Dynamic Load Balancing Based on Constrained K-D Tree Decomposition for Parallel Particle Tracing

Jiang Zhang, Hanqi Guo, *Member, IEEE*, Fan Hong, Xiaoru Yuan, *Senior Member, IEEE*, and Tom Peterka, *Member, IEEE*

Abstract—We propose a dynamically load-balanced algorithm for parallel particle tracing, which periodically attempts to evenly redistribute particles across processes based on k-d tree decomposition. Each process is assigned with (1) a statically partitioned, axis-aligned data block that partially overlaps with neighboring blocks in other processes and (2) a dynamically determined k-d tree leaf node that bounds the active particles for computation; the bounds of the k-d tree nodes are constrained by the geometries of data blocks. Given a certain degree of overlap between blocks, our method can balance the number of particles as much as possible. Compared with other load-balancing algorithms for parallel particle tracing, the proposed method does not require any preanalysis, does not use any heuristics based on flow features, does not make any assumptions about seed distribution, does not move any data blocks during the run, and does not need any master process for work redistribution. Based on a comprehensive performance study up to 8K processes on a Blue Gene/Q system, the proposed algorithm outperforms baseline approaches in both load balance and scalability on various flow visualization and analysis problems.

Index Terms—Parallel particle tracing, dynamic load balancing, k-d trees, performance analysis

1 Introduction

Distributed and parallel particle tracing, which computes the movements of many massless particles that are released in the flow field, is a fundamental technique in large-scale flow visualization and analysis. Applications include visualizing streamlines and pathlines, generating streamsurfaces [9], computing finite-time Lyapunov exponents (FTLEs) and Lagrangian coherent structures (LCSs) [11], studying teleconnections [17], and analyzing differences in numerical ensembles [13]. Parallel particle tracing enables these analyses to be run on clusters or supercomputers and to handle large-scale data generated from computational fluid dynamics, combustion, climate, weather, and biomedical simulations.

We focus on the load balance—a known hard problem and key to achieving scalability—in parallel particle tracing. The load-balancing problem exists in both task-parallel and data-parallel particle tracing methods, which are the two basic parallelism strategies. In the task-parallel methods, particles are statically distributed to parallel processes; each process has access to the whole data. Processes are unbalanced because of the early termination of some particles that travel out of the domain or hit critical points where the velocity is zero. In the data-parallel methods, the flow data are statically partitioned into blocks and distributed to processes; particles exchange between processes to finish the tasks. Processes are unbalanced because some blocks may contain more complex features than others, such as vortices that trap particles locally.

In this paper, we propose a dynamically load-balanced algorithm for parallel particle tracing using k-d (short for k-dimensional) trees. Our motivation is based on the successful use of k-d trees to balance workloads in N-body simulations [6], Delaunay tessellations [23], clustering [10], and sort-first parallel rendering [21]. In these applications, k-d trees are used to evenly (re)distribute particles, data points, or pixels across parallel processes. As illustrated in a four-process run (Figure 1(a)), each process has a k-d tree leaf node, and particles are evenly distributed into different processes periodically. However, the redistribution requires full data duplication over all parallel processes, because processes may have particles that are located anywhere in the data domain, making the algorithm not scale on supercomputers.

We overcame this problem by a novel redesign of k-d tree decomposition, namely, the *constrained k-d tree*, to redistribute particles in the data-parallel particle tracing. In this design, each process is assigned with (1) a statically partitioned, axis-aligned data block that partially overlaps with neighboring blocks in other processes and (2) a dynamically determined k-d tree leaf node that bounds the active particles for computation. The bounds of k-d tree nodes are constrained by the geometries of the data blocks. In static data partitioning, we initially subdivide the domain into n non-overlapping, equal-sized, and axis-aligned blocks, where n equals the number of processes. We then expand the blocks to overlap with other blocks as much as possible, given the memory limit of the process. The expanded parts, called ghost layers, essentially maximize the overlaps between blocks under the memory limit and thus enable the k-d tree decomposition with constraints. During run time, we periodically redistribute particles based on the constrained k-d tree decomposition to balance the workload. The splitting planes in the k-d tree decomposition are limited to the overlapped regions of the ghost layers. Thus the decomposition ensures that the redistributed particles are inside the bounds of the corresponding blocks, as shown in Figure 1(b). In the extreme case that each process can fit the whole data, our algorithm can evenly redistribute the particles because there are no constraints.

The proposed method can improve load balancing in both full- and local-range flow visualization and analysis techniques for steady and unsteady flows. Full-range analyses such as FTLE computation need to trace densely-seeded particles over the whole domain, while local-range analyses such as source-destination queries place seeds only locally in the domain. The constrained k-d tree can be used for both static and time-varying flows in both 2D and 3D meshes.

We evaluate our method with various flow visualization and analysis tasks on Vesta, a Blue Gene/Q supercomputer at Argonne National Laboratory. We show the performance benchmarks with up to 8K parallel processes. Compared with the baseline data-parallel particle tracing method, our constrained k-d tree approach significantly improves the performance in both load balancing and scalability. Compared with other load-balancing algorithms for parallel particle tracing, the proposed method does not require any preanalysis, does not use any heuristics based on flow features, does not make any assumptions

• *Jiang Zhang, Fan Hong, and Xiaoru Yuan are with Key Laboratory of Machine Perception (Ministry of Education), School of EECS, Peking University. E-mail: {jiang.zhang, fan.hong, xiaoru.yuan}@pku.edu.cn.*
• *Hanqi Guo and Tom Peterka are with the Mathematics and Computer Science Division, Argonne National Laboratory, Lemont, IL 60439, USA. E-mail: hguo@anl.gov, tpeterka@mcs.anl.gov.*

Manuscript received 31 Mar. 2017; accepted 1 Aug. 2017.
Date of publication 28 Aug. 2017; date of current version 1 Oct. 2017.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TVCG.2017.2744059

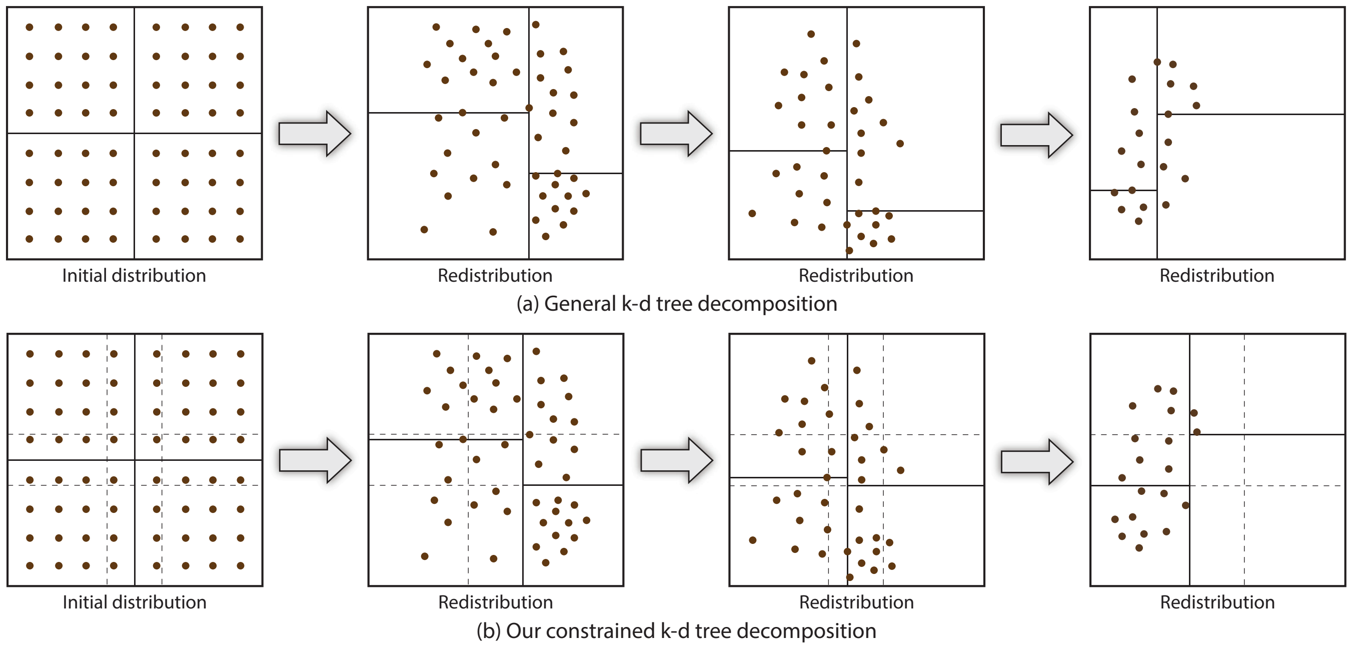

Fig. 1. Example of particle redistribution during parallel particle tracing through the general k-d tree decomposition and our constrained k-d tree decomposition, respectively. In (a) and (b), the solid lines split the particles as balanced as possible. In (b), the dashed lines show the overlapped regions of ghost layers that limit the splitting lines.

about initial particle distribution, does not move any data blocks during the run, and does not need any master process for work redistribution.

In summary, the contributions of this paper are twofold:

- A dynamically load-balanced parallel particle tracing algorithm based on the redesigned k-d tree decomposition that is constrained by static data partitions;
- A comprehensive performance study of our algorithm with different datasets on a Blue Gene/Q system.

In Section 2 we briefly introduce the background of this work. In Section 3, we give an overview of our method. The initialization and run-time load-balanced parallel particle tracing are described in Section 4 and Section 5, respectively. A detailed performance analysis is given in Section 6 to demonstrate the effectiveness of our method, followed by discussion in Section 7. In Section 8, we conclude our paper and discuss future work.

2 Related Work

In this section, we review related work on load balancing in parallel particle tracing and on load-balancing algorithms based on k-d trees. We also review advection-based flow visualization and analysis applications.

2.1 Load Balancing in Parallel Particle Tracing

Load-balanced parallel particle tracing in large-scale flow data is difficult because of the varying computation and communication workloads in both task- and data-parallel methods.

In task-parallel methods, particles are statically distributed to processes. The workload among the processes is unbalanced because the particles have different trajectories. Work stealing and work requesting [8, 24] can be used to redistribute tasks among processes. These methods, however, require a central master process for scheduling, which becomes a bottleneck to scale up. Moreover, task-parallel methods usually have higher I/O cost than data-parallel methods have, because processes need to load data out of core when the data is larger than the memory in each process. Our method is instead based on data-parallel methods. Compared with the task-parallel load-balancing methods [8, 24], our method has much lower I/O overhead, and we do not need any master processes to redistribute tasks.

In data-parallel methods, the input data is partitioned and distributed to processes statically; particles are exchanged between processes to finish the tasks. Both static and dynamic load-balancing algorithms are proposed for data-parallel particle tracing. Static load-balancing algorithms usually require data preprocessing and flow feature analysis. Nouanesengsy et al. [26] proposed a method to construct flow graphs that characterizes both workload and flow directions of the data in preprocessing and then statically distributes the data blocks based on the optimized partitioning of flow graphs. Chen and Fujishiro [7] instead partitioned the flow data in irregular shapes based on flow feature extraction. The irregular partitions can also be achieved by hierarchical clustering [34]. Dynamic load-balancing algorithms redistribute and move data blocks periodically during the run. The major costs are twofold: from the scheduling algorithm that decides how blocks should be redistributed and from the data movement. For example, Peterka et al. [28] used a recursive coordinate bisection to repartition the distribution of data blocks. The repartitioning is performed periodically, and data blocks are moved on the fly. Lu et al. [19] proposed a work-stealing approach to achieve dynamic load balancing in streamsurface generation. Data blocks are moved to the processes that have lower workload.

To improve scalability, researchers have proposed hybrid methods that combine task and data parallelism, but the load-balancing issues are not well studied in these methods. For example, Kendall et al. [17] proposed the DStep framework that schedules tasks in a multitiered manner based on static data partitioning. DStep has also been extended to visualize ensemble differences by pathline advection analysis [13] and to analyze multivariate unsteady flow data with Lagrangian-based attribute space projection [12]. Zhang et al. [36] also modified the DStep framework to generate high-order access dependencies in a data-preprocessing stage.

We regard our load-balanced particle tracing algorithm as a hybrid method that distributes data blocks statically and redistributes particles dynamically. Comprehensive comparisons with previous methods are discussed in Section 7.

2.2 Load Balancing Based on K-D Trees

A comprehensive review of load balancing for distributed and parallel computing in general is out of the scope of this paper. We instead focus on load-balancing algorithms that are based on k-d trees.

The k-d tree, which was invented by Bentley [4], is a data structure

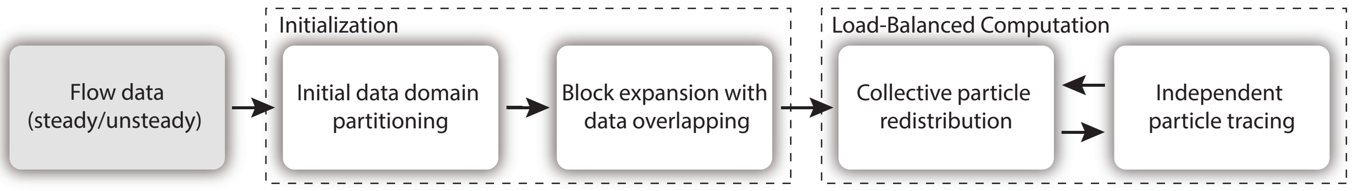
Initialization
Load-Balanced Computation
Flow data (steady/unsteady)
Initial data domain partitioning
Block expansion with data overlapping
Collective particle redistribution
Independent particle tracing

Fig. 2. Workflow of our load-balanced parallel particle tracing method. The raw flow data is first partitioned into blocks in the initialization stage. These blocks are further expanded by adding ghost layers on them to maximize the overlap with other blocks as much as possible. In the computation stage, collective particle redistribution and independent particle tracing are alternately executed to achieve load balance dynamically.

that splits k-dimensional data for efficient range queries and k-neighbor queries. The k-d tree data structure is essentially a balanced binary tree; each leaf node of the tree is a subdomain of its parent.

K-d trees have been used in load balancing of various applications in computational sciences, database, data analysis, and visualization. In computational sciences, k-d trees are used to build hierarchies of particles in order to improve scalability in large N-body cosmology simulations [3,6] and molecular dynamics simulations [31]. In database applications, k-d trees can be used for fast query of images [1]. In data analysis, k-d trees are widely used in distributed and parallel clustering algorithms, such as k-means [10] and the DBSCAN algorithm [27]. In visualization, the community has been extensively using k-d trees to address various load-balancing problems. For example, Morozov and Peterka [23] proposed a load-balanced Delaunay tessellation algorithm. Input points are partitioned with k-d trees and distributed to different processes for load-balanced computation. The positions of these points are statically fixed when performing the computation of Delaunay tessellation. In sort-first volume rendering, k-d trees are used to split the image space to balance the parallel rendering costs [21]. Bounding volume hierarchies [33] or particle distribution schemes based on z-ordering [30] that are similar to k-d trees are also used in parallel volume rendering. The path of each ray in these volume-rendering methods is determined by its initial position. On the contrary, in particle tracing, the positions of the particles always change over time, making it difficult to achieve load balance statically. Our method addresses this problem by dynamically performing k-d tree decomposition during run time to redistribute particles.

The implementation of k-d trees must be efficient and scalable for distributed applications. Serial k-d tree implementations include `nanoflann`[1], but they are not able to scale in parallel. The previous version of our work [37] used `nanoflann` to implement k-d tree decomposition. Zhang et al. [35] proposed the first scheme to distribute k-d trees in peer-to-peer systems. Morozov and Peterka [23] further improved the algorithm to scale k-d tree decomposition for efficient Delaunay tessellation.

In our study, we redesigned the parallel tree construction algorithm for constrained k-d trees. We further use the constrained k-d trees to balance the number of particles in parallel processes in order to achieve dynamic load balancing.

2.3 Advection-Based Flow Visualization and Analysis

We review advection-based flow visualization and analysis applications that can benefit from scalable and parallel particle tracing. Advection-based flow visualization and analysis methods include texture-based [18], geometry-based [20], and part of feature extraction and tracking techniques [29]. Based on the distribution of particle seed locations, we subdivide the techniques into two categories: full-range analysis and local-range analysis.

In local-range analyses, particles are seeded in sparse and local regions in order to understand part of the data. For example, seeds of a streamsurface are usually distributed on a line or curve in 3D flows [9]. A source-destination query [17] needs to seed particles only in a local region, instead of the whole domain.

In full-range analyses, particle are seeded densely in the entire data domain in order to analyze and visualize comprehensive features in the flow. For example, in texture-based visualization, line integral convolution (LIC) [5] and unsteady flow LIC (UFLIC) [32] need to compute streamlines and pathlines, respectively, from all locations. Although methods have been proposed to reduce the computational cost of tracing densely-seeded particles, such as partial path reuse [15] and adaptive refinement [2], comprehensive analysis of the computation of FTLEs and LCSs [14] still needs scalable and parallel tracing of densely-seeded particles [25].

Our study can be used to improve load balancing in both full- and local-range analyses. Our method is also compatible with both static and time-varying flows. More details and evaluations are in the following sections.

3 Overview

As described in Algorithm 1 and illustrated in Figure 2, the pipeline of our method consists of a initialization stage and a computation stage.

The initialization stage has several steps. We first partition the input data into non-overlapping, equal-sized, axis-aligned blocks and then expand the ghost layers of each block up to the memory limit of each process. The blocks are loaded into memory, and the particles are initialized in the corresponding blocks. More details are explained in Section 4.

The computation stage is an iterative process that alternately executes the particle redistribution and particle tracing. In the particle redistribution phase, the parallel processes collectively exchange unfinished particles based on the constrained k-d tree decomposition. In the particle tracing phase, each parallel process independently traces its unfinished particles without communication. More details are given in Section 5.

Algorithm 1 Main function of each parallel process in our method, where comm is the communicator, and local_block and local_particles are the block and particles that are distributed to the process, respectively.

```
initialize(comm, local_block, local_particles)                ▷ Section 4
while !all_done do
    redistribute_particles(comm, local_particles)           ▷ Section 5.1
    trace_particles(local_block, local_particles)           ▷ Section 5.2
end while
```

4 Algorithm Initialization

We initialize data blocks and particles for the dynamically load-balanced parallel particle tracing.

Domain partitioning We partition the input data by iteratively splitting the domain in each dimension; the order of dimensions is consistent with that of k-d tree decomposition. Without loss of generality, we assume the number of processes n is a power of 2; otherwise we can subdivide each dimension based on the prime factorization of the number. For example, in 3D data we evenly subdivide the domain along the x-axis into two blocks and then further evenly split each block along the y and z axes. After the z-axis, the next dimension to subdivide is the x-axis. The iteration stops when the number of blocks equals n. Each process then owns single block. The outputs of the domain partitioning are n non-overlapping, equal-sized, axis-aligned data blocks.

Block expansion We maximize the overlaps between blocks by adding ghost layers on these blocks. As shown in Figure 3, each block is expanded in all dimensions so that the block overlaps with its neighbor

[1]https://github.com/jlblancoc/nanoflann

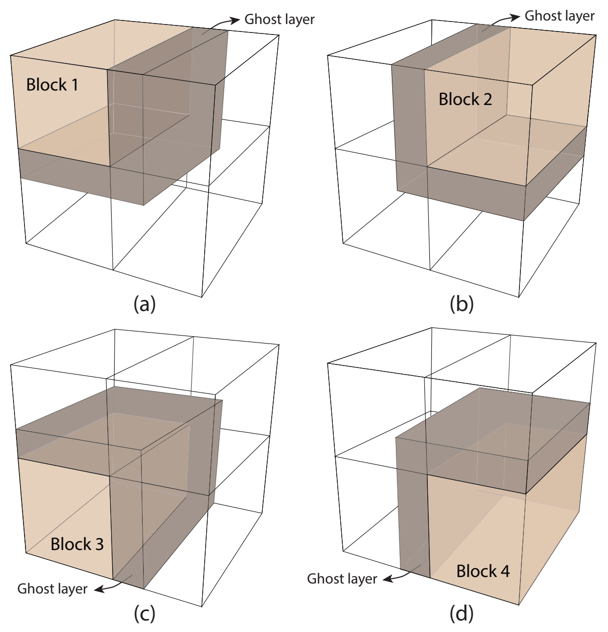

Fig. 3. Illustration of four expanded blocks in 3D.

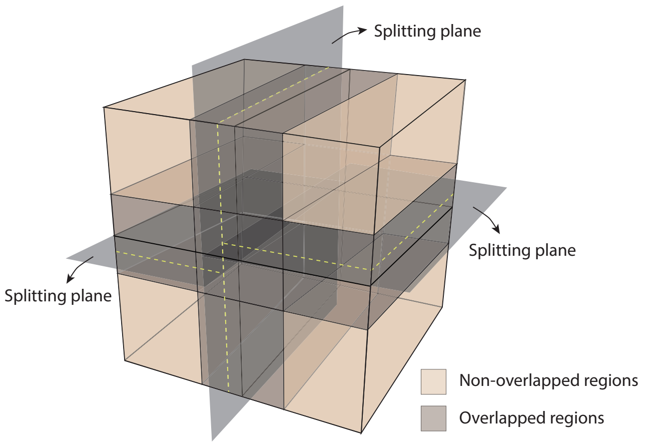

Fig. 4. Illustration of constrained k-d tree decomposition in 3D. The splitting planes are constrained in the overlapped regions of ghost layers.

blocks. The block expansion enables the k-d tree to split the domain in the overlapped regions, as explained in the next section. The thickness of the block is bounded by the available memory in the process. In the extreme case when the memory is large enough to fit the whole dataset, each process keeps a complete copy of the data.

Block I/O We load the expanded data blocks in parallel. The parallel I/O is handled by the block I/O layer (BIL) [16], which essentially uses parallel I/O to load expanded data blocks in a scalable manner.

Particle initialization We load input particles and distribute them into different processes. Each particle is assigned to the block whose "core" region excluding ghost layers contains the particle.

5 Load-Balanced Parallel Particle Tracing

The computation stage (the loop in Algorithm 1) of our algorithm alternates the collective particle redistribution and the serial particle tracing for dynamic load balancing.

5.1 Particle Redistribution Based on Constrained K-d Trees

Constrained k-d tree construction is illustrated in Figures 4 and 5. The algorithm cycles through each dimension to split the data domain. For each dimension, the algorithm attempts to find an axis-aligned median plane that can split particles evenly. If the plane is not in the overlapped regions between the two neighbor blocks, we use the boundary of either block that can balance the number of particles. As shown in the 2D example in Figure 5, if the coordinate of the plane is greater than the upper bound of the red block in x direction, we split the dimension with the right boundary of the red block; if the coordinate of the plane is less than the lower bound of the blue block in x direction, we split the dimension with the left boundary of the blue block; otherwise we choose the median plane in the overlapped region of the two blocks. The rule is similar in 3D, as illustrated in Figure 4.

We redesign the distributed k-d tree decomposition [23] to constrain the splitting planes, as detailed in Algorithm 2. Each iteration has two key parts: median selection and particle exchange. Initially, we group all processes in a group. In the first iteration, each process computes a local histogram of its particles and sends it to a designated root process within the group for gathering. The root process then picks up a median plane that is constrained in the overlapped regions, as described above, and broadcasts it to all other processes within the group. The selected median value splits all the particles into two parts, trying to make each part have particles as even as possible under the geometry constraints. For particle exchange, the processes are divided into two subgroups. Each process from one subgroup chooses a partner from the other subgroup to exchange particles, so that half the processes receive the particles whose projection onto the first coordinate is less than the picked median value, while the other half receives particles whose projection is more than the value. Given n processes, the algorithm will repeat in each process subgroup by cycling through the coordinates for splitting until $\log_2 n$ iterations are executed.

Algorithm 2 redistribute_particles(comm, local_particles)

```
dim ← 0
rank ← comm.rank
domain ← data.domain                    ▷ domain of the input flow data
group ← $\{0, 1, \ldots, n-1\}$              ▷ n is the number of processes
$N_r \leftarrow log_2 n$                            ▷ number of iterations
for $i < N_r$ do
    root ← group[0]              ▷ rank of the root process in the group
    local_histogram ← compute_local_histogram(local_particles)
    global_histogram ← comm.gather(group, root, local_histogram)
▷ gather to the root
    if rank = root then
        range ← compute_overlapped_region(domain, dim)            ▷
overlapped region of ghost layers
        median ← compute_bounded_median(global_histogram,
range, dim)
    end if
    median ← comm.broadcast(group, root, median)    ▷ broadcast
from the root
    partner ← root + (rank + group.size/2) mod group.size   ▷ rank
of the partner process in the group
    particles_low ← $\{p \in$ local_particles $\mid p[\text{dim}] <$ median$\}$
    particles_up ← $\{p \in$ local_particles $\mid p[\text{dim}] \geqslant$ median$\}$
    comm.sendrecv(rank, partner, particles_low, particles_up)      ▷
swap particles
    if rank < group.size/2 then                      ▷ bisect the group
        local_particles ← particles_low
        group ← $\{x \in$ group $\mid x <$ group.size/2$\}$
    else
        local_particles ← particles_up
        group ← $\{x \in$ group $\mid x \geqslant$ group.size/2$\}$
    end if
    domain ← bisect_domain(domain, median, dim)
    dim ← (dim + 1) mod $k$        ▷ k is the dimensionality of k-d tree
    $i \leftarrow i + 1$
end for
```

The constrained k-d tree decomposition splits the particles as evenly as possible while also ensuring that the particles are bounded in the data block of the corresponding process. A thicker ghost layer leads to more even decomposition of the number of particles and better workload

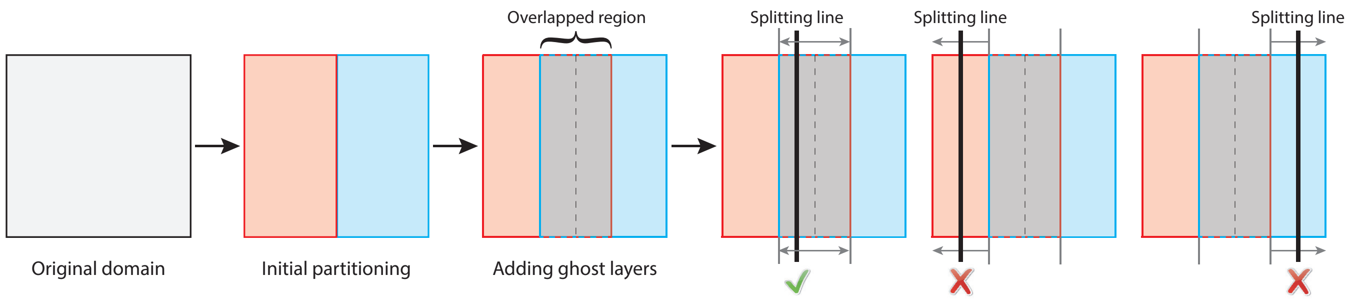

Fig. 5. 2D example to show constrained k-d tree decomposition with data overlapping between blocks. The splitting line is limited in the overlapped region of ghost layers.

balance.

5.2 Particle Tracing within Processes

We trace particles inside the local block of each process independently without communication, as shown in Algorithm 3. After tracing, each particle is either finished or unfinished. A particle is marked as finished if it goes out of the data domain, hits a critical point where the velocity is zero, or reaches the maximum integral steps required by the visualization and analysis algorithm; otherwise the particle is marked as unfinished. Unfinished particles will be redistributed in the next cycle, as detailed in the preceding subsection.

Algorithm 3 trace_particles(local_block, local_particles)

```
for each particle $p$ in local_particles do
    $N \leftarrow 0$
    while $N \leqslant N_{max}$ do
        $p \leftarrow$ RK4(local_block, $p$)  ▷ one single step of fourth-order Runge-Kutta numerical integration
        $N \leftarrow N+1$
        if check_finish($p$) then ▷ check if the particle goes out of the domain or hits a critical point
            local_particles.remove($p$)
            finish_particle($p$)
        else if !local_block.contains($p$) then  ▷ check if the particle goes out of the local block
            break
        end if
    end while
end for
```

We limit the maximal number of integration steps (N_{max}) that a particle can be traced in the cycles. This is an important parameter in our method because it indirectly defines the frequency of particle redistribution. If the redistribution is performed too frequently, the workload can be more balanced, but the redistribution cost will increase. On the contrary, if the redistribution is performed infrequently, our method cannot sufficiently balance the workloads. More discussion and evaluation on N_{max} are in Sections 6 and 7, respectively.

6 Performance Analysis

We conducted a comprehensive performance study with three applications: tracing densely-seeded streamlines in thermal hydraulics simulation (Nek5000) data, querying source-destinations in GEOS-5 simulation data, and computing FTLE of Hurricane Isabel data. The specifications of the data are detailed in Table 1 and the following subsections.

The implementation of our method is based on C++11. We use the DIY2 library [22], which wraps MPI for interprocess communication, to decompose the domain and exchange messages between processes. We also use the BIL library [16] to efficiently read disjoint data blocks from different NetCDF files collectively.

The benchmark platform is Vesta, an IBM Blue Gene/Q system at Argonne National Laboratory. Vesta has 2,048 compute nodes: each has a 16-core 1.6 GHz PowerPC A2 processor and 16 GB of DDR3 RAM. The interconnection between compute nodes is a 5D torus network. The storage of Vesta is GPFS (IBM General Parallel File System). In our test, we execute 8 processes per compute node, and we use up to 8,192 processes. We study the performance of our method with the following benchmarks.

- **Load balancing** We use the maximal workload divided by the average workload as the load-balancing indicator. The workload in each process is evaluated as the number of numerical integral steps for particle tracing. The load is more balanced if the indicator is closer to 1. We also use a Gantt chart to visualize the computation time of different processes.
- **Strong scaling** We measure how the total execution time varies with the number of processes for a fixed total number of particles. The strong scaling indicates the efficiency of our method by distributing a constant-size problem on all processes. A linear speedup is optimal.
- **Weak scaling** We measure how the total execution time varies with the number of processes for a fixed number of particles per process. The weak scaling indicates the efficiency of our method by assigning each process a constant-size problem. An optimal weak scaling is achieved if the total execution time is constant as the number of processes increases.

We also alter the conditions that affect performance:

- **Available memory** (M). We arbitrarily choose different memory limits for data blocks per process to test the scalabilities. In our work, the memory limit indicates the available memory for accommodating the loaded data block. The thickness of the ghost layers grows with the memory limit;
- **Choice of 3D/4D trees** (k). We study whether 3D or 4D trees perform better in time-varying flows.
- **Maximum integral steps per cycle** (N_{max}). We study the only parameter that controls the frequency of particle redistribution.

The baseline approach in the performance study is a data-parallel particle tracing implementation. The initialization stage of the baseline approach is the same as in Section 4. The computation stage alternates the independent particle tracing and the collective particle exchange phases. In the first phase, each particle does not stop until it goes out of the local block; in the second phase, particles that move to remote processes are exchanged collectively with `MPI_Alltoall`. The program exits after all tasks in the system are finished.

6.1 Thermal Hydraulics Data

The thermal hydraulics dataset is from the output of the Nek5000 solver, which is a large-eddy Navier-Stokes simulation developed by Argonne National Laboratory. We use a single timestep of the data for the static flow analysis and trace streamlines from all grid points. Figure 6 shows the Gantt chart visualization of a run, and Figure 7 shows the timings of both strong- and weak-scaling results with different memory limits.

Load balance The workload in our method is much more balanced than in the baseline approach. We collect and visualize in Figure 6

Table 1. Datasets and analyses for the performance study.

Dataset	Resolution	Size	Analysis	Seeding	Benchmark	Alternating Conditions
Nek5000	1024^3	12 GB	Streamlines	Full-range	Strong/weak scaling	M
GEOS-5	$288 \times 181 \times 72 \times 24$	1.34 GB	Source-destination	Local-range	Strong scaling	M, N_{max}
Isabel	$500 \times 500 \times 100 \times 48$	13.4 GB	FTLE	Full-range	Strong scaling	M, k

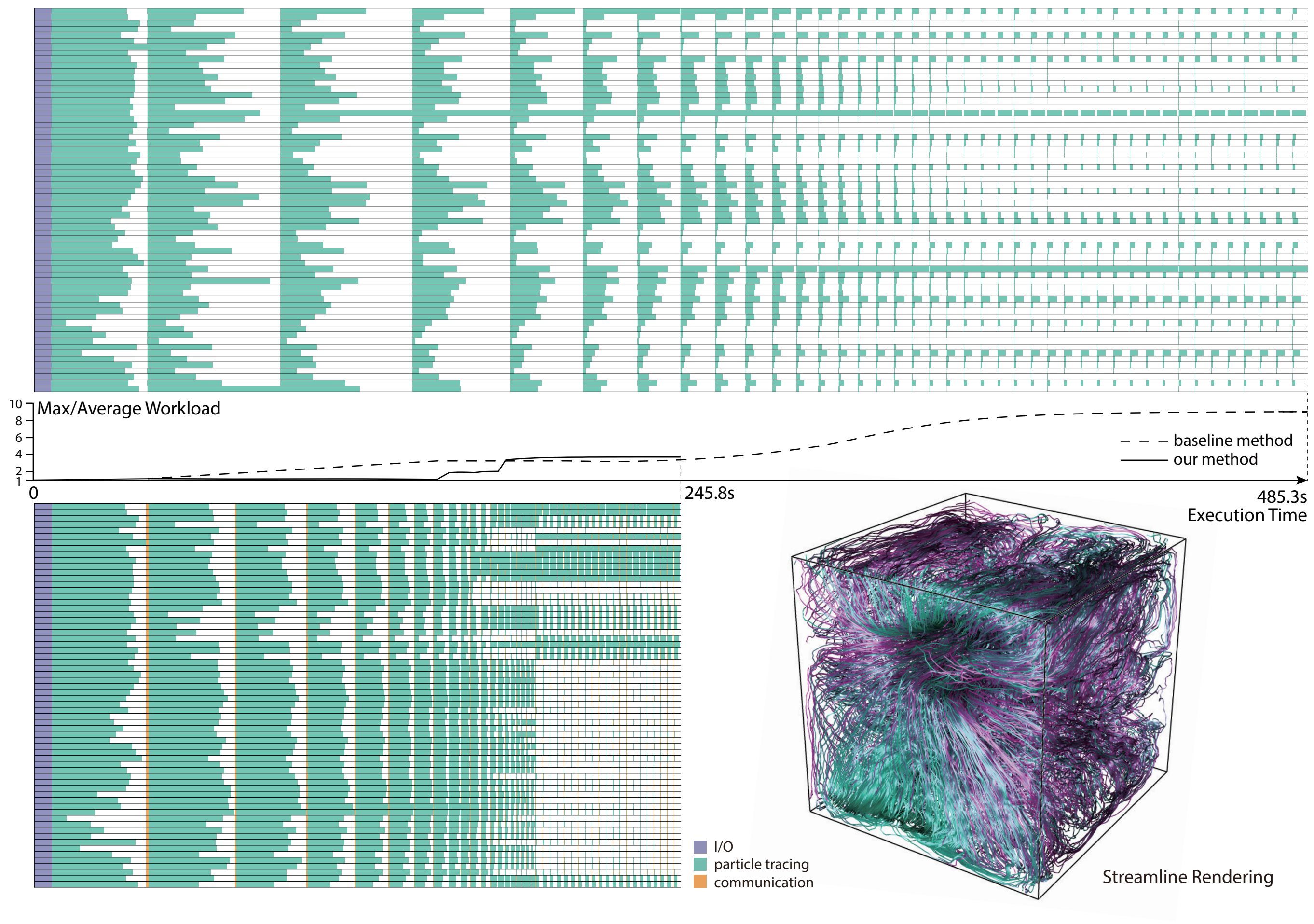

Fig. 6. Performance of 64 processes tested by downsampled Nek5000 data. We record the time for I/O, particle tracing computation, and communication in the Gantt charts and the load balance indicator in the line chart. Top: Gantt chart using the baseline method. Each row in the Gantt chart represents a process. Middle: Line chart showing the evolution of workload balance. The dashed line represents the baseline method, while the solid line represents the k-d tree method. Bottom left: Gantt chart using our k-d tree method. The time axises of the two Gantt charts and the line chart are aligned. Bottom right: Rendering result of the Nek5000 data with 2,000 streamlines. The color encodes the integration steps during the streamline generation.

the performance data from two 64-process runs with our method and the baseline method, respectively. We use the down-sampled version of the original data in order to fit into the small number of processes (64) for Gantt chart visualization. In the Gantt chart, the horizontal axis is the execution time, and the vertical axis is the ID of the process. We also visualize the load balance indicator over the execution time in the line chart of Figure 6. We can see that each time after the k-d tree decomposition is performed, the workload becomes more balanced, and thus the particle tracing takes less time compared with the baseline method. The Gantt chart of the baseline method shows that one or two processes are always busy computing while other processes are idle. We can also see that the particle redistribution time is much less than the particle tracing time.

Strong scaling We trace 128 million particles in total with 512, 1K, 2K, 4K, and 8K processes to study the strong scalability. As shown in Figure 7(a), as the number of processes increases, the total execution time of our method decreases faster than the baseline approach does. The strong scalability of our method is better than the baseline. With 8K processes, our method performs more than twice as fast as the baseline. The parallel efficiency of 2K processes with a 48 MB, 96 MB, and 384 MB memory limit is 46.0%, 48.3%, 76.6%, respectively. In the baseline method, the parallel efficiency is 39.7%, 41.4%, 42.7%, respectively. From Figure 7(b), we can also see that the particle tracing time dominates the execution time, even if the percentage of I/O and particle redistribution increases as the number of processes increases.

Weak scaling We trace 16K particles per process with 512, 1K, 2K, and 8K processes in this test. As the number of processes increases, the execution time of the baseline approach increases much faster than that of our method does, as shown in Figure 7(c). In other words, the weak scalability of our method is also better than that of the baseline. The parallel efficiency of 2K processes with a 48 MB, 96 MB, and 384 MB memory limit in our method is 47.8%, 50.2%, 77.1%, respectively. In the baseline method, the corresponding parallel efficiency is 40.3%, 41.9%, 43.6%, respectively.

From all tests in Figure 7, we can see that our method performs better with larger memory limits. Although larger memory limits lead to thicker ghost layers and eventually result in higher I/O costs, I/O takes less than 10% of the total execution time in all cases. Larger memory

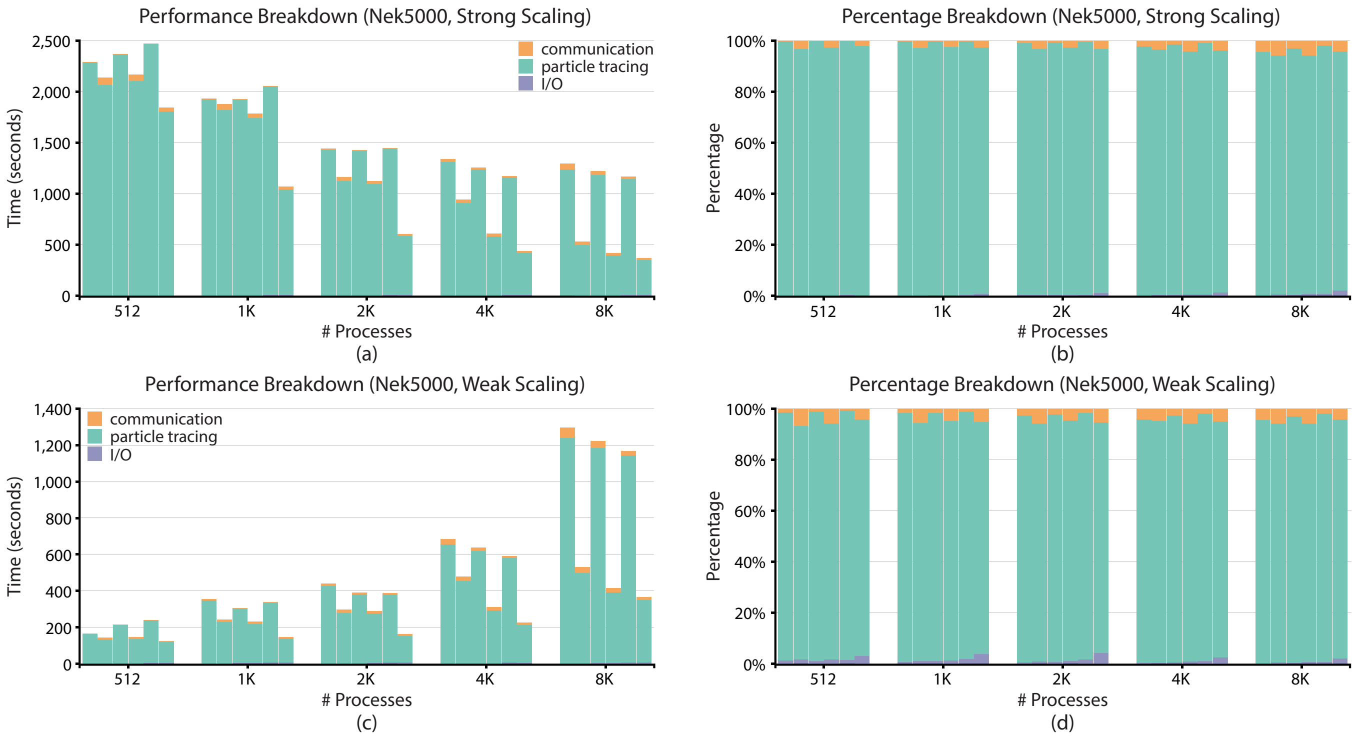

Fig. 7. Performance breakdown for Nek5000 data analysis. (a) and (b) show the strong-scaling tests, while (c) and (d) show the weak-scaling tests. Time for I/O, particle tracing computation, and communication are encoded in different colors to reflect the performance. At each kind of process count, the six stacked histograms represent the baseline method with 48 MB memory, the k-d tree method with 48 MB memory, the baseline method with 96 MB memory, the k-d tree method with 96 MB memory, the baseline method with 384 MB memory, and the k-d tree method with 384 MB memory, respectively.

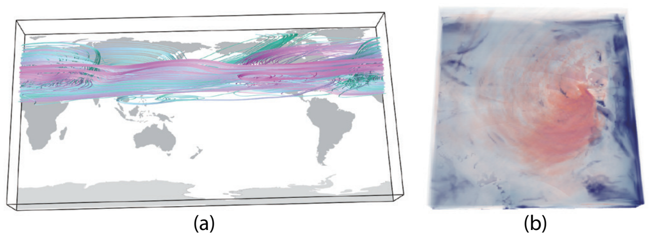

Fig. 8. (a) Source-destination query on GEOS-5 data. 300 pathlines are shown in this visualization. (b) FTLE field of Isabel data at time steps 0 within a finite time scope 12.

limits improve both strong and weak scalabilities, because thicker ghost layers weaken the constraints in our k-d tree decomposition, allowing the particles to be redistributed more evenly.

6.2 GEOS-5 Simulation Data

The GEOS-5 data is from the simulation output of an atmospheric model developed by the NASA Goddard Space Flight Center. The spatial resolution of the model is a $1^\circ \times 1.25^\circ$ latitude-longitude grid with 72 vertical pressure levels that ranges from 1 atm (near the terrain surface) to 0.01 hPa (about 80 km). The dataset in our experiment consists of the monthly averaged output from January 2000 to December 2001.

We conduct a source-destination query analysis with 8M particles in the GEOS-5 data. The query results are shown in Figure 8(a). In this experiment, seeds are densely placed in North America, and we visualize the distribution of the particle trajectories over time.

Strong scaling The first row of Figure 9 shows the strong-scaling tests of this case. The optimal scaling curves in this figure show the linear speedup for reference. Figure 9(a) shows that our method scales as the number of processes increases, whereas the scalability of the baseline method is poor. With 8K processes, our method is 3.58 times faster than the baseline method. Figure 9(c) shows the overall level of load balance in the run-time particle tracing. In our method, the load balance indicator slowly increases as the number of processes increases. It is consistent with the time cost of the constrained k-d tree decomposition, as shown in Figure 9(b). We can also see that the curves that represent our k-d tree method are lower and more stable than those as the baseline method. These results demonstrate that compared with the baseline approach, our method has better load balance and scalability.

Parameter N_{max} We compare the timings and load balance using different maximal number of steps in each cycle of tracing (N_{max}). To evaluate the effect of the execution frequency of the constrained k-d tree decomposition, we use 64 MB memory limit for this test. In the second row of Figure 9, as N_{max} increases, the total running time decreases continually, and the workload becomes more balanced in most cases, except when N_{max} is 10. Although frequently performing the constrained k-d tree decomposition can achieve more balanced workload, it also brings larger overhead. In this case, the optimal N_{max} is 20. More discussion of N_{max} is in the next section.

6.3 Hurricane Isabel Data

The Isabel dataset is from an atmospheric simulation conducted by the National Center for Atmospheric Research. The resolution of this mesh is $500 \times 500 \times 100$, and the dataset has 48 time steps. We compute the FTLE field (shown in Figure 8(b)) based on the wind (U, V, and W components) by tracing 24 million particles that are uniformly seeded in the domain. The optimal N_{max} in this experiment is 50.

Strong scaling We also make strong-scaling tests of this case. As shown in the first row of Figure 10, our method improves the performance and load balance. In this case, we extract the computation time of particle tracing from the total running time in Figure 10(b). The computation is 2.1 times faster compared with the baseline method using 8K processes. From these results we can also see that the larger memory limit leads to higher I/O cost because the expanded blocks are larger, but our method scales better with larger memory limit in general.

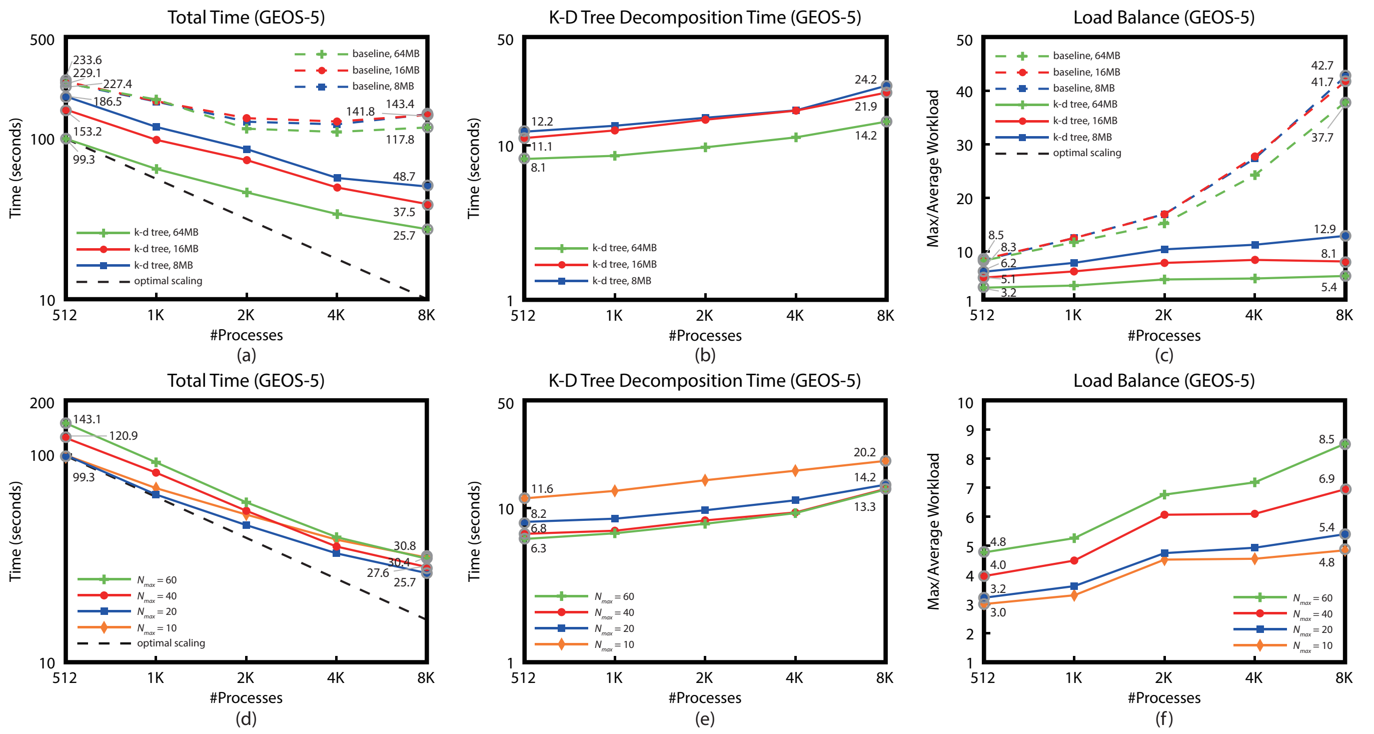

Fig. 9. Strong-scaling tests using GEOS-5 data with different number of processes. Panels (a), (b), and (c) show the tests with different memory limits. The maximal number of tracing steps, N_{max}, is set to 20 in panels (a), (b), and (c). Panels (d), (e), and (f) show the tests with different N_{max} under 64 MB memory limit.

Choice of 3D/4D trees We test whether 3D or 4D tree can achieve better load balance with our method. The results are shown in the second row of Figure 10. In general, we can see from Figure 10(e) that the performance is better if we consider only the space dimensions in this experiment. As shown in Figure 10(f), we conclude that the workload of using 4D trees is less balanced than using 3D trees, because the time of spatiotemporal particles always increases and leads to imbalance in the time dimension.

7 Discussion

In this section, we first discuss the advantages of our method compared with other load-balancing methods. We then discuss the limitations of our method.

7.1 Comparison with Other Methods

We compare our method with other load-balancing algorithms for parallel particle tracing in different aspects, which are detailed in Table 2 and below.

Data preprocessing Our method does not require any data preprocessing. In previous studies, such as flow graph-based workload estimation [26] and irregular data partitioning [7, 34], the whole input dataset needed to be processed and analyzed in a pilot run before the actual particle tracing runs. As the data scale continues to grow in present and future computing, however, it becomes prohibitive to perform the preprocessing because of high I/O costs.

Feature analysis Our method does not rely on any flow feature analysis. Flow partitioning [7] and clustering [34] can be expensive and sensitive to parameter selection. It is still an open challenge to well define features in fluid flows, especially time-varying datasets. It is even harder to detect features in distributed environments with scalability. In contrast, our constrained k-d tree method is light-weight and scalable.

Seed distribution We make no assumptions about the initial particle distribution. Most load-balancing algorithms in data-parallel particle tracing [7, 26, 34] are designed for full-range analyses that trace densely-seeded particles in the whole domain. Our method can handle arbitrary distributed seeds for load-balanced particle tracing.

Data dimensionality Our method supports both static and time-varying data in both 2D and 3D meshes, because k-d trees can decompose spaces in arbitrary dimensions. To the best of our knowledge, most load-balancing algorithms in data-parallel settings are designed for static datasets. Task-parallel algorithms can support unsteady flows, but they require out-of-core I/O scheduling for large-scale data.

Data movement The data movement of our method is minimal. Unlike previous studies that move data blocks for dynamic load balancing [19, 28], we do not need to move flow data back and forth after the initialization. Moving data blocks can be expensive, especially for clusters that have lower network bandwidths. As demonstrated in Peterka et al.'s study [28], the additional costs associated with data repartitioning caused by data movement (including the exchange of data blocks and particles) and updating of data structures may lower the overall performance. The only data movement in our method is the particle exchange, which is also necessary in all data-parallel particle tracing algorithms.

Communication patterns The distributed k-d tree decomposition is self-consistent and decentralized. In previous work-stealing [8] and work-requesting [24] algorithms for task-parallel particle tracing, a dedicated process is necessary in order to schedule the workloads in all other processes. Frequently requesting tasks and sending information for workload scheduling in work requesting [24] significantly increase the communication overhead. The 1-to-n communication pattern makes it difficult to scale in large systems. We instead do not need any master processes to schedule tasks.

7.2 Limitations

Our method does not guarantee optimal and perfect load balancing. First, we assume that an equal number of particles leads to the ideal balance, but it is only an approximation. In the independent particle tracing phase, some particles may stop earlier than others, because they go out of the local bounds of the block or hit critical points. The computation time may also vary because of data locality. As a result, processes have different workloads even if they are assigned the same number of particles. Second, the constrained k-d tree decomposition does not guarantee an even distribution of particles. Because the splitting planes of k-d trees are limited to the overlapped regions of ghost

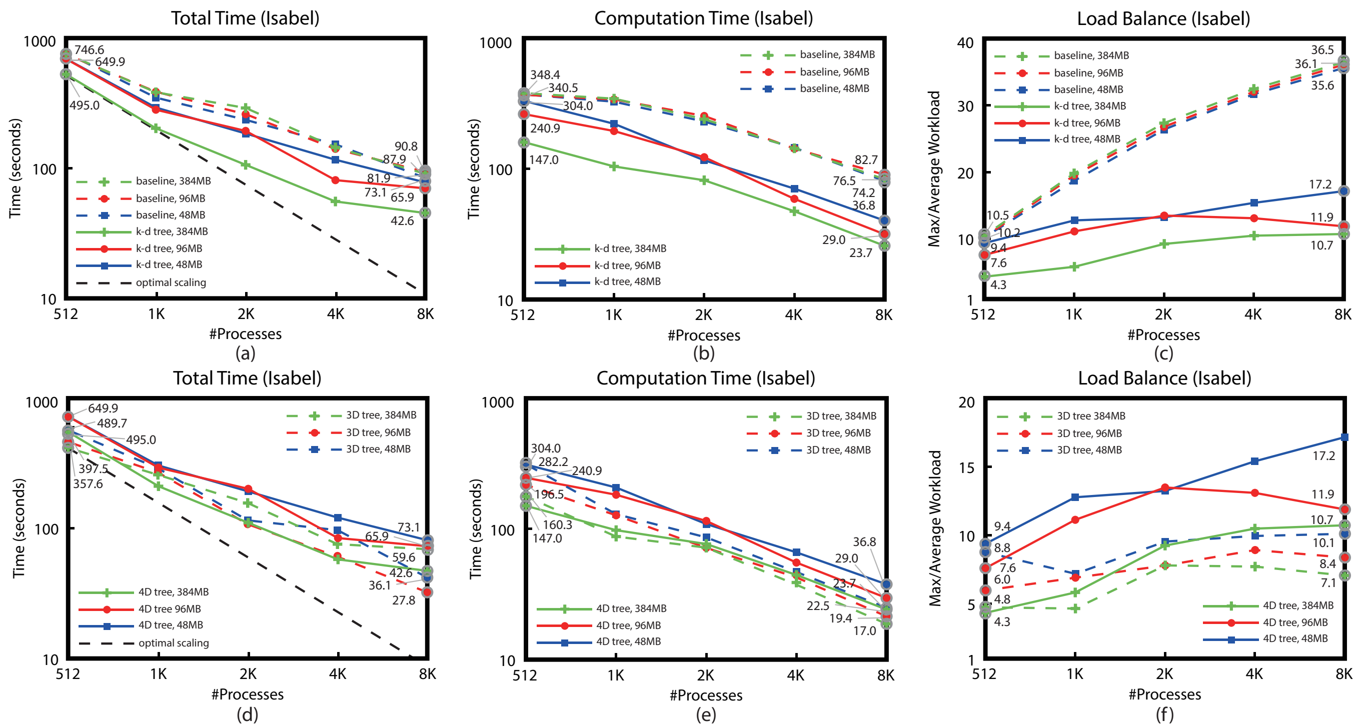

Fig. 10. Strong-scaling tests using Isabel data with different number of processes. Panels (a), (b), and (c) show the tests with different memory limits that consider both space and time dimensions used for splitting (i.e., 4D trees), while panels (d), (e), and (f) show the comparison between considering both space and time dimensions for splitting (i.e., 4D trees) and considering only space dimensions for splitting (i.e., 3D trees).

Table 2. Comparison with other load-balancing algorithms for parallel particle tracing. The plus (+) means that the method meets a certain kind of category or requirement, while minus (-) means that it does not meet the corresponding kind of category or requirement.

	Category				Requirement				
	Parallel over Data	Parallel over Seeds	Static Load Balancing	Dynamic Load Balancing	Data Preprocessing	Heuristics on Flow Features	Assumptions about Seed Distribution	Flow Data Movement	Scheduler Process
Our Work	+	+	-	+	-	-	-	-	-
Nouanesengsy et al. [26]	+	-	+	-	+	-	+	-	-
Chen and Fujishiro [7]	+	-	+	-	+	+	+	-	-
Yu et al. [34]	+	-	+	-	+	+	+	-	-
Lu et al. [19]	+	-	-	+	-	-	-	+	-
Peterka et al. [28]	+	-	-	+	-	-	-	+	-
Dinan et al. [8]	-	+	-	+	-	-	-	-	+
Müller et al. [24]	-	+	-	+	-	-	-	-	+

layers, the k-d tree could be imbalanced. If there are no constraints on the decomposition, that is, the entire data can fit into memory, we can have optimal distribution. But at worst, when the ghost layers just overlap, our method degenerates to the baseline method because the splitting planes will be in fixed positions.

Our method needs the parameter configuration of N_{max}. As the single important parameter, it determines how frequently the particles are redistributed. If N_{max} is too small, the k-d tree decomposition may bring large overhead. If N_{max} is too large, the workload will be uneven because the redistribution is not performed frequently enough. We have shown the optimal values in our experiments on Blue Gene/Q systems, but the number may vary with different data and different architectures.

Our method is bulk-synchronous. The constrained k-d tree decomposition is a collective operation that involves all processes in a synchronous manner. However, the cost of synchronization grows as the number of processes increases. We currently cannot overlap computation with communication.

8 Conclusions and Future Work

In this work, we present a novel approach to balance workloads in parallel particle tracing with constrained k-d trees. The static data partitioning with partial data replication makes it possible to redistribute the unfinished particles during particle tracing through the k-d tree decomposition. A balanced workload is achieved dynamically with our method, which improves the performance of parallel particle tracing. We evaluate our method with different flow visualization and analysis problems through a comprehensive performance analysis. Results show that the proposed method improves both load balance and scalability in particle tracing.

In the future, we will generalize our algorithm to handle unstructured mesh data. Our method also has the potential to visualize and analyze flows in situ. We would also like to combine our algorithm with existing static load-balancing strategies. Furthermore, we plan to relax the synchronizations in the parallel k-d tree decomposition.

Acknowledgments

We thank the anonymous reviewers for their comments. This work is supported by NSFC No. 61672055 and the National Program on Key Basic Research Project (973 Program) No. 2015CB352503. This work is also supported by the U.S. Department of Energy, Office of Science, under contract number DE-AC02-06CH11357.

REFERENCES

[1] M. Aly, M. Munich, and P. Perona. Distributed kd-trees for retrieval from very large image collections. In *Proceedings of the British Machine Vision Conference (BMVC)*, pages 1–11, 2011.

[2] 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.

[3] J. E. Barnes and P. Hut. A hierarchical O(N log N) force calculation algorithm. *Nature*, 324(6096):446–449, 1986.

[4] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

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

[6] J. Carrier, L. Greengard, and V. Rokhlin. A fast adaptive multipole algorithm for particle simulations. *SIAM Journal on Scientific and Statistical Computing*, 9(4):669–686, 1988.

[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*, pages 87–94, 2008.

[8] J. Dinan, D. B. Larkins, P. Sadayappan, S. Krishnamoorthy, and J. Nieplocha. Scalable work stealing. In *SC'09: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 53:1–53:11, 2009.

[9] 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.

[10] G. D. Fatta and D. Pettinger. Dynamic load balancing in parallel kd-tree k-means. In *Proceedings of the 10th IEEE International Conference on Computer and Information Technology*, pages 2478–2485, 2010.

[11] 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.

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

[13] 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.

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

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

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

[17] W. Kendall, J. Wang, M. Allen, T. Peterka, J. Huang, and D. Erickson. Simplified parallel domain traversal. In *SC'11: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 10:1–10:11, 2011.

[18] 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.

[19] K. Lu, H.-W. Shen, and T. Peterka. Scalable computation of stream surfaces on large scale vector fields. In *SC'14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1008–1019, 2014.

[20] 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.

[21] B. Moloney, D. Weiskopf, T. Möller, and M. Strengert. Scalable sort-first parallel direct volume rendering with dynamic load balancing. In *EGPGV07: Proceedings of Eurographics Symposium on Parallel Graphics and Visualization*, pages 45–52, 2007.

[22] D. Morozov and T. Peterka. Block-parallel data analysis with DIY2. In *Proceedings IEEE Symposium on Large Data Analysis and Visualization 2016*, pages 29–36, 2016.

[23] D. Morozov and T. Peterka. 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, 2016.

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

[25] 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 *SC'12: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 61:1–61:11, 2012.

[26] 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.

[27] M. M. A. Patwary, S. Byna, N. R. Satish, N. Sundaram, Z. Lukic, V. Roytershteyn, M. J. Anderson, Y. Yao, Prabhat, and P. Dubey. BD-CATS: big data clustering at trillion particle scale. In *SC'15: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 6:1–6:12, 2015.

[28] 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 *IPDPS'11: Proceedings of IEEE International Symposium on Parallel and Distributed Processing*, pages 580–591, 2011.

[29] 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.

[30] S. Rizzi, M. Hereld, J. A. Insley, M. E. Papka, T. D. Uram, and V. Vishwanath. Large-scale parallel visualization of particle-based simulations using point sprites and level-of-detail. In *Proceedings of Eurographics Symposium on Parallel Graphics and Visualization 2015*, pages 1–10, 2015.

[31] S. Seckler, N. Tchipev, H. Bungartz, and P. Neumann. Load balancing for molecular dynamics simulations on heterogeneous architectures. In *SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 101–110, 2016.

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

[33] I. Wald, S. Boulos, and P. Shirley. Ray tracing deformable scenes using dynamic bounding volume hierarchies. *ACM Transactions on Graphics*, 26(1):6, 2007.

[34] H. Yu, C. Wang, and K.-L. Ma. Parallel hierarchical visualization of large time-varying 3D vector fields. In *SC'07: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 24:1–24:12, 2007.

[35] C. Zhang, A. Krishnamurthy, and R. Y. Wang. Brushwood: Distributed trees in peer-to-peer systems. In *IPTPD'05: Proceedings of International Workshop on Peer-to-Peer Systems*, pages 47–57, 2005.

[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*, pages 80–87, 2016.

[37] J. Zhang, H. Guo, X. Yuan, and T. Peterka. Dynamic load balancing based on constrained k-d tree decomposition for parallel particle tracing. In *Proceedings of IEEE Pacific Visualization Symposium 2017 (Posters)*, pages 310–311, 2017.