Noname manuscript No.
(will be inserted by the editor)

A Survey of Parallel Particle Tracing Algorithms in Flow Visualization

Jiang Zhang · Xiaoru Yuan

Received: date / Accepted: date

Abstract Particle tracing is a very important method in flow field data visualization and analysis. By placing particle seeds in the flow domain and tracing the trajectory of each particle, users can explore and analyze the hidden local or global features in the flow field. However, particle tracing is computational complex and intensive. As the size and complexity of data continue to increase, tracing particles efficiently through parallel computing for flow field visualization and analysis becomes a popular trend in recent years. In this paper, we summarize the state-of-the-art researches on parallel particle tracing algorithms in flow visualization. According to the problems and challenges in the parallelization of particle tracing, methods are divided into three categories, including task parallelism, data parallelism, and hybrid methods that combine task and data parallelism. We show the pros and cons of these algorithms and their relationships for summarization. At the end of this survey, we also look into the research trends and discuss the remaining challenges for the possible future work.

Keywords Flow visualization · Large-scale data · Parallel particle tracing

1 Introduction

Particle tracing is a fundamental method for the visualization and analysis of flow field data. It is widely used in many flow visualization methods that are based on texture [Laramee et al., 2004], geometry [McLoughlin et al., 2010], and feature extraction [Post et al., 2003]. The idea is to release a certain amount of massless particles at specified positions in the flow field domain and then track the trajectories of these particles in the domain. In steady and time-varying flow fields, the generated trajectories are called streamlines and pathlines, respectively. These trajectories visually reflect the local or global changes of the flow fields and help users extract important features which are inherently hidden in the flow fields. Therefore, particle tracing has become a research focus in flow visualization and analysis.

The computation of particle tracing is complex and intensive. The final result of trajectories is often much larger than the original data itself in certain applications that require tracing massive densely-seeded particles (e.g., from each spatio-temporal sampling point in the flow field) with a large number of integration steps. Besides, due to the rapid development of modern science and computing technology, the data scale of flow field data is growing explosively. The flow field itself and the related analysis tasks are becoming more and more complicated. In particular, in some cases based on particle tracing, such as line integral convolution [Cabral and Leedom, 1993, Shen and Kao, 1997], FTLE computation for LCS extraction [Haller, 2001, Garth et al., 2007], flow surface calculations [Edmunds et al., 2012], and source-destination analysis [Kendall et al., 2011], complex calculations of particle tracing are often required. In practice, it is difficult for a single processor to meet needs of this large-scale particle tracing calculation due to the constraints of the memory and computing power. The traditional out-of-core technology used

Jiang Zhang, Xiaoru Yuan
Key Laboratory of Machine Perception (Ministry of Education), and School of EECS, Peking University
E-mail: xiaoru.yuan@pku.edu.cn

by earlier researchers [Silva et al., 2002,Bruckschen et al., 2001,Ellsworth et al., 2004] is also becoming less applicable because of their I/O bottlenecks when dealing with large-scale flow field data.

With the development of high-performance computing technology, researchers began to utilize parallel computing clusters or supercomputers with their strong computing power and collaboration capabilities to trace particles, and it has become a popular trend recently. However, the parallelization of particle tracing poses new challenges in scalability due to the issues in I/O, communication, load balance, and so on. This is also a classical and hard problem in visualization and other research areas. So how to design proper parallel algorithms to efficiently trace particles in flow visualization is attracting the attention of researchers.

1.1 Problems and Challenges

The problems and challenges in parallel particle tracing mainly come from two aspects, i.e., the flow field and the particle seeds initially specified.

Flow field There are two main problems brought by the flow field itself, including the data scale and the complexity of the flow field. As introduced before, the flow field data generated from recent numerical simulations is always with large scale, causing the entire data cannot fit into the main memory. Although we can load the data on demand, it brings more I/O operations and thus heavier I/O burden. On the other hand, the vector field of the flow data can be very complex. This will greatly impact the tracing process because particle tracing is highly dependent on the flow field. Due to some significant flow characteristics, such as the critical points or vortices that attract particles to be traced with much more integration steps in their vicinity, the computation load will not be uniform over the flow data domain. Moreover, for certain flow visualization tasks such as streak surface computation, the working set is only a very small portion of the entire data [Guo et al., 2014b]. These problems pose the challenges about how to efficiently distribute the data to processes.

Particle seeds The problems of initial particle seeds also come from two aspects: the size of the seeds and the distribution of the seeds over the flow domain. A larger seed size increases the probability of unbalanced workload distribution, especially in complex flow fields. The seed distribution indicates the seeding strategy in particle tracing based flow visualizations. In practice, some applications require densely seeding in the entire data domain to explore the overview features of the flow field, while some others require placing seeds in some specific local regions for visualizing local flow features. Different seeding strategies also have different needs of the amount of data accesses. More importantly, each particle can be considered as a computation task in particle tracing, which means the particles directly relate to the workload of processes. So it is still challenging to design efficient parallel algorithms for the workload distribution.

The flow field and particle seeds actually are related tightly because the computation of particle tracing is conducted within the flow data domain. When we develop the parallel strategies of particle tracing, these two aspects should be seriously considered. We give the classification of existing algorithms in the followings.

1.2 Algorithm Classification

Based on the problems and challenges described above, we take the flow field and the initial particle seeds as the objects that need to be parallelized. The rationale is to decrease the granularity of the entire procedure of particle tracing by distributing it on multiple processes. Specially, we follow the classification mentioned in previous work [Pugmire et al., 2009], and divide the parallel particle tracing algorithms into three categories, namely

- **Data Parallelism:** The flow field is statically divided into blocks, and these blocks are then assigned to different processes, as shown in the left of Figure 1.
- **Task Parallelism:** The particle seeds are statically divided into sets, and these seed sets are then assigned to different processes, as shown in the right of Figure 1.
- **Hybrid Parallelism:** The combination of the above two parallel methods, i.e., dividing both flow field and particle seeds and then assigning them to different processes, to further improve the computational efficiency of parallel particle tracing.

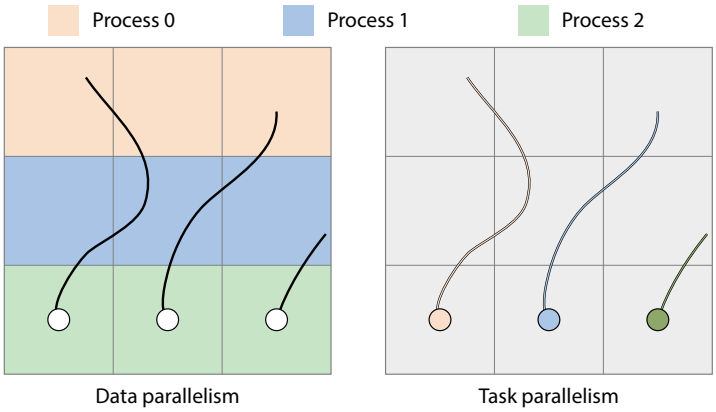

Fig. 1 Illustration of data-parallel and task-parallel particle tracing algorithms. The former one is the parallelism over the data, while the latter one is the parallelism over the particle seeds.

For the parallelization of particle tracing, the scale and complexity of the flow fields, and the size and distribution of the initial particle seeds specified in different applications, can directly affect the efficiency of I/O operations, memory, and communication in the parallel programs. It poses a huge challenge for the design of scalable algorithms. Therefore, researchers have proposed many algorithms to solve the problems about how to make reasonable use of computing resources to improve the scalability of the parallel particle tracing computation.

Table 1 Categories and requirements of several representative parallel particle tracing algorithms (including data parallelism, task parallelism, and hybrid methods that combines data and task parallelism). The plus symbol (+) indicates that the corresponding algorithm is in a certain kind of categories or meets a certain kind of requirements, while the minus symbol (-) means that it does not correspond to a certain kind of categories or requirements.

Representative Algorithms	Categories				Requirements						
	Data Parallelism		Task Parallelism		Pre-analysis			Run time			
	Static Load Balancing	Dynamic Load Balancing	Dynamic Load Balancing	Data Prefetching	Data Pre-processing	Heuristics on Flow Features	Assumptions on seed distribution	Particle Exchange	Data Movement among Proc.	On Demand Data Loading	Scheduler Processes
[Peterka et al., 2011]: round-robin	+	-	-	-	-	-	-	+	-	-	-
[Nouanesengsy et al., 2012]	+	-	-	-	-	-	-	+	-	+	-
[Nouanesengsy et al., 2011]	+	-	-	-	+	-	+	+	-	-	-
[Chen and Fujishiro, 2008]	+	-	-	-	+	+	+	+	-	-	-
[Yu et al., 2007]	+	-	-	-	+	+	+	+	-	-	-
[Peterka et al., 2011]: data repartitioning	-	+	-	-	-	-	-	+	+	+	-
[Zhang et al., 2018]	-	-	+	-	-	-	-	+	-	-	-
[Dinan et al., 2009]	-	-	+	-	-	-	-	+	-	+	+
[Müller et al., 2013]	-	-	+	-	-	-	-	+	-	+	+
[Chen et al., 2012, Chen and Shen, 2013]	-	-	-	+	+	-	-	-	-	+	-
[Guo et al., 2014b]	-	-	-	+	-	-	-	-	+	+	-
[Zhang et al., 2016]	-	-	-	+	+	-	-	-	+	+	-
[Lu et al., 2014]	-	+	+	-	-	-	-	+	+	-	-
[Pugmire et al., 2009]	-	+	+	-	-	-	-	+	-	+	+
[Kendall et al., 2011]	+	-	+	-	-	-	-	+	-	-	-

For the related algorithms in parallel particle tracing, each one has its focus such as the communication, I/O efficiency, and the load balance. Based on the different focuses, each algorithm also has

different requirements when initializing. The requirements include two aspects, i.e., during pre-analysis and run time, which can be used to reveal the characteristics of the methods and show their similarity and difference. Table 1 lists some representative parallel particle tracing algorithms and the corresponding requirements for comparison. In the following sections, we will follow this to summarize the related parallel particle tracing algorithms proposed by the researchers recently. The pros and cons of these algorithms will be described in details.

2 Data Parallelism

Data-parallel methods are related to the static data partitioning and allocation strategy. It requires the entire flow field data to be divided into several data blocks. These blocks are then allocated to each process so that each process gets a portion of data. During the run-time particle tracing, when a particle leaves the data block where the process is responsible for, it will be sent to the process corresponding to the target data block to continue the advection. In general, this method assumes that the data can be loaded into memory at one time statically, and I/O operations are not required during the computation phase, so it has minimal I/O overhead. However, because of the need to frequently exchange particles between processes, the cost of intensive communications has a great impact on the performance.

Due to the unpredictability of particle trajectories and the complexity of the flow field, it is difficult to ensure that each process has the same workload after data allocation. In particular, when the flow field contains relatively strong local features (such as critical points or vortices, etc.), or when the seeds are densely distributed only in some local areas, the workload between processes may be seriously imbalanced and the scalability may be reduced. The data-parallel methods aim to solve these problems. These methods mainly include two types: static load balancing, i.e., the static partitioning and allocation of data according to certain rules; dynamic load balancing, i.e., dynamic adjustment of data allocation during particle tracing.

2.1 Static Load Balancing

In data-parallel methods, in order to make the processes have equal workload during particle tracing, it often requires to design effective data partitioning and allocation strategy initially. Because the particles are traced within the data blocks, each block actually records a number of workloads. We need to partition the data into blocks, so that each process can be allocated for one or multiple blocks with equal workloads. Therefore, the rule of data partitioning is critical for static load balancing in data parallelism. We divide the existing methods into two categories, namely, regular data repartitioning based on the rule of regular grids and irregular data partitioning based on the characteristics of the flow fields.

Regular Data Partitioning

Partitioning the data in regular grids is a common strategy. The entire flow field is partitioned evenly into several data blocks, with each block containing equal extent in each dimension. The granularity of partitioning is an important parameter that affects the performance and is always determined in combination with the particle distribution.

Suppose there are n processes and m blocks, a relatively straightforward approach is using round-robin assignment [Peterka et al., 2011], i.e., process i is assigned blocks $\{b_i, b_{i+m/n}, b_{i+2*m/n}, \ldots\}$. This round-robin method ensures that each process is assigned equal (or near equal) number of blocks in different locations of the flow fields. It is easy to use and does not require any additional computational overhead and prior knowledge. So this method is used very commonly in practical applications. The effectiveness of this method has been demonstrated in a previous study [Peterka et al., 2011]. Typical application of round-robin assignment is in the FTLE computation by Nouanesengsy et al. [Nouanesengsy et al., 2012]. In their method, the data is divided and assigned to different groups of processes by round-robin to support the computation of FTLE in a pipelined manner, as shown in Figure 2. Round-robin introduces a certain degree of randomness, increasing the probability that each process holds the blocks with relative even workloads. It is very straight-forward and does not require any pre-analysis and additional run-time

processing. But this kind of data allocation method is rough, in which the improvement of load balance is limited.

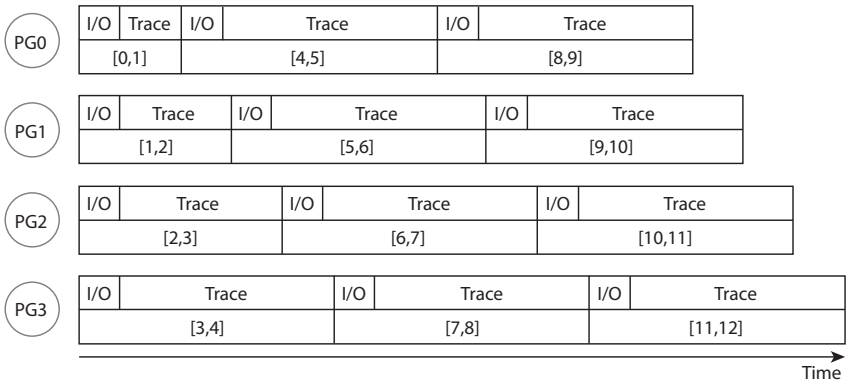

Fig. 2 A pipelined parallel particle tracing method for FTLE computation [Nouanesengsy et al., 2012]. Different time intervals are loaded by different process groups based on the round-robin assignment.

In order to achieve balanced workload, Nouanesengsy et al. [Nouanesengsy et al., 2011] present a workload-aware method to estimate the workload of each data block statically and assign blocks to the processes accordingly. In their method, the access dependencies between each pair of blocks are first computed in the preprocessing stage, according to which the workload of each block during run time is predicted given the initial particle seeds. To gain more even distribution of workloads on processes, a block can be assigned to more than one process. As shown in Figure 3, each process is only responsible for one portion of tracing workloads in this block, making the total workload of each process equal. However, this method is only suitable for the case that requires to trace a large number of seeds. When the number of particle seeds is small, it is not cost-effective due to the relative high overhead of preprocessing. Besides, the initial distribution of particle seeds is also involved in the estimation of workloads, which is not scalable for the generalization of this method.

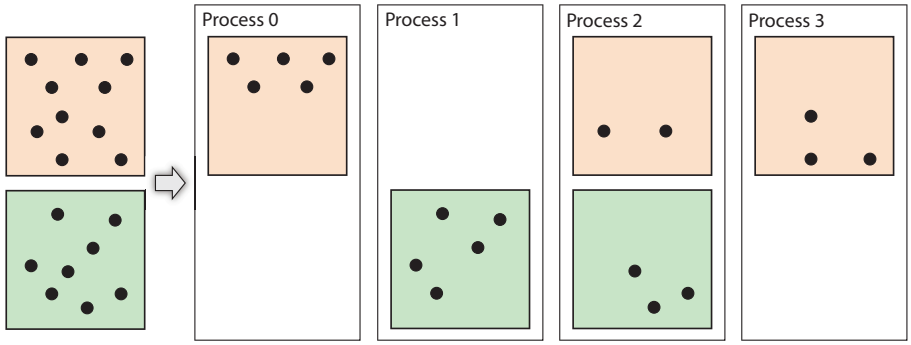

Fig. 3 A workload-aware method to estimate the workload of each data block for load-balanced streamline computation [Nouanesengsy et al., 2011]. In this example, the two data blocks are duplicated to ensure balanced partitioning.

Irregular Data Partitioning

Regular data partitioning is relatively simple but does not consider the characteristics of the flow field itself. So it is a coarse partitioning method. For some flow fields that have very obvious features, partitioning the data irregularly based on the flow characteristics is more efficient. Figure 4 gives an example to show this kind of partitioning. In this figure, the small circles indicate seed points, and the black solid lines indicate the flow direction of the flow field. The figures with red dashed lines are the results of regular partitioning that causes the particles frequently entering the data blocks that are handled by

different processes, resulting in additional communication overhead. On the other hand, from the distribution of particle seeds in the figures, some processes will be idle while other processes are very busy with tracing particles, i.e., the workloads are unbalanced. Therefore, under this circumstance we need to take into account the nature of the flow field, putting forward another kind of partitioning strategy, namely irregular data repartitioning.

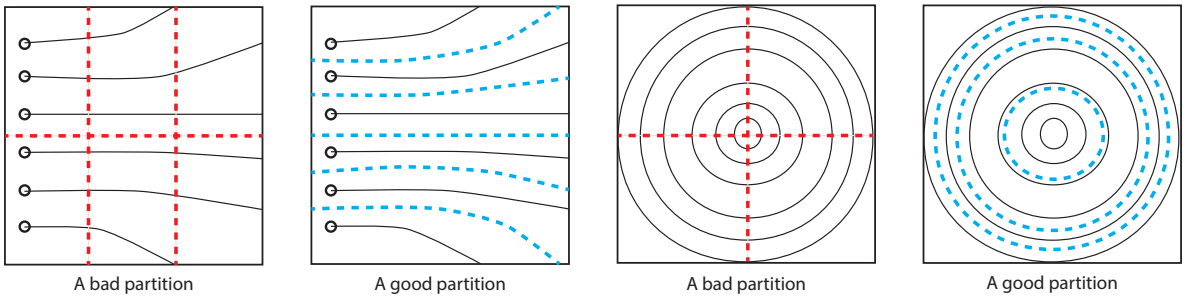

Fig. 4 Comparison between the regular data partitioning and flow direction based partitioning [Chen and Fujishiro, 2008]. The latter one reduces the communication and synchronization.

The figures with blue dashed lines in Figure 4 show a data partitioning method according to the flow direction [Chen and Fujishiro, 2008]. This method takes the distribution of seeds and the local characteristics of the flow field (such as vortices) into consideration. It tries to ensure each particle is always traced by one process or less number of processes, which will reduce the heavy overhead caused by communication and process synchronization. Another irregular data partitioning is using adaptive method of hierarchical clustering based on flow features [Yu et al., 2007]. As shown in Figure 5, the unit grids in the data are first considered as the smallest clusters. Adjacent clusters that have similar flow pattern will be merged in pairs from bottom to up, which forms a hierarchical structure of the binary tree. The depth of the binary tree actually reflects the granularity of the flow features. Then in the data partitioning stage, we can flexibly select clusters (shown as the black dots in Figure 5) from different levels in the binary tree and estimate the workload of each corresponding region for the data assignment. However, these two methods both rely on the analysis of flow features, which is also a new challenge due to the difficulty in feature definition in distributed environments. The consequent data preprocessing and the involvement of seed distribution also bring additional computational overhead.

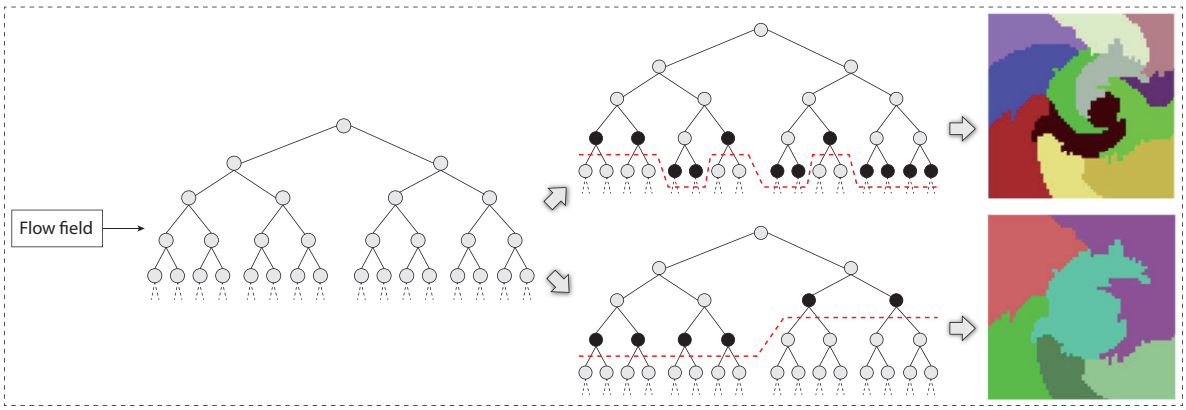

Fig. 5 A data partitioning method based on adaptive hierarchical clustering [Yu et al., 2007].

Summary

The static load balancing methods can effectively balance the workload of the process and reduce the communication overhead. However, apart from the round-robin method, other methods including both

partitioning in regular grids and based on flow features always require a data preprocessing stage before partitioning the flow data and make some assumptions on the seed distribution. The pre-analysis of the flow data induces additional costs and impacts the overall performance. In the cases that we just need to conduct some lightweight visualization tasks, it is not recommendable to use the algorithms with complex pre-analysis because the gains do not make up for the losses. But in practice, we should also see that for one flow data in some applications, this pre-analysis stage only needs to be done once in most cases and the result of pre-analysis can be used for subsequent particle tracing with arbitrary distribution of initial particles. This can also make the pre-analysis worthwhile.

2.2 Dynamic Load Balancing

In order to overcome the cons of static data allocation methods, researchers have also proposed some dynamic load balancing methods. In contrast to the static methods, the dynamic ones aim at dynamically subdividing and then reassigning the data during the run-time particle tracing to achieve load balance, as shown in Figure 6.

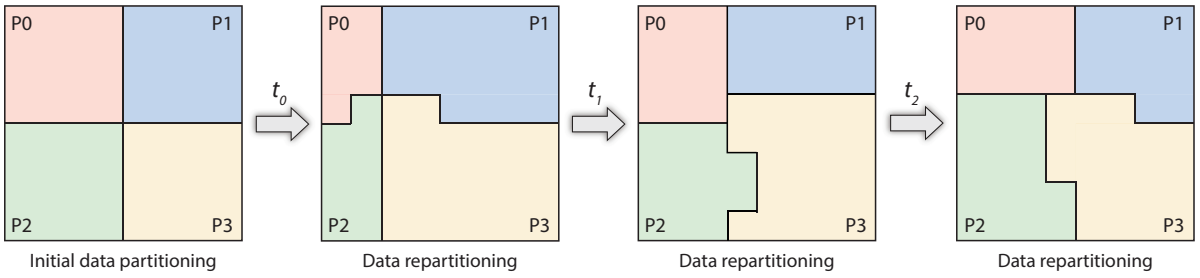

Fig. 6 Illustration of dynamic load balancing based on data repartitioning. At regular intervals during run time, the entire data is repartitioned and then reassigned to processes.

Dynamic Data Repartitioning

A typical method is proposed by Peterka et al. [Peterka et al., 2011], which dynamically repartitions the data using recursive coordinate bisection (RCB) algorithm [Berger and Bokhari, 1987]. In their work, the number of integration steps traced in each block is recorded as the historical workload of the corresponding block. The historical workload can be used to guide the repartitioning of data at regular intervals during the run-time particle tracing. The rationale of this method is that the workload of each block during adjacent time periods can be considered as similar. So after data repartitioning, each process is reassigned blocks with near equal workload which will be computed in the next period. Here the data repartitioning is performed by RCB method [Berger and Bokhari, 1987]. Besides RCB, similar algorithms include recursive inertial bisection (RIB) [Simon, 1991], space-filling curves [Pilkington and Baden, 1994], and partitioning based on graph topologies [Karypis and Kumar, 1996, Çatalyürek et al., 2007].

Summary

The advantage of the dynamic data allocation methods is that the data preprocessing stage is not required, and the initial data partitioning and allocation strategy does not affect the overall algorithms. Moreover, these methods do not use any heuristics on flow features and make any assumptions on the distribution of initial particle seeds. However, there are still some additional costs during run time, such as the calculation of the data repartitioning algorithm and the expensive communication of data movement between processes [Peterka et al., 2011]. As studied in Peterka et al. [Peterka et al., 2011], although the load is more balanced and the pure computation time of particle tracing is less, the overall execution time of dynamic data repartitioning method is still more than the static round-robin partitioning method. So the additional costs associated with the data repartitioning should be addressed. Recently, researches on this kind of dynamic data repartitioning algorithms in parallel particle tracing are still lacking.

3 Task Parallelism

The task-parallel particle tracing algorithms consider the computation of each particle as a task. The basic idea is to statically allocate all the particles so that each process can be assigned a certain number of particles. During the run-time particle tracing, each process is independently responsible for the calculation of the particles allocated. Unlike data parallelism, this approach does not emphasize the need to allocate data statically, but requires loading the data on demand during particle tracing. Because particles do not need to be exchanged between processes, task-parallel methods always require minimal communication.

In order to ensure load balance, the task-parallel algorithms initially assign an equal number of particles to each process. However, the completion time of different tasks is not equal and is difficult to predict. Even if we have even distribution of initial particle seeds, it is still difficult to ensure that each process has equal workload during the run-time particle tracing, resulting in load imbalance. So achieving even workloads statically in task parallelism is impracticable. To solve this problem, the current researches mainly use some dynamic workload adjustment (i.e., dynamic load balancing) methods. On the other hand, task-parallel particle tracing typically uses on-demand data loading strategies, which can cause frequent I/O operations. The initial particle seeds can be assigned based on the spatial proximity to reduce the I/O cost because the spatially close particles are more likely to travel in the same regions. However, the huge gap between I/O performance and computing power still makes the I/O time dominate the entire computation. Therefore, it is necessary to reduce the I/O overhead in task-parallel particle tracing. Addressing this problem, researchers usually use data prefetching technique, i.e., loading the required data in advance, so the I/O operation time can be "hidden" in the computation as much as possible.

3.1 Dynamic Load Balancing

In task parallelism, dynamic load balancing is a way to balance the workload of each process by dynamically redistributing tasks (i.e., particles) during the run-time particle tracing. Because the distribution of initial particle seeds is relatively random and their tracing time is always considered unpredictable, it is difficult to statically assign particles with equal workloads to processes. So we need to redistribute the unfinished particles during run time to re-balance the workload. In general, the tasks are transferred from the processes with heavy workloads to the processes with light workloads to minimize the idle time of processes. In existing methods, dynamic task redistribution is mainly done through work stealing/requesting and k-d tree decomposition.

Dynamic Task Redistribution

Work stealing is a common method to dynamically balance the workload. The principle of this method is that when a process completes all its tasks, it will steal some tasks from other busy processes to help them execute the tasks. This procedure continues unless all processes have finished their tasks. Specifically, each process maintains a task queue, and the tasks are fetched to be executed from the front of the queue. If one process finishes the tasks, it becomes a thief and selects a victim process according to some schemes, such as random victim selection [Blumofe, 1994], and then transfers some tasks from the tail of the victim's task queue if the victim's tasks are available. This method has been proved efficient [Dinan et al., 2009] and is applicable to the computation of parallel particle tracing. In Lu et al.'s work [Lu et al., 2014], they use work stealing to balance the tasks among processes during the stream surface computation.

Müller et al. [Müller et al., 2013] propose a similar method called work requesting to achieve dynamic load balance in parallel particle tracing. The difference comes from the activity of the victim process in the procedure of transferring the tasks. In work requesting the thief process will first ask the victim process for tasks and the victim process is actively involved in providing tasks, while in work stealing the thief process directly accesses the task queue of the victim process and the victim process is not involved. Work requesting brings more communication, but it is easier to implement, especially in distributed memory architectures.

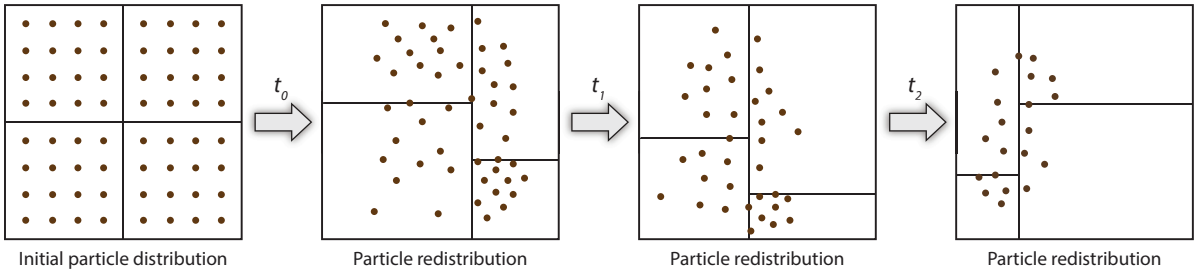

Fig. 7 Illustration of dynamic particle redistribution through the general k-d tree decomposition [Zhang et al., 2018]. At regular intervals during run time, the unfinished particles is repartitioned and then reassigned to processes.

Besides work stealing/requesting, another general idea to redistribute particles dynamically is through the k-d tree decomposition. As shown in Figure 7, at regular time intervals, the unfinished particles are split into several parts so that each part has equal number of particles. Each process is then assigned one part of particles and continue to trace these particles. This k-d tree decomposition has been successfully applied in load balancing for Delaunay tessellation [Morozov and Peterka, 2016]. However, this method requires full data duplication over all the processes because the the redistributed particles may locate anywhere in the data domain. Furthermore, Zhang et al. [Zhang et al., 2018] propose a novel constrained k-d tree decomposition to overcome this problem. In their method, the entire data domain is initially partitioned into several non-overlapping, equal-sized, and axis-aligned blocks, with each assigned to a process. Each block is then extruded to overlap with its neighboring blocks under the memory limit of the corresponding process. The extruded data parts are called the ghost layers of the data blocks. A 3D illustration of this method is shown in Figure 8. When performing the k-d tree decomposition to redistribute the unfinished particles during run time, the splitting planes are limited in the overlapped regions made up of the ghost layers. This ensures the particles after redistribution are bounded in the corresponding blocks. The constrained k-d tree decomposition method does not require any additional costs, except for the communication brought by the decomposition for particle redistribution. The performance study shows it achieves great load balance and scalability on various flow visualization and analysis problems.

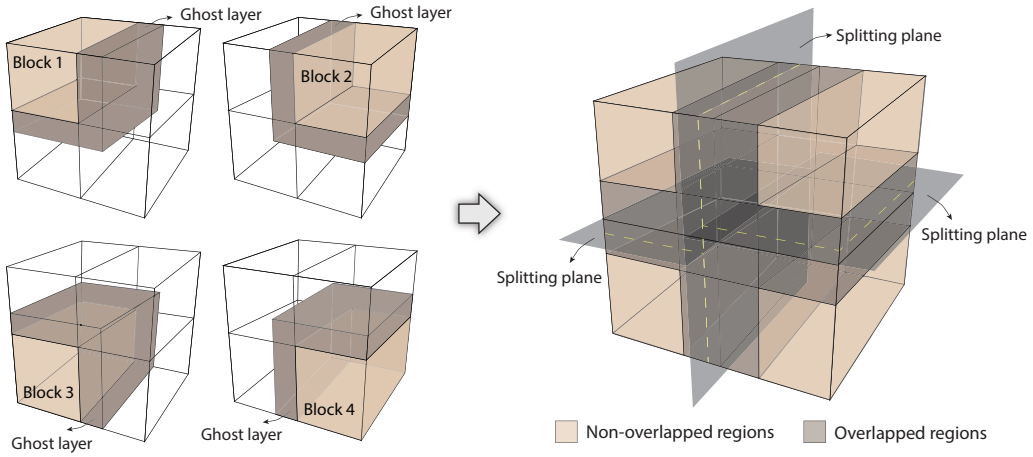

Fig. 8 3D illustration of the constrained k-d tree decomposition [Zhang et al., 2018]. The splitting planes are constrained in the overlapped regions of the ghost layers. The ghost layers come from the extruded parts on the initial data blocks under the memory limit.

Summary

The work stealing and work requesting can achieve better efficiency than the naive task-parallel methods because they do not require any data preprocessing, heuristics on flow features, and involvement of

initial seed distribution in algorithms. The data is loaded on demand, indicating no data movement among processes. However, the on-demand data loading strategy is I/O inefficient. It also causes that data blocks may be loaded by multiple processes because different processes may trace particles in the same blocks. This will further increase the I/O overhead. On the other hand, these methods always require dedicated processes or threads to schedule the task transfer, which brings huge communication overhead. Therefore, the scalability of these methods is limited to some extent. The constrained k-d tree decomposition requires no data pre-analysis and additional run-time processing. However, this method is built upon the assumption that the ideal balance can be achieved through distributing equal number of particles over processes. Actually the load balance is directly determined by the number of integration steps traced by particles, but not the number of particles. This means that it cannot guarantee to achieve optimal load balance. So it is still very necessary to develop more efficient task redistribution methods.

3.2 Data Prefetching

As mentioned earlier, the I/O issue is one of the main bottlenecks that constrains the performance of parallel particle tracing, especially in task parallelism that loads data on demand. Due to the rapid development of technology, the computing power of processors has improved greatly. But the I/O speed is far behind the computation, which becomes a heavy burden on the performance of task-parallel particle tracing algorithms. Even we use parallel file systems, such as PVFS [Carns et al., 2000], Lustre [Inc., 2002], and GPFS [Schmuck and Haskin, 2002], the efficiency is comparably low when dealing with large-scale, small, and discontinuous I/O requests in parallel particle tracing. So in order to improve the performance of data accesses, data prefetching is often used in task-parallel methods.

The idea of data prefetching is that when loading the data that is required for immediate particle advection, it will also fetch the data that may be accessed in the near future into memory. This technique decreases the waiting time due to the absence of required data and thus improves the efficiency of particle tracing. The effectiveness of data prefetching depends on the accuracy of predictions on the data accesses. In the applications based on parallel particle tracing, the I/O access patterns are often recorded to provide the basis for data prefetching. As early as 2005, Rhodes et al. [Rhodes et al., 2005] take the access patterns as priori knowledge and use caching and prefetching mechanisms to dynamically improve the I/O performance. This idea is also applied in the unstructured grid data by combining with data reorganization [Akande and Rhodes, 2013]. In the following, we summarize the related work in recent years with different categories of I/O access patterns, including access dependency graph, high-order access dependencies, and other more complex I/O patterns.

Access Dependency Graph

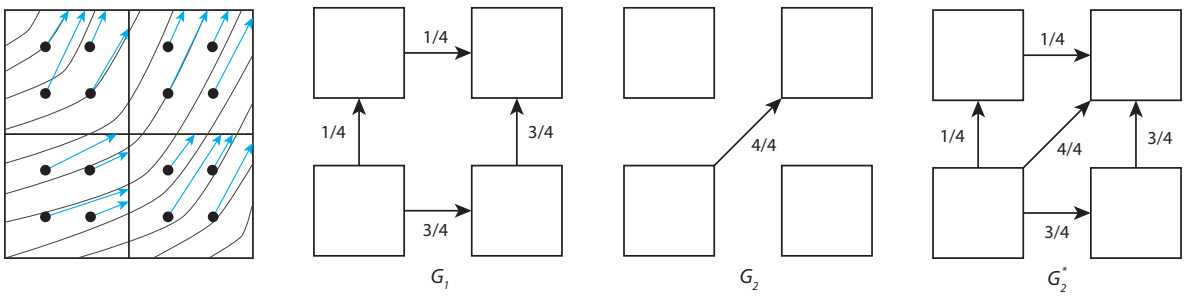

Fig. 9 The access dependency graph (ADG) in particle tracing [Chen et al., 2012b]. This example shows the ADGs that particles travel from the originating blocks to the next blocks (G_1) and continuously to the third blocks (G_2), respectively. G_2^* is the mixture of these two graphs.

Recently, targeting to the computation of streamlines, pathlines, and the related FTLE computation, the visualization group from the Ohio State University proposes a series of algorithms based on access dependencies among data blocks for optimizing the file layout to improve the efficiency [Chen et al., 2012b,

Chen et al., 2012a,Chen and Shen, 2013]. In the preprocessing stage of their methods, they build an access dependency graph (ADG) that indicates the data access dependencies. As shown in Figure 9, each node represents a data block, and the directed edges between two nodes represent the access transition from one block to the other block. Each edge is with a weight recording the access transition probability, i.e., the probability that a particle in one block travels to the other block. According to the ADG, they reorganize the data for an optimal file layout. This makes it possible to precisely prefetch more blocks that may be accessed in the next few integration steps. However, because the accuracy of ADG and the preprocessing cost are sensitive to the block size and seeding strategy, these methods are more susceptible to the parameters.

Depending on the data access dependencies, Guo et al. [Guo et al., 2014b] propose an advection-based sparse data management for unsteady flow visualization. Addressing the problem that coarse-grained partitioning causes low efficiency of data accesses, their method partitions the data into blocklets with fine granularity. The data blocklets are used for on-demand data accesses with parallel key-value store. Figure 10 shows the data structures in their sparse data management. During the run-time particle tracing, the access dependency graph among blocklets is built on-the-fly and is then used to guide the data prefetching precisely for the following tracing. In order to improve the I/O efficiency, the required data is fetched from first the local, and then the parallel key-value store, and finally the file system through MPI/IO. Results show that this method is both memory- and I/O efficient in some application cases, including source-destination queries, FTLE computation, and streak surface computation.

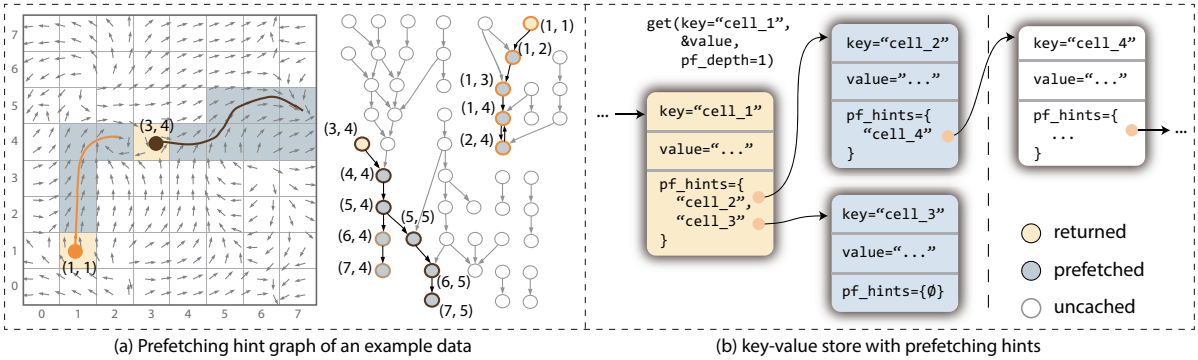

Fig. 10 The data structures in the advection-based sparse data management, including (a) the prefetching hint graph and (b) the key-value store [Guo et al., 2014b].

High-Order Access Dependencies

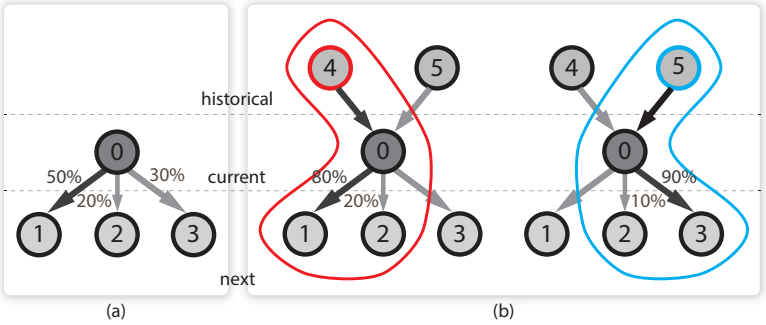

Fig. 11 Comparison between the first-order (a) and second-order (b) access dependencies [Zhang et al., 2016]. By integrating with the historical data accesses, the prediction of the next possible data access is more accurate.

Data prefetching in the above methods is just based on the access transition between pairs of data blocks, which we called it the first-order access dependencies. In fact, there exist more sophisticated

access patterns that can be used to provide more accurate prediction of data accesses in particle tracing. Zhang et al. [Zhang et al., 2016] present an idea of high-order access transitions, which integrates the historical data access information to calculate the access dependencies. Different from the methods based on the first-order access dependencies [Gerndt et al., 2004, Chen et al., 2012b, Chen et al., 2012a, Chen and Shen, 2013, Guo et al., 2014b], in Zhang et al.'s work [Zhang et al., 2016], the prediction of data blocks that may be accessed next is not only related to the current block, but also depends on the sequence of several blocks that has been accessed, as shown in Figure 11. They further apply the high-order access dependencies in a parallel particle tracing framework [Guo et al., 2014b] to enable high-order data prefetching. Experiments show that this method can greatly improve the usage of prefetched data, and thus achieve higher efficiency of particle tracing compared with the first-order methods.

Other I/O Access Patterns

In addition to these access dependencies between data blocks, some more complex I/O access patterns, such as spatial patterns, temporal intervals, and repetitions, can also be detected and extracted. Byna et al. [Byna et al., 2008] record these patterns as I/O signatures. Combined with the exact data accesses in the run-time computation, the I/O signatures are used to provide the guidance of data prefetching in MPI-IO operations. When the I/O patterns are very random or unpredictable, speculative prefetching can always predict the I/O accesses reliably. A good example is pre-execution prefetching [Chen et al., 2008], which can overlap the computation and I/O accesses to hide the I/O latency, thus improving the I/O performance of parallel applications. These techniques are also applicable to parallel particle tracing for improving the computation efficiency.

Summary

Data prefetching can improve the data locality and thus achieve higher I/O efficiency during particle tracing. However, in order to obtain accurate prediction for prefetching, it is often necessary to perform complex preprocessing on the data, such as extracting the I/O patterns (e.g. data access dependencies). This incurs additional computational burden on parallel particle tracing. Moreover, the complexity of data preprocessing is often proportional to the size and complexity of the flow fields, which limits the use of this technique in large-scale flow data. Besides, data prefetching is always related to the visualizations in an out-of-core manner. So it is more suitable for the applications using desktop computers [Chen et al., 2012b, Chen et al., 2012a, Chen and Shen, 2013] or the supercomputers with limited computing resources [Guo et al., 2014b, Zhang et al., 2016].

4 Hybrid Parallelism

The data- and task-parallel methods improve the scalability of parallel particle tracing at different angles, but they also have their own limitations in some aspects, as described before and shown in Table 1. In recent years, visualization researchers have combined these two strategies, and they have proposed a series of hybrid strategies that combine data and task parallelism to further improve the performance of parallel particle tracing. The integration of static/dynamic data (re)partitioning and dynamic task redistribution is always used in the general hybrid-parallel methods and in the specific applications. We introduce the hybrid parallelism in the followings.

Hybrid Strategies

Addressing the challenges in parallel particle tracing, Pugmire et al. [Pugmire et al., 2009] have analyzed the characteristics of data and task parallelism in their work. They further propose a hybrid scheduling method based on a master-slave mode to dynamically adjust the data and tasks for load balancing. In this method, the processes are divided into multiple groups, with each group containing a master process and several slave processes. The data is decomposed statically and loaded on demand. During run time, the master process monitors whether the workload is balanced in the program, according to which it assigns particles to the slave processes. This method does not rely on the prior knowledge, such as the flow

features and the initial distribution of particle seeds. However, the heavy overhead of communication and I/O are induced due to the scheduling of dedicated processes and data loading during run time, respectively. These become the bottlenecks on the scalability.

DStep [Kendall et al., 2011], a framework similar to MapReduce [Jeffrey and Sanjay, 2008], is proposed recently to solve the parallel domain traversal problems, including parallel particle tracing. DStep effectively integrates the advantages of data and task parallelism. Specifically, it uses static round-robin data assignment in data-parallel aspect and a task management based on dedicated task queues with different types of tasks in task-parallel aspect to improve the scalability of parallel particle tracing. No special requirements are needed in the DStep framework, making it very suitable for the parallel computation of particle tracing. Results show this framework is scaled well to 64K BlueGene/P cores. On the basis of DStep, the visualization group in Peking University inherits and redesigns the DStep framework, in order to generate and manage large-scale pathlines for ensemble data visualization [Guo et al., 2013,Liu et al., 2016]. Furthermore, to introduce Lagrangian-based attribute space projection into multivariate flow data analysis, they refine the DStep framework and integrate it with Pivot MDS projection algorithm [Brandes and Pich, 2006] to support the computation and analysis of large-scale pathlines [Guo et al., 2014a]. These methods both achieve good scalability.

Hybrid Parallelism in Specific Applications

In some specific flow visualization applications, the hybrid parallel strategy has also been well studied. Camp et al. [Camp et al., 2012] use the data parallelism and task parallelism respectively on the parallel computation of stream surface. They test these two methods by using different datasets to estimate the efficiency on I/O, communication, and computational workload. Experiments demonstrate that these two methods both encounter load imbalance problem, because the waiting of processes and the I/O operations cost much time. Furthermore, Lu et al. [Lu et al., 2014] present a hybrid-parallel method. They partition the data and seeding curves respectively and assign them to processes. Each process is responsible for computing one part of the stream surface. During the computation, when the required data is not in the memory, a data request will be issued to other processes for a data copy. So this method also needs to move data between processes during run time. For the task aspect, when the seeding curves grow wide due to flow expansion, they will be subdivided into two segments dynamically. The segmented curves are assigned to idle processes through work stealing. Compared with Camp et al.'s work [Camp et al., 2012], the load becomes more balanced because the time is mostly spent on computation of particle tracing.

Summary

The hybrid parallelism fully combines the advantages of data- and task-parallel methods. It can overcome some of the inherent shortcomings in data and task parallelism. Hybrid-parallel methods do not require any pre-analysis. In the existing algorithms, the additional costs come from the communication of data movement [Lu et al., 2014], or the I/O operations of on-demand data loading and the scheduling of mater processes [Pugmire et al., 2009]. There is an increased complexity in the design and implementation of hybrid-parallel particle tracing methods, which also requires more research efforts. However, for some specific applications, such as the stream surface computation mentioned above [Lu et al., 2014], we can take advantage of the nature of the problem to be solved and study the appropriate hybrid-parallel strategy accordingly.

In the context of ever growing scale and complexity of flow fields, hybrid-parallel methods can gain great benefits in particle tracing and thus become gradually the focus of researchers. It is foreseeable that there will appear more hybrid-parallel algorithms on efficient computation of particle tracing in the near future.

5 Discussions

With the rapid development of supercomputing technology, tracing particles in the parallel environment has become an important development trend. Particle tracing itself is computational intensive and complex. In the parallelization of particle tracing, the size and complexity of the flow fields and the analysis

tasks (i.e., particles) pose more challenges, including the efficiency of I/O and communication, and the load balance over the processes. These always result in the scalability problem that needs to be seriously addressed.

Researchers have proposed many algorithms addressing the problems and challenges in parallel particle tracing. These methods have their pros and cons, as well as different requirements for initializing that are shown in Table 1. Among them, a lot of these algorithms are conducted in high performance computing environment. In Table 2, we list the scale of the problem and the number of the cores used in the experiments. As the scale and the complexity of flow fields and seed size is dramatically increasing, using high performance computing with larger-scale computing nodes will become more popular. But the challenges also increase as the scale of the problems becomes more complicated. More efforts are needed in the future research of parallel particle tracing. It requires more efforts to develop more scalable analysis methods for parallel particle tracing. Specifically, we need to decrease the granularity of data parallelism and task parallelism to improve the parallel efficiency.

Table 2 The maximal data size, seed size, and the number of cores used in the experiments of some algorithms that are performed in supercomputing environment.

Algorithm	Data Size	Seed Size	# Cores
[Peterka et al., 2011]	608 GB	120 K	32 K
[Nouanesengsy et al., 2012]	140 GB	288 M	16 K
[Nouanesengsy et al., 2011]	18.69 GB	256 K	4 K
[Yu et al., 2007]	720.8 GB	1 M	256
[Zhang et al., 2018]	13.4 GB	128 M	8 K
[Müller et al., 2013]	275 GB	1 M	1 K
[Guo et al., 2014b]	843.75 GB	143 M	512
[Lu et al., 2014]	96 GB	128 K	8 K
[Pugmire et al., 2009]	0.49 GB	20 K	512
[Kendall et al., 2011]	410 GB	40 M	64 K

On the basis of the traditional parallel particle tracing algorithms, we should combine the current development of new technology and put forward more new methods in the future. For example, we have the following hints:

First, we can put efforts to further improve the traditional parallel algorithms with suitable strategies that address the actual issues. As mentioned before, the challenges come from the flow fields and the particle seeds. For parallel particle tracing algorithms, the I/O, communication, and load balance have large impacts on the scalability, which affect the efficiency of the parallel computation. The general methods first decompose the problem into several small subproblems, and then distribute them over all the processes. Good partitioning and assignment strategies are very critical. We need to further decrease the granularity of data parallelism and task parallelism to improve the parallel efficiency. A fine granularity often contributes to more balanced work distribution to a great extent. To reduce the I/O and communication cost, there are also some strategies that can be adopted, such as organizing small and frequent I/O (or communication) requests into larger one. On the other hand, for the specific applications based on particle tracing, the design of parallel particle tracing algorithms should be more flexible. For example, in FTLE or stream surface computation, the seeding of particles is often adaptive, i.e., dynamically determined during the computation. This increases the difficulty of choosing good parallel strategy. Nevertheless, the characteristics of the application itself can be integrated in parallel particle tracing. Examples include the pipelined FTLE computation [Nouanesengsy et al., 2012] and dynamic seeding curve subdivision [Lu et al., 2014], as detailed in the preceding subsection. So in this case, it also provides greater flexibility when designing suitable algorithms.

Second, we can choose/design proper parallel methods based on the hardware environment. For example, in the implementation of data and task parallelism, each core of a CPU node always runs one thread

and owns independent computing resources, which indicates a distributed manner between CPU nodes and between cores. Actually, as shown in Figure 12, we can use multiple threads (including I/O threads and computation threads) in each core with shared resources between these threads [Camp et al., 2011]. Compared with the traditional methods, I/O operations and computation can be run asynchronously, according to which higher performance is achieved. Although the implementation of this kind of methods is more complicated and more challenging, it will become a future trend in the supercomputing environment due to the significant gains of performance and efficiency. Besides, except for these CPU-based parallel methods, GPU acceleration can also be applied in parallel particle tracing. Camp et al. [Camp et al., 2013] study this kind of method to explore the performance characteristics of GPU in dealing with parallel particle tracing. This method is data-parallel, and it uses two threads, i.e., one for communication and the other for computation. Compared with the CPU implementation, in most cases using GPU acceleration performs better, except for the case with less particles. So GPU is more suitable for the computation of high activity integration. Chen et al. [Chen et al., 2016] also use GPU computation to improve the performance by reorganizing particles into spatially coherent bundles. However, GPU-based parallel particle tracing is difficult to scale up. The I/O and communication limit its scalability. Currently there are few researches on this aspect, so challenges are still remaining.

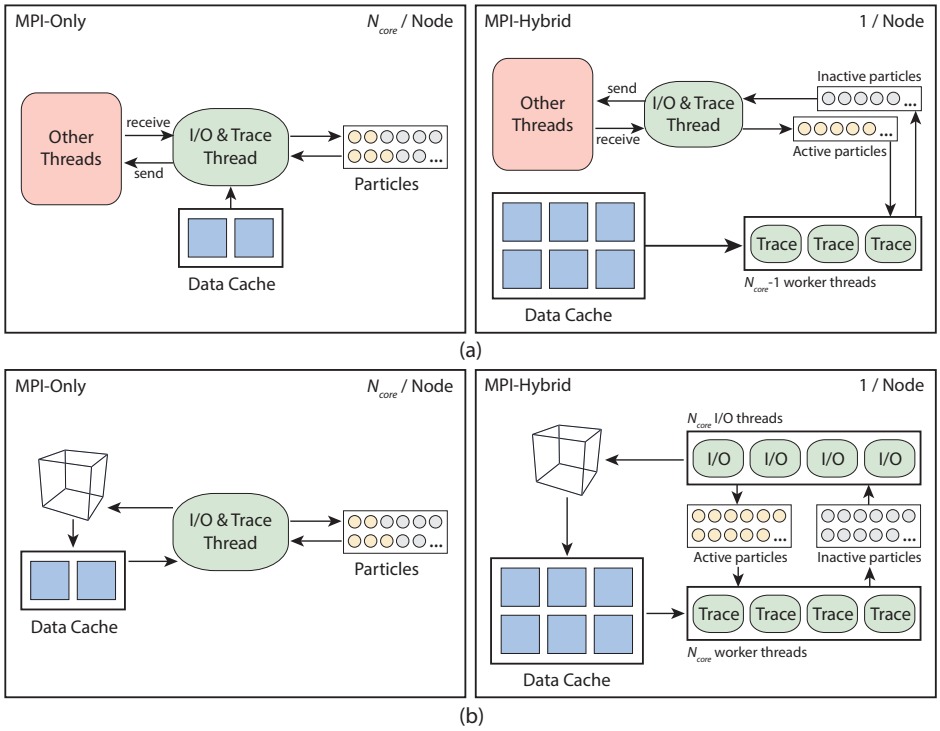

Fig. 12 Two implementation ways of data parallelism (a) and task parallelism (b) [Camp et al., 2011]. Using multiple threads in each core with shared computing resources achieves higher performance.

Third, because the I/O issues severely constrain the efficiency of large-scale particle tracing, we need to find more efficient methods in data storage and processing. As mentioned earlier, the scale of the flow field and the generated field lines by particle tracing is very huge. Traditional post-processing methods store the data in the file system and then analyze it. The efficiency of processing is limited by the I/O bottleneck. In practice, the data used in the post-processing methods is often a small part of the original data generated by simulations. For the flow visualization based on particle tracing, the accuracy and applications are always affected a lot. We can consider to use in-situ [Ma, 2009] or in-transit [Bennett et al., 2012] to solve this problem. In-situ methods conduct visualization and numerical simulation simultaneously on the simulation nodes, sharing the hardware and software resources. In-transit methods transfer the data generated during simulations from supercomputers directly to visualization machines through high-speed I/O devices. These methods support parallel visualization directly within the original data during the

simulation. On the other hand, for the applications that users need to analyze the data carefully after the simulation, they can first reduce the data by subsampling, quantization, and compression, or directly extract important features from the original data. The in-situ or in-transit processing results can be used for further explorations. In general, we can trace particles within the original data or the reduced data in in-situ or in-transit. The impact of the performance of I/O devices on parallel particle tracing will be reduced a lot. However, it is still challenging to couple the particle tracing code with numerical simulation code and minimize the interference of particle tracing on the simulation, as well as reduce the cost of particle tracing during this phase. These problems are challenging and are also remained to be solved in the near future.

6 Conclusions

In this paper, we review the recent researches of parallel particle tracing algorithms in flow visualization and analysis and discuss the research trends in the future. During the run-time particle tracing, it is hard to maintain balanced workload due to the unpredictable advection of particles. On the other hand, the I/O efficiency of data accesses and inter-process collaboration between processes often lead to some scalability problems. Therefore, the researchers put forward targeting solution in their methods for improving the I/O efficiency, reducing the communication costs, and essentially balancing the load to improve the scalability of parallel particle tracing. The parallel particle tracing algorithms consist of three categories, including data-parallelism, task-parallelism, and hybrid-parallelism. In data-parallelism, data is statically partitioned and assigned to processes, while in task-parallelism, particle seeds are statically assigned to processes. The hybrid-parallel methods combines data and task parallelism. Different algorithms have different characteristics, which are reflected on the requirements in pre-analysis and in run-time. We show the pros and cons of these algorithms and summarize the relationships and difference among them in the survey.

With the continuous advancement of high performance computing, the use of distributed computing resources to handle large-scale flow data with large-scale computational tasks (i.e., particles) has become an increasingly popular trend. It also becomes the focus of the research work in flow visualization and analysis. In the context of the continuous growth in the size and complexity of flow field data and the variety of visualization tasks, parallel particle tracing will have a wider range of applications in flow visualization in the future. More challenges in terms of the load balance and essentially the scalability will encounter. Therefore, researchers are required to continue to make efforts on designing more efficient algorithms on the basis of the existing methods for parallel particle tracing.

Acknowledgements This work is supported by NSFC No. 61672055 and the National Program on Key Basic Research Project (973 Program) No. 2015CB352503.

References

[Akande and Rhodes, 2013] Akande, O. O. and Rhodes, P. J. (2013). Iteration aware prefetching for unstructured grids. In *Proceedings of the 2013 IEEE International Conference on Big Data*, pages 219–227.

[Bennett et al., 2012] Bennett, J., Abbasi, H., Bremer, P., Grout, R. W., Gyulassy, A., Jin, T., Klasky, S., Kolla, H., Parashar, M., Pascucci, V., Pébay, P. P., Thompson, D. C., Yu, H., Zhang, F., and Chen, J. (2012). Combining in-situ and in-transit processing to enable extreme-scale scientific analysis. In *SC12: Proceedings of the ACM/IEEE Conference on Supercomputing*, page 49.

[Berger and Bokhari, 1987] Berger, M. J. and Bokhari, S. H. (1987). A partitioning strategy for nonuniform problems on multiprocessors. *IEEE Trans. Computers*, 36(5):570–580.

[Blumofe, 1994] Blumofe, R. D. (1994). Scheduling multithreaded computations by work stealing. In *35th Annual Symposium on Foundations of Computer Science, Santa Fe, New Mexico, USA, 20-22 November 1994*, pages 356–368.

[Brandes and Pich, 2006] Brandes, U. and Pich, C. (2006). Eigensolver methods for progressive multidimensional scaling of large data. In *Graph Drawing, 14th International Symposium, GD 2006, Karlsruhe, Germany, September 18-20, 2006. Revised Papers*, pages 42–53.

[Bruckschen et al., 2001] Bruckschen, R., Kuester, F., Hamann, B., and Joy, K. I. (2001). Real-time out-of-core visualization of particle traces. In *Proceedings of the IEEE 2001 Symposium on Parallel and Large-Data Visualization and Graphics*, pages 45–50.

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

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

[Camp et al., 2011] Camp, D., Garth, C., Childs, H., Pugmire, D., and Joy, K. I. (2011). Streamline integration using MPI-hybrid parallelism on a large multicore architecture. *IEEE Transactions on Visualization and Computer Graphics*, 17(11):1702–1713.

[Camp et al., 2012] Camp, D., Garth, C., Childs, H., Pugmire, D., and Joy, K. I. (2012). Parallel stream surface computation for large data sets. In *Proceedings of IEEE Symposium on Large Data Analysis and Visualization 2012*, pages 39–47.

[Camp et al., 2013] Camp, D., Krishnan, H., Pugmire, D., Garth, C., Johnson, I., Bethel, E. W., Joy, K. I., and Childs, H. (2013). GPU acceleration of particle advection workloads in a parallel, distributed memory setting. In *EGPGV13: Eurographics Symposium on Parallel Graphics and Visualization*, pages 1–8.

[Carns et al., 2000] Carns, P. H., III, W. B. L., Ross, R. B., and Thakur, R. (2000). PVFS: A parallel file system for linux clusters. In *4th Annual Linux Showcase & Conference 2000*.

[Çatalyürek et al., 2007] Çatalyürek, Ü. V., Boman, E. G., Devine, K. D., Bozdag, D., Heaphy, R. T., and Riesen, L. A. (2007). Hypergraph-based dynamic load balancing for adaptive scientific computations. In *IPDPS07: Proceedings of IEEE International Symposium on Parallel and Distributed Processing*, pages 1–11.

[Chen et al., 2012a] Chen, C., Nouanesengsy, B., Lee, T., and Shen, H. (2012a). 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.

[Chen and Shen, 2013] Chen, C. and Shen, H. (2013). 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.

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

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

[Chen et al., 2016] Chen, M., Shadden, S. C., and Hart, J. C. (2016). Fast coherent particle advection through time-varying unstructured flow datasets. *IEEE Trans. Vis. Comput. Graph.*, 22(8):1959–1972.

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

[Dinan et al., 2009] Dinan, J., Larkins, D. B., Sadayappan, P., Krishnamoorthy, S., and Nieplocha, J. (2009). Scalable work stealing. In *SC09: Proceedings of the ACM/IEEE Conference on Supercomputing*, pages 53:1–53:11.

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

[Ellsworth et al., 2004] Ellsworth, D., Green, B., and Moran, P. J. (2004). Interactive terascale particle visualization. In *Proceedings of IEEE Visualization 2004*, pages 353–360.

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

[Gerndt et al., 2004] Gerndt, A., Hentschel, B., Wolter, M., Kuhlen, T., and Bischof, C. H. (2004). 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.

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

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

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

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

[Inc., 2002] Inc., C. F. S. (2002). Lustre: A scalable, high performance file system. *whitepaper*.

[Jeffrey and Sanjay, 2008] Jeffrey, D. and Sanjay, G. (2008). Mapreduce: Simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113.

[Karypis and Kumar, 1996] Karypis, G. and Kumar, V. (1996). Parallel multilevel k-way partitioning scheme for irregular graphs. In *SC96: Proceedings of the ACM/IEEE Conference on Supercomputing*, page 35, Washington, DC, USA. IEEE Computer Society.

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

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

[Liu et al., 2016] Liu, R., Guo, H., Zhang, J., and Yuan, X. (2016). Comparative visualization of vector field ensembles based on longest common subsequence. In *Proceedings of IEEE Pacific Visualization Symposium 2016*, pages 96–103.

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

[Ma, 2009] Ma, K. (2009). In situ visualization at extreme scale: Challenges and opportunities. *IEEE Computer Graphics and Applications*, 29(6):14–19.

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

[Morozov and Peterka, 2016] Morozov, D. and Peterka, T. (2016). Efficient delaunay tessellation through K-D tree decomposition. In *SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 728–738.

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

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

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

[Peterka et al., 2011] Peterka, T., Ross, R. B., Nouanesengsy, B., Lee, T., Shen, H., Kendall, W., and Huang, J. (2011). 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.

[Pilkington and Baden, 1994] Pilkington, J. R. and Baden, S. B. (1994). Partitioning with space-filling curves. In *CSE Technical Report Number CS94-349*.

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

[Pugmire et al., 2009] Pugmire, D., Childs, H., Garth, C., Ahern, S., and Weber, G. H. (2009). Scalable computation of streamlines on very large datasets. In *SC09: Proceedings of the ACM/IEEE Conference on Supercomputing*, pages 16:1–16:12.

[Rhodes et al., 2005] Rhodes, P. J., Tang, X., Bergeron, R. D., and Sparr, T. M. (2005). Iteration aware prefetching for large multidimensional datasets. In *SSDBM2005: proceedings of the 17th International Conference on Scientific and Statistical Database Management*, pages 45–54.

[Schmuck and Haskin, 2002] Schmuck, F. B. and Haskin, R. L. (2002). GPFS: A shared-disk file system for large computing clusters. In *Proceedings of the FAST '02 Conference on File and Storage Technologies*, pages 231–244.

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

[Silva et al., 2002] Silva, C., Chiang, Y.-J., El-Sana, J., and Lindstrom, P. (2002). Out-of-core algorithms for scientific visualization and computer graphics. *IEEE Visualization Course Notes*.

[Simon, 1991] Simon, H. D. (1991). Partitioning of unstructured problems for parallel processing. *Computing Systems in Engineering*, 2(2-3):135–148.

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

[Zhang et al., 2018] Zhang, J., Guo, H., Hong, F., Yuan, X., and Peterka, T. (2018). Dynamic load balancing based on constrained k-d tree decomposition for parallel particle tracing. *IEEE Trans. Vis. Comput. Graph.*, 24(1):954–963.

[Zhang et al., 2016] Zhang, J., Guo, H., and Yuan, X. (2016). Efficient unsteady flow visualization with high-order access dependencies. In *Proceedings of IEEE Pacific Visualization Symposium 2016*, pages 80–87.