Efficient Unsteady Flow Visualization with High-Order Access Dependencies

Jiang Zhang[1] [*] Hanqi Guo[3] [†] Xiaoru Yuan[1,2] [‡]

1) Key Laboratory of Machine Perception (Ministry of Education), and School of EECS, Peking University
2) Beijing Engineering Technology Research Center of Virtual Simulation and Visualization, Peking University
3) Mathematics and Computer Science Division, Argonne National Laboratory

Abstract

We present a novel high-order access dependencies-based model for efficient pathline computation in unsteady flow visualization. By taking longer access sequences into account to model more sophisticated data access patterns in particle tracing, our method greatly improves the accuracy and reliability in data access prediction. In our work, high-order access dependencies are calculated by tracing uniformly seeded pathlines in both forward and backward directions in a preprocessing stage. The effectiveness of our approach is demonstrated through a parallel particle tracing framework with high-order data prefetching. Results show that our method achieves higher data locality and hence improves the efficiency of pathline computation.

1 Introduction

Field line tracing is a fundamental technique in flow visualization and analysis. Various applications require massively tracing streamlines or pathlines, such as flow surface computation [8], finite-time Lyapunov exponent (FTLE) computation [14], and source-destination analysis [16]. However, field line computation is both data- and computational-expensive, especially for large and complex unsteady flow data. The dominant cost in field line tracing is I/O, which may take up to 90% time [15]. The key to mitigating the data I/O burden is to improve the data locality, which has been studied extensively [5, 3, 6]. One major solution is to incorporate data access patterns in field line tracing. In flow visualization, however, modeling data access patterns is challenging because they are implicitly determined by flow field features. The access patterns are regarded as random in most studies.

One approach to model data access patterns is using Markov chains. In stochastic models, the probability of accessing one block (chunk of data) depends only on the previously visited block. The first-order access patterns are recorded by calculating the state transition probabilities for each pair of blocks. Then in the field line tracing process, the next data accesses can be evaluated in advance according to which block the particle currently sits. This first-order Markov model was successfully applied to computational fluid dynamics (CFD) postprocessing for reducing data access time [10]. Moreover, in flow visualization, a few attempts had been made to use this model for both streamline computation in steady flow fields [5] and pathline computation in unsteady flow fields [3, 13], in order to predict data accesses and thereby improve data localities. The limitation of the Markov chain-based method is that the state may transit from one data block to various other data blocks, which means a particle may have various probabilities to access in the next step. As a result, determining the next data access is difficult because a lot of data blocks may be possible. In other words, accurate and reliable data access patterns cannot be achieved.

In this work, we introduce a novel access dependency model based on high-order Markov chains [25]. The high-order access dependencies are calculated in order to improve the data localities and efficiency in the field line computation. Specifically, historical access information is taken into account in high-order access dependencies, which improves the prediction accuracy of the next data accesses. Different from the first-order access dependencies-based method commonly employed [10, 5, 3, 13], the prediction of the next data accesses relies on both current and several previously visited data blocks in our method. By incorporating historical data accesses, the number of probabilities for the next data accesses is less, and the precision of each probability is higher. Hence, more reliable predictions of the next access patterns can be achieved during particle advection.

In our implementation, high-order access dependencies are computed in a preprocessing stage and are reusable for further pathline-based visualization tasks. The data is first partitioned into regular and small blocks. By uniformly seeding particles in each block, we perform a round of particle tracing in both forward and backward directions in the domain. The high-order access dependencies between data blocks are calculated based on the data access information recorded in these pathlines. To further explore the advantages of high-order access dependencies in pathline computation, we apply a task-parallel particle tracing framework with high-order data prefetching. In our work, the high-order access dependencies are organized together with corresponding data blocks. During particle advection, when a block is requested, the system precaches the blocks with higher probability to be visited in the immediate future. High-order access dependencies achieve higher prefetching accuracy and cache hit ratio, thus reducing I/O requests.

In the remainder of this paper, we review the background of this work in Section 2. In Section 3, the high-order access dependencies in unsteady flow fields are described in detail. Section 4 depicts the parallel particle tracing framework with data prefetching, which applies high-order access dependencies. Results are shown in Section 5 to demonstrate the effectiveness of our method. In Section 6, we present our conclusions and discuss future work.

2 Background

We summarize related work on advection-based flow visualization, data access dependencies in flow visualization, and parallel particle tracing and data prefetching in general.

2.1 Advection-Based Flow Visualization

Field line advection plays a vital role in flow visualization and analysis. Many flow visualization tasks require field line-based analysis, such as texture-based visualization [17], geometry-based visualization [19], and feature extraction and tracing in flow visualization [24]. Based on different strategies of particle placement, these

[*]e-mail: jiang.zhang@pku.edu.cn
[†]e-mail: hguo@anl.gov
[‡]e-mail: xiaoru.yuan@pku.edu.cn (corresponding author)

visualization methods can be divided into two categories: full-range analysis and local-range analysis.

In full-range analysis, particles are seeded over the entire domain for exploring overview features of flow fields. Numerous applications, such as line integral convolution [2], unsteady flow LIC [26], and Lagrangian coherent structures extraction [9], are computed by densely seeding particles in the entire spatiotemporal domain. The computation in full-range analysis is always expensive because of tracing a large number of particles. In contrast, local-range analysis requires seeding particles in a local subdomain. Methods in this category are typically used to explore local features in flow fields, such as source-destination queries [16] and flow surfaces computation [8].

2.2 Data Access Dependencies in Flow Visualization

Based on the fact that certain data access patterns exist in particle advection, recent research has focused on the access dependencies among data blocks. The access dependency graph (ADG) [5] was proposed for file layout optimization of static flow field data. In ADG, each node represents a data block, and each direct edge between two nodes records the access probability for a particle seeded in one block traveling to the other block. Furthermore, ADG was extended to compute pathlines in unsteady flow field [3], as well as to effectively schedule particle seeds for efficient out-of-core FTLE computation [4]. Recently Guo et al. [13] introduced a novel advection-based sparse data management for efficient and scalable unsteady flow visualization based on data access dependencies. In that work, the access dependencies were learned and recorded as hints during particle tracing. This on-the-fly construction was also employed to build a probability graph that represents the successor relation of blocks for CFD datasets [10]. However, the access dependencies in these methods are all first order, and the predictions of data accesses are not accurate and reliable. More sophisticated access patterns exist in particle tracing. Therefore, our work also takes historical access information into account. By matching the high-order historical information of a pathline, next data accesses can be predicted with higher accuracy.

2.3 Data Prefetching and Parallel Particle Tracing

Our work is related to data prefetching, a widely used technique to improve the I/O access performance for particle tracing. The effectiveness of data prefetching depends on the predictions of next data accesses. In order to hide I/O latency for parallel applications, a pre-execution prefetching framework was also proposed [7]. Moreover, an I/O signature-based strategy [1] used a predetermined I/O signature generated automatically to guide prefetching and reduce the overhead of future I/O read prediction.

In order to efficiently generate high-order access dependencies, our method employs parallel computation in the preprocessing stage. Tracing particles in parallel is a common solution in visualizing large unsteady flow fields. There are three parallel strategies for parallel particle tracing: task-parallelism, data-parallelism, and hybrid-parallelism. The task-parallel methods focus on workload distribution through various schedule strategies. Static workload estimation [22] and dynamic load-balancing [20] were both well studied. In data-parallel methods, the strategies of data partitioning and distribution are dominant. Data can be partitioned based on flow features [6], over time steps [21], or be distributed by round-robin [23], hierarchical clustering [28]. Hybrid-parallel method that combines task- and data-parallelism was introduced recently in scalable stream surface computation for balancing workload distribution [18]. Moreover, a MapReduce-like hybrid-parallel particle tracing framework, DStep [16], was proposed, which employs a two-layer synchronous communication to improve scalability. Further LASP [11] and *e*FLAA [12] redesigned the DStep framework [16] for Lagrangian-based attribute space projection and ensemble flow analysis, respectively. In this work, we modify the DStep framework [16] to generate high-order access dependencies, and we demonstrate their advantages in both full-range and local-range analyses by a task-parallel particle tracing framework with high-order data prefetching.

3 High-Order Access Dependencies in Unsteady Flow Field

The pipeline of our work is illustrated in Figure 1. In our approach, we first trace pathlines in a preprocessing stage in order to compute the high-order access dependencies. The raw data is partitioned into small blocks indexed by their spatiotemporal locations, which are also units in the computation of high-order access dependencies. For each block, the high-order access dependencies are computed according to the pathlines starting from this block. The data access patterns then are recorded in high-order access dependencies permanently.

The high-order access dependencies support more accurate predictions of next data accesses in pathline computation because of the higher data locality achieved. Many unsteady flow visualizations can take advantage of the computed high-order access dependencies. In this work, we apply a parallel particle tracing framework that performs high-order data prefetching for efficient pathline computation. When advecting pathlines, the next data accesses can be predicted and prefetched with higher accuracy compared with the first-order access dependencies-based method (called the first-order method for short). Details are described in Section 4.

3.1 Basics of High-Order Access Dependencies

The theoretical and mathematical foundation of high-order access dependencies can be modeled with a high-order Markov chain [25]. A Markov chain is a mathematical system that describes a series of states and the transitions from one state to another. The conditional probabilities associated with state transitions are called transition probabilities. For a high-order Markov chain, the next state not only depends on the current state but also relies on the past states. In our work, accessing one data block is called one *data access*, which is similar to the definition of a state in a Markov chain.

Suppose there are $n+1$ data accesses, $B_1,\ B_2,\ \ldots,\ B_n,\ B_{n+1}$. Among them, $B_1,\ B_2,\ \ldots,\ B_{n-1}$ are the historical data accesses, B_n is the current data access, and B_{n+1} is the next data access. In previous methods based on first-order access dependencies [10, 5, 3, 13], the next data access and historical data accesses are independent. The relationship between these data accesses is formulated as

$$P(B_{n+1}=b \mid B_n=b_n,\ B_{n-1}=b_{n-1},\ \ldots,\ B_1=b_1) = P(B_{n+1}=b \mid B_n=b_n), \quad (1)$$

which means that the next data access depends only on the current data access in first-order access dependencies. In high-order access dependencies, however, the historical data accesses are also taken into account to guide more accurate prediction of the next data access. Similar to transition relationships between states in a high-order Markov chain, we have the following equation for the mth-order access dependencies $(m>1)$:

$$P(B_{n+1}=b \mid B_n=b_n,\ B_{n-1}=b_{n-1},\ \ldots,\ B_1=b_1) = P(B_{n+1}=b \mid B_n=b_n,\ B_{n-1}=b_{n-1},\ \ldots,\ B_{n-m+1}=b_{n-m+1}), \quad (2)$$

which means the next data access is determined by both the historical $(m-1)$ data accesses and the current data access.

Figure 2 illustrates the difference between a first-order and high-order (second-order in the figure) access dependency model. In the directed graphs, each node represents a data access, and each

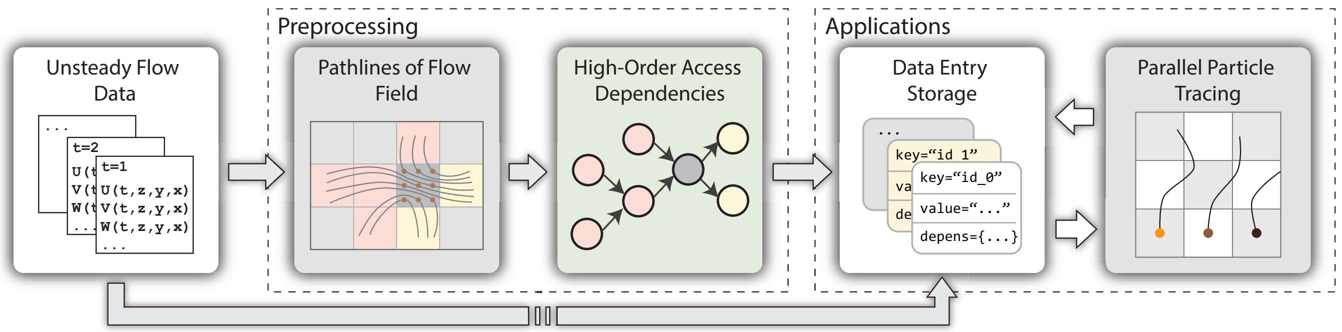


Figure 1: Pipeline of our work. In the preprocessing stage, particles are uniformly seeded and traced by the input of the raw flow data. The high-order access dependencies are further computed according to the generated pathlines. We further integrate high-order access dependencies into data blocks. A parallel particle tracing framework that performs high-order data prefetching is employed to demonstrate that our method achieves better efficiency than that of the first-order method.

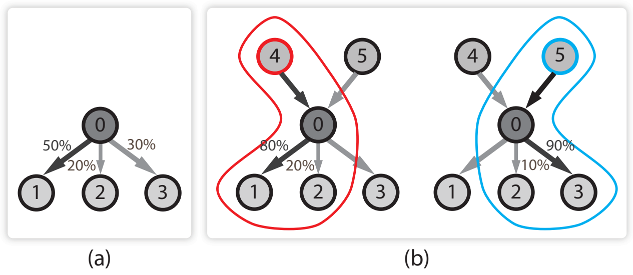


Figure 2: Comparison between the (a) first-order and (b) second-order access dependencies using graph models.

edge represents the dependency between two data accesses. In Figure 2(a), node 0 has an access-dependent relationship with three neighboring nodes 1, 2, and 3. Thus, there are three different probabilities for a particle to continue advecting from node 0. In Figure 2(b), according to the historical data access, the number of probabilities of next data access is reduced to two. If the historical data access is node 4, we can consider just nodes 1 and 2 as the next possible data accesses and ignore node 3. If the historical data access is node 5, nodes 2 and 3 are taken into consideration while node 1 is ignored. The prediction accuracy is improved when integrating the historical data access information.

3.2 High-Order Access Dependency Definition

In high-order access dependencies, the next data access depends on both the historical data accesses and the current data access. According to Equation 2, we define one mth-order access dependency $(m > 1)$ formally as

$$\{B_{p_{m-1}}, B_{p_{m-2}}, \dots, B_{p_1}, B_c\} \xrightarrow{p} B_f, \tag{3}$$

where $B_{p_{m-1}}, B_{p_{m-2}}, \dots, B_{p_1}$ are the $m-1$ $(m > 1)$ data blocks accessed in the past, B_c is the current data block, and B_f is the data block that may be accessed in the immediate future. p is the access transition probability from $B_{p_{m-1}}, B_{p_{m-2}}, \dots, B_{p_1}$ and B_c to B_f, which generally represents the probability that a particle travels from $B_{p_{m-1}}, B_{p_{m-2}}, \dots, B_{p_1}$ and B_c to B_f. In other words, a high-order access dependency provides the prediction of the next data access with certain probability according to the historical and current data access information.

The number of high-order access dependencies for a data block is dominated by the variety of historical and next data access information. In the preprocessing stage, the high-order access dependencies of each data block all will be computed.

3.3 High-Order Access Dependencies Computation

Given an unsteady flow data, the entire data domain is evenly partitioned into blocks, each containing an equal-size spatial range with sequential timesteps. Particles are uniformly seeded in different positions within each data block. Because the historical access information is also needed in high-order access dependencies, tracing particles along one direction all the time is no longer applicable. Instead, each particle is traced in both forward and backward directions starting from the same seed position. The corresponding pathline generated is called *forward pathline* and *backward pathline*, respectively. Step by step, forward pathlines are traced by increasing the time while backward pathlines are traced by decreasing the time. Figure 3 shows forward and backward pathlines. The forward and backward pathline seeded in the same position can be considered as one pathline that integrates both historical and next access information, as shown in Figure 3(c).

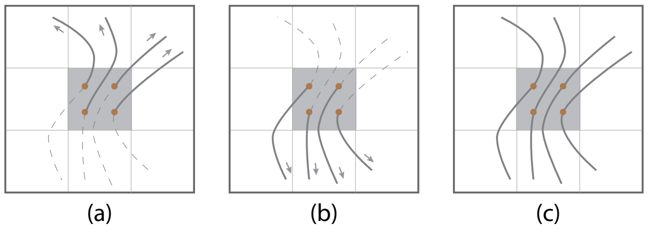
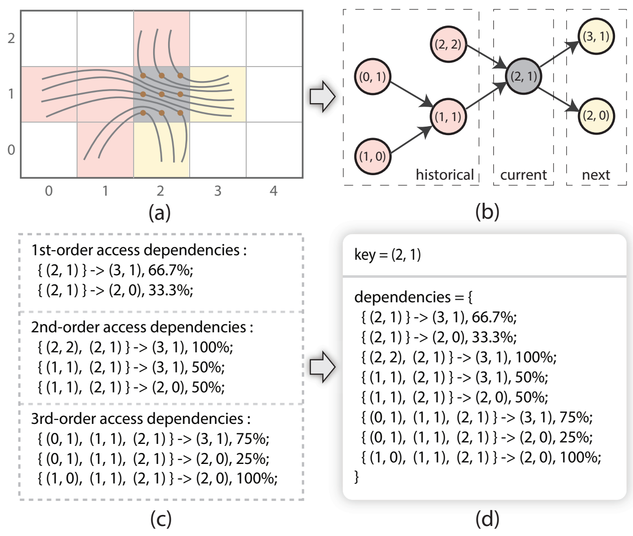


Figure 3: Illustration of (a) forward pathlines and (b) backward pathlines. In (c), the forward and backward pathline seeded in the same position are considered as one pathline.

There are several strategies to reduce the preprocessing time when tracing pathlines. Because high-order access dependencies are generated according to the data access information recorded in pathlines rather than the pathlines themselves, we do not need to trace complete and accurate pathlines. For a forward pathline, when it travels from the originating block to next block, the advection is stopped, because we want to know only which is the next data access. For a backward pathline, the number of blocks it traveled should not be larger than the order of access dependencies which is specified in advance. During pathline tracing, the first-order Runge-Kutta (RK1) instead of an RK4 method can be used, and the step size can be set relatively larger.

Considering the current data block, all pathlines including forward and backward pathlines started from this block are gathered for the computation of corresponding high-order access dependencies. The data access information has been recorded in these pathlines. As illustrated in Figure 3(c), we merge each pair of forward and backward pathlines that are seeded in the same position to form a new pathline. For the merged pathlines, excluding the originating block, the data blocks accessed by corresponding backward pathlines are considered as the historical access information, while the blocks accessed by corresponding forward pathlines are seen as the possible access information in the immediate future.



1st-order access dependencies :
{ (2, 1) } -> (3, 1), 66.7%;
{ (2, 1) } -> (2, 0), 33.3%;

2nd-order access dependencies :
{ (2, 2), (2, 1) } -> (3, 1), 100%;
{ (1, 1), (2, 1) } -> (3, 1), 50%;
{ (1, 1), (2, 1) } -> (2, 0), 50%;

3rd-order access dependencies :
{ (0, 1), (1, 1), (2, 1) } -> (3, 1), 75%;
{ (0, 1), (1, 1), (2, 1) } -> (2, 0), 25%;
{ (1, 0), (1, 1), (2, 1) } -> (2, 0), 100%;

key = (2, 1)

```
dependencies = {
  { (2, 1) } -> (3, 1), 66.7%;
  { (2, 1) } -> (2, 0), 33.3%;
  { (2, 2), (2, 1) } -> (3, 1), 100%;
  { (1, 1), (2, 1) } -> (3, 1), 50%;
  { (1, 1), (2, 1) } -> (2, 0), 50%;
  { (0, 1), (1, 1), (2, 1) } -> (3, 1), 75%;
  { (0, 1), (1, 1), (2, 1) } -> (2, 0), 25%;
  { (1, 0), (1, 1), (2, 1) } -> (2, 0), 100%;
}
```

Figure 4: 2D example to show the computation of first-order and high-order access dependencies. (a) The nine pathlines originated from block (2, 1); (b) The graph model of access dependencies recorded in the pathlines from (a); (c) The individual access dependencies from 1st- to 3rd-order of block (2, 1); (d) The 3rd-order "accumulated" access dependencies of block (2, 1), which include all access dependencies with 1st, 2nd, and 3rd order.

In order to generate access dependencies with specified order m ($m > 1$), the merged pathlines with the amount of historical accessed blocks equal to $m-1$ are taken into account. Because the historical access information of these pathlines may still be different, we cluster these pathlines into groups. For the pathlines in each group, the historical access information is the same, but the probabilities of the next data access may be various. Suppose there are n probabilities of the next data access in total, in our method, each possible next data access B_{f_i} associated with the same access history B_p (short for the sequence of $B_{p_{m-1}}$, $B_{p_{m-2}}$, $\ldots$, B_{p_1}) and current access B_c corresponds to one high-order access dependency. The corresponding access transition probability p_i is defined as the ratio between the number of pathlines traveling to this next block and the total number of pathlines traveling from the same historical blocks to all possible next blocks, formally

$$p_i = \frac{N_{\{B_p,\, B_c\} \to B_{f_i}}}{\sum_{k=1}^{n} N_{\{B_p,\, B_c\} \to B_{f_k}}},\ i = 1, 2, \ldots, n. \tag{4}$$

The computation of first-order and high-order access dependencies with a 2D example is shown in Figure 4.

In the preprocessing stage, the order is specified before tracing pathlines. To make full use of pathlines, we calculate all access dependencies with the order lower than the specified value, and we store them using the data structure implemented by Google protobuf library[1]. For example, if the order is set to m ($m > 1$), the access dependencies with order m, $m-1$, $\ldots$, 1 are all computed. With this approach, the access dependencies with different orders can be employed in real applications after only one round of computation of a specified order. In the rest of this paper, the high-order access dependencies mean this kind of "accumulated" access dependencies unless otherwise noted. An example is shown in Figure 4(d).

3.4 Parameter Evaluation

Two important parameters influence the effectiveness of the generated high-order access dependencies. One is the block size. A suitable block size setting is important, because it may affect the

[1]http://code.google.com/p/protobuf

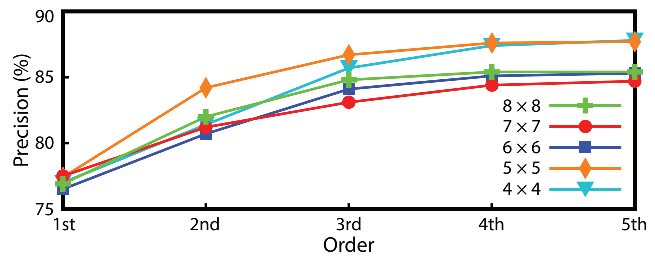


Figure 5: Prediction precision of next data accesses under different block size and order settings using simple 2D synthetic data.

prediction accuracy of high-order access dependencies and the cost in preprocessing. It also has strong effects on the number of I/O requests and the memory footprint in runtime applications. The second vital parameter is the order. Theoretically it would be better to use a higher order to achieve more accurate prediction of next data accesses. However, the computation of higher-order access dependencies will induce a larger cost in the preprocessing stage. On the other hand, the prediction accuracy do not always increase as the order increases. Figure 5 shows the precision of predicting next data accesses tested by simple 2D synthetic flow data with a resolution of 70×70 under different block size and order settings. The value of the prediction precision is calculated by dividing the total number of hit (i.e., correct prediction) blocks by the total number of blocks all particles have traveled when tracing pathlines randomly in the entire domain. With each block size setting, the precision increases as the order increases. But when the order is high enough, the precision becomes stable gradually because few particles have enough historical access information to contribute a higher precision. Therefore, it is important but also difficult to select a proper level of border for these parameters to balance the gain in prediction accuracy and the computational cost in real applications. This issue is discussed in Section 5.

4 Parallel Particle Tracing with High-Order Access Dependencies

We apply high-order access dependencies to parallel particle tracing to demonstrate the effectiveness of our method. The parallel computation framework is a modified version of Guo et al.'s work [13], which employs a two-layer cache scheme for task-parallel pathline advection in an out-of-core manner with limited computing resources. In our work, high-order data prefetching is employed to improve I/O efficiency. The high-order access dependencies are calculated only in the preprocessing stage. No access dependencies will be newly generated during runtime pathline advection, unlike Guo et al.'s work [13].

4.1 Structure of Data Entry

The high-order access dependencies constructed in the preprocessing stage are independent of the storage of data. When performing data prefetching, the data and the high-order access dependencies are both required. To avoid accessing them in different files, we integrate high-order access dependencies into data blocks during data partition in the application of parallel particle tracing.

In our method, each block corresponds to a data entry in the storage. The data entry maintains the information of the block itself and the high-order access dependencies associated with it. Formally, based on the traditional model of key-value store, the structure of each data entry is organized as `<key, value, dependencies>`, where `key` and `value` represent the spatiotemporal index and the actual data of corresponding block, respectively. In a data entry, `dependencies` consists of the high-order access dependencies associated with the corresponding block.

According to Equation 3, each mth-order access dependency ($m > 1$) in `dependencies` that is associated with the data en-

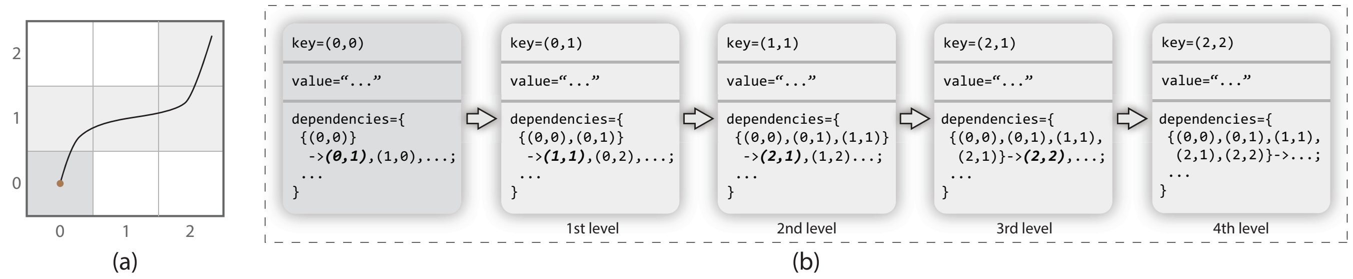


Figure 6: Recursive prefetching of multiple data entries at a time. In this example, we set the prefetching depth to four and prefetch only one data block at each prefetching depth level. When requesting block (0, 0), blocks (0, 1), (1, 1), (2, 1) and (2, 2) will be predicted one by one and are then prefetched together. At each prefetching depth level, the historical access information is updated.

try indexed by a *key* can be formulated as

$$\{key_{m-1},\ \ldots,\ key_2,\ key_1,\ key\} \xrightarrow{p} key', \tag{5}$$

where $key_{m-1},\ \ldots,\ key_2,\ key_1$ are the indices of the $m-1$ historical accessed data blocks, key' is the index of the data block possibly accessed next, and p is the access transition probability. Because the same historical access information may correspond to various probabilities of the next data access, the mth-order access dependency $(m > 1)$ can be further expressed as

$$\{key_{m-1},\ \ldots,\ key_2,\ key_1,\ key\} \rightarrow key'_1,\ key'_2,\ \ldots,\ key'_n, \tag{6}$$

where $key'_1,\ key'_2,\ \ldots,\ key'_n$ are the indices of the n probabilities of data blocks that may be accessed next. For convenience, these keys are sorted in descending order of access transition probability. In `dependencies` of each data entry, all associated access dependencies are structured by using an unordered map for fast retrieve.

4.2 Runtime Data Prefetching

In runtime applications, particles are assigned evenly to processes and are advected in a task-parallel mode. Each process maintains a cache with least-recently used (LRU) policy, and only `key` and `value` parts of data entry (i.e., the own information of data block) are kept in the cache. During pathline computation, the corresponding data is loaded on demand from the disk according to the block that the particle currently sits. When loading a data entry that is not in the cache, a prefetch request is issued. By using the exact historical data accesses of the particle to match the high-order access dependencies in `dependencies` of the loaded data entry, the indices of data blocks that have a higher access transition probability than others are found. These blocks are considered as the next possible accessed blocks and are then prefetched together at a time. In practice, a particle may not have enough historical data accesses to match the access dependencies with the specified order. In this case, we use the whole historical access information to do the matching. When the number of historical data accesses is equal to or larger than the order, we use the latest access information. The number of historical data accesses used is always controlled to be no more than the order. This is also the reason that we compute all access dependencies with the order lower than a specified value in the preprocessing stage.

We can specify the maximum depth of prefetching in order to prefetch multiple data blocks at a time for the next few advection steps rather than just one step in the immediate future. The data entries are prefetched recursively by a breath-first search according to the high-order access dependencies in `dependencies`. The prefetched result at the lower prefetching depth level is taken as a part of new historical access information to guide the prefetching at the higher prefetching depth level. On the other hand, according to Expression 6 that there may be more than one possible data blocks to be accessed next, we can also prefetch multiple data blocks with higher probability than others at each prefetching depth level. So a prefetch request actually consists of two parameters, i.e., the prefetching depth D and the number of blocks prefetched at each prefetching depth level N. The total number of blocks prefetched at a time for a prefetch request is the product of these two parameters, i.e., $D \times N$. Figure 6 shows an example of high-order data prefetching. In this example, only one block with highest probability is prefetched at each prefetching depth level. Theoretically, in order to ensure that the required data is hit, prefetching more blocks seems to be necessary. However, loading too much data at a time will incur a large number of I/O requests and decrease the data usage because of the finite-size memory in each process. In Section 5, we evaluate and discuss these parameters.

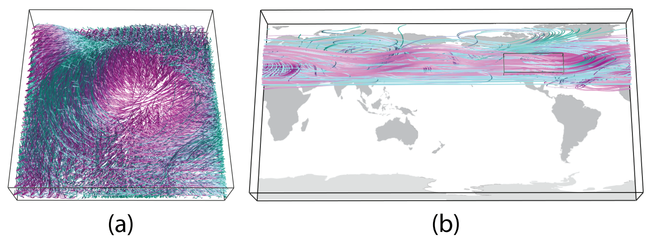


Figure 7: The (a) Isabel dataset and (b) GEOS-5 dataset used in our experiments.

5 Results

In this section, we present the evaluation results. Two datasets are used: Isabel (Figure 7(a)) and GEOS-5 (Figure 7(b)). The Isabel data is a hurricane simulation from the National Center for Atmospheric Research in the United States. The spatial resolution of this dataset is $500 \times 500 \times 100$, which covers a physical range of $2,139$ km $\times\ 2,004$ km $\times\ 19.8$ km. There are 48 timesteps, which are hourly saved in separated files. The GEOS-5 simulation is an atmospheric model from the NASA Goddard Space Flight Center. This model has a spatial resolution of $1^\circ \times 1.25^\circ$ with 72 vertical pressure levels ranging from 1 atm (near to the terrain surface) to 0.01 hPa (about 80 km). The output of this model consists of 24 monthly averaged simulation results from January 2000 to December 2001 and is stored in hybrid-sigma pressure grid.

We test the performance and scalability on a cluster with a parallel computing environment. The platform consists of eight computing nodes, one I/O node, and a RAID6 disk array that includes sixteen 2TB HDDs. Each computing node is equipped with two Intel Xeon E5520 CPUs (operating at 2.26GHz) and 48GB RAM. A Lustre parallel file system is shared by the I/O node to the computing nodes. The intercommunication link between nodes is InfiniBand QDR, which has a 40Gbps theoretical bandwidth.

5.1 Preprocessing

In the preprocessing stage, we calculate high-order access dependencies by employing a redesigned MapReduce-like parallel com-

Dataset	Dimension	Data Size	# Blocks	Storage Overhead				Time Cost			
				1st	2nd	4th	6th	1st	2nd	4th	6th
Isabel	$500 \times 500 \times 100 \times 48$	13.4GB	2,425,059	72MB	0.9GB	5.7GB	20GB	230s	515s	622s	736s
GEOS-5	$288 \times 181 \times 72 \times 24$	1.34GB	171,396	6MB	54MB	388MB	1.2GB	22s	38s	43s	51s

Table 1: Properties and preprocessing performance of two datasets used in our experiments. In the columns of storage overhead and time cost, the four numbers represent 1st-, 2nd-, 4th-, and 6th-order access dependencies, respectively.

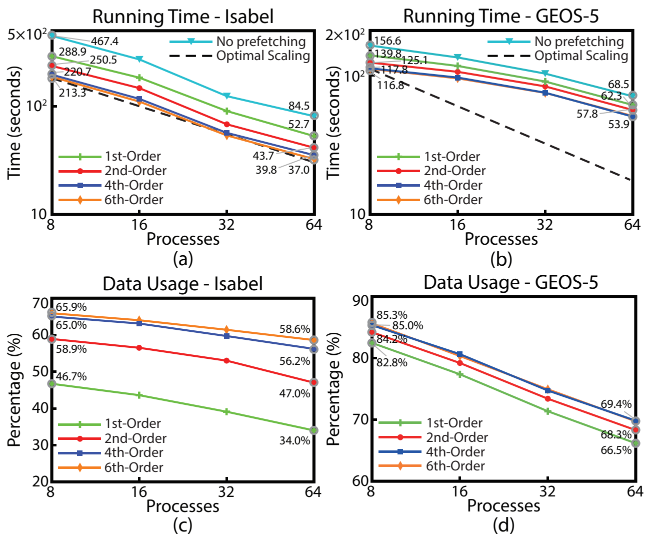


Figure 8: Running time and data usage of full-range analysis with different numbers of processes. Here the number of blocks prefetched at each prefetching depth level is initially specified as one.

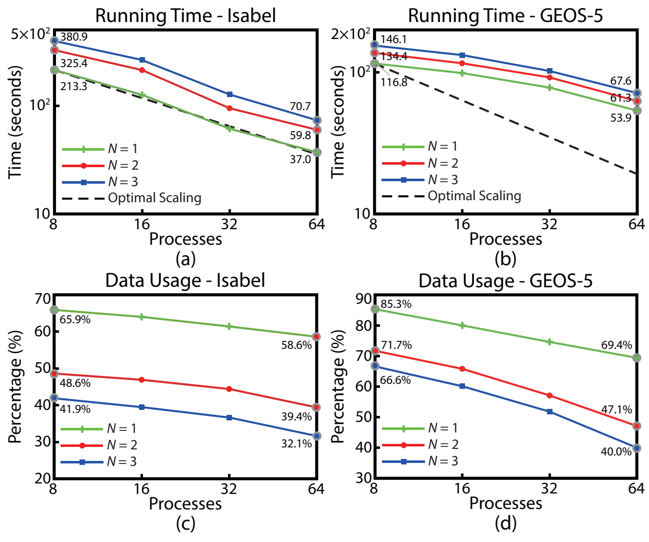


Figure 9: Running time and data usage of the sixth-order method in full-range analysis when the number of blocks prefetched at each prefetching depth level N is different.

puting framework [16]. For each dataset, a block is set to contain $8 \times 8 \times 8 \times 1$ cells ($9 \times 9 \times 9 \times 2$ grid size). Seeds are placed per $2 \times 2 \times 2 \times 1$ grids in each block. Table 1 shows the preprocessing performance of both datasets with 64 processes, including the storage overhead and time cost for the computation of access dependencies with different orders. The number of access dependencies directly relates to the number of blocks and the characteristics of flow field. As mentioned in Section 3.3, it is unnecessary to trace complete and accurate pathlines, which largely reduces the cost in the preprocessing stage. Note that the preprocessing is done only once for each dataset, and the generated high-order access dependencies are reusable for further various applications of pathline tracing.

5.2 Runtime Performance

In our experiments, both full-range and local-range analyses are performed on two datasets by using the parallel particle tracing framework. The cache in each process is set to 1GB, which can accommodate 61,624 Isabel blocks and 45,996 GEOS-5 blocks, respectively. For prefetching parameters, we choose the prefetching depth as two for Isabel and five for GEOS-5 because of different data properties, and the number of blocks prefetched at each prefetching depth level is evaluated. We explore the benefits of high-order (second-, fourth-, and sixth-order) access dependencies in parallel pathline computation and compare them with two baseline methods: the first-order method and the method without data prefetching.

5.2.1 Full-Range Analysis

The full-range analysis gives an overview distribution of the flow field. Because particles are uniformly seeded over the entire field, the I/O requests in the full-range analysis are always intensive. We place 6,250 seeds evenly in the domain and advect particles for each dataset. Initially, we specify the number of blocks prefetched at each prefetching depth level as one, which means the data block with highest probability will be selected.

The performance results under different numbers of processes are displayed in Figure 8(a) and 8(b). The scalability is shown to be good as the number of processes increases. For both datasets, we can clearly see that the running time decreases as the order increases, which demonstrates that the high-order method achieves better efficiency. We can also observe that the improvement rate of efficiency decreases gradually when the order changes to be higher. The result is compatible with the discussion in Section 3.4 that the prediction precision becomes stable as the order increases to a certain level. In this experiment, the optimal order for Isabel is sixth. For GEOS-5, the curves that represent the fourth- and sixth-order methods almost coincide with each other. Considering the preprocessing cost, the fourth-order method is determined to be optimal for this dataset.

For further exploration, we compare the usage of prefetched data between different-order methods. The data usage is the percentage of prefetched data that is really used. With higher data usage, less data will be loaded because the data is more likely to be used instead of being replaced by new data. Therefore, the data usage has great impact on the I/O cost. Figures 8(c) and 8(d) show that using higher-order access dependencies achieves higher prefetched data usage. As a result, for Isabel, compared with the first-order method that loads around 0.38 million data entries, the highest-order method saves about 15% data entries to read from the storage. Thus, our method can reduce I/O requests, and improve the efficiency of pathline computation.

From these results, we notice that the improvement of performance and data usage from the first-order method to higher-order method for GEOS-5 is not as significant as that for Isabel. In the best case, the sixth-order method saves 31.1% and 55.6% time for Isabel compared with the first-order method and the method without data prefetching, respectively. The corresponding time savings for GEOS-5 are just 16.5% and 25.4%. Actually, the effectiveness of high-order access dependencies largely depends on the variety

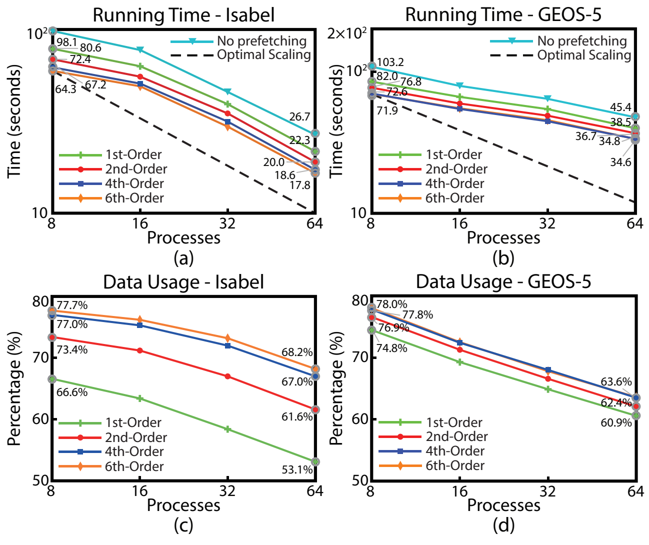


Figure 10: Running time and data usage of local-range analysis with different numbers of processes. Here the number of blocks prefetched at each prefetching depth level is initially specified as one.

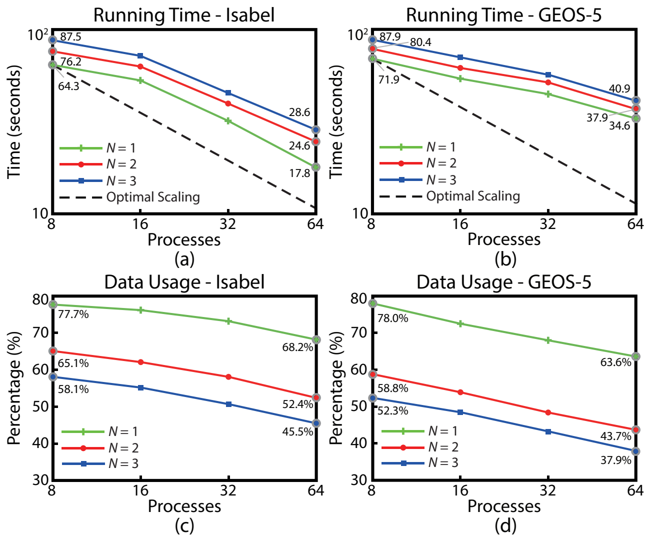


Figure 11: Running time and data usage of the sixth-order method in local-range analysis when the number of blocks prefetched at each prefetching depth level N is different.

of historical access information. If pathlines started from one data block always share common historical data accesses, there will be no distinction between the first-order and high-order access dependencies which are generated according to these pathlines. In our experiments, the number of high-order access dependencies generated from GEOS-5 is much less than that from Isabel. This is also the reason that the difference of data usage for GEOS-5 between first-order and higher-order methods is small, as shown in Figure 8(d).

We also compare the timings and data usage of prefetching different numbers of blocks at each prefetching depth level. Figure 9 shows the corresponding results of the sixth-order method. We can see that the performance and data usage of prefetching multiple blocks at each prefetching depth level are even worse than that of prefetching one block for both datasets. The reason is that at each prefetching depth level, only one data block is required for the next advection of a particle. Prefetching more blocks will induce larger number of I/O requests and much lower data usage, therefore reducing the efficiency of the pathline computation. So for the number of blocks prefetched at each prefetching depth level, the optimal value is one.

5.2.2 Local-Range Analysis

The local-range analysis provides a local visualization and analysis for flow fields, which is typically used in source-destination queries [16]. Different from full-range analysis, particles are required to be seeded in a specified spatiotemporal domain, thus increasing the probability of particles passing through the same data blocks during advection. In our experiment, we randomly select a region and trace 4,096 pathlines starting from this region for each dataset. We use the same prefetching strategy initially as in the full-range analysis.

Figure 10 shows the performance results with different numbers of processes. Similar to the results shown in Section 5.2.1, our method always achieves higher performance compared with the first-order method and the method without data prefetching. As shown in Figure 11, for local-range analysis, prefetching more than one blocks at each prefetching depth level also decreases the data usage and is infeasible for improving the efficiency. In this experiment, the optimal order is also sixth for Isabel and fourth for GEOS-5.

We can also see that the improvement of performance and data usage in Figures 10(a) and 10(c) is smaller than that in Figures 8(a) and 8(c). For example, with 8 processes, the full-range analysis of Isabel saves 26.2% time in the sixth-order method compared with the first-order method, whereas the local-range analysis saves only 20.2% time. The corresponding increases of data usage are 19.2% and 11.1%, respectively. Unlike full-range analysis that spreads particles over the entire domain, in local-range analysis particles are originated from a local region and advected in a relatively concentrated area. Usually, the data prefetched inaccurately by one particle is more likely to be used by other neighboring particles before being dumped. Hence, the efficiency of the high-order method in local-range analysis is not as significant as in full-range analysis. This is always reasonable (e.g., in the experiments of Isabel). However, for datasets with less dispersive flow characteristics to generate enough valuable high-order access dependencies, this difference between full-range and local-range analyses is not so apparent, as shown in Figures 10(b), 10(d) and Figures 8(b), 8(d) for GEOS-5.

5.3 Discussions

By taking the historical access information into account, high-order access dependencies enable more accurate predictions of data access patterns than the first-order access dependencies. With higher data locality achieved by high-order access dependencies, pathline computation for various visualization applications becomes more efficient. In our experiments, both full-range and local-range analyses can benefit from high-order access dependencies and achieve higher performance than the first-order method does.

In addition to the impact of the dataset itself on high-order access dependencies, there are two other limitations in our work. First, in order to obtain more comprehensive high-order access dependencies, the preprocessing stage requires tracing a large number of pathlines uniformly in the domain, which makes it time-consuming. For large-scale unsteady data, preprocessing will spend even more time. Therefore, better seeding strategies that sample smaller number of seeds while maintaining the effectiveness of high-order access dependencies are worth further study.

The second limitation concerns the selection of parameters. The block size and prefetching depth are two parameters that impact the performance of our experiments. As mentioned in Section 3.4, achieving optimal parameter settings is difficult. Because we focus on the analysis of fractal flow patterns, the block size should not be too large. However, the block size should also not be too small. A smaller block size setting leads to a larger number of blocks partitioned, which greatly increases the time and storage costs in the preprocessing stage. For the prefetching depth, a large value results in overwhelming the bandwidth [27, 13]. In addition, because data blocks are prefetched recursively based on previous predictions, the accuracy will decrease gradually as the prefetching depth increases. In our experiments, we empirically choose different parameter settings for the two datasets according to the data properties. We will study the parameters and their relationship to further improve the efficiency of high-order access dependencies.

6 Conclusions and Future Work

In this work, we exploit the advantages of high-order access dependencies for efficient unsteady flow visualization. The historical access information is integrated to support more accurate predictions of data access patterns, thus greatly improving the data locality. We apply high-order access dependencies to a parallel particle tracing framework with high-order data prefetching for pathline computation. Results demonstrate that our method achieves better efficiency in both full-range and local-range analyses than the first-order access dependencies-based method does.

We would like to extend our work to support irregular and unstructured grid data for more complex visualization tasks. Currently, the high-order access dependencies are directly associated in data blocks that are stored in random order. To achieve better I/O efficiency, we would like to reorganize the data according to the high-order access dependencies for sequential data access.

Acknowledgements

We thank the anonymous reviewers for their comments. This work is supported by NSFC No. 61170204 and partially by the Strategic Priority Research Program - Climate Change: Carbon Budget and Relevant Issues of the Chinese Academy of Sciences Grant No. XDA05040205. This work is also supported by the U.S. Department of Energy, Office of Science, under contract number DE-AC02-06CH11357.

References

[1] S. Byna, Y. Chen, X.-H. Sun, R. Thakur, and W. Gropp. Parallel I/O prefetching using MPI file caching and I/O signatures. In *SC08: Proceedings of the ACM/IEEE Conference on Supercomputing*, pages 44:1–12, 2008.

[2] B. Cabral and L. C. Leedom. Imaging vector fields using line integral convolution. In *Proceedings of SIGGRAPH 1993*, pages 263–270, 1993.

[3] C.-M. Chen, B. Nouanesengsy, T.-Y. Lee, and H.-W. Shen. Flow-guided file layout for out-of-core pathline computation. In *Proceedings of IEEE Symposium on Large Data Analysis and Visualization 2012*, pages 109–112, 2012.

[4] C.-M. Chen and H.-W. Shen. Graph-based seed scheduling for out-of-core FTLE and pathline computation. In *Proceedings of IEEE Symposium on Large Data Analysis and Visualization 2013*, pages 15–23, 2013.

[5] C.-M. Chen, L. Xu, T.-Y. Lee, and H.-W. Shen. A flow-guided file layout for out-of-core streamline computation. In *Proceedings of IEEE Pacific Visualization Symposium 2012*, pages 145–152, 2012.

[6] L. Chen and I. Fujishiro. Optimizing parallel performance of streamline visualization for large distributed flow datasets. In *Proceedings of IEEE Pacific Visualization Symposium 2008*, pages 87–94, 2008.

[7] Y. Chen, S. Byna, X.-H. Sun, R. Thakur, and W. Gropp. Hiding I/O latency with pre-execution prefetching for parallel applications. In *SC08: Proceedings of the ACM/IEEE Conference on Supercomputing*, pages 40:1–10, 2008.

[8] M. Edmunds, R. S. Laramee, G. Chen, N. Max, E. Zhang, and C. Ware. Surface-based flow visualization. *Computers & Graphics*, 36(8):974–990, 2012.

[9] C. Garth, F. Gerhardt, X. Tricoche, and H. Hagen. Efficient computation and visualization of coherent structures in fluid flow applications. *IEEE Computer Graphics and Applications*, 13(6):1464–1471, 2007.

[10] A. Gerndt, B. Hentschel, M. Wolter, T. Kuhlen, and C. H. Bischof. VIRACOCHA: An efficient parallelization framework for large-scale CFD post-processing in virtual environments. In *SC04: Proceedings of the ACM/IEEE Conference on Supercomputing*, pages 50:1–12, 2004.

[11] H. Guo, F. Hong, Q. Shu, J. Zhang, J. Huang, and X. Yuan. Scalable Lagrangian-based attribute space projection for multivariate unsteady flow data. In *Proceedings of IEEE Pacific Visualization Symposium 2014*, pages 33–40, 2014.

[12] H. Guo, X. Yuan, J. Huang, and X. Zhu. Coupled ensemble flow line advection and analysis. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2733–2742, 2013.

[13] H. Guo, J. Zhang, R. Liu, L. Liu, X. Yuan, J. Huang, X. Meng, and J. Pan. Advection-based sparse data management for visualizing unsteady flow. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2555–2564, 2014.

[14] G. Haller. Distinguished material surfaces and coherent structures in three-dimensional fluid flows. *Physica D: Nonlinear Phenomena*, 149(4):248–277, 2001.

[15] W. Kendall, J. Huang, T. Peterka, R. Latham, and R. B. Ross. Toward a general I/O layer for parallel-visualization applications. *IEEE Computer Graphics and Applications*, 31(6):6–10, 2011.

[16] W. Kendall, J. Wang, M. Allen, T. Peterka, J. Huang, and D. Erickson. Simplified parallel domain traversal. In *SC11: Proceedings of the ACM/IEEE Conference on Supercomputing*, pages 10:1–11, 2011.

[17] R. Laramee, H. Hauser, H. Doleisch, B. Vrolijk, F. Post, and D. Weiskopf. The state of the art in flow visualization: dense and texture-based techniques. *Computer Graphics Forum*, 23(2):203–222, 2004.

[18] K. Lu, H.-W. Shen, and T. Peterka. Scalable computation of stream surfaces on large scale vector fields. In *SC14: Proceedings of the ACM/IEEE Conference on Supercomputing*, pages 1008–1019, 2014.

[19] T. McLoughlin, R. Laramee, R. Peikert, F. Post, and M. Chen. Over two decades of integration-based, geometric flow visualization. *Computer Graphics Forum*, 29(6):1807–1829, 2010.

[20] C. Müller, D. Camp, B. Hentschel, and C. Garth. Distributed parallel particle advection using work requesting. In *Proceedings of IEEE Symposium on Large Data Analysis and Visualization 2013*, pages 1–6, 2013.

[21] B. Nouanesengsy, T.-Y. Lee, K. Lu, H.-W. Shen, and T. Peterka. Parallel particle advection and FTLE computation for time-varying flow fields. In *SC12: Proceedings of the ACM/IEEE Conference on Supercomputing*, pages 61:1–11, 2012.

[22] B. Nouanesengsy, T.-Y. Lee, and H.-W. Shen. Load-balanced parallel streamline generation on large scale vector fields. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):1785–1794, 2011.

[23] T. Peterka, R. B. Ross, B. Nouanesengsy, T.-Y. Lee, H.-W. Shen, W. Kendall, and J. Huang. A study of parallel particle tracing for steady-state and time-varying flow fields. In *IPDPS11: Proceedings of IEEE International Symposium on Parallel and Distributed Processing*, pages 580–591, 2011.

[24] F. Post, B. Vrolijk, H. Hauser, R. Laramee, and H. Doleisch. The state of the art in flow visualization: Feature extraction and tracking. *Computer Graphics Forum*, 22(4):1–17, 2003.

[25] A. E. Raftery. A model for high-order Markov chains. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 528–539, 1985.

[26] H.-W. Shen and D. L. Kao. UFLIC: a line integral convolution algorithm for visualizing unsteady flows. In *Proceedings of IEEE Visualization 1997*, pages 317–322, 1997.

[27] R. Sisneros, C. Jones, J. Huang, J. Gao, B.-H. Park, and N. F. Samatova. A multi-level cache model for run-time optimization of remote visualization. *IEEE Transactions on Visualization and Computer Graphics*, 13(5):991–1003, 2007.

[28] H. Yu, C. Wang, and K.-L. Ma. Parallel hierarchical visualization of large time-varying 3D vector fields. In *SC07: Proceedings of the ACM/IEEE Conference on Supercomputing*, pages 24:1–12, 2007.