides in adjacent data blocks by comparing their correspondent boundary surface.

Since the data is distributed, vortex data in adjacent blocks are invisible to the current processor. Though intuitively, exchanging the array of all the voxices on the boundary surface should work for finding the possible matches, we choose to exchange more abstracted data, the min-max coordinate of the boundary box and the geometric centroid of the boundary surface, to reduce the communication load over the network. The min-max coordinates are not optional that they ensure correct connectivity for cases that a sectional surface is surrounded by a concave or hollow surface whose centroid happen to be the same. see figure x.

A voxel-width "ghost surface" that stores boundary surface belonging to neighboring blocks might also help to achieve voxel-wise matching of partial features. However, for non-post-processing cases where dataset are not predetermined, maintaining such "ghost surface" requires interprocess communication and thus was considered expensive for data generated in real-time. Instead, a 1-voxel tolerance was given when matching geometric centroids on the boundary surfaces. That is, if the neighboring centroid is within the eight direct neighbor of a domestic centroids, they are considered a match.

Some other special cases, though they rarely happen in real world dataset, also need to considered. As shown in Figure x, in certain time steps there might exist features with very sharp edge surface right on the boundary, or even more extreme, parttwo parts of features reside in two adjacent data blocks with an sharp gap in shape on the correspondnet boundary surface (Figure x), this will also be considered as a single feautre as a whole.

The creation of partial connectivity graph will not introduce extra computational cost as it could be done along with the region growing process. A new edge is appended to the existing graph when a feature touches the block boundary, taking the ID of the target block that shares a same boundary surface as the ending vertex, as well as the global index of the centroids point as edge value.

And the detail algorithm is given in Table x.

// TODO Memory cost, each feature on boundary will use two INTs, 1 as global centroids coordinate, the other as target node number. even if there's 1000 features on the boundary, it wont cost more than 1000 * 2 * 4 = 8mb to store this graph, neglectable compare to the volume data itself. (But kind of large if used for communication)

C. Creating Global connectivity graph

1) The naive solution: To obtain the global descriptions of features, partial connectivity information need to be merged together after they were individually created. A naive solution to gather partial connectivity graphs is to send all edges sharing the same ending vertex to that target processor, and to merge edges sent from other processor with local partial connectivity graph. Two edges are merged if 1. their starting and ending vertices are *reversely matched* and 2. both edge index are located within direct neighbors. Recall that the starting vertex of an edge represents the current processor ID and ending vertex the ID of target processor, whose data block is adjacent to the one that reside in the current processor, and the edge index is encoded by the global coordinate of the geometric centroid on the shared boundary surface. If two edges match to each other, the two partial features share a same boundary surface with the same centroid and bounding region. In other word, these two partial features were cut off when partitioning the original dataset, and should be considered the same feature sharing the same feature ID.

The detail algorithm of this REDUCE process is given in Table x.

The naive solution may works well for those dataset whose feature size is small. However if we consider the case that there exist a long curly feature inside the volume and was partitioned evenly over the grid. In order to gather and merge the whole feature, the processors need to talk to its neighbor and spread the edge exchange operation one by one like that in the "telephone" game. This takes O(N) times communication to connect a single feature, where N is the number of total number of processors in the grid. And consequently O(num of feature * N) times communication for all the features within one time step. Also, since one processor has no idea if it will receive any edges from its adjacent processors, it is hard to schedule the communication process.

2) The centralized approach: A possible solution to reduce the number of times required for merging all edges is to employ the master-slave hierarchy by introducing a separate host processor. When the feature extraction process is done, partial connectivity graphs with edges representing how many features have touched the block boundary and where they are located on the surface within each individual

processor will be gathered to the host processor. After the GATHER operation is done, i.e. host processor has collected all partial connectivity graphs, the REDUCE operation starts to merge edges from all partial graphs to a single global connectivity graph.

The merit of this centralized solution is that it only requires sending the data only once. Also, a global graph of feature information could be preserved in the host processor, which made it easy for the host to answer connectivity queries directly without pulling information from slave processors a second time. However the drawback of this approach is also obvious. Since all partial graphs preserved in each processor will be sent to one single processor, there exists potential bottleneck, both communicational and computational, on the host processor.

3) Two decentralized approaches: A better solution is to decentralize the merging process from a single host to all processors that are available, as depicted in Figure x. After the feature extraction process is done and so does the creation of partial connectivity graph, the ALL-GATHER process starts to exchange all partial connectivity graphs within each processor to all the others. This time, each processor will first collect a full copy of all partial connectivity graphs followed by the same REDUCE process to merge the edges into the concise graph.

Though the ”redundant” host processor is not required when applying for computation resources, this approach does not actually solve the bottleneck problem since now every processor is act like a host while they still need to gather all the partial graphs and try merging them. For real world dataset, however, it is rarely the case that a single feature will span over every processor. In other word, it is unnecessary to gather edges of feature that are not reside in current processor. To reduce the redundant communications with every other processor in the grid, the decentralized approach could be further improved in that we only consider those processors that are directly adjacent to the current one. That is, we only talk to our neighbors and share information with them regardless what is happening outside the community.

For a partitioned volumetric dataset, there are only 6 direct neighbors for each processor (for processor on the edge of the grid there are even less). Instead of gathering partial connectivity graphs from all processors in the grid, only graphs reside in neighboring processors will be exchanged. This could be regarded as a higher level of region growing, starting from one seeding processor and grows to all adjacent processors, exchange and merge partial connectivity graphs in a bread-first fashion until all residing processors of the same feature have been connected. For the worst case that a feature span over all of the processors, it takes O(C(—N—)) time to finish the search for connected processors, much faster than the previous O(N) time for all gathering among all processors.

4) The hybrid approach: We can still take a step further to optimize the aforementioned decentralized approach. As volume data evolves over time, the internal features may vary but should not change drastically in size and shape nor location if the time interval for sampling is short enough. Thus we can apply the prediction-correction approach to further reduce the number of times required to complete the connectivity graph.

Every time (say, ti) when the global connectivity graph is completed, new local communicators will be updated for the next time step (ti+1), with the union of processors that share a same edge with the current processor, as shown in figure x. Edges from these processors are required to complete the global connectivity graph anyway, no matter which approach is used. Hence, for these must-involve processors, we apply the Decentralize-I approach, allowing the minimum one-time synchronization to finish gathering all edges necessary for updating the connectivity graph based on the graph created at previous time step (t). Then, the processor-level region growing explained in 3.3.3 is applied to extend the ”boundary” of processes, obtaining newly connected processors caused by the evolution of the original volume. Beside, only those edges that are changed and not synchronized will be sent. This makes sure we keep the amount of data being sent over network to the minimum necessity. The detail algorithm of the hybrid approach is given in Table x.

IV. Application

A. Feature selection and refinement

Since the feature extraction are done independently within each PE, one of the potential function need to be addressed is that how do they know if their adjacent processor was has the feature to be highlighted and whether it has the other part of the features. Intuitively, sending a message to the adjacent PE whenever a feature was detected touching the surface boundary can do this. But if the target feature spans over multiple PEs, this sending/receiving procedure would take several rounds to end. This is potentially a big problem when the PE/Volume ratio is relatively high such that each feature spans over a lot of PEs. Another problem is that, if a PEs has two selected features whose connectivity info arrives in different rounds, it requestes to calculate twice, which again, will become a problem when PE/Volume ratio is large.

By introducing the global connectivity graph in our approach, whenever part of the feature was selected, the unique feature id will be sent back to host processor and then broadcast to all PEs contains it, and only in one rounds, the selection can be finished.

Based on the coordinate user specified or clicked on the volume rendering result, the residual processor as well as the feature, if the point was included by, could be obtains. Then the host processor can simply broadcast the selected

feature id to all those processors who has the partial feature such that they can be highlighted.

B. Feature tracking

Before you begin to format your paper, first write and save the content as a separate text file. Keep your text and graphic files separate until after the text has been formatted and styled. Do not use hard tabs, and limit use of hard returns to only one return at the end of a paragraph. Do not add any kind of pagination anywhere in the paper. Do not number text heads-the template will do that for you.

Finally, complete content and organizational editing before formatting. Please take note of the following items when proofreading spelling and grammar:

V. Result

A. Performance Result

B. Visualiztion Result

VI. Conclusion

In this paper, we proposed a decentralized approach that all feature connectivity information are created and preserved in distributed processors. Traditional approaches perform connectivity test on each processor and subsequently correspond them in a host processor after gathering all or partially merged connectivity information. Our approach does not follow this paradigm. Rather, instead of sending local connectively information back to the host, they are computed and preserved only in processors where correspondent feature resides in. There is no copy of the global feature information preserved in the host processors and it only acts as the interface from where the criterion of feature of interest is broadcasted. By doing this the computation of merging local connectivity information is distributed to the non-host s and it effectively reduces the potential communication bottleneck on the host processor. What's more, there's no need to set a barrier and wait before all connectivity information was send back to the host, thus if one of the features spans over a large number of processors but was not selected by the user, the potentially long computation time for this feature will not be considered. This makes it ideal for an interactive system, where users can select the feature of interest and instantly receive the visual feedback as the feature evolves.

VII. Conclusion

The conclusion goes here. this is more of the conclusion

Acknowledgment

The authors would like to thank... more thanks here

References

[1] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.