Eurographics Symposium on Parallel Graphics and Visualization (2013)
F. Marton and K. Moreland (Editors)

Scalable Parallel Feature Extraction and Tracking for Large Time-varying 3D Volume Data

Yang Wang[1], Hongfeng Yu[2], Kwan-Liu Ma[1]

[1]University of California, Davis
[2]University of Nebraska-Lincoln

Abstract

Large-scale time-varying volume data sets can take terabytes to petabytes of storage space to store and process. One promising approach is to process the data in parallel, and then extract and analyze only features of interest, reducing required memory space by several orders of magnitude for following visualization tasks. However, extracting volume features in parallel is a non-trivial task as features might span over multiple processors, and local partial features are only visible within their own processors. In this paper, we discuss how to generate and maintain connectivity information of features across different processors. Based on the connectivity information, partial features can be integrated, which makes it possible to extract and track features for large data in parallel. We demonstrate the effectiveness and scalability of our approach using two data sets with up to 16384 processors.

Categories and Subject Descriptors (according to ACM CCS): I.3.2 [Computer Graphics]: Graphics Systems—Distributed/network graphics I.4.6 [Image Processing And Computer Vision]: Segmentation—Region growing

1. Introduction

The accessibility to supercomputers with increasing computing power has enabled scientists to simulate physical phenomena of unprecedented complexity and resolution. These simulations generate large-scale time-varying data that can take tera- or even peta-bytes of space to preserve. Such storage requirements will be not sustainable towards the forthcoming exascale computing. One promising solution to the problem is to reduce the data by storing only features of interest. Extracted features require storage space that can be several orders of magnitude smaller than raw data.

However, it is a non-trivial task to extract and track features embedded in large data. Large simulation data are typically presented and processed in a distributed fashion, simply because of the shear size. A feature can span over multiple distributed data blocks, and its distribution can evolve over time. Existing research effort on feature-based data visualization has mostly focused on extracting features using quantitative measures, such as size, location, shape, and topology information. These methods can extract partial features among individual data blocks, but cannot directly assemble partial features to provide integrated descriptions, unless the distribution of partial features can be captured and traced efficiently over time.

Efficiently capturing the distribution of features is challenging with respect to increasing numbers of features and computing nodes. In this paper, we present a scalable approach to generating feature information and tracking feature connectivity information using parallel machines. Compared with the existing approaches that gather the global feature information in a single host node, our approach only involves local covered data blocks of target features. This requires less communication overhead and avoids the potential link contention. We demonstrate the effectiveness and scalability of our method with two vortical flow data sets on large parallel supercomputers with up to 16384 processors.

2. Related Work

2.1. Feature Extraction and Tracking

Feature extraction and tracking are two closely related problems in feature-based visualization. Conventional approaches extract features from individual time steps and then associate them between consecutive time steps. Silver and Wang [SW97] defined threshold connected components as

DOI: 10.2312/EGPGV/EGPGV13/017-024

their features, and tracked overlapped features by calculating their differences. Reinders [RPS] introduced a prediction verification tracking technique that calculates a prediction by linear extrapolation based on the previous feature path, and a candidate will be added to the path if it corresponds to that prediction. Theisel and Seidel [TS03] represented dynamic behavior of features as steam lines of critical points in a higher dimensional vector field, such that no correspondence analysis of features in consecutive time step is required. Ji and Shen [JSW03] introduced a method to track local features from time-varying data using higher-dimensional iso-surfacing. They also used a global optimization correspondence algorithm to improve the robustness of feature tracking [JS06]. Caban et al. [CJR07] estimated a tracking window and compared feature distance of textural properties to find the best match within the window. Bremer et al. [BBD*07] described two topological feature tracking methods where one employs Jacobi sets to track critical points and the other uses distance measures on graphs to track channel structures. Muelder and Ma [MM09] introduced a prediction-correction approach that first predicts a feature region based on the centroid location of that in the previous time steps, and then corrects the predicted region by adjusting the surface boundaries via region growing and shrinking. This approach is appealing for its computing efficiency and the reliability in an interactive system. Ozer and Wei presented a group feature tracking framework [OWS*] that clusters features based on similarity measures and tracks features of similar behavior in groups. However, it is difficult to obtain the global feature descriptions if a single processing node cannot hold the whole volume, unless the descriptions can be shared and merged in an efficient way.

2.2. Parallel Feature Extraction and Tracking

To boost the speed for feature tracking in data-distributed applications, Chen et al. [CSP03] developed a two-stage partial-merge strategy using the master-slave paradigm. The slaves first exchange local connectivity information using Binary-tree merge, and then a visualization host collects and correlates the local information to generate the global connectivity. This approach is not scalable since half of the processors will become idle after each merge. It is also unclear how the host can efficiently collect local connectivity information from the slaves, since gathering operations can be expensive given a large number of processors.

2.3. Parallel Graph Algorithm and Applications

Graph-based algorithms have long been studied and used for a wide range of applications, typically along the line of divide-and-conquer approaches. Grundmann et al. [GKHE10] introduced a hierarchical graph-based approach for video segmentation, a closely related research topic to 3D flow feature extraction as video can be treated as a space-time volume of image data [KpJSFC02]. In their

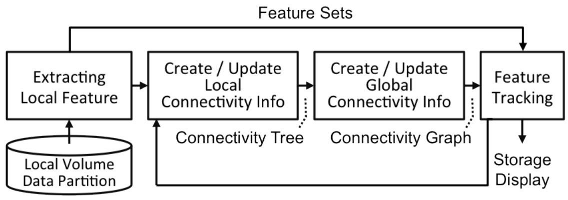

Figure 1: *The major steps of our parallel feature extraction and tracking process.*

work, a connected sequence of time-axis-aligned subsets of cubic image volumes are assigned to a set of corresponding processors, and the incident regions are merged if they are inside the volumes window. The incident regions on window boundary are first marked as ambiguous and later connected by merging the neighboring windows to a larger window, which consists of the unresolved regions form both windows on their common boundary. This approach is not applicable for a memory intensive situation since the allocated volume size before merging might already reached the memory capacity. Liu and Sun [LS10] made a parallelization of the graph-cuts optimization algorithm [BK04], in which data are uniformly partitioned and then are adaptively merged to achieve fast graph-cuts. These approaches are suitable for shared-memory but not message-passing parallelization due to their frequent shifting on data ranges.

3. Methodology

Figure 1 depicts the major steps of our parallel feature tracking process. The data is loaded and distributed among processors. Along with features extraction, the local feature connectivity information is generated within each processor and then merged to obtain the global description. Finally, the features are tracked based on the global connectivity information, and the connectivity information is updated over time accordingly. We represent the global connectivity information in a distributed fashion to avoid communication bottlenecks and to enable scalable feature extraction and tracking.

3.1. Overview

It is challenging to extract and track features of large time-varying volume data in parallel. Although a feature can be extracted partially on a processor using the conventional methods, we first need to build the connectivity information of the feature across multiple processors. Such information allows us to obtain the global description of a feature from a set of neighboring processors, and enables more advanced operations such as statistical analysis and feature similarity evaluation. Second, such connectivity information can be dynamically changed with features evolving over time. We need to update and maintain the connectivity of features in an efficient fashion to track highly intermittent phenomena.

However, building and maintaining connectivity information of features typically requires intensive data exchanges among processors, and thus incurs extra communication costs. To address this issue, we adopt the master-slave paradigm [CSP03], but carefully design our parallel feature representation and schedule inter-processor communication to prevent the host from becoming the bottleneck. The local connectivity information is computed and preserved only by the slaves where the correspondent features reside. Hence, there is no global connectivity information maintained at the host. The host only serves as an interface to broadcast the criterion of features to the slaves. In this way, the computation of merging local information is distributed to the slaves, effectively reducing the potential communication bottleneck on the host.

In addition, our approach does not need to set a barrier to wait for all connectivity information to be sent back to the host. Therefore, if there exist features that span over a large number of nodes but are not explored by a 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 features of interest and instantly receive visual feedback as the features evolve.

Without loss of generality, for each time step, we partition the volume data into a regular grid of blocks. We then distribute the data blocks among processors with each processor is assigned to one block.[†] In general, a feature can be any interesting object, structure, or pattern that is considered relevant for investigation. Here, a feature is defined as the collection of voxels enclosed by a certain iso-surface. Given a sufficiently fine grained partitioning, some features can cross multiple data blocks.

We consider the following two factors in our communication scheme design for better performance and scalability:

- N_{com} : The number of communications required to build the connectivity information;
- $N_{proc/com}$: The number of processors involved in each communication.

3.2. Extracting Partial Local Features

Volume features can be extracted using conventional techniques such as region growing, geometry or topology based clustering, or other domain specific algorithms. In this work, we use a standard region-growing algorithm [Loh98] to identify partial features in each data block. This is done by first spreading a set of seed points inside each data block, and then growing the voxels into a set of separate regions, each regarded as a single feature. Since the data is distributed, a feature can cross multiple blocks, and each processor is not aware of the partial features identified on the other processors in this stage.

[†] Our method can be easily extended to the case that each processor is assigned to multiple data blocks.

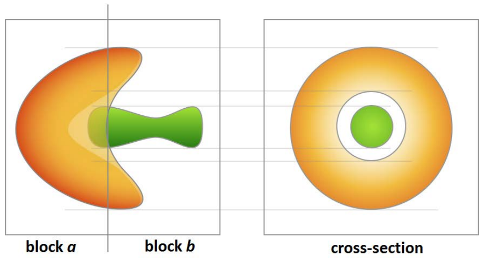

Figure 2: *Two features cross two blocks and share the same centroid on the cross-section.*

Algorithm 1 Match of two partial features f and f'

if $abs(P_{centroid} - P'_{centroid}) \leq 1$ **and** $abs(P_{min} - P'_{min}) \leq 1$ **and** $abs(P_{max} - P'_{max}) \leq 1$ **then**
  return f **matches** f'
end if

3.3. Matching Partial Local Features

For the features across multiple blocks, their cross-sections in both sides of the adjacent blocks should match. Therefore, we can connect two partial features by comparing their cross-sections on the corresponding boundary surfaces. That is, two adjacent processors can find the possible matches of the partial features through exchanging and comparing their boundary voxels. Using a ghost area that stores the boundary surface belonging to a neighbor may help to achieve voxel-wise matching for the partial features. However, maintaining such ghost areas requires frequent inter-process communication and is considerably expensive for interactive applications.

To reduce communication cost and accelerate comparison, we use a simplified method to detect matches. We first represent the cross-section on a boundary surface as:

- $P_{centroid}$: The geometric centroid of the cross-section of the feature;
- P_{min} and P_{max}: The minimal and maximal coordinates of the cross-section area.

For two partial features, we then compare their geometric centroids. If the difference is larger than 1-voxel offset, we consider that they belong to different features. However, only considering geometric centroids is not sufficient to match two features. In some special cases, two different features can have the same geometric centroid on the boundary surface, as shown in Figure 2. Therefore, we also need to consider the min-max coordinates of the cross-section areas to detect bipartite matching of partial features, as shown in Algorithm 1. In this way, we only need to exchange 3 coordinate values, which in most cases are sufficient to detect feature connection across a boundary in practice.

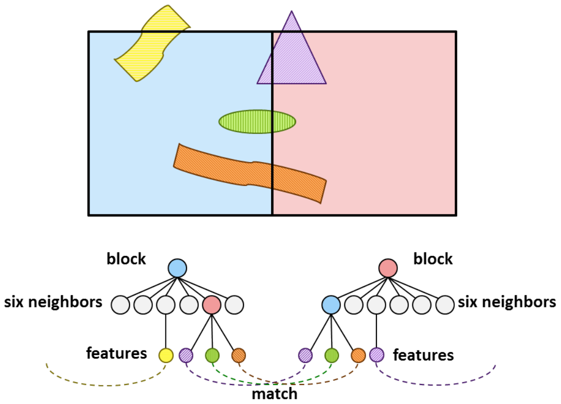

Figure 3: *Top: Two blocks with four features across the blocks. Bottom: The tree structure used for maintaining local connectivity information, where the root node is encoded with the block index, its child nodes are encoded with the indexes of its neighboring blocks, and the leaves of each first level child node represent the local partial features. The leaves should match the ones residing on the corespondent neighboring block, which are indicated by the dashed lines.*

3.4. Creating Local Connectivity Tree

Based on our method to match the partial local features, we can abstract the local connectivity information using a tree structure. As shown in Figure 3, each data block has six direct neighbors (the outermost blocks have less), each with a shared boundary surface. The connectivity tree is constructed by taking the block as root and its six adjacent blocks as its first level child nodes. A leaf is appended to a first level node if and only if a local feature touches the corresponding boundary surface. Note that a feature can touch multiple boundary surfaces and thus be attached to multiple first level nodes.

Each voxel in the local data block has a unique global index, and thus each leaf can be encoded using 3 integers (global index for $P_{centroid}$, P_{min} and P_{max}). We use $P_{centroid}$ as the feature index, and sort the sibling leaves according to the indexes in ascending order. In addition, the root and the first level child nodes can be encoded with the indexes of corresponding data blocks, which are irrelevant to the number of features-on-boundary (henceforth referred as N_{fb}). Therefore, the overall spatial complexity of a local connectivity tree for each data block is $\theta(3 * N_{fb})$, which is typically negligible compared to the volume size.

From the perspective of temporal complexity, the creation of a local connectivity tree does not introduce an extra computational cost as it can be done along with the region growing process. The values of $P_{centroid}$, P_{min} and P_{max} are updated only if a feature reaches the boundary surface.

3.5. Creating Global Connectivity Information

After a local connectivity tree is created within each data block, their leaves need to be exchanged and merged to obtain the overall description of a partitioned feature. The exchanging and merging process is decisive in that its effectiveness largely affects the overall performance and scalability of the feature tracking algorithm as a whole.

3.5.1. Representation of Connectivity Information

Based on the tree structure of the local connectivity information, the global connectivity information can be described as a graph that connects the local connectivity trees, as shown in Figure 3. To facilitate data exchanges among processors, we adopt the linear representation techniques [Sam90] and represent the global connectivity information into a feature table. Each feature has a global unique ID. The table is indexed by the feature IDs and each entry lists the processors that contains the corresponding partial local features. Given this simple representation, once a user selects a feature, each related processor can query the table to identify the other processors that need to be communicated to collectively operate on the selected feature.

3.5.2. The Centralized Approach

One possible solution to build the global feature table is to directly use the master-slave paradigm. When the feature extraction process is done, all local connectivity trees are gathered to the host processor. Then the host starts to merge the leaves from each connectivity tree and matches the partial features to build the global feature table.

The merit of this centralized approach lies in that it requires inter-processor communication only once; that is, $N_{com} = 1$ for each processor. Moreover, the global feature table can be preserved in the host, and it can directly respond to feature queries without collecting information from the slaves again. However, this approach has an obvious drawback. Since all local connectivity trees are sent to the host, the number of processors involved in each communication is $N_{proc/com} = N_p$, and there exists potential contention, both in communication and computation, on the host.

3.5.3. The Decentralized Approach

A better solution is to decentralize the gathering and merging process from using a single host processor to exploiting all available processors. After the feature extraction process and the creation of local connectivity tree are done, an *all-gather* process starts to exchange all local connectivity trees among processors. Each processor first collects a full copy of all local trees and merges the leaves to obtain the global feature table. However, this approach does not actually resolve the contention problem since every processor acts like the host and still need to gather and merge all local trees.

We observe that for a real-world data set, it is rarely the

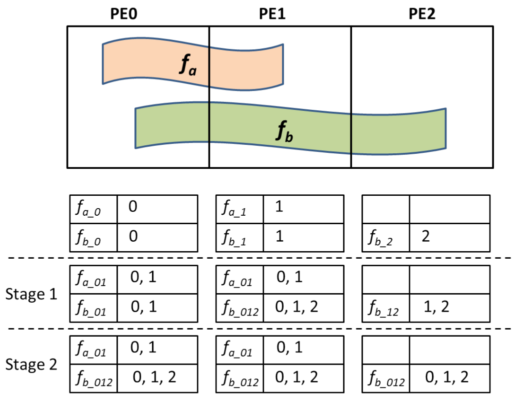

Figure 4: *Construction of partial global feature table with three processors and two features. There are two communication stages. At each stage, each processor only communicates with its immediate neighbors. Each entry of the table is indexed by the feature IDs and lists the processors that contain the corresponding feature.*

case that all features span over every data block. In addition, it is unnecessary for each processor to construct a global feature table to contain all features. Each processor only needs to construct a partial table that records the other processors sharing the same set of features. Thus, it is possible for a processor to communicate with a small set of processors to construct the needed portion of the table. However, we also observe that each processor has no information of the partial features identified on the other processors. Thus, a processor is not initially aware of other processors that can be directly communicated with gather the partial features.

Based on these observations, we design an iterative approach that uses a multi-stage communication scheme. During each stage, each processor only communicates with its immediate neighbors to exchange and propagate the feature connectivity information. This could be considered as a higher level of region growing process that starts from one seeding block and grows to adjacent blocks by exchanging and merging connectivity information in a breadth-first fashion until all cross-boundary features are connected.

Figure 4 gives an example of the procedure to construct a partial global feature table with three processors and two features.[‡] We can see that the feature f_a is identified by the processors PE0 and PE1, and the feature f_b is identified by PE0, PE1 and PE2. Initially, each processor constructs a partial global feature table initialized with only the local features with their local IDs, such as f_{a_0}, f_{b_2}, and so on. In the first stage, PE0 exchanges the local connectivity tree with PE1, and PE1 exchanges the tree with PE2. After exchanging trees, each processor independently matches the partial features, and updates the corresponding feature IDs and entries in its table. For example, the ID of f_a has been changed to f_{a_01} on both PE0 and PE1, and the entry contains the same processor list. However, since the information for f_b has not been propagated between PE0 and PE2, its ID is different on the three processors. In the second stage, each processor still only communicates with its immediate neighbors, and the information of f_b has been propagated to PE0 and PE2 through PE1. Now the f_b ID and its processor list are all the same on the three processors. After an extra communication, each processor detects there is no further information sent from its neighbors, and the construction of the partial global feature table is completed.

After constructing its partial global connectivity table for any selected features, each processor can easily find other corresponding processors. For example, in Figure 4, if f_a is selected, PE0 and PE1 can mutually find that one another belongs to the same communicator, while PE2 is excluded.

The reason we choose the six-direct-neighbor paradigm is because it can minimize the communication cost. It takes a maximum of $3n-1$ communications, where n denotes the maximum processor number among the axes. This corresponds to the maximum communications needed for propagating the information of a feature that covers the whole domain, although this is extremely unlikely in practice. The temporal complexity for garnering all necessary leaves is hence as low as $O(\sqrt[3]{N_{proc}})$, and the number of processors involved in each communication is a constant of maximum six, i.e., $N_{proc/com} \leq 6$.

Another optional paradigm is to let each processor communicate with its 26 neighbors, including the adjacent diagonal blocks. Communication with the adjacent diagonal block takes as much as half the time for any block to reach its furthest diagonal. However, $N_{proc/com}$ is also increased to 26. For data sets where features only span over a small number of blocks, the 6-direct-neighbor paradigm outweighs the 26-neighbors paradigm in communication complexity.

3.6. Updating Global Connectivity Information

To track features, we can construct the global connectivity table for each time step. However, if the time interval is sufficiently small for generating the data, volumetric features may drift but should not change drastically in size, shape, or location. We assume that the changes of each feature are within the range of one block. Based on this assumption, we can optimize feature tracking by incrementally updating the global connectivity information over time.

As depicted in Figure 5, each processor constructs a partial global feature table at time step t_i. Meanwhile, we maintain a communicator, **C**, which contains the corresponding processors for each feature. For example, feature f_c spans over PE0, PE1, and PE2. These three processors have the same table entry with respect to f_c. The table of PE3 is empty at t_i. PE0, PE1 and PE2 belong to the same communicator **C**.

[‡] For simplicity, we use an example of 1D partitioning. However, the procedure can be easily extended to 3D cases.

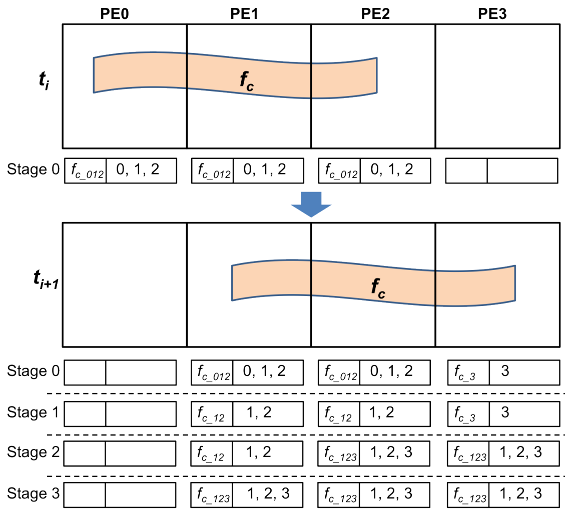

Figure 5: *Update of partial global feature table with four processors and one feature. Feature f_c is extracted and adjusted in Stage 0 followed by three communication stages to record the possible shrinking and expanding of the feature.*

For the next time steps, t_{i+1}, each processor continues to predict and correct boundaries as to extract partial local features. For the existing features, their IDs are retained in the partial global feature table. For the new features, their IDs are added into the table. Lastly, IDs are erased from the table if the corresponding features drift away from that block. As shown in Figure 5, f_c leaves PE0 and enters PE3. In this case, the table of PE0 becomes empty, and the table of PE3 adds a new entry. At this step, the feature ID on PE3 is not the same as the others, as the feature has not been matched yet. In addition, PE0, PE1 and PE2 still belong to **C**.

Next we start to update the connectivity information. In the first stage, PE0, PE1, and PE2 perform an *all-gather* operation within their communicator **C** to update the connectivity. PE0 is then removed from the corresponding entry on PE1 and PE2, and is also removed from **C**. In the second stage, each processor exchanges the local connectivity information with the immediate neighbors as the decentralized approach in Section 3.5.3. The information of f_c is propagated between PE2 and PE3. In the third stage, all the processors in the communicator **C** perform an *all-gather* operation again to update the connectivity. The information of f_c is propagated to the rest of processors **C**, and now PE1, PE2, and PE3 have the same table at t_{i+1}. Given the unified information, we can then update the communicator **C** by including PE3 with respect to f_c.

This update procedure can be easily extended to the circumstances with more processors and features. We note that the cost of collective communication is marginal within a communicator of limited size. By leveraging this nice property for each feature, we only need at most three stages to update the connectivity information, independent of the length of the feature.

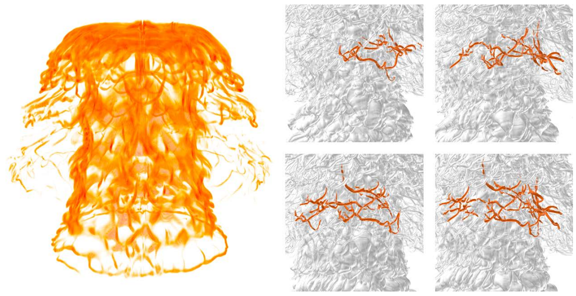

Figure 6: *Left: The volume rendering of a single time step of the combustion data set; Right: Selected features of interest extracted and tracked overtime.*

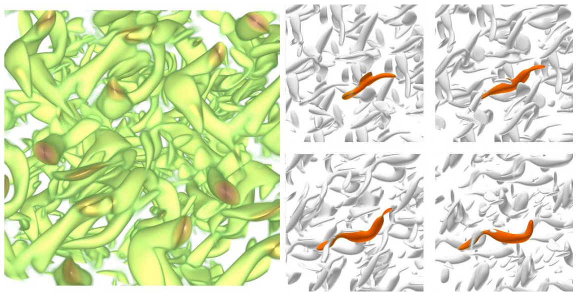

Figure 7: *Left: The volume rendering of a single time step of the vortex data set; Right: Selected features of interest extracted and tracked overtime.*

4. Results

We test our feature extraction and tracking algorithm using the NERSC Hopper cluster on two datasets. A 400 time steps 256^3 vortex data set obtained from a combustion solver that can simulate turbulent flames, and a 100 time steps 1024^3 vortex data set synthesized from the 128^3 volume data set used in the other works [SW97, JSW03, JS06]. In the combustion data set, each voxel contains the magnitude value of vorticity derived from velocity using a curl operator. As time evolves, vortical features may vary from small amassed blob features to long curly features that span over large portion of the volume. Figures 6 and 7 show the examples of identified and tracked features in these two data sets, which match the non-parallelized tracking results. We ignore the I/O cost and only focus on the computation time in our study.

Time for extracting features ($T_{extract}$):

Because we use region-growing based algorithm to extract features, given a fixed specification of feature, the computation time is determined by the size of the volume as well

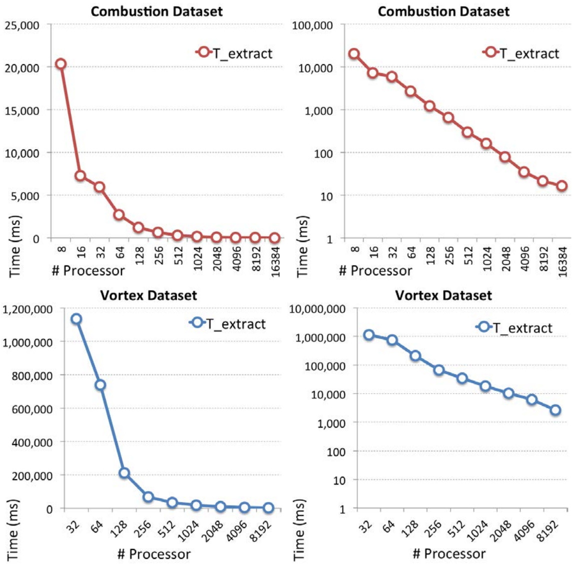

Figure 8: *Average computation cost per time step for feature extraction. The left two plots are shown in linear scale, and the right plots are shown in logarithmic scale. The speedup is nearly linear with the number of processors.*

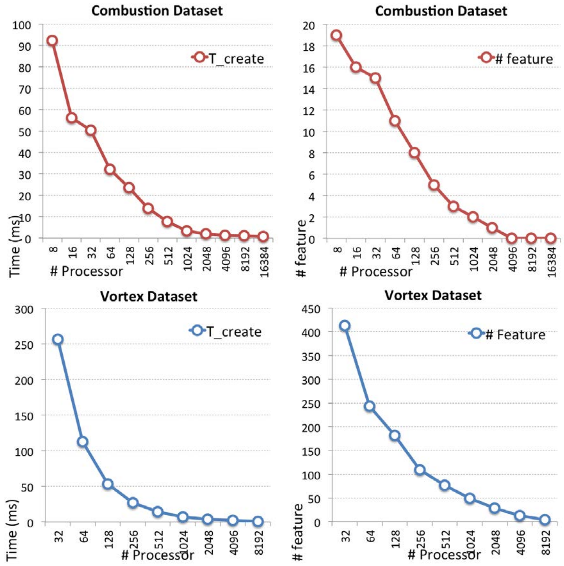

Figure 9: *Average computation cost per time step for creating local connectivity tree. The speedup is nearly linear with the number of processors. The time cost is approximately proportional to the number of features-on-boundary.*

as the number of processors being used. Once the volume data and its partitioning (the size of each data block) is determined, the computation time for extracting residing features remains approximately the same. In post-processing, the size of each data block decreases with the increasing number of processors, and hence so does the time spent on extracting features. As depicted in Figure 8, $T_{extract}$ decreases log-linearly as the number of processors increases.

Create Local Connectivity Tree (T_{create}):

Despite the size of each data block, the computation cost for creating and updating local connectivity tree is dependent on the number of the features extracted within the original volume, or more precisely, the number of features that touches the boundary surfaces of their residing data block. As shown in Figure 9, similar to $T_{extract}$, T_{create} decreases as the number of processors increases, and as the number of features-on-boundary decreases. For both the combustion and vorticity data set, it takes an average of 0.1 seconds to create the local connectivity tree, approximately 0.5% of the time of $T_{extract}$ using the same amount of processors. The $T_{create}/T_{extract}$ ratio increases but does not exceed 1% in our test, and hence, T_{create} is not considered as a bottleneck.

Create Global Connectivity Information (T_{merge}):

We also compared the performance for both centralized and decentralized approach in creating global connectivity information, which is the major factor related to the scalability of our algorithm. Though the number of features-on-boundary decreases as more processors involve, the communication cost for the centralized approach increases as N_p

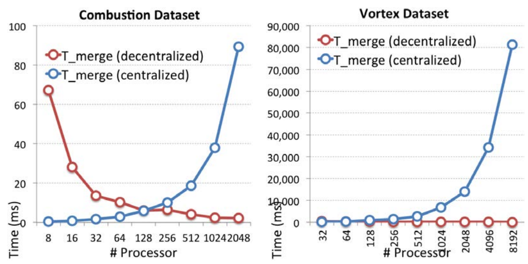

Figure 10: *The comparison of the average computation cost per time step between the centralized and the decentralized approach. The centralized approach works well for a small number of processors while the decentralized approach exceeds after a certain number, e.g. 128 processors for the combustion data set, is used.*

increases. As shown in Figure 10, the centralized approach is suitable for scenarios that only a small number of processors are required, while the decentralized approach outperforms when a large amount of processors are used. From the overall performance perspective, when T_{merge} exceeds $T_{extract}$ after using a certain amount of processors, 2048 for instance in Figure 11, the overall execution time rebounds for the centralized approach. On the other hand, the decentralized approach scales well up to 16384 processors for the combustion data set, as the communication cost is as low as $O(\sqrt[3]{N_p})$.

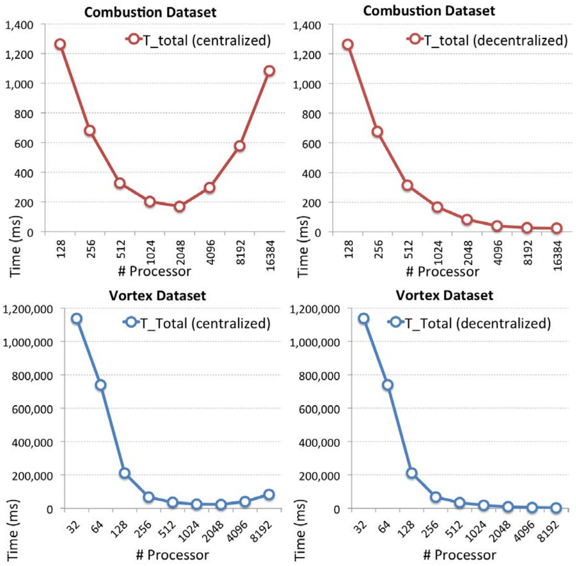

Figure 11: *The comparison of the average computation cost per time step for different approaches to global connectivity information generation. The centralized approach scales up to 2048 processors but the merging time outweighs the extraction time when using more processors; The decentralized approach scales linearly up 16384 processors for the combustion data set.*

5. Conclusion

We present a scalable approach to extracting and tracking features for large time-varying 3D volume data using parallel machines. We carefully design the communication scheme such that only minimal amounts of data need to be exchanged among processors through local communications. The features are tracked in parallel by incrementally updating the connectivity information over time. Compare to the naive centralized solution, our decentralized approach can significantly reduce the communication cost and ensure the scalability with up to 16384 processors. To the best of our knowledge, no prior approaches can extract and track features at this scale (in terms of the number of processors).

Our approach shows performance that is as scalable as large scientific simulations. In the future, we plan to integrate our approach with large simulations and conduct experimental studies on in situ feature extraction and tracking during a simulation execution. The study can possibly enable scientists to capture highly intermittent transient phenomena which could be missed in post-processing. In addition, we would like to investigate the feature-base data reduction and compression techniques to significantly reduce simulation data but retain the essential features for scientific discovery. Our parallel feature extraction and tracking approach builds a solid foundation for these future studies.

6. Acknowledgment

This work has been supported in part by the U.S. National Science Foundation through grants OCI-0749227, CCF-0811422, OCI-0850566, and OCI-0905008, and also by the U.S. Department of Energy through the SciDAC program with Agreement No. DE-FC02-06ER25777 and DE-FC02-12ER26072, program manager Lucy Nowell.

References

[BBD*07] Bremer P.-T., Bringa E. M., Duchaineau M. A., Gyulassy A. G., Laney D., Mascarenhas A., Pascucci V.: Topological feature extraction and tracking. *Journal of Physics: Conference Series 78*, 1 (2007), 012007. 2

[BK04] Boykov Y., Kolmogorov V.: An experimental comparison of min-cut/max- flow algorithms for energy minimization in vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence 26*, 9 (2004), 1124–1137. 2

[CJR07] Caban J., Joshi A., Rheingans P.: Texture-based feature tracking for effective time-varying data visualization. *IEEE Transactions on Visualization and Computer Graphics 13*, 6 (2007), 1472 –1479. 2

[CSP03] Chen J., Silver D., Parashar M.: Real time feature extraction and tracking in a computational steering environment. In *Proceedings of High Performance Computing Symposium* (2003), pp. 155–160. 2, 3

[GKHE10] Grundmann M., Kwatra V., Han M., Essa I.: Efficient hierarchical graph-based video segmentation. In *Proceedings of Computer Vision and Pattern Recognition* (2010). 2

[JS06] Ji G., Shen H.: Feature tracking using earth mover's distance and global optimization. In *Proceedings of Pacific Graphics* (2006). 2, 6

[JSW03] Ji G., Shen H.-W., Wenger R.: Volume tracking using higher dimensional isosurfacing. In *Proceedings of IEEE Visualization* (2003), pp. 209–216. 2, 6

[KpJSFC02] Klein A. W., pike J. Sloan P., Finkelstein A., Cohen M. F.: Stylized video cubes. In *Proceedings of ACM SIGGRAPH Symposium on Computer Animation* (2002). 2

[Loh98] Lohmann G.: *Volumetric Image Analysis*. Wiley & Teubner Press, 1998. 3

[LS10] Liu J., Sun J.: Parallel graph-cuts by adaptive bottom-up merging. In *Proceedings of Computer Vision and Pattern Recognition* (2010), pp. 2181–2188. 2

[MM09] Muelder C., Ma K.-L.: Interactive feature extraction and tracking by utilizing region coherency. In *Proceedings of IEEE Pacific Visualization Symposium* (2009), pp. 17 –24. 2

[OWS*] Ozer S., Wei J., Silver D., Ma K.-L., Martin P.: Group dynamics in scientific visualization. In *Large Data Analysis and Visualization (LDAV), 2012 IEEE Symposium on*. 2

[RPS] Reinders F., Post F. H., Spoelder H. J. W.: Visualization of time-dependent data using feature tracking and event detection. 2

[Sam90] Samet H.: *Applications of Spatial Data Structures: Computer Graphics, Image Processing and GIS*. Addison-Wesley Publishing Company, 1990. 4

[SW97] Silver D., Wang X.: Tracking and visualizing turbulent 3d features. *IEEE Transactions on Visualization and Computer Graphics 3*, 2 (1997), 129–141. 1, 6

[TS03] Theisel H., Seidel H.: Feature flow fields. *Proceedings of the symposium on Data . . .* (2003). 2