Dynamic Data Repartitioning for Load-Balanced Parallel Particle Tracing

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

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

ABSTRACT

We present a novel dynamic load-balancing algorithm based on data repartitioning for parallel particle tracing in flow visualization. Instead of static data assignment, we dynamically repartition the data into blocks and reassign the blocks to processes to balance the workload distribution among the processes. Block repartitioning is performed based on a dynamic workload estimation method that predicts the workload in the flow field on the fly as the input. In our approach, we allow data duplication in the repartitioning, enabling the same data blocks to be assigned to multiple processes. Load balance is achieved by regularly exchanging the blocks (together with the particles in the blocks) among processes according to the output of the data repartitioning. Compared with other load-balancing algorithms, our approach does not need any preprocessing on the raw data and does not require any dedicated process for work scheduling, while it has the capability to balance uneven workload efficiently. Results show improved load balance and high efficiency of our method on tracing particles in both steady and unsteady flow.

Keywords: Parallel particle tracing, dynamic load balancing, data repartitioning

1 INTRODUCTION

In flow visualization and analysis, particle tracing is a fundamental technique. By tracing particles in the data domain, users can conduct many applications, such as computing streamlines in steady flow and pathlines in unsteady flow, extracting flow features based on predicates [31, 32], studying the relationships between scalars and vectors in the flow field [14], and analyzing the ensemble differences [15]. As the data scale and complexity grow explosively, tracing particles on large-scale distributed and parallel computing environment has become a popular trend in recent years. The parallelization of particle tracing enables handling large-scale flow data on clusters or supercomputers, making the visualization and analysis of these data feasible.

However, parallel computing often suffers from workload imbalance. In this paper, we also address this problem in parallel particle tracing, which is challenging because the advection of particles over the flow field domain is considered unpredictable. Large flow data is often partitioned into blocks, which are subsequently assigned to processes. During run time, the unfinished particles will be exchanged among processes. Because the blocks may have different features that require different computational workloads in particle tracing, processes will be unbalanced. Although some static load balancing methods exist, such as static workload estimation [28] and irregular flow partitioning [7, 34], a time-consuming preprocessing step is needed before running applications. Therefore, achieving a balanced workload during parallel particle tracing is often difficult, especially when tracing massive particles in large and sophisticated flow fields. As a result, the scalability in parallel particle tracing is constrained.

In this work, we present a novel dynamic load-balancing method based on data repartitioning for parallel particle tracing. Unlike static load-balancing methods, our method starts with an arbitrary data partitioning and assignment strategy that does not require any preprocessing costs. The run-time parallel particle tracing process is divided into several rounds. The goal is to achieve a balanced workload in each round of tracing. Before tracing a round (except the first round), we dynamically estimate the workload of each block using the information of its historical workload and the number of particles that will be traced in this block. Data repartitioning is performed based on the workload estimation. Each process is then reassigned a new data partition and continues to trace particles within the data blocks in this round.

In order to estimate larger workloads in each round and reduce the frequency of data repartitioning, our method allows data duplications between the new partitions. During run time, we dynamically build a graph model that records the access dependencies between neighboring blocks according to the historical data accesses of the particles. This graph enables us to predict how many particles will travel from one block to the other. With this graph, we can estimate the workload of particles originating from one block that travel through multiple blocks in a round. In other words, each particle can be traced in a specified number of blocks instead of only the current block in each round. Data is duplicated in the new partitions after repartitioning because the particles of each process may access the same blocks in a round.

Our method can be applied to both steady and unsteady flow data that require tracing streamlines and pathlines, respectively. For tracing streamlines in steady flow, the data is loaded only once and is exchanged after data repartitioning. For tracing pathlines in unsteady flow, because the data consists of multiple time steps and the memory may not accommodate all the data, we divide the data into several time intervals and load them successively. Only one time interval is in memory each time, and we follow the approach used in steady flow to trace particles and perform data repartitioning for this time interval.

We study our method with different data sets on Vesta, a Blue Gene/Q supercomputer at Argonne National Laboratory. Compared with the baseline round-robin data assignment method, we show that our method significantly improves the load balance and hence the performance in parallel particle tracing. Compared with other load-balancing algorithms, our method does not need any preprocessing on the raw data and does not require any dedicated process for work scheduling, while having the capability to balance uneven workload efficiently.

The remainder of this paper is organized as follows. In Section 2, we review related work. In Section 3, we present our parallel particle tracing framework. Section 4 describes our dynamic data-repartitioning method during parallel particle tracing. The results of the performance study are presented in Section 5 to demonstrate the effectiveness of our method. We also compare our method with

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

other methods and discuss the limitations of our method in Section 6. In Section 7, we present the conclusions and discuss possible future work.

2 Background

In this section, we review the background of our work, including advection-based flow visualization, load balancing in parallel particle tracing, and load balancing based on data partitioning.

2.1 Advection-Based Flow Visualization

Particle advection computes field lines from given seed points. It is widely used in many flow visualization applications, including texture-based [23] and geometry-based methods [25], and the extraction and tracking of flow features [30]. According to the seeding strategies, advection-based visualization methods can be subdivided into two categories: full-range analysis and local-range analysis.

In local-range analysis, particles are initially seeded in a specified spatial-temporal subdomain. The subdomain can be local regions in source-destination analysis [22] or seeding curves for flow surfaces computation [12]. The local-range analysis is used for visualizing regional flow features.

Full-range analysis places particle seeds densely in the entire data domain. In contrast to local-range analysis, full-range analysis is used to explore the overview features of flow fields. Examples include line integral convolution (LIC) [3], unsteady flow LIC [33], and the extraction of Lagrangian coherent structures with finite-time Lyapnov exponents (FTLE) [13]. The densely seeding strategy in these methods indicates that tracing massive particles is required, which makes the computation expensive. Some methods also accelerate particle advection, such as reuse of partial advection path [18] and adaptive seeding refinement [1]. However, more scalable methods are still needed for applications with densely seeded particles.

Our work is more suitable for densely seeded full-range analysis because each part of the data domain is involved during particle tracing in our method. In Section 5, we show a detailed performance analysis for the evaluation. In Section 6, we discuss the application scope of our method.

2.2 Load Balancing in Parallel Particle Tracing

Parallel particle tracing algorithms can be divided into three categories: task parallelism, data parallelism, and hybrid methods that combine task and data parallelism. We review the algorithms that focus on solving the problem of load imbalance in parallel particle tracing.

In task-parallel methods, particle seeds are statically assigned to processes, and then each process traces particles. Since each particle has a different trajectory, the workload among different processes is unbalanced. Some dynamic load-balancing methods, such as work stealing [11,24] and work requesting [26], are employed to allow the process to request tasks from other busy processes when it becomes idle. However, a center process is required in order to schedule the tasks, which induces high communication cost and limits the scalability.

In data-parallel methods, the data are partitioned into blocks, and the blocks are assigned to processes. During run time, unfinished particles are exchanged among processes to continue the advection. In order to improve load balance, the data can be partitioned based on static workload estimation [28], according to flow direction and flow features [7], over time steps in FTLE computation [27], or can be assigned statically by round-robin methods [29] and hierarchical clustering [34].

Recently, researchers introduce hybrid parallel methods for more scalable parallel particle tracing. In stream surface computation, the data and the seeding curve segments are both statically assigned [24]. Kendall et al. [22] proposed a MapReduce-like framework, DStep, for hybrid-parallel particle tracing, which improves the scalability in large-scale parallel environment. DStep [22] has been extended to visualize the differences between vector field ensembles with pathline advection [15] and to analyze multivariate unsteady flow by coupling Lagrangian-based attribute space with projection techniques [14]. It also has been modified to calculate high-order access dependencies in a preprocessing stage for efficient pathline computation [36].

We focus on data-parallel particle tracing and propose a dynamic load-balancing method targeting the problem. A comprehensive comparison with other load-balancing methods is given in Section 6.

2.3 Data Repartitioning for Load Balancing

The focus of our work is dynamic data repartitioning for load balancing. The balanced workload is obtained by partitioning the data regularly during the run time of the applications. The algorithms of data partitioning that are applied in load-balancing problems can be divided into two categories: geometric methods and topological methods.

Geometric methods partition the data based on the geometric coordinates of partitioning objects. We can perform geometric partitioning by recursive coordinate bisection [2], recursive inertial bisection [19], and space-filling curves [8]. Although these methods require the coordinate information, they are simple, fast, and computationally inexpensive.

Topological methods require constructing topological structures for partitioning objects, including graph [20] and hypergraph partitioning [4]. The problem of topological partitioning is solved by techniques such as quadratic programming [17] and multilevel scheme [21]. These methods have high-quality partitions because they can meet criteria other than load balance, such as minimizing the interprocessor communication. However, they are more sophisticated and expensive than geometric methods, especially hypergraph partitioning [10].

In this work, we refine the recursive coordinate bisection algorithm (RCB) for data repartitioning in parallel particle tracing. The estimated workloads among blocks are balanced with this method for dynamic load balancing.

3 Parallel Particle Tracing Framework

Figure 1 illustrates the workflow of our parallel particle tracing framework. Given the initial partitions of the raw flow data and the particle seeds, we start to trace particles in parallel. The parallel particle tracing process consists of multiple rounds. After each round of tracing is complete, we repartition the data based on the dynamic workload estimation on all blocks for the subsequent round. Then, we migrate the data and exchange the particles among processes according to the new partitions. The subsequent round starts with new data assignments. This multiround process repeats until all particles are finished. Because load imbalance occurs during particle tracing, our goal is to maintain a balanced workload among all processes in each round through dynamic data repartitioning.

3.1 Initialization

In this work, we use fine granularity to manage the raw data into data entries. As demonstrated in Guo et al.'s work [16], using fine-grained data management improves the I/O efficiency in particle tracing. The data is partitioned into smaller blocks, with each consisting of a combination of several continuous data cells. The data block is also the object unit in the data-repartitioning model.

Our method handles particle tracing in both steady and unsteady flow data, that is, computing streamlines and pathlines, respectively. For 3D steady flow, the data is partitioned into spatial blocks. Each block is a chunk of data with $x \times y \times z$ data cells, where x, y, z are the size along each spatial dimension, respectively. The total number of spatial blocks is proportional to the number of processes. In order to ensure enough data blocks for repartitioning, the ratio between the total number of spatial blocks and the number of processes, bp,

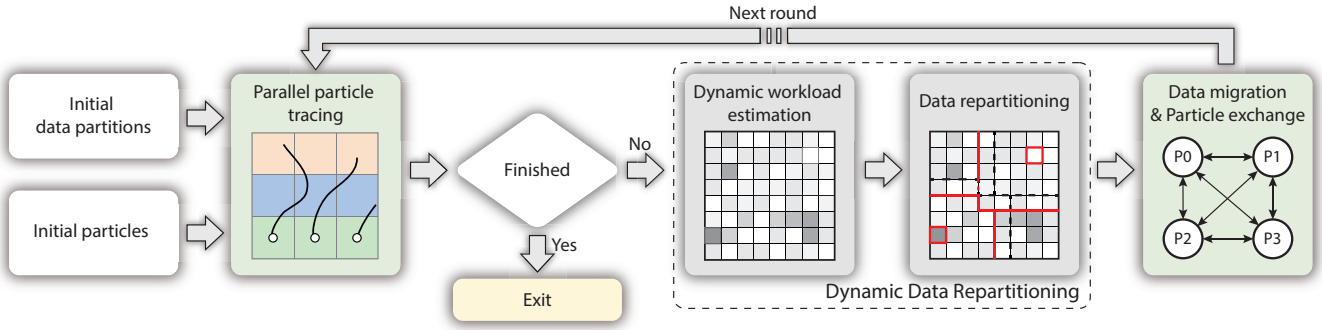

Figure 1: Workflow of our parallel particle tracing framework. Starting with the initial assignments of the raw data, particles are traced through a multiround process. Dynamic data repartitioning is performed after each round of tracing. Then with the new data assignments, the subsequent round repeats to trace particles. When all particles are finished, this process will terminate.

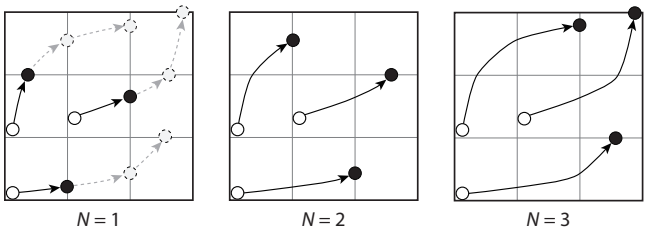

Figure 2: Illustration of one round of particle tracing. When N is 1, it needs three rounds for particles to trace through 3 blocks. But if N is 3, only one round of tracing is enough.

is a user-defined parameter and is directly related to the block size. In our tests, the size of each spatial block ranges from 8^3 to 64^3. For 4D unsteady flow data, the blocks are considered as 3D spatial blocks $\times$ 1D temporal blocks. The time dimension t is divided into several time intervals. Each interval is then decomposed into spatial blocks as is done in 3D steady flow. The number of time intervals, tb, is also a parameter that can affect the efficiency of dynamic data repartitioning. More evaluations of bp and tb are in Section 5.

Initially, we load the data in parallel for particle tracing. For steady flow data, all the data is loaded at the beginning. For unsteady flow data, which has multiple time steps and cannot fit into memory, we load and process the time intervals separately, that is, when a time interval finishes the particle tracing within it, the next time interval is loaded into the memory, until all time intervals are complete. The loaded data is decomposed into several partitions evenly. The number of partitions is equal to the number of processes specified to trace particles. Each partition is one part of the data, which contains a number of blocks. We then assign each process one partition. Because we apply dynamic data repartitioning during the run-time parallel particle tracing, the initial data assignment is not vital. The input particles are also loaded and distributed into processes according to the initial data assignment.

3.2 Multiround Tracing

During run time, each process traces the particles in its own blocks. A particle is advected until it goes out of the entire data domain, hits a critical point, or has traveled the maximum number of advection steps. When all particles meet this termination criterion, the parallel particle tracing process finishes. We consider the whole tracing process as a multiround model.

In previous work [28, 29], a round was considered complete when all particles traveled outside the blocks they initially began in or when they finished advection. However, a particle may be traced only a small number of advection steps in one block. With this definition of one round, a large number of rounds are needed in order to finish the particle tracing process. However, data repartitioning is more frequently performed, which increases the overhead due to the costs related to the data-repartitioning algorithm. Therefore, we extend the round definition to allow each particle to travel through multiple blocks, excluding the originating block. This allows particles to be traced with more integration steps in one round.

We define the number of blocks a particle can travel through in one round as the *tracing depth*, represented by a parameter N. For each round, a particle is traced until it reaches the boundary of the Nth block from the originating block or has already finished before that time. The round is complete when all particles have been traced in this manner. Figure 2 illustrates the particle tracing in one round with N ranging from 1 to 3. Note that when the particles have traveled through N blocks but have not yet met the termination criterion, they will stop, even if the next blocks for the particles to continue the advection are still in the memory of the same processes. Larger tracing depth reduces the number of rounds for tracing particles in the same situation. In the following data repartitioning, however, it also induces larger overhead to estimate the trajectory and the number of advection steps of a particle if the tracing depth is large. This estimation is explained in Section 4.1.

Each process traces particles independently using the fourth-order Runge-Kutta (RK4) method in each round. In order to ensure that each process has equal computation time, the workload distribution on different processes in each round should be as even as possible. If not, some processes will be waiting for other processes that still have not finished the current round, thus causing load imbalance. Maintaining balanced workloads in each round is therefore critical. This is also our goal of dynamic data repartitioning.

After each round of tracing, if particles are still unfinished, we perform data repartitioning to reassign data blocks to processes, so that each process will be redistributed balanced workload again in the subsequent round. For unsteady flow data, after a time interval is finished, we repartition the domain for loading the data of the next time interval, and the current time interval is removed from the memory. The data repartitioning is the key part of our dynamic load-balancing method, which is described in detail in Section 4.

Data migration and particle exchange take place after data repartitioning. As the new partitions are computed according to data repartitioning, the data blocks together with the particles inside owned by each process should be exchanged for continuing the advection in the next round.

Two ways exist for communicating among different processes for data migration and particle exchange. One is via point-to-point nonblocking communication with `MPI_Isend` and `MPI_Irecv` in order to overlap the communication and computation, which is commonly used in previous methods [28, 29]. However, particle tracing requires processes to be coordinated. A process will not work if other processes have not finished tracing and sent the unfinished particles to it. Hence, the goal of nonblocking communication is

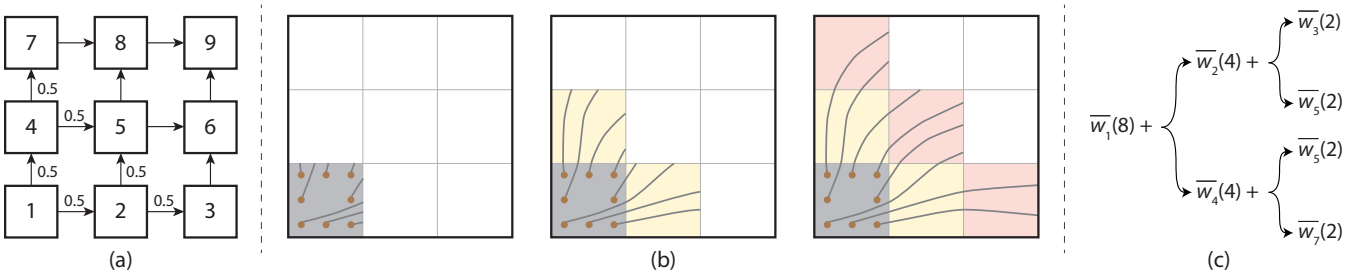

Figure 3: 2D example of workload estimation based on ADG. (a) During particle tracing, we build the access dependency graph on the fly; (b) For the originating block #1 with 8 particles in it, we predict the blocks that will be accessed and the number of particles in these blocks at each tracing depth level; (c) The workload of the originating block #1 is estimated as the sum of all the involved blocks in (b).

actually difficult to achieve. In our tracing model, the number of particles received in the latest round is almost always inconsistent with the number of particles intended to be sent after the preceding round if using this kind of communication. Therefore, we employ the second method, all-to-all communication using `MPI_Alltoall`. The collective all-to-all communication ensures that in the latest round each process will receive all particles and blocks sent from the preceding round. This is important because we want to know the exact number of particles that would be traced in each round in order to estimate the workload in dynamic data repartitioning. This also is explained in Section 4.1.

4 Dynamic Data Repartitioning

Dynamic data repartitioning is performed after each round of tracing, which targets balancing the workload in the subsequent round. In our method, we dynamically estimate the workloads of unfinished particles in each block for the subsequent rounds. Then, based on the workload estimation, we perform data repartitioning to reassign data to processes. Because the unsteady flow data can be considered as multiple steady flow data with different time steps, we take the data repartitioning of steady flow as the general case to describe our method.

4.1 Dynamic Workload Estimation

In this work, we use the number of advection steps in particle tracing to measure the workload. In each round, the workload of a block is defined as the total number of advection steps of particles that pass through the block in this round. The workload of a partition is the sum of workloads of all blocks that the partition contains. It is also the workload of the corresponding process in this round. We need to estimate the workloads in each round so that we can balance the workloads among processes by data repartitioning according to the estimation results.

In previous methods, two estimation measurements are mentioned. One directly uses the number of particles as the workload. However, it does not consider the advection difference between particles. In actual situations, some particles may require more steps than others to advect in certain blocks (e.g., they are involved in a vortex). Hence, the number of particles cannot reflect the actual workloads of these particles. The other method uses the historical workloads of the blocks [29]. However, the historical workload cannot accurately represent the workload of the future particle advectionand thus will induce a large error in the workload estimation. In our work, we use both the number of particles and the historical workload to compute the estimated workloads.

During the run-time particle tracing, each block keeps the historical tracing information by recording the number of particles and the number of advection steps traced in it in the preceding rounds. After one round of particle tracing is finished, we can calculate the average number of advection steps of a particle in each block. Moreover, according to the positions of unfinished particles, their current distribution on blocks can also be computed. Therefore, the workload of each block in the subsequent round can be estimated. Suppose we want to estimate the workload in round j after $j-1$ rounds of tracing. For block k in this round, if it has $n_{k,j}$ particles, then the estimated workload is computed as

$$\overline{w_{k,j}}(n_{k,j}) = \frac{\sum_{i=0}^{j-1} w_{k,i}}{\sum_{i=0}^{j-1} n_{k,i}} \times n_{k,j}, \tag{1}$$

where $w_{k,i}$ is the actual workload of block k in preceding round i, and $n_{k,i}$ is the actual number of particles passing through block k in preceding round i.

In our definition of Section 3.2, a particle can travel through multiple blocks in each round. For the particles originating from one block, we should estimate the total workload when each of these particles travels N blocks. Therefore, we need to know which blocks the particles will travel through when they go out of the originating block. To achieve this goal, we build an access dependency graph (ADG) among blocks on the fly during the run-time particle tracing. ADG records access transitions from one block to its neighboring blocks. It has been successfully used for data prefetching in out-of-core streamline and pathline computation [5, 6]. In our work, each block also keeps a part of ADG with several <`key, value`> pairs, where `key` is the index of one of its neighboring blocks and `value` is the corresponding access transition probability. During the tracing, the access dependencies related to each block are calculated by the corresponding process. We compute the ratio of the number of particles that travel from this block to each of its neighboring blocks and the total number of particles traced in this block, and we take it as the access transition probability between the two blocks. After each round of particle tracing, each process gathers the ADG from all other processes through communication and then obtains the entire ADG covering all blocks. By using this method, we can predict which blocks particles will travel through and how many particles will travel to those blocks in the subsequent round. Specifically, according to the access transition from the originating block, we can predict the blocks that will be accessed at the second tracing depth level. Based on the access transition probabilities, we can also predict the number of particles that will be traced to these blocks, respectively. Then we can compute the workload according to Equation 1 in each of these blocks. From these blocks, we can further predict the data accesses in the next tracing depth level. This process is repeated until the tracing depth N is reached. Figure 3 shows an example of workload estimation when the tracing depth is 3. After that, the workload calculated in each involved block is summed as the estimated workload of the originating block.

4.2 Repartitioning Model

After dynamic workload estimation, the data can be repartitioned based on the estimated workload of each block. For this goal, we first define a repartitioning model to figure out the problem we should resolve. In the repartitioning model, each block is considered as

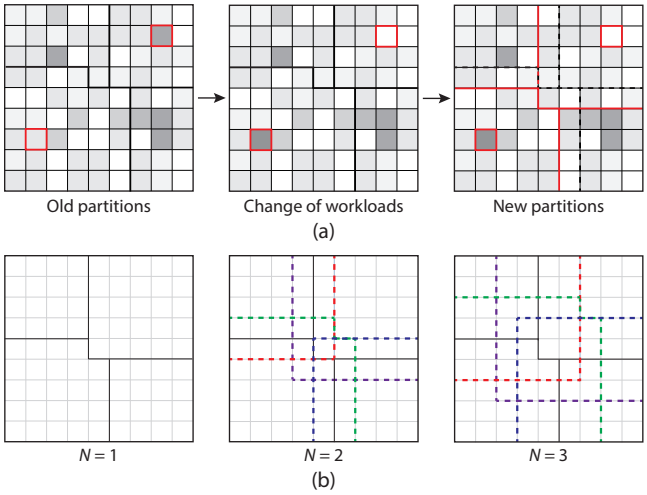

Figure 4: (a) Data repartitioning according to the change of workloads (shown in the squares with red outline) estimated after one round of particle tracing; (b) maximal data duplication between partitions with different tracing depths. Each partition is bounded by a colored dashed line. The copies of duplicated blocks will be also added to the corresponding partitions. Note that data is duplicated only when the tracing depth is larger than one.

an object, which is also the basic unit for data repartitioning. The weight of the object corresponds to the workload of the block.

Suppose the original data has $m(m > 0)$ blocks, namely, $C = \{C_0, C_1, C_2, \cdots, C_{m-1}\}$, with a corresponding estimated workload $w_k = \{w_{k,0}, w_{k,1}, w_{k,2}, \cdots, w_{k,m-1}\}$ before the kth data repartitioning (i.e., after k rounds of particle tracing). For n processes, we want to partition these blocks into n groups, namely, $G = \{G_0, G_1, G_2, \cdots, G_{n-1}\}$, where $G_i = \{C_{i_0}, C_{i_1}, C_{i_2}, \cdots\}$. The workload of each group is the sum of the workloads of all blocks it contains: $w_{k,G_i} = w_{k,i_0} + w_{k,i_1} + w_{k,i_2} + \cdots$. In order to achieve load balance for each process, each partition group should have equal workload. This requirement leads to an optimization problem: to minimize the workload difference between each two partition groups:

$$\min \mathscr{W}_k = \sum_{i=0}^{n-1} \left| w_{k,G_i} - \frac{1}{n} \sum_{j=0}^{n-1} w_{k,G_j} \right|. \quad (2)$$

For data repartitioning, this optimization problem also has a constraint. In order to reduce data migration among processes after repartitioning, the overlap between the old and new partitions should be maximal. The blocks in the new partition should not change significantly.

4.3 Repartitioning Algorithm

As described in Section 2.3, two kinds of data-repartitioning algorithms exist. Topological methods are always more costly than geometric methods. In our work, the cost of data repartitioning itself should be lowered because it may affect the overall performance of parallel particle tracing. Having much additional cost on data repartitioning is infeasible. Therefore, we employ the recursive coordinate bisection (RCB) algorithm [2] from the Zoltan Parallel Data Services Toolkit [9] to solve the optimization problem in Equation 2, as also used in Peterka et al.'s study [29]. The RCB method is illustrated in Figure 4(a). In this method, each block is weighted with the estimated workload with its spatial coordinate and is considered as a repartitioning object.

The RCB method is a collective operation involving all processes. It produces a new data-partitioning result. Each process inputs its old partition consisting of the weighted blocks it is responsible for and

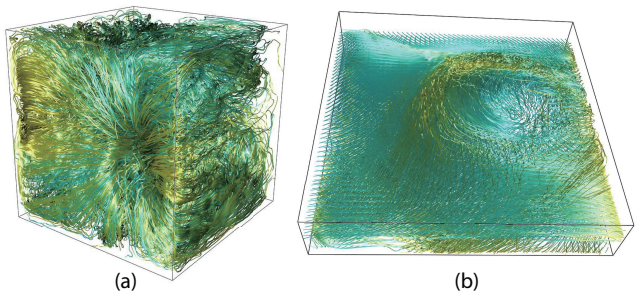

Figure 5: (a) Streamline rendering of the Nek5000 data; (b) Pathline rendering of the Isabel data.

then gets a new partition that has similar workload with others. The RCB method maximizes the overlap between the old and new partitions, which meets the constraint in our repartitioning model. The workload estimation of each block involves multiple other blocks according to ADG. In order to make sure that particles in the subsequent round will be traced consistently with our estimation, a copy of each involved block is added to the corresponding process through data migration. This will induce data duplication among different processes. Generally, with larger tracing depth, more data will be duplicated. Because the blocks in the new partition are almost adjacent, however, data duplication will occur mostly in the same process. It is actually the boundary blocks between partitions that will result in data duplication among different processes, as can be seen in Figure 4(b).

In our work, the parallel particle tracing framework targets computing both streamlines in steady flow and pathlines in unsteady flow. The general methods that can be directly applied to steady flow data were discussed above. For unsteady flow, the blocks are 4D with spatial and temporal dimensions. We load and process the time intervals successively. The blocks in a time interval are treated as spatial blocks. We assume that the data with adjacent time steps is temporally coherent, with little change. For each time interval, we build a new ADG and use it to estimate the workload and perform data repartitioning as we did for steady flow data. After a time interval is complete, we load the next time interval in parallel according to the repartitioning result. The earlier time interval is then removed from the memory. In this way, all the data need not to be read at one time, thus improving the memory efficiency.

Several strategies are available to improve the overall performance when performing dynamic data repartitioning. First, we reduce the workload of each particle in the first round of tracing. Load balance cannot be guaranteed because data repartitioning cannot be applied in the first round. With larger workload, it is more likely to result in load imbalance. Second, we do not perform data repartitioning when the total number of unfinished particles is small. Data repartitioning leads to data migration, which incurs additional overhead. The overhead may exceed the benefit brought from the data repartitioning if few particles are left (e.g., in the last few rounds of particle tracing). In this case, performing data repartitioning is not worthwhile. In our work, we empirically determine the use of these strategies depending on different applications.

5 Performance Study

We evaluated our method through a comprehensive performance analysis with two cases. The first case is a detailed analysis that traces streamlines in thermal hydraulics simulation (Nek5000) data to study strong and weak scalability and load balancing. The second case is tracing pathlines in Hurricane Isabel data to examine the strong scalability of particle tracing in unsteady flow. In these cases, we also evaluated the conditions that affect performance, including the tracing depth N and parameters bp and tb.

The performance study was conducted on Vesta, an IBM Blue

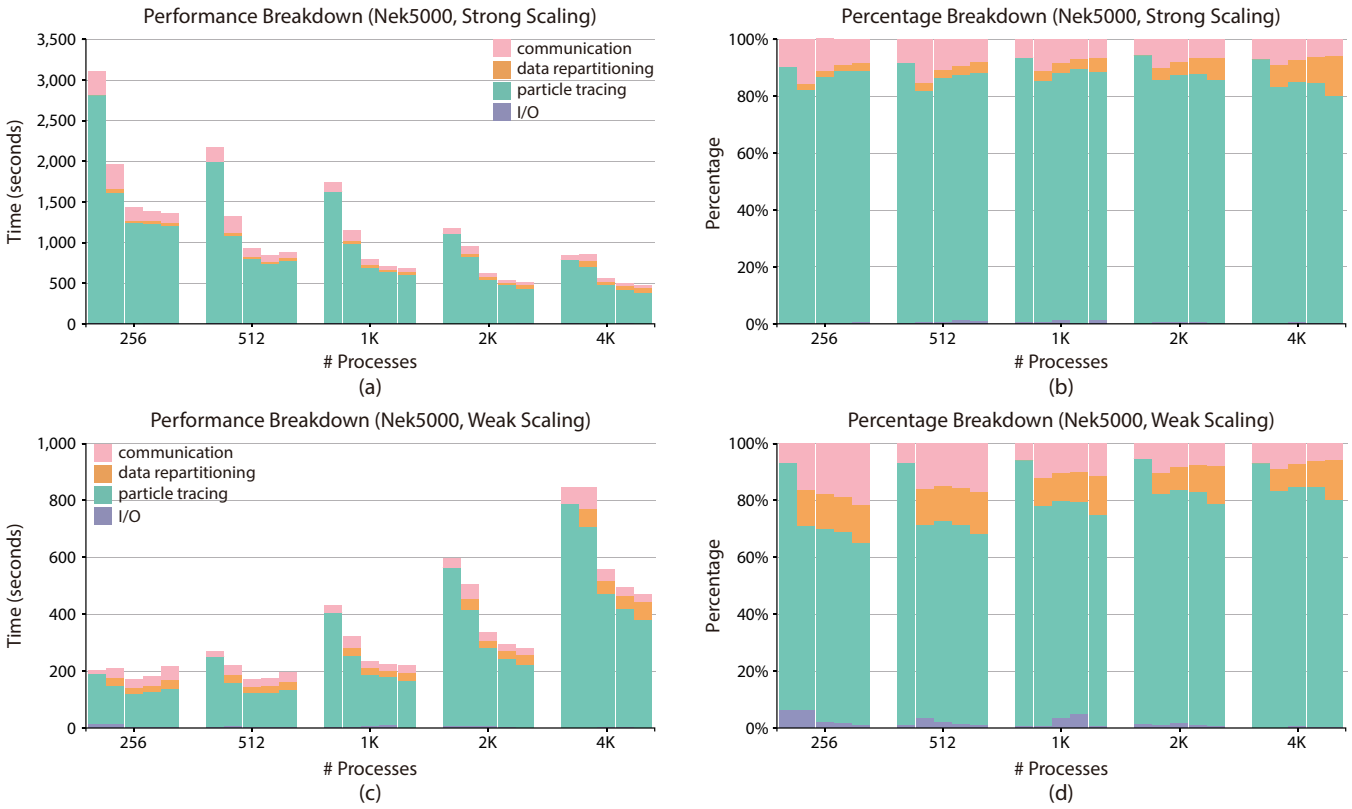

Figure 6: Performance and percentage breakdown for the analysis of the Nek5000 data. Figures (a) and (b) show the results of strong-scaling tests, while (c) and (d) show the results of weak-scaling tests. In each stacked bar, time for I/O, data repartitioning, particle tracing, and communication are encoded in different colors. At each kind of process count, the five stacked histograms represent the baseline method and our method with $N = 1, 2, 3, 4$, respectively.

Gene/Q platform at Argonne National Laboratory. It has 2,048 nodes, and each node is equipped with a 16-core 1600 MHz PowerPC A2 processor and 16 GB RAM. The intercommunication link between nodes is a 5D torus proprietary network. In our experiments, we ran 8 processes per node and used up to 4,096 processes.

In the performance study, we also compared our method with the baseline data-parallel particle tracing method. The baseline method uses static round-robin assignment, as studied in Peterka et al.'s work [29]. Within each round of particle tracing, the unfinished particles are traced until they reach the boundary of the current blocks. We directly exchange particles collectively using `MPI_Alltoall` and do not repartition the data after each round. The process exits when all particles are finished.

5.1 Streamline Computation: Thermal Hydraulics Data

The first case is a streamline computation with thermal hydraulics data. This data comes from the output of a large-eddy Navier-Stokes simulation generated by the Nek5000 solver at Argonne National Laboratory. A single time step of the data with a resolution of $512 \times 512 \times 512$ (about 1.5 GB) is used in this case. The streamline rendering result is shown in Figure 5(a). We placed densely seeded particles in the entire data domain for all the experiments.

Strong and Weak Scaling

We first traced 64M particles in total with 256, 512, 1K, 2K, and 4K processes to study the strong scalability. As shown in Figure 6(a), under different process count, the total running time of our method is less than the baseline. In most cases, the improvement of the performance in our method is more significant with larger tracing depth. In the best case with 1K processes, our method is 1.59 times faster than the baseline. Data repartitioning takes more time as the number of processes increases. The reason is that the number of blocks increases accordingly, incurring larger overhead in the workload estimation and repartitioning algorithm. Communication time decreases when the tracing depth is large enough. Larger tracing depth indicates fewer rounds of particle tracing, resulting in decreased frequency of particle exchange and data migration. However, larger tracing depth also induces more computation in the workload estimation and increases the memory overhead because of the data duplication. Thus the improvement in performance becomes stable as the tracing depth increases. From Figure 6(b), we can also see that the dominant time cost is in particle tracing as the number of processes increases.

For weak scaling, we traced 8K particles per process with 256, 512, 1K, 2K, and 4K processes. As shown in Figure 6(c), the total running time of our method changes more stably than that of the baseline. This demonstrates that the weak scalability of our method is better than that of the baseline. Taking a tracing depth of 4 as an example, the parallel efficiency with 1K processes in our method is 97.5%, while in the baseline method it is 47.2%.

Load Balancing

To show the load balance, we collected the performance data of running 64 processes with our method and the baseline method. We then visualized the activity of each process using the Gantt chart, as shown in Figure 7(a) (the baseline method) and Figure 7(b) (our method with tracing depth of 4). In the Gantt chart, the horizontal direction indicates the execution time, and the vertical axis is the ID of each process that is stacked vertically. The run-time activity consists of I/O, particle tracing, data repartitioning, and communication, which are encoded in different colors. The line chart in Figure 7(d) shows the load-balancing indicator over the execution time. In this

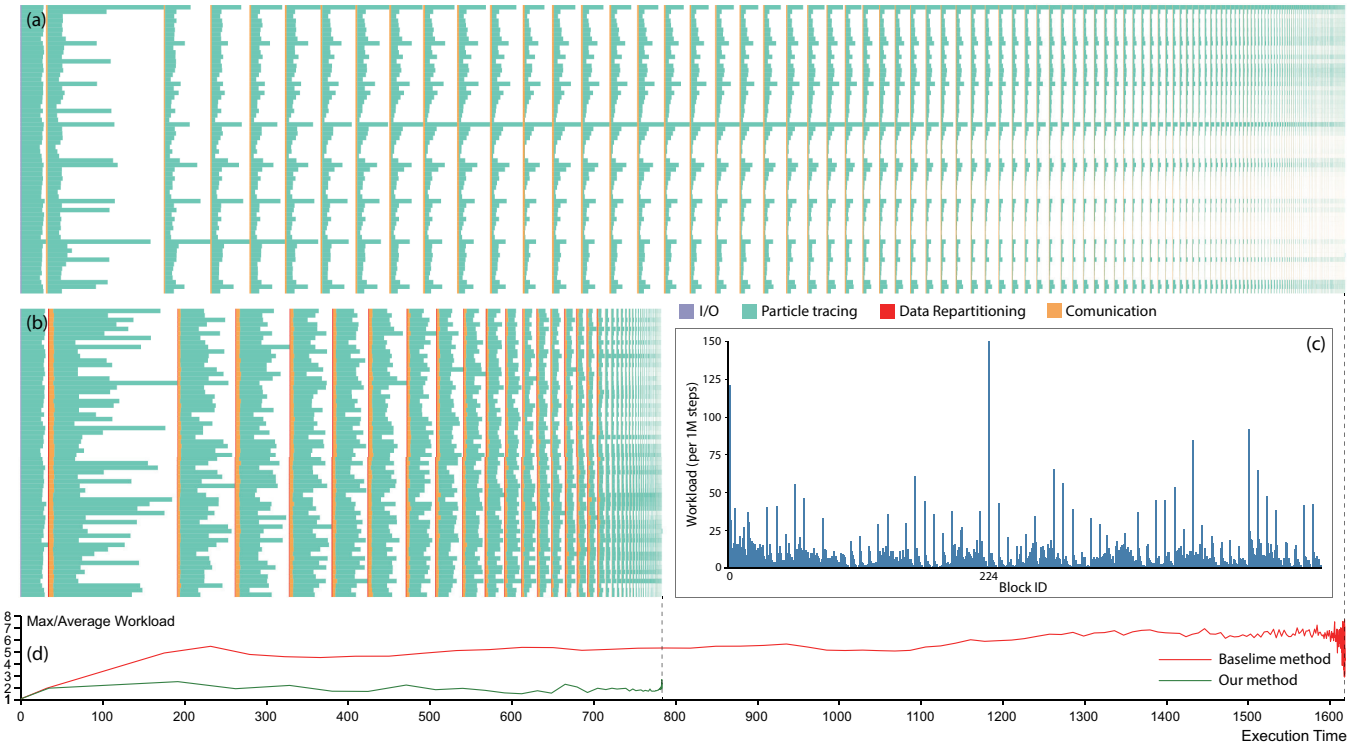

Figure 7: Performance of a 64-process run with the Nek5000 data. (a) Gantt chart using the baseline method. (b) Gantt chart using our method. In the Gantt charts, each row represents a process, and the times for I/O, data repartitioning, particle tracing, and communication are encoded in different colors. (c) Bar chart showing the workload distribution of the blocks. (d) Line chart showing the evolution of load-balancing indicator. In the line chart, the dashed line represents the baseline method, while the solid line represents our method. The time axes of the two Gantt charts and the line chart are aligned for comparison.

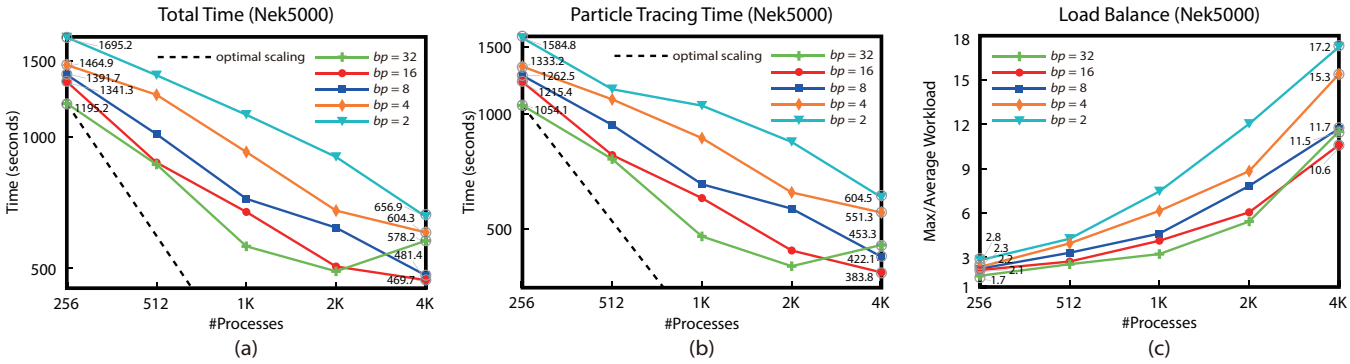

Figure 8: Strong-scaling tests using the Nek5000 data with different numbers of processes to evaluate the parameter bp. Panels (a), (b), and (c) show the test results with different bp under tracing depth of 4.

work, we use the maximal workload divided by the average workload as the load-balancing indicator. If this indicator is closer to 1, the load is more balanced. From the Gantt charts and line chart in Figure 7, we can see that in each round of particle tracing, the workload is more balanced in our method than in the baseline. The Gantt chart of the baseline method shows that processes 0 and 26 are always busy computing while others are relatively idle during the whole procedure. We further visualized the workload distribution of each block in Figure 7(c), which shows that the block workloads are nonuniform, with two peaks in blocks 0 and 224. We checked these two blocks and found vortices in it. The vortices trap particle locally, leading to more integration steps to finish the tracing in each round. The baseline round-robin assignment cannot distribute workload balanced to processes. In contrast, our dynamic repartitioning with data duplication distributes the workload more evenly in each round of particle tracing, as demonstrated in the corresponding Gantt chart.

Parameter bp

We also evaluated the parameter bp by comparing the timings and load balance. In this test, we used a tracing depth of 4 to show the effect of the number of total blocks on the performance.

As shown in Figure 8, the running time (either the total time or only the particle tracing time) decreases continually and the workload becomes more balanced as bp increases, except when bp is 32 with 4K processes. With a larger number of total blocks, it is more likely to have balanced repartitioning, leading to better performance compared with coarser granularity. A smaller bp indicates a larger block size, which may increase the communication overhead due to data migration after repartitioning. As bp continues to increase, however, the frequency of data repartitioning will also increase,

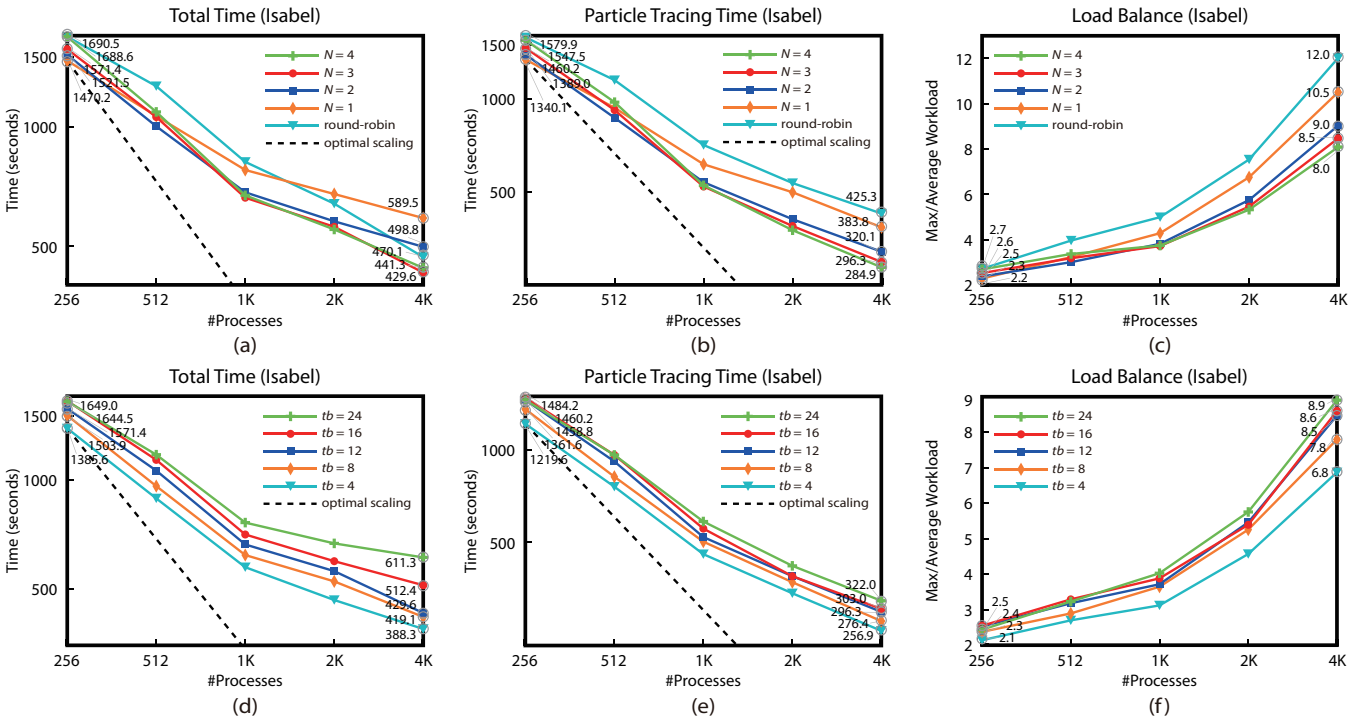

Figure 9: Strong-scaling tests using the Isabel data with different numbers of processes. The performance benchmarks include the total running time, particle tracing time, and the overall level of load balance. Panels (a), (b), and (c) show the test results with different tracing depths, while panels (d), (e), and (f) show the comparison results using different numbers of time intervals (i.e., parameter tb) under a tracing depth of 3.

which induces larger overhead associated with the repartitioning. A smaller block size (i.e., a larger bp) also results in a smaller size of messages communicated among processes, which usually leads to worse bandwidth. In Figure 8, we can see that the performance does not improve from 2K to 4K processes when bp is 32. The reason is that with larger bp, the blocks become too small so that there is not enough computation of particle tracing, which affects the accuracy of block workload estimation. Combing these factors, we conclude that bp should also not be too large.

5.2 Pathline Computation: Hurricane Isabel Data

The Isabel data is from a hurricane simulation conducted by the National Center for Atmospheric Research in the United States. The spatial resolution of this data is $500 \times 500 \times 100$, representing a physical range of $2,139$ km $\times$ $2,004$ km $\times$ 19.8 km. The data has 48 time steps, which are saved per hour in separate files. The total data size is about 13.4 GB. Figure 5(b) shows the rendering of pathlines by this data. We traced 54 million particles that were densely seeded in the domain.

Strong Scaling

The strong-scaling tests are shown in the first row of Figure 9. We can see that our method has better computational efficiency and load balance compared with the baseline method. As the number of processes increases, the performance gain with larger tracing depths becomes more significant. When the tracing depth is 1 with larger process counts, the total time in our method is more than that in the baseline because of the additional costs caused by data repartitioning. But with larger tracing depth, the performance gain brought from data repartitioning becomes larger than the additional costs, resulting in improved overall performance. Considering only the particle tracing time in Figure 9(b), the improvement in efficiency is more obvious. We also found that the performance gain in this test is not as large as that with the Nek5000 data. In the best case, our method with a tracing depth of 4 saves 33.0% time in particle tracing compared with the baseline method. As shown in Figure 9(c), the overall level of load balance between our method and the baseline method is closer than that in Figure 8(c). This is because the workload distribution on blocks in the Isabel data is more even than that in the Nek5000 data, so the performance gap between the baseline round-robin method and our method is narrowed in this case.

Parameter tb

We also compared the timings and load balance using different numbers of time intervals (i.e., parameter tb). In this test, we used a tracing depth of 3 to evaluate the effect of this parameter. As shown in the second row of Figure 9, with a smaller number of time intervals (i.e., more time steps per time interval), the load is more balanced and the efficiency is improved in parallel particle tracing. The reason is that the average workload of each block in the Isabel data in this test is relatively small. A time interval with more time steps indicates larger blocks partitioned. Hence, the workload of each block also becomes larger, which can have efficient computation and provide more information for the workload estimation of the next rounds. However, the number of time intervals should not be too small, because each time larger data will be loaded into the memory as the number of time intervals decreases, which increases the memory overhead. Using the tracing information that is done in much earlier time steps may also have a lower prediction accuracy for the workload estimation in unsteady flow, resulting in a low-quality partitioning result.

6 Discussion

In this section, we discuss the advantages of our method by comparing with other methods. We also describe the limitations of our method.

6.1 Comparison with Other Methods

We first compare our method with the work of Peterka et al. [29], which also studied a dynamic data repartitioning method for parallel

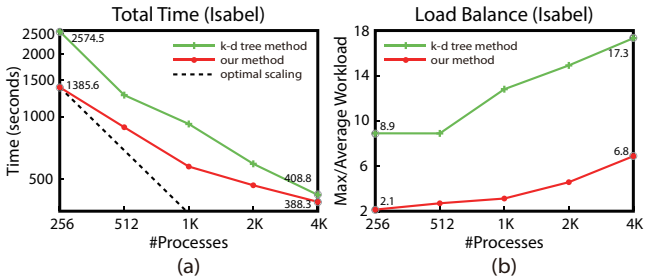

Figure 10: Performance comparison between our method and the k-d tree method [35] using the Isabel data with different numbers of processes.

particle tracing. In that work, only the historical computation workload of each block is used for repartitioning the domain. Tracing particles in steady flow that requires data migration after repartitioning was not implemented. They tested only the case of performing domain repartitioning between time intervals in unsteady flow data. The case studies demonstrated that the additional overheads caused by the updating of data structures in their method may instead lower the overall performance compared with the round-robin method. Our dynamic data repartitioning method significantly improves their method. First, we realize the dynamic data repartitioning for tracing both streamlines and pathlines in steady and unsteady flow data. Data is repartitioned after each round of tracing, instead of only before loading a time interval for unsteady flow. Second, we propose a new dynamic workload estimation method. We estimate the workload of a block based on the historical workload and the number of unfinished particles in it. Moreover, we build an access dependency graph on the fly to predict more blocks (except the originating block) that a particle may travel through in each round (i.e., new round definition with the tracing depth), which controls the frequency of performing repartitioning. The results show significant performance improvement with our method compared with the baseline round-robin method.

We then compare our method with the dynamic load-balancing method proposed by Zhang et al. [35]. In that work, load is balanced through dynamic particle redistribution based on constrained k-d tree decomposition (we refer to it as the k-d tree method). We used the Isabel data and set tb to 4 in our method since it performs best in the experiments. From Figure 10, we can see that our method achieves load balance and efficiency than does the k-d tree method. In the k-d tree method, the splitting planes are constrained by the geometry constraints, and it uses an approximation that assumes that a balanced distribution of particles leads to the ideal balance of workload. When the positions of unfinished particles become localized, the improvement of load balance is also limited. In contrast, our method achieves a balanced workload by freely exchanging data blocks for workload redistribution without constraints. It can balance uneven workload distribution during run time. This can also be seen in Figure 7(a), where the workload in the last few rounds is unevenly distributed. With our method, the workload can still be balanced well.

We also compared our method with other static and dynamic load-balancing methods for parallel particle tracing. We show the merits of our method through the comparison. Specifically, our method does not require data preprocessing and prior knowledge about the seed point distribution and flow features. Existing static load-balancing methods, such as flow graph-based static workload estimation [28] and irregular data partitioning [7, 34], require a time-consuming preanalysis stage on the input raw data before the run-time parallel particle tracing. In order to achieve balanced workload distribution for static data partitioning, the preprocessing stage should also integrate the initial seed distribution and rely on flow feature analysis. This induces additional costs and thus impacts the overall performance of parallel particle tracing. Our method achieves load balance dynamically and thus does not have these requirements.

Moreover, our method does not require a dedicated process for task scheduling during run time. For some dynamic load-balancing methods, including work stealing [11] and work requesting [26], a center process is needed to schedule the workload of each computing process. The communication overhead is significantly increased because the scheduling mode in these methods induces frequent information requesting and sending, which also becomes a bottleneck and thus limits the scalability of these methods. Our method does not use this 1-to-n pattern during communication. By traveling multiple blocks for particles in each round and allowing data duplication, the cost of information exchange in our method is lowered.

6.2 Limitations

We discuss the limitations of our method in order to point out the application scope and possible improvements for future work.

Our method is better suited for densely seeded full-range analysis. The dynamic estimation of workload relies on on-the-fly access dependency construction. With sparse seeding, it cannot provide accurate predictions and thus may affect the efficiency of data repartitioning. Moreover, historical tracing information of blocks is required in our dynamic workload estimation. For local-range analysis, the workload of the data blocks that will be newly accessed cannot be estimated.

Our method is a bulk-synchronous implementation. Each time when performing the data repartitioning and subsequently the data migration and particle exchange, all processes are involved in a synchronous manner for a collective execution. As the number of processes increases, the cost of synchronization also rises, which affects the scalability of our method. This is also the reason that the difference of total running time between our method and the k-d tree method in Figure 10 becomes not so significant as the number of processes increases. The communication in our work currently cannot be overlapped with the computation.

7 Conclusions and Future Work

In this work, we presented a dynamic data-repartitioning algorithm for load balancing in parallel particle tracing. The whole particle tracing process is decomposed into multiple rounds. Based on the dynamic workload estimation on each block, we dynamically repartition the domain with data duplication and redistribute the new data partitions to processes. Load balance is achieved because the workload of each new partitions is balanced in each round of particle tracing. We evaluated our method by conducting a comprehensive performance study with both steady and unsteady flow data. Results demonstrate that our method improves load balance and efficiency in parallel particle tracing.

We would like to extend our method to support unstructured mesh data for more generalized applications. We would also like to integrate our method with existing load-balancing methods to work on various flow visualization and analysis problems, including both full- and local-range seeding. To further improve the flexibility and scalability, we plan to develop an asynchronous realization of the parallel data repartitioning instead of using synchronizations.

Acknowledgments

We thank the anonymous reviewers for their valuable comments. This work is supported by NSFC No. 61672055, the National Program on Key Basic Research Project (973 Program) No. 2015CB352503, and the National Key Research and Development Program of China (2016QY02D0304). This work is also supported by PKU-Qihoo Joint Data Visual Analytics Research Center and the U.S. Department of Energy, Office of Science, under contract number DE-AC02-06CH11357.

REFERENCES

[1] S. S. Barakat and X. Tricoche. Adaptive refinement of the flow map using sparse samples. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2753–2762, 2013.

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

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

[4] Ü. V. Çatalyürek, E. G. Boman, K. D. Devine, D. Bozdag, R. T. Heaphy, and L. A. Riesen. Hypergraph-based dynamic load balancing for adaptive scientific computations. In *IPDPS07: Proceedings of 21th IEEE International Parallel and Distributed Processing Symposium*, pp. 1–11, 2007.

[5] 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*, pp. 109–112, 2012.

[6] 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*, pp. 145–152, 2012.

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

[8] J. M. Dennis. Partitioning with space-filling curves on the cubed-sphere. In *IPDPS03: Proceedings of 17th IEEE International Parallel and Distributed Processing Symposium*, p. 269, 2003.

[9] K. D. Devine, E. G. Boman, R. T. Heaphy, B. Hendrickson, and C. T. Vaughan. Zoltan data management services for parallel dynamic applications. *Computing in Science and Engineering*, 4(2):90–96, 2002.

[10] K. D. Devine, E. G. Boman, R. T. Heaphy, B. A. Hendrickson, J. D. Teresco, J. Faik, J. E. Flaherty, and L. G. Gervasio. New challenges in dynamic load balancing. *Applied Numerical Mathematics*, 52(2-3):133–152, Feb. 2005.

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

[12] 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.

[13] 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.

[14] 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*, pp. 33–40, 2014.

[15] 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.

[16] 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.

[17] W. W. Hager and Y. Krylyuk. Graph partitioning and continuous quadratic programming. *SIAM Journal on Discrete Mathematics*, 12(4):500–523, oct 1999.

[18] M. Hlawatsch, F. Sadlo, and D. Weiskopf. Hierarchical line integration. *IEEE Transactions on Visualization and Computer Graphics*, 17(8):1148–1163, 2011.

[19] H. D. S. I, H. D. Simon, and H. D. Simon. Partitioning of unstructured problems for parallel processing. In *Computing Systems in Engineering*, vol. 2, pp. 135–148, 1991.

[20] G. Karypis and V. Kumar. Parallel multilevel k-way partitioning scheme for irregular graphs. In *Proceedings of the 1996 ACM/IEEE Conference on Supercomputing*, p. 35, 1996.

[21] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, dec 1998.

[22] 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*, pp. 10:1–11, 2011.

[23] 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.

[24] 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*, pp. 1008–1019, 2014.

[25] 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.

[26] 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*, pp. 1–6, 2013.

[27] 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*, pp. 61:1–11, 2012.

[28] 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.

[29] 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*, pp. 580–591, 2011.

[30] 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.

[31] T. Salzbrunn, C. Garth, G. Scheuermann, and J. Meyer. Pathline predicates and unsteady flow structures. *The Visual Computer*, 24(12):1039–1051, 2008.

[32] T. Salzbrunn and G. Scheuermann. Streamline predicates. *IEEE Transactions on Visualization and Computer Graphics*, 12(6):1601–1612, 2006.

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

[34] 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*, pp. 24:1–12, 2007.

[35] J. Zhang, H. Guo, F. Hong, X. Yuan, and T. Peterka. Dynamic load balancing based on constrained k-d tree decomposition for parallel particle tracing. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):954–963, 2018.

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