Extreme-Scale Stochastic Particle Tracing for Uncertain Unsteady Flow Visualization and Analysis

Article *in* IEEE Transactions on Visualization and Computer Graphics · July 2018

DOI: 10.1109/TVCG.2018.2856772

7 authors, including:

Hanqi Guo
Argonne National Laboratory
43 PUBLICATIONS **415** CITATIONS
SEE PROFILE

Wenbin He
The Ohio State University
7 PUBLICATIONS **26** CITATIONS
SEE PROFILE

Emil M. Constantinescu
Argonne National Laboratory
95 PUBLICATIONS **1,387** CITATIONS
SEE PROFILE

Tom Peterka
Argonne National Laboratory
76 PUBLICATIONS **906** CITATIONS
SEE PROFILE

Extreme-Scale Stochastic Particle Tracing for Uncertain Unsteady Flow Visualization and Analysis

Hanqi Guo, *Member, IEEE,* Wenbin He, Sangmin Seo, *Member, IEEE,* Han-Wei Shen, *Member, IEEE,* Emil Mihai Constantinescu, Chunhui Liu, and Tom Peterka, *Member, IEEE*

Abstract—We present an efficient and scalable solution to estimate uncertain transport behaviors—stochastic flow maps (SFMs)—for visualizing and analyzing uncertain unsteady flows. Computing flow maps from uncertain flow fields is extremely expensive because it requires many Monte Carlo runs to trace densely seeded particles in the flow. We reduce the computational cost by decoupling the time dependencies in SFMs so that we can process shorter sub time intervals independently and then compose them together for longer time periods. Adaptive refinement is also used to reduce the number of runs for each location. We parallelize over tasks—packets of particles in our design—to achieve high efficiency in MPI/thread hybrid programming. Such a task model also enables CPU/GPU coprocessing. We show the scalability on two supercomputers, Mira (up to 256K Blue Gene/Q cores) and Titan (up to 128K Opteron cores and 8K GPUs), that can trace billions of particles in seconds.

Index Terms—Parallel particle tracing, Uncertain flow visualization, CPU-GPU hybrid parallelism.

1 Introduction

Visualizing and analyzing data with uncertainty are important in many science and engineering domains, such as computational fluid dynamics, climate, weather, and materials sciences. Instead of analyzing deterministic data resulted from statistical aggregation, scientists can gain more understanding by investigating uncertain data that are derived and quantified from experiments, interpolation, or numerical ensemble simulations. For example, typical analyses of uncertain flows involve finding possible pollution diffusion paths in environmental sciences with uncertain source-destination queries [1] and locating uncertain flow boundaries [2] in computational fluid dynamics models with uncertain Lagrangian analysis.

In this work, we develop a scalable solution to compute *stochastic flow maps* (SFMs), which characterize transport behaviors in uncertain unsteady flows. SFMs are the generalization of *flow maps* of deterministic data and hence are the basis for uncertain flow analysis. Formally, the flow map is a function that maps the start spatiotemporal location to the end location for the given time in a flow field; the SFM follows the same definition except that the end location is stochastic.

Applications based on SFMs include uncertain flow separatrix extraction and topology analysis. For example, finite-time Lyapunov exponent (FTLE) analysis can be generalized to understand uncertain transport behaviors in uncertain flows [3]. The distribution of Lagrangian coherent structures (LCS)—the flow boundaries in unsteady flows—can be further extracted as the ridges in stochastic FTLE fields [2]. The concept of vector field topology is also generalized to uncertain datasets in recent studies [4], [5]. Likewise, vortices [6] and closed streamlines [1] are studied in uncertain flows. All of these uncertain flow analysis methods require SFM computation.

The main obstacle in uncertain flow analysis is the high computational cost of SFMs. Currently, the only practical solution for computing SFMs is to perform Monte Carlo runs, which trace the particles stochastically in the uncertain data. However, one must trace billions or even trillions of particles for a typical analysis even for small scale data. For example, if the number of grid points and Monte Carlo runs is 10^6 and 10^3, respectively, and if the data has 10^3 time steps, the overall number of particles will be 10^{12}. As documented in previous studies [2], [5], it may take hours to days to run a small problem, even with GPU acceleration.

We focus on scalable and parallel SFM computation with supercomputers. In recent publications, the parallel efficiency is about 45% on 16K cores for 162M particles [7] and 35% on 16K cores for 40M particles [8]. Tracing billions or even trillions of particles at extreme scale is still challenging.

- *Hanqi Guo is with the Mathematics and Computer Science Division, Argonne National Laboratory, Lemont, IL 60439, USA. E-mail: hguo@anl.gov*
- *Wenbin He is with the Department of Computer Science and Engineering, the Ohio State University, Columbus, OH 43210, USA. E-mail: he.495@buckeyemail.osu.edu*
- *Sangmin Seo is with Ground X Inc., Seoul, Korea. E-mail: seo.sangmin@gmail.com.*
- *Han-Wei Shen is with the Department of Computer Science and Engineering, the Ohio State University, Columbus, OH 43210, USA. E-mail: shen.94@osu.edu*
- *Emil Mihai Constantinescu is with the Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439, USA. E-mail: emconsta@mcs.anl.gov.*
- *Chunhui Liu is with the Department of Mathematics, Faculty of Science, Kyoto University, Kyoto, 606-8502, Japan. E-mail: chunhui.liu@math.kyoto-u.ac.jp.*
- *Tom Peterka is with the Mathematics and Computer Science Division, Argonne National Laboratory, Lemont, IL 60439, USA. E-mail: tpeterka@mcs.anl.gov*

We have observed two major differences between deterministic and stochastic flow map computations. First, the task dependencies in deterministic flow map computation are strict, but they can be relaxed in SFM computation. By default, one must trace a particle based on its current location. In the stochastic case, the "current" location is also stochastic; thus the strong dependency can be released by transforming the problem into a probabilistic model. Second, the problem size of SFM computation is much larger. Existing parallel algorithms do not scale for the numbers of particles required by the Monte Carlo runs. The challenges are the memory footprint, the designs of task models, load balancing, and communication patterns. Solving these challenges requires a new parallel framework for SFM computation.

We propose a decoupled SFM computation that removes the time dependencies, and in turn to reduce communication and improving scalability. For time-varying uncertain flow data, we can decompose the time domain into subintervals, independently compute SFMs for each subinterval, and then compose the results for long time intervals of interest. The rationale for the composition is the the Chapman-Kolmogorov equation [9], and the computation is based on sparse matrix multiplication. Because the working data (two adjacent time steps) is much smaller than the whole sequence, we can replicate the working data across parallel processes as much as possible, so that more data are locally available. Decoupling the advection into short time intervals also shortens the travel distances of particles, and thus less communication is required. In addition, we introduce adaptive refinement over the number of Monte Carlo runs for each seed location. Experiments show that computing decoupled SFMs combined with adaptive refinement is more efficient.

We also propose a novel task parallelism that enables CPU/GPU coprocessing when GPUs are available in computing nodes. The philosophy of coprocessing is to dynamically schedule complex and heavy tasks for GPUs while leaving lighter tasks for CPUs. The granularity of a task is a packet of particles associated with the same block. Within each process, a dedicated thread is used to schedule the tasks and to exchange tasks between processes in a nonblocking and asynchronous manner.

We demonstrate the scalability of our methods on two supercomputers: Mira at Argonne National Laboratory and Titan at Oak Ridge National Laboratory. On Mira, we test the performance up to 1 million Blue Gene/Q cores over 16,384 nodes. On Titan, we test up to 131,072 AMD Opteron cores cooperating with 8,192 NVIDIA K20X GPUs. On these supercomputers, our method allows tens of billions of particles to be traced in a few seconds. Our system thus can help scientists analyze uncertain flows in greater detail with higher performance than what was previously possible. In summary, the contributions of this paper are as follows.

- A decoupled scheme that makes it possible to compute SFMs in a highly parallelized manner
- An adaptive refinement algorithm to reduce SFM estimate cost
- A fully asynchronous parallel framework for stochastic parallel tracing based on thread pools, nonblocking communication, and lock-free data structures
- A parallel CPU/GPU coprocessing particle tracing implementation

TABLE 1
Nomenclature.

Symbol	Domain	Meaning
n	2 or 3	Spatial dimension of the flow
i, j, k, l, q, r, s	$\mathbb{N}$	Integer indices
t	$\mathbb{R}$	Time
n_c	$\mathbb{N}$	Number of cells
C_i	$\mathbb{R}^n$	ith cell
$\bar{C}_i$	$\mathbb{R}^n$	Centroid of the ith cell
$V(C_i)$	$\mathbb{R}$	Volume of the ith cell
δ	$\mathbb{R}$	Maximum cell size in any dimension
D	$\mathbb{R}^n$	Spatial domain $D = \cup_i C_i$
$\Pr(\cdot)$	$[0, 1]$	Probability of a random event
$O(\cdot)$	$\mathbb{R}$	Big O notation
$\mathbf{O}$	$\mathbb{R}^{n_c \times n_c}$	Matrix of $O(\cdot)$
$\|\| \cdot \|\|$	$\mathbb{R}^n \to \mathbb{R}$	Euclidean norm in $\mathbb{R}^n$
$\mathbf{v}(t, \mathbf{x})$	$\mathbb{R}^{n+1} \to \mathbb{R}^n$	Deterministic flow
$\phi(t_0, t_1, \mathbf{x})$	$\mathbb{R}^{n+2} \to \mathbb{R}^n$	Deterministic flow map
$\mathbf{V}(t, \mathbf{x})$	$\mathbb{R}^{n+1} \to \mathbb{R}^n$	Stochastic flow
$\Phi(t_0, t_1, \mathbf{x})$	$\mathbb{R}^{n+2} \to \mathbb{R}^n$	Stochastic flow map
$\rho(t_0, t_1, \mathbf{x}_0, \mathbf{x}_1)$	$\mathbb{R}^{2n+2} \to [0, 1]$	Transition probability from $(t_0, \mathbf{x}_0)$ to $(t_1, \mathbf{x}_1)$
$p(t_0, t_1, i, j)$	$\mathbb{R}^2 \times \mathbb{N}^2 \to [0, 1]$	Transition probability from the ith cell at t_0 to the jth cell at t_1
L	$\mathbb{R}$	Lipschitz constant of ρ
$H(t_0, t_1, i)$	$\mathbb{R}^2 \times \mathbb{N} \to \mathbb{R}$	Entropy of $p(t_0, t_1, i, \cdot)$
$\mathbf{P}_{t_0}^{t_1}$	$\mathbb{R}^2 \to [0, 1]^{n_c \times n_c}$	Matrix form of $p(t_0, t_1, \cdot, \cdot)$
m	$\{m \in \mathbb{N} \| m \geq 2\}$	Number of time spans that are coupled
$E(t_0, \ldots, t_m; i_0, i_m)$	$\mathbb{R}^m \times \mathbb{N}^2 \to \mathbb{R}$	Absolute error of coupled SFM estimate over m time spans
$\mathbf{E}(t_0, \ldots, t_m)$	$\mathbb{R}^m \to \mathbb{R}^{n_c \times n_c}$	Matrix form of E
$\mathbf{abs}(\cdot)$		Element-wise absolute value of a matrix
$\preccurlyeq$		Element-wise less or equal to

2 BACKGROUND

We formalize the concepts of SFMs and review the related work on uncertain flow visualization and parallel particle tracing. The notation in this paper is enumerated in Table 1.

2.1 Deterministic and Stochastic Flow Maps

We review the concepts of flow maps in deterministic data and then describe their generalization in uncertain flows.

Formally, in a deterministic time-varying flow field $\mathbf{v} : \mathbb{R}^{n+1} \to \mathbb{R}^n$, the flow map ϕ maps the $(n + 2)$-dimensional tuple $(t_0, t_1, \mathbf{x}_0)$ into $\mathbb{R}^n$, where n is the data dimension and t_0, t_1 are time. As illustrated in Figure 1(a), the physical meaning of $\phi(t_0, t_1, \mathbf{x}_0)$ is the location at time t_1 of the massless particle released at the spatiotemporal location $(t_0, \mathbf{x}_0)$. Assuming $\mathbf{v}$ satisfies the Lipschitz condition, the flow map is defined by the initial value problem

$$\frac{\partial \phi(t_0, t_1, \mathbf{x}_0)}{\partial t_1} = \mathbf{v}(t_1, \phi(t_0, t_1, \mathbf{x}_0)), \text{ and } \phi(t_0, t_0, \mathbf{x}_0) = \mathbf{x}_0. \tag{1}$$

In analyses such as FTLE, the flow map is usually computed at various resolutions to capture ridges of increasing

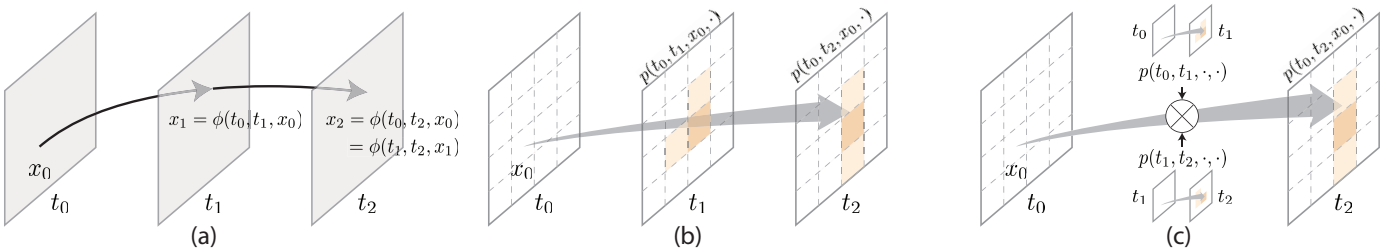


Fig. 1. (a) Flow map computation in a deterministic flow; (b) direct SFM estimate in an uncertain flow; (c) decoupled SFM estimate.

complexity. Particles are densly seeded and traced over time t_1. Numerical methods, such as Euler or Runge-Kutta, are usually used in the particle tracing.

The uncertain flow field $\mathbf{V} : \mathbb{R}^{n+1} \rightarrow \mathbb{R}^n$ and its flow map Φ are stochastic. As shown in Figure 1(b), for a given seed $(t_0, \mathbf{x}_0)$, the final location of this particle at time t_1 is a random variable denoted as $\Phi(t_0, t_1, \mathbf{x}_0)$. The transition probability density function (PDF) of $\Phi(t_0, t_1, \mathbf{x}_0)$ is defined as

$$\rho(t_0, t_1, \mathbf{x}_0, \mathbf{x}_1) \stackrel{\text{def}}{=} \Pr(\mathbf{x}_1 = \Phi(t_0, t_1, \mathbf{x}_0)), \tag{2}$$

where ρ is a $(2n + 2)$-dimensional scalar function and $\int_D \rho(t_0, t_1, \mathbf{x}_0, \mathbf{x}_1) d\mathbf{x}_1 = 1$ for any given t_0, t_1, and $\mathbf{x}_0$. We assume that ρ is Lipschitz continuous in order to approximate the error of our methods (see Appendix for more details). We use the discretized form of SFM for computation and storage:

$$\begin{aligned} p(t_0, t_1, i, j) &\stackrel{\text{def}}{=} \Pr(\mathbf{y} = \Phi(t_0, t_1, \bar{C}_i) \in C_j) \\ &= \int_{C_j} \rho(t_0, t_1, \bar{C}_i, \mathbf{y}) d\mathbf{y}, \end{aligned} \tag{3}$$

where C_i and C_j are the ith and jth cell in the mesh discretization, respectively; $\bar{C}_i$ is the centroid of C_i; and $\sum_j p(t_0, t_1, i, j) = 1$. The straightforward approach to estimate p is direct Monte Carlo simulation based on the Euler-Maruyama method. For each cell C_i, we trace a number of particles seeded from the centroid $\bar{C}_i$ and then estimate the density of particles as the output.

Our adaptive SFM computation is related to but different from the adaptive refinement of deterministic flow maps. Sadlo and Peikert [10] proposed an adaptive mesh refinement (AMR) approach to filter ridges in the deterministic FTLE fields. Barakat and Tricoche [11] reconstruct flow maps based on the reconstruction of sparse samples. For SFM computation, instead of controlling the density of seed locations, we adaptively control the number of stochastic runs for each input seed location, as explained in Section 3.1.

Our decoupled SFM estimate is related to the hierarchical line integration technique [12], which decouples deterministic flow map computation with reduced accuracy for GPU parallelism. The error is bounded by the second order of the cell size if the flow map ϕ is Lipschitz continuous. We generalize the decoupling scheme to estimate SFMs in uncertain unsteady flows and theoretically derive the error bound based on the Lipschitz continuity of ρ. More details on decoupled SFM estimate are in Section 3.2 and Appendix.

2.2 Uncertain and Ensemble Flow Visualization

Comprehensive reviews of uncertainty visualization can be found in [13], [14], and reviews of flow visualization are in [15], [16], and [17].

We categorize uncertain flow visualization techniques into two major types: Eulerian and Lagrangian methods. This classification based on fluid dynamics considers flow fields at specific spatiotemporal locations and at individual moving parcels, respectively. Eulerian uncertain flow visualizations usually directly encode data into visual channels, such as colors, glyphs [18], and textures [19]. Our focus instead in this paper is on the Lagrangian methods that analyze transport behaviors in uncertain unsteady flows.

Lagrangian uncertain flow visualization includes topology analysis for stationary data and FTLE-based analysis for time-varying data. Otto et al. [4] extend vector field topology to 2D static uncertain flow. Monte Carlo approaches are used to trace streamlines that lead to topological segmentation. The same technique is applied to 3D uncertain flows in a later work [5]. For 3D unsteady flows, vector field topologies are no longer feasible because they are unstable and overwhelmingly complicated. FTLE and LCS are alternatives for analyzing unsteady flows. One use of FTLE in uncertain unsteady flows is finite-time variance analysis [3], which is based on the variance of particles advected from the same locations over a time interval of interest. Recently, Guo et al. [2] proposed two metrics to generalize FTLE in uncertain unsteady flows: D-FTLE and FTLE-D. In this paper, we address the common problem of these methods: the high computational cost of Monte Carlo particle tracing.

Our study is also related to ensemble flow visualization. For example, Guo et al. [20] characterized differences between ensemble members based on the analysis of pathlines in flow fields. Höllt et al. [21] proposed a method to visualize flow trajectories in ensemble simulations. The trajectories can be further aggregated and visualized with various visual representations such as contour boxplots [22].

2.3 Parallel Particle Tracing

Parallel particle tracing is a challenging problem in both the HPC and visualization communities. A comprehensive review of this topic can be found in [23]. Parallel particle tracing algorithms can be categorized into two basic types—parallel over data and parallel over seeds, as illustrated in Figure 2.

Parallel-over-data algorithms rely on data partitioning for load balancing. A common practice of data partitioning

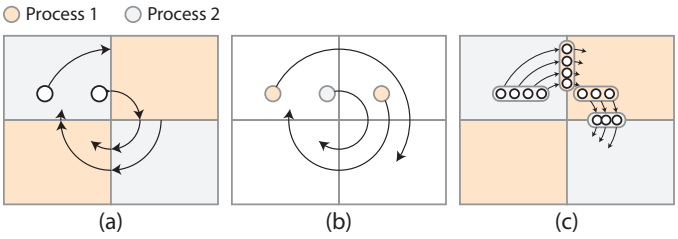


Fig. 2. Existing parallel particle tracing paradigms include (a) parallel over data and (b) parallel over seeds. We employ a task-parallel scheme (c) in this paper. Our task granularity is a packet of particles associated with the same block.

is to subdivide the domain into regular blocks. Peterka et al. [24] show that static round-robin block assignments with fine block partitioning can lead to good load balancing in tracing streamlines in 3D vector fields. The static load balancing can be further improved by assigning blocks based on estimated workloads [25]. In addition to regular blocks, irregular partitioning schemes are used to improve load balancing. For eample, Yu et al. [26] propose a hierarchical representation of flows, which defines irregular partitions for parallel particle tracing. Similarly, mesh repartitioning algorithms are used to balance the workload across processes [27]. For time-varying data, Nouanesengsy et al. [7] partition processes into groups that independently compute multiple FTLE fields with different time spans, in order to reduce global synchronization and decrease I/O overhead. In our study, SFMs can be estimated by decoupling SFMs into short time spans that are independent and can be efficiently computed, instead of tracing particles in long time intervals.

In parallel-over-seeds algorithms, seeds are distributed over processes. Pugmire et al. [28] explore this strategy to load data blocks on demand; thus no communication occurs between processes to exchange particles. Guo et al. [29] present a framework to manage the on-demand data access based on a key-value store. Fine-grained block partitioning and data prefetching are employed to improve the parallel efficiency. The parallel-over-seeds paradigm shows better performance in applications such as 3D stream surface computation [30], but it often suffers from load-balancing issues because flow behaviors are complicated and unpredictable. Work stealing has been used to improve the load balancing in 3D stream surfaces computation [31]. Mueller et al. [32] propose a work-requesting approach that uses a master process to dynamically schedule the computations. Recently, Zhang et al. [33] use k-d trees to dynamically redistribute particles to balance the workload across different processes, but the k-d tree decomposition requires frequent and expensive global synchronization for task scheduling. We instead dynamically schedule the tasks between worker threads within single processes in an asynchronous manner.

Hybrid methods combine both parallelization paradigms. For example, a hybrid master/worker model can be used to dynamically schedule both particles and blocks [28]. DStep [8] employs multitiered task scheduling combined with static data distribution. Camp et al. [34] develop a hybrid implementation based on an MPI/threads programming model, which is also used in a distributed GPU-accelerated particle tracing implementation [35].

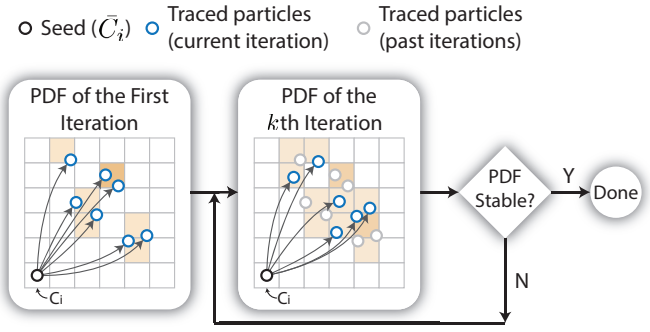


Fig. 3. Adaptive refinement of the SFM of cell C_i. In each iteration, a batch of particles is traced from the cell center C_i, and then the PDF is estimated from all traced particles including the current and all past iterations. The iteration loop exits if the PDF converges.

We regard our system as a hybrid method. Our MPI/threads model defines tasks as packets of particles instead of single particles that are used by Camp et al. [34]. This model also enables us to trace massive amounts of particles on all available CPU and GPU resources simultaneously. Our study is also related to other particle-based Monte Carlo simulations, such as stochastic differential equations [9], particle physics, and ray tracing [36]. However, SFM estimate is a unique problem that requires a tailored parallel algorithm design for uncertain unsteady flow data.

3 Adaptive and Decoupled SFM Estimate

In this section, we introduce the adaptive and decoupled schemes to estimate SFMs with Monte Carlo simulations. We use a synthetic uncertain double gyre dataset to demonstrate the decoupled SFM estimate results. The original double gyre[1] is a 2D, deterministic, time-varying, closed-form vector field dataset defined on $[0,0] \times [2,1]$; and we synthetically injected uncertainties by adding independent Gaussian noises into the u and v components:

$$\begin{aligned} f(x,t) &= \sin^2 tx^2/3 + (1 - \sin t/3)x \\ u(x,y) &= -\pi \sin(\pi f(x)) \cos(\pi y) + N(0, 0.02^2) \\ v(x,y) &= \pi \cos(\pi f(x)) \sin(\pi y) \tfrac{df}{dx} + N(0, 0.02^2) \end{aligned} \quad , \qquad (4)$$

where N is a Gaussian random variable with the variance of 0.02^2. We uniformly discretized the spatial domain into three different resolutions—100×50, 200×100, and 400×200—to evaluate our SFM estimate results.

3.1 Adaptive Refinement of SFMs

We propose an adaptive refinement approach to dynamically control the number of Monte Carlo runs for each seed location in order to reduce computational cost. The adaptive refinement for each seed is illustrated in Figure 3. Without loss of generality, we estimate the SFM of the ith cell (C_i) in the bottom left corner in the figure. In every iteration, a batch of particles is traced from the cell center $\bar{C}_i$; and then the probability density from all traced particles—including

1. http://shaddenlab.berkeley.edu/uploads/LCS-tutorial/

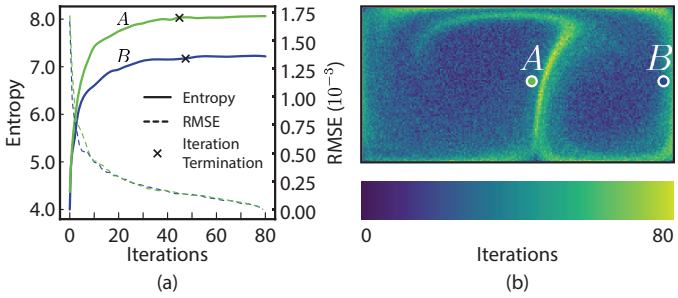


Fig. 4. Experiment results of the synthetic uncertain double gyre data: (a) entropy-error-iteration plot showing the changing SFM entropy and RSME values at locations A and B (marked in b) with respect to the number of iterations; (a) adaptive refinement level (number of iterations) in the synthetic uncertain double gyre data. In this experiment, the time interval is $[0, 2]$; the mesh discretization is 200×100; and the stop criterion ΔH is 0.05.

the current and all past iterations—is estimated as a discrete histogram in array $p_k(t_0, t_1, i, \cdot)$, where $\cdot$ is the cell index and k is the index of the current iteration. In general, any density estimator can be used, and we use the simplified cloud-in-cell (CIC) [37] approach in our experiments. The density value in each cell is proportional to the number of particles that fall into the cell. The iteration loop exits if the array $p_k(t_0, t_1, i, \cdot)$ converges, and then we use $p_k(t_0, t_1, i, \cdot)$ as the estimation results.

In this study, we use the difference of information entropies between $p_{k-1}(t_0, t_1, i, \cdot)$ and $p_k(t_0, t_1, i, \cdot)$ as the stop criterion. As mentioned by Hofmann et al. [38], the number of iterations can be determined by any measurable stop criterion. We use the difference of entropies, because the computational and space complexity of computing entropy are less expensive than other metrics such as and earth mover's and point-wise distances. Formally, the entropy of $p_k(t_0, t_1, i, \cdot)$ is defined as

$$H_k(t_0, t_1, i) = -\sum_j p_k(t_0, t_1, i, j) \log p_k(t_0, t_1, i, j). \quad (5)$$

If the difference $\Delta H_k = |H_k(t_0, t_1, i) - H_{k-1}(t_0, t_1, i)|$ is greater than a preset threshold ΔH, we inject a number of particles at $\bar{C}_i$ to estimate the probability density; otherwise we exit the iteration.

The threshold ΔH is empirically determined to trade off the computational cost and the precision. In practice, before conducting large runs on supercomputers, one can sample the problem space and determine a proper ΔH based on the *entropy-error-iteration* plot, as shown in Figure 4(a). In this figure, the SFM entropy of each location A and B converges to a constant because of the Law of Large Numbers. The root mean squared error (RMSE) measures the differences of SFMs between each iteration and the last iteration in the plot. Based on the figure, we choose to use $\Delta H = 0.05$ to terminate iterations before the entropy starts to converge. As a result, different numbers of iterations, which are color-coded in Figure 4(b), are used to estimate the SFMs.

Figure 5 visually compares the results with adaptive refinements and fixed numbers of Monte Carlo runs. The adaptive refinement uses fewer particles (278.24 per cell on average) to achieve roughly the same quality as uniformly using 1,600 particles, while uniformly using 300 particles per cell leads to an under-sampled result. Considering the SFM estimated with a large number (1,600) of particles as the ground truth, the RMSE of the results using 300 and adaptive samples are 3.61×10^{-4} and 2.15×10^{-4}, respectively; the corresponding peak signal-to-noise ratio (PSNR) values are 68.70 dB and 73.21 dB, respectively.

3.2 Decoupled Estimate of SFMs

As illustrated in Figure 1, the idea of decoupled SFM estimate is to avoid the direct computation of $p(t_0, t_2, i, k)$ by divide-and-conquer. First, we decompose the time interval $[t_0, t_2]$ into two subintervals $[t_0, t_1]$ and $[t_1, t_2]$ ($t_0 < t_1 < t_2$). Second, we independently compute $p(t_0, t_1, i, j)$ and $p(t_1, t_2, j, k)$ for any j. Third, we approximate $p(t_0, t_2, i, k)$ with the following equation:

$$p(t_0, t_2, i, k) \approx \sum_j p(t_0, t_1, i, j) p(t_1, t_2, j, k). \quad (6)$$

The rational of Eq. (6) is based on the Chapman-Kolmogorov equation and detailed in the Appendix. Eq. (6) can also be written in the matrix form:

$$\mathbf{P}_{t_0}^{t_2} \approx \mathbf{P}_{t_0}^{t_1} \mathbf{P}_{t_1}^{t_2}. \quad (7)$$

By generalizing Eq. (7), we can approximate $\mathbf{P}_{t_0}^{t_m}$ for the given time interval $[t_0, t_m]$. We first decompose the time domain $[t_0, t_m]$ into m subintervals $[t_0, t_1]$, $[t_1, t_2]$, $\cdots$, $[t_{m-1}, t_m]$ ($t_0 < t_1 < \cdots < t_m$), and then independently compute the SFM for each subinterval. The SFM $\mathbf{P}_{t_0}^{t_m}$ can be estimated as the multiplication of the results for all subintervals:

$$\mathbf{P}_{t_0}^{t_m} \approx \mathbf{P}_{t_0}^{t_1} \mathbf{P}_{t_1}^{t_2} \cdots \mathbf{P}_{t_{m-1}}^{t_m}. \quad (8)$$

We study the approximation error of decoupled SFM estimate both theoretically and empirically. The theoretical error analysis, detailed in the Appendix, is based on end locations of particles being discretized into the indices of cells. The absolute error is no larger than $O(m\delta^{n+1})$, where δ is the maximum cell size in any dimension and m is the number of subintervals that are coupled. Finer mesh discretization (smaller δ) and smaller number of subintervals m will lead to preciser results.

Figure 6 empirically studies the approximation error of the synthetic uncertain double gyre data by sweeping δ and m. We use the fixed 1,600 particles per cell for all (sub)intervals in this experiment. We first study the discretization error by using different spatial resolutions of SFMs. Figure 6 (a), (b), and (c) use three different spatial resolution and thus δ equals to 2, 1, and 0.5, respectively. We can see that the estimate results significantly improves by decreasing δ. We also study the approximation error introduced by m. The first row ($m = 1$) is the direct estimate of $\mathbf{P}_0^2$. In every next row, we increase m by the factor of two. Each row uniformly subdivides the time interval $[0, 2]$ into m subintervals and then estimate $\mathbf{P}_0^2$. For example, in the third row ($m = 4$), $\mathbf{P}_0^2$ is derived from $\mathbf{P}_0^{0.5}$, $\mathbf{P}_{0.5}^1$, $\mathbf{P}_1^{1.5}$, and $\mathbf{P}_{1.5}^2$. We can observe that the more subintervals that are composed, the more errors are introduced. Overall all,

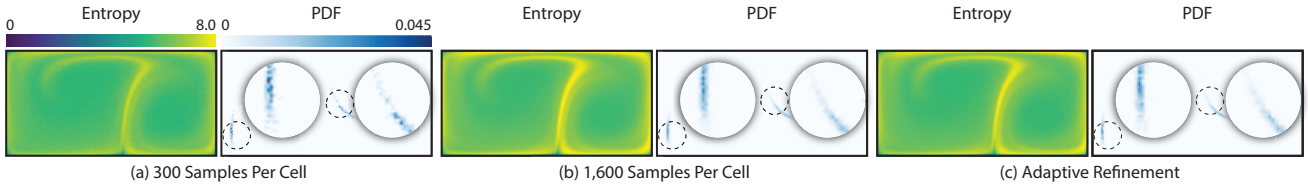


Fig. 5. Experiment results of the synthetic uncertain double gyre data using 300 (a), 1600 (b), and adaptive numbers (c) of particles per cell. The time interval is $[0, 2]$ and the mesh discretization is 200×100. Each PDF plot visualizes and magnifies two SFMs of A (left) and B (right).

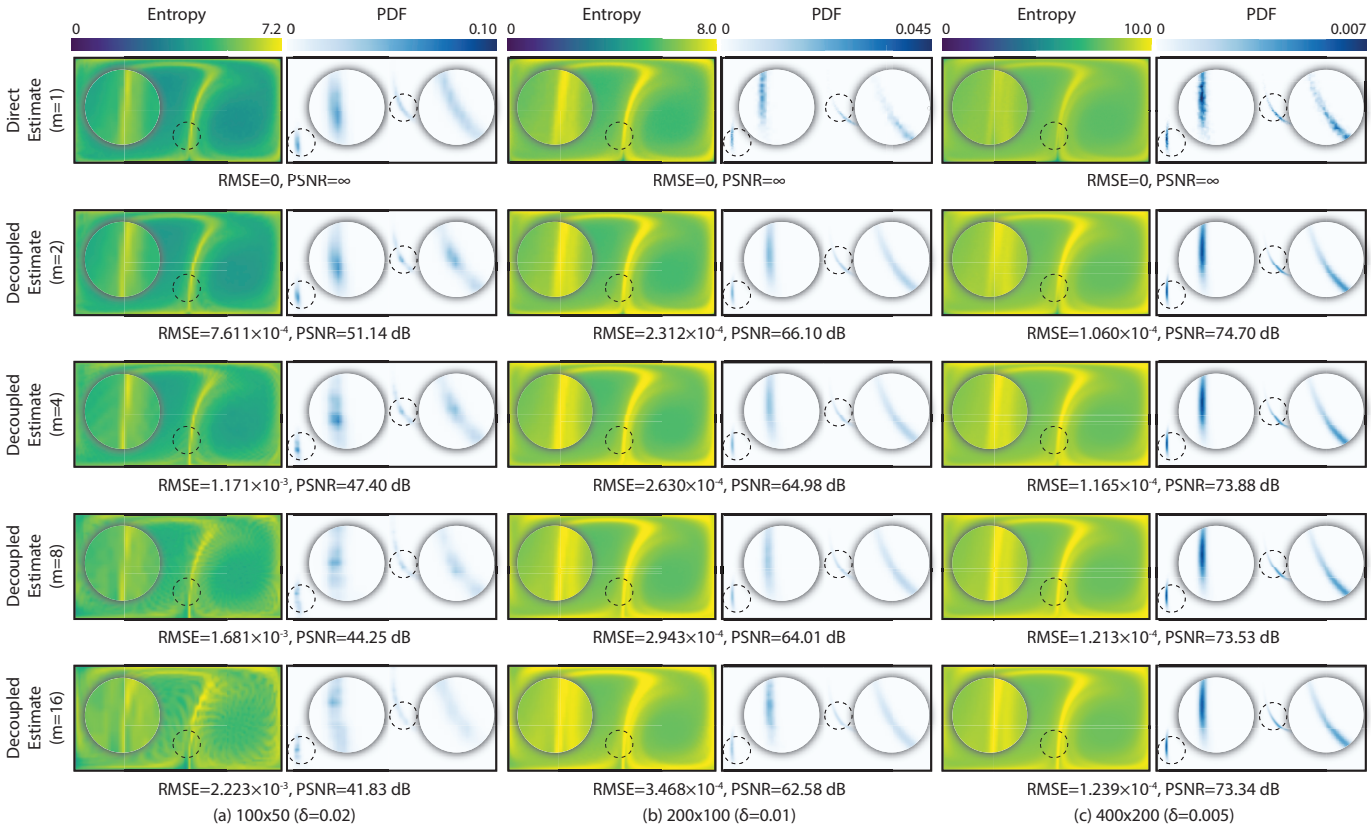


Fig. 6. Comparison between direct (first row) and decoupled SFM estimate (other rows) of the synthetic uncertain double gyre data with three spatial resolutions 100×50 (a), 200×100 (b), and 400×200 (c). Each PDF plot visualizes and magnifies two SFMs of A (left) and B (right). (The color maps for a, b, and c are different because SFM and SFM entropies are computed with different mesh discretization.)

we can see that the decoupled SFM estimate results approximate those of the direct estimate, and the error matches our theoretical analysis. More details on the bounded error approximation can be found in Appendix.

The decoupled SFM estimate is the key to achieving scalability in our study. Decoupling removes the time dependencies, so that we can first independently compute SFMs for subintervals and then compose the results. Decoupling has two benefits. First, it reduces the communication cost because the lifetimes and travel distances of particles are less than those in long time periods. Second, it reduces memory cost because the working datasets in the decoupled estimate are smaller. More details on the scalable computation are in next sections.

4 Software Architecture Design

Our parallel particle tracing framework exploits hierarchical parallelization. At the top level, the processes are divided into groups. Each group replicates the working data and traces a different set of seeds. Inside each group, we parallelize over the data. Each process has a portion of data blocks. A novel task model based on MPI/thread hybrid parallelization is used. The rationale for our hierarchy is based on adaptive and decoupled SFM estimate. First, the decoupling makes it possible to have higher degrees of data replication for better scalability because the working data of two adjacent time steps are smaller than the data of the whole dataset. Second, the adaptive refinement allows asynchronous processing, which also boosts the scalability of parallel particle tracing.

We implement a novel task model design—packets of particles—to achieve high parallel efficiency in the MPI/thread model. Within each process, the tasks are scheduled and processed by a pool of threads in parallel. The interprocess task exchange is managed by a dedicated thread, which handles nonblocking MPI communication. Lock-free data structures are used to exchange

data between threads. In general, this design is fully asynchronous—communication and computation are overlapped, and threads are synchronization-free. This design improves data locality and enables CPU/GPU coprocessing.

4.1 Initialization

Because the decoupled SFM estimate yields smaller working data, typically two adjacent time steps, we can replicate data in order to improve data locality. We first partition the data into blocks and then determine how many processes to assign to each group for the given memory limit. For example, given 4 processes, 64 total blocks, and a maximum of 32 blocks per process, we would create 2 process groups.

Within each group, the blocks are distributed across processes. As in Peterka et al. [24], we statically assign blocks to processes by a round-robin scheme. Each process is in charge of one or more blocks. In addition, threads and lock-free data structures for task exchanging are created upon the initialization.

4.2 Task Model

We define a task as a tuple `(blkID, type, particles[])`, where `particles[]` is a packet of particles associated with only one block (`blkID`). The granularity of a task is one or more particles, up to a given limit. Each particle is a tuple $(\mathbf{x}_0, \mathbf{x})$ consisting of its initial and current spatiotemporal locations, respectively.

Four types of tasks are depicted in the model: initialization (`INIT`), tracing (`TRACE`), return (`RETURN`), and checking (`CHECK`). Initialization tasks are used to initialize particles for a list of seed locations in the given block. Particles are created either by the system for bootstrapping or by the `CHECK` tasks when more particles are necessary to refine the SFMs. Tracing tasks start or continue to trace a packet of particles that are not finished yet. If particles are moving out of the current block, new tracing tasks associated with the target blocks are created. Return tasks are created by tracing tasks to send finished particles to their home blocks. Checking tasks check termination for particles released at the same location $\mathbf{x}_0$. The density is written to the output sparse matrix if it is converged; otherwise new initialization tasks are created to refine the density.

4.3 Thread Model

We use a thread pool for parallelism within a single process. Figure 7 illustrates the thread model in our design. Two types of threads exist: the communicator/scheduler (comm/sched) threads and the worker threads. Several lock-free producer-consumer queues are used to schedule and exchange tasks between threads. Two groups of queues, the work queues (Q_{work}^{CPU} and Q_{work}^{GPU}) and the send queues (Q_{send}), keep the pending tasks for the local and remote processes, respectively.

Algorithm 1 shows the pseudo code of the worker thread main loop. The worker threads function as both producers and consumers. Worker threads consume tasks and also produce new tasks to deliver particles to their next or final destinations. The new task is enqueued to the work queue

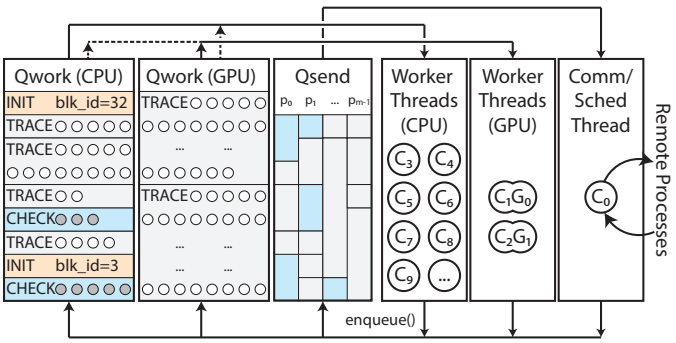


Fig. 7. Thread model on single processes. The comm/sched thread exchanges tasks with remote processes and schedules tasks on CPUs and GPUs. The worker threads consume and produce tasks for SFM estimate. The work queues and send queues buffer the pending tasks.

Algorithm 1 Worker thread loop. The process_task() function processes an input task and returns a list of new tasks to continue the computation.

```
while !all_done do
    if Q_work.pop(task) then
        new_tasks[] = process_task(task)
        for all task in new_tasks[] do
            comm.enqueue(task.blkID, task)
        end for
    end if
end while
```

if the current process owns the destination block; otherwise the task is appended to the send queues.

Algorithm 2 simplifies the task routing by providing a unified interface to enqueue tasks from both CPU/GPU worker threads and the comm/sched thread. Tasks are either routed to local work queues or remote processes. The enqueue function also schedules tasks for CPU/GPU coprocessing, which is detailed in Section 4.5.

The maximum number of particles for each task (`max_size_CPU`), which defines the granularity of a `TRACE` task, is the most important parameter that determines the scalability of the thread pool. Figure 8 shows a scalability benchmark using different `max_size_CPU` values. The optimal `max_size_CPU`, which may differ between datasets and hardware architectures, can be determined by parameter sweeping on a number of compute nodes. We found that 64 is the near optimal selection in our experiments. A similar parameter

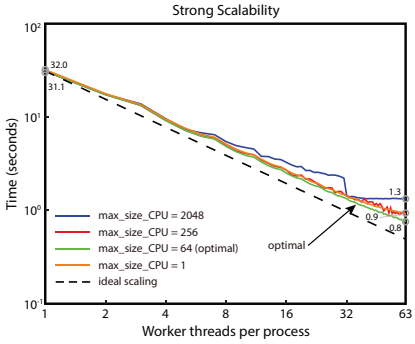


Fig. 8. Benchmark of the flow map computation in tornado simulation data (32K particles) with different `max_size_CPU` and different numbers of worker threads per process on 32 Blue Gene/Q nodes. A proper selection of `max_size_CPU` leads to better performance and scalability.

Algorithm 2 Task enqueue function, which routes tasks to different local and remote CPU/GPU processors.

```
function ENQUEUE(blkID, task)
    i ←blkID_to_rank(blkID)
    if i=comm.rank then
        if task.size≥max_size_GPU then
            split_tasks[]=task.split(max_size_GPU)
            enqueue_all(split_tasks[])
        else if task.size≥min_size_GPU then
            Q_work^(GPU).push(task)
        else if task.size≥max_size_CPU then
            split_tasks[]=task.split(max_size_CPU)
            enqueue_all(split_tasks[])
        else
            Q_work^(CPU).push(task)
        end if
    else
        Q_send^i.push(task)
    end if
end function
```

(max_size_GPU) needs to be configured when a GPU is available for coprocessing with CPUs. The goal is to set max_size_GPU to have approximately equivalent processing time on GPUs as CPUs, so max_size_GPU is usually larger than max_size_CPU. More details on the parameter setting in CPU/GPU coprocessing are in Section 4.5.

The comm/sched thread consumes tasks in the send queues by sending them to the destination process and enqueues to work queues tasks that are received from remote processes. Our thread model uses a dedicated thread for communication, a common practice in the implementation of high-level task-parallel programing models. In addition, our comm/sched thread schedules tasks for load balancing and CPU/GPU coprocessing.

4.4 Asynchronous Communication

Algorithm 3 Comm/sched thread loop

```
while !all_done do
    for all i in comm.world do                    ▷ outgoing tasks
        if Q_send^i.pop_bulk(tasks, max_size_send) then
            comm.isend(i, serialize(tasks))
        end if
    end for
    while comm.iprobe() do                        ▷ incoming tasks
        tasks = unserialize(comm.recv())
        for all task in tasks do
            enqueue(task.blkID, task)
        end for
    end while
    comm.iexchange(all_done)                      ▷ exchange status
end while
```

The comm/sched thread executes and manages non-blocking MPI requests without any synchronization, in order to fully overlap between computation and communication as illustrated in Figure 9. The pseudo code of the comm/sched thread main loop is listed in Algorithm 3. Each process maintains a list of lock-free send queues $\{Q^i_{\text{send}}\}$, where i is the destination rank.

For m processes, we use $m-1$ send queues, which yield better performance than a single queue. In our design,

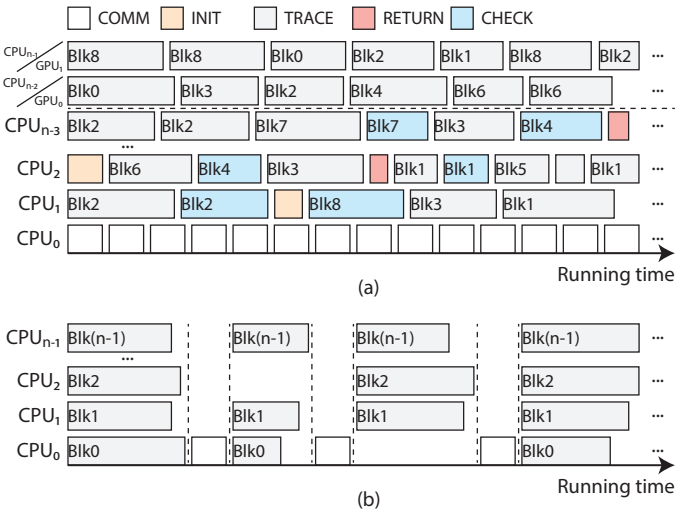


Fig. 9. Gantt chart of (a) our task model and (b) the bulk synchronous parallel model. Each row represents a thread.

a set of tasks with the same destination rank is obtained with the pop_bulk() function in the lock-free queue. Thus, we can send a larger message that contains multiple tasks to the same destination rank, instead of multiple smaller messages each with a single task. We do so because a larger message size usually leads to better bandwidth utilization than smaller messages do. The loop exits when all tasks across all processes are finished.

4.5 CPU/GPU Coprocessing

The thread pool model enables hybrid CPU/GPU parallelization, which fully utilizes the computation power of both CPUs and GPUs in compute nodes. Our task-scheduling strategy is to fill GPUs with larger tasks and assign complex or small tasks for CPUs. We associate a GPU worker thread (running on the CPU) with each GPU. The data blocks in the main memory are copied into the GPU in the initialization stage. A designated GPU task queue is also set up for task scheduling. In the enqueue() function, larger tasks and smaller tasks are pushed into the GPU and CPU queues, respectively.

Similar to the rationale of max_size_CPU for CPU workers, we also need to limit the task size for the GPU, that is, min_size_GPU and max_size_GPU. We usually set max_size_CPU $\leq$ min_size_GPU $<$ max_size_GPU.

4.6 Implementation

We implemented the prototype system with C++11 and CUDA. MPI is used for interprocess communication. For each process, the worker threads are created with Pthreads, and the main thread plays the role of the comm/sched thread. We use a lock-free concurrent queue implementation[2] to exchange tasks between threads. DIY2 [39] is used for domain decomposition. The Block I/O Layer (BIL) library [40] is used to efficiently load disjoint block data across different files and processes collectively. After the computation, we store the SFMs in a sparse matrix that is managed by the PETSc library [41].

2. https://github.com/cameron314/concurrentqueue

5 Application Results

We applied our method to two weather simulation datasets: uncertain Hurricane Isabel data and ensemble Weather Research and Forecasting (WRF) data.

5.1 Input Data

Uncertainty arises in the Hurricane Isabel data from temporal down-sampling. In climate and weather simulations, a common practice is to dump average data hourly or daily instead of every time step. Such data down-sampling reduces the I/O cost but sacrifices accuracy. We follow Chen et al. [42] who use quadratic Bezier curves to quantify the uncertainty of the original Isabel data from the IEEE Visualization Contest 2004. The down-sampled dataset we used in the experiment keeps the full spatial resolution but aggregates every 12 time steps into one. The parameters of the quadratic Bezier curves and the Gaussian error are used to reconstruct the uncertain flow field.

The uncertainty of the ensemble WRF data arises from averaging the ensemble members. The input data, courtesy of the National Weather Service, is simulated with the High Resolution Rapid Refresh model [43]. The model is based on the WRF model and assimilates observations from National Oceanic and Atmospheric Administration and other sources. The spatial resolution of the model is $1799 \times 1059 \times 40$, and we use 10 ensemble members with 15 hourly averaged outputs. The uncertainty is modeled as Gaussian—the mean and covarances of the ensemble members are computed for every grid point location.

5.2 Uncertain Source-Destination Queries

Scientists can investigate and explore the uncertain transport behaviors by queries. Figure 10(a) shows the uncertain source-destination query results. We create particles along a line in the domain and visualize the distributions of these particles after every hour by volume rendering. The blue line at the 0th hour indicates the distribution of the seeds, which is deterministic. As the time evolves, we can see that the uncertainties of SFMs grow throughout the advection. In addition, the uncertain transport behaviors in different regions are different.

5.3 Uncertain FTLE and LCS Visualization

FTLE and LCS are the most important tools for analyzing deterministic unsteady flow. The FTLE was formalized for time-varying flows by Haller [44], and it measures flow convergence or divergence for the time interval of interest. In our previous study [2], we generalized FTLE and LCS to analyze uncertain unsteady flows based on SFMs. Three new concepts were introduced: D-FTLE (distributions of FTLE), FTLE-D (FTLE of distributions), and U-LCS (uncertain LCS). We compute FTLE-D and U-LCS from the uncertain Isabel data and the WRF ensembles in Figure 11 and Figure 10, respectively.

The FTLE-D and U-LCS in Figures 11(a) and (b) show convective bands of the uncertain Isabel data. The spiral arm that extends to the east coast separates two different motions: the flow going upwards and the flow remaining horizontal. In general, the U-LCS field characterizes the probability of belonging to an LCS, or ridges of the FTLE field, for each spatiotemporal location. Because there is more uncertainty in updraft and downdraft flows, the boundary of the two features is fuzzy, as shown in the volume rendering of U-LCS and the FTLE-D.

In the WRF ensembles, likewise, we can observe that the upward and downward air flows lead to uncertainties in U-LCS and FTLE-D. These are due mainly to the land surface variability. We can see four distinct regions in Figures 10(b) and 10(c): the on-shore flow from the Pacific Ocean to the Cascade mountains (①), a cold front from Oklahoma to the Dakotas (②), and two unstable troughs in the Midwest and the East (③ and ④). The visualizations of FTLE-D and U-LCS, which are confirmed by meteorologists, highlight these unstable zones.

6 Performance Evaluation

We study the scalability of our method on two supercomputers: Mira and Titan. We also compare our parallel particle tracing scheme with previous studies.

6.1 Scalability Study on the Blue Gene/Q Systems

We conducted a scalability study on Mira, an IBM Blue Gene/Q system at Argonne National Laboratory. The theoretical peak performance of Mira is 10 petaflops. Each compute node has 16 1.6 GHz PowerPC A2 cores, which support 64 hardware threads in total. The memory on each node is 16 GB, and the interconnect is a 5D torus network.

We ran one MPI process on each node, with one comm/sched thread and 63 worker threads for computation. These choices are based on the experiments in Section 4.3. We limited the memory for data blocks to 1 GB per process. For the uncertain Isabel data, we used both fixed numbers of Monte Carlo runs and adaptive refinements for comparison. For the fixed sampling, the number of runs is 256; thus, the total number of particles is about 6.5 billion.

Figures 12(a) and 12(c) show the timings of SFM computation on both datasets with different numbers of processes on Mira. Ideal scaling curves based on linear speedup are shown for reference. From the benchmark we can see that the strong scaling speedup is nearly linear. The parallel efficiency of 4K, 8K, and 16K processes is 92%, 85%, and 72%, respectively.

6.2 Scalability Study on CPU/GPU Hybrid Architectures

We benchmarked the CPU/GPU coprocessing on Titan, which is a Cray XK7 supercomputer at Oak Ridge National Laboratory. Titan has 18,688 compute nodes, each equipped with an AMD Opteron 16-core CPU that operates at 2.2 GHz with 32 GB of memory. In addition to the CPU, each node contains an NVIDIA K20X GPU with 6 GB memory. The number of CUDA cores on a single GPU is 2,688, running at 732 MHz. We use up to 8,192 compute nodes in our experiments.

Figure 12(b) shows the strong scalability benchmark on the uncertain Isabel dataset. The problem size is the same as that on Mira, 6.5 billion particles. In the experiments, we fully used the CPU resources by running 15 worker threads and one comm/sched thread per process on each node.

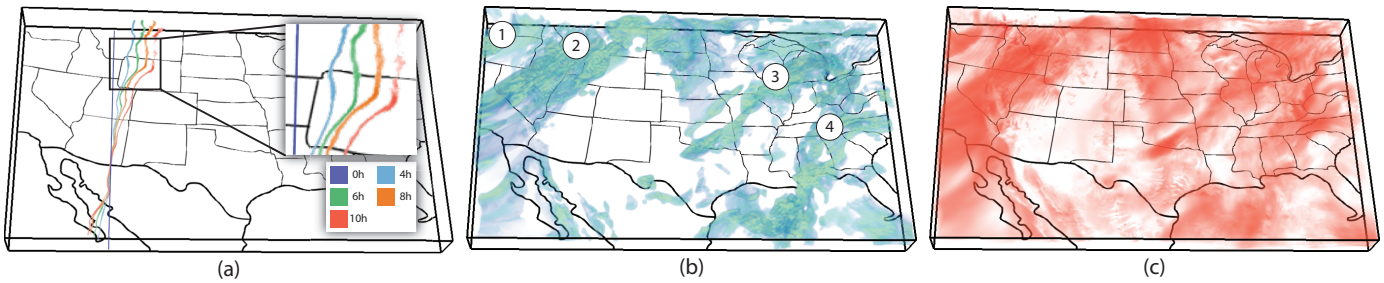


Fig. 10. Experiment results of the WRF ensemble simulation data: (a) uncertain source-destination queries; (b) uncertain LCSs; (c) FTLE-D.

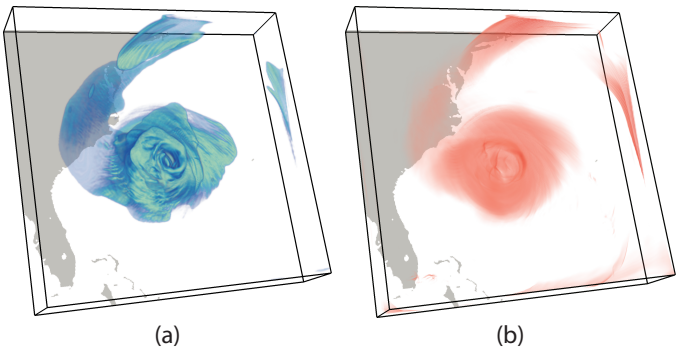


Fig. 11. Volume rendering of (a) uncertain LCSs and (b) FTLE-D of the uncertain Isabel data.

In the CPU/GPU coprocessing mode, one of the worker threads managed the GPU. We conducted three runs to study the effectiveness of CPU/GPU hybrid parallelization: pure CPU mode, pure GPU mode, and hybrid mode.

Results show that the computation time of the hybrid mode is about $2.5\times$ faster than with the pure CPU mode. For reference, Camp et al. [35] report a speedup of $1\times$ to $10.5\times$ on a distributed-memory GPU particle tracer compared with a CPU-only code on 8 nodes. Based on this study and other previous studies, we believe that our $2.5\times$ speedup is promising. Our hybrid parallelization design enables the full use of available hardware resources on compute nodes, including all CPU and GPU cores. The scheduling of CPUs and GPUs is also adaptive, capable of balancing working time between CPU and GPU workers. Moreover, the hybrid implementation is scalable up to 131,072 Opteron cores with 8,192 NVIDIA K20 GPUs in our test. At this scale, tracing billions of particles takes less than 10 seconds.

6.3 Discussion

Our approach reduces network bandwidth utilization for high scalability. Because of the decoupled SFM estimate, we only need to handle small subintervals as the working dataset for each run. Because of the process grouping in the software design, the small working dataset can be replicated over different groups for independent processing. All communications happen within each group; there is no intergroup communication, thus avoiding scalability bottlenecks.

We compared our task-parallelism particle tracing with existing algorithms. The baseline approaches are those of Peterka et al. [45] and Camp et al. [34]. Both algorithms partition data into blocks for parallel processing and use MPI/thread hybrid parallelization. We implemented these algorithms and compared their performance on the same dataset and problem size. In the experiment, we used the deterministic tornado dataset and 32 threads per process for computation. Notice that our task parallel scheme not only works for stochastic particle tracing, but also capable of accelerating deterministic particle tracing and FTLE computation as well. The timings with respect to different numbers processes are shown in Figure 13. We can see that our method outperforms the others.

The parallel model used by Peterka et al. [45] is bulk synchronous (Figure 9(b)). In this model, each block of data is associated with a thread in single process. The particles are traced in the current block until they cross the block bounds, and then they are exchanged between neighbor blocks collectively. Compared with the bulk synchronous parallel model, our model does not associate blocks with threads. We also fully overlap the communication and computation in our framework.

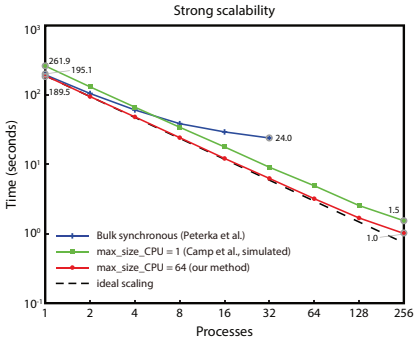


Fig. 13. Performance comparison with two parallel particle tracing methods [34], [45]

The thread pool pattern is used by Camp et al. [34], but the major differences in our design are the task model and the software design. Their task granularity is limited to a single particle instead of a packet of particles as in our method. Therefore, in Figure 13 we simulate their method with the `max_CPU_size` of 1 (for one particle) and compare the performance with our method with larger `max_CPU_sizes`.

7 Conclusions and Future Work

In this paper, we presented a scalable SFM estimate method for uncertain flow visualization and analysis. The keys to achieving high scalability are the decoupled and adaptive algorithms, the MPI/thread hybrid parallelization, and the unique task design that assembles packets of particles. The decoupling allows us to estimate SFMs of adjacent

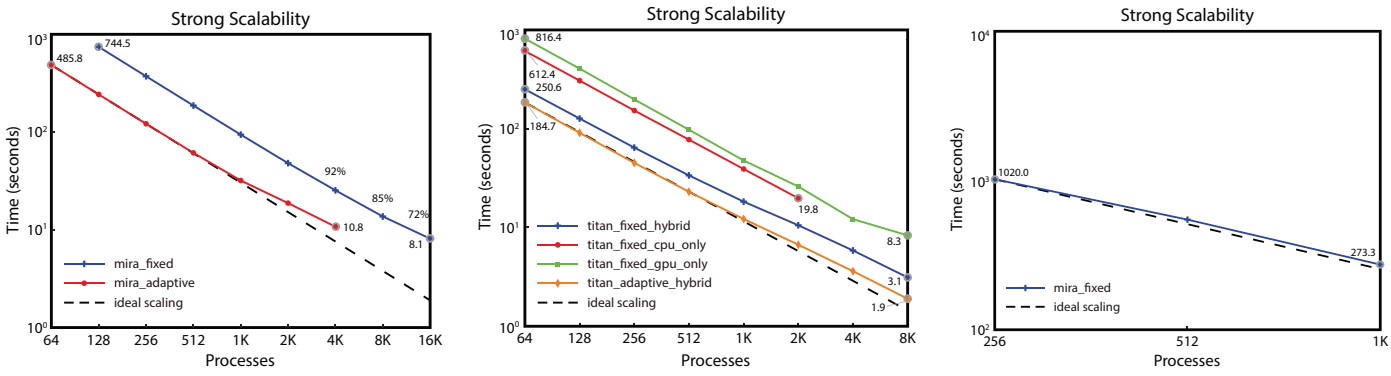


Fig. 12. Strong scalability studies of our method: (a) uncertain Isabel data on Mira; (b) uncertain Isabel data on Titan; (c) ensemble WRF data on Mira.

time steps and then compose them together. The number of stochastic runs can be adaptively configured for better efficiency and precision. We parallelize over tasks, which are packets of particles, to achieve high efficiencies in the MPI/thread hybrid programming. Our parallelization design also enables CPU/GPU coprocessing when GPUs are available. Results show that our method can help scientists analyze uncertain flows with higher performance than previously possible.

In future, we would like to use our approach to accelerate more flow analyses in uncertain unsteady flows. We are going to further derive the estimate error of our adaptive refinement as well. We would also like to further reduce the computational cost for the SFM estimate on resource-limited platforms, which are more accessible to users.

APPENDIX

We review the rationale of the decoupled SFM estimate and then evaluate its error in this Appendix.

Definitions, Lemmas, and Assumptions

Based on the approximation in Eq. (8), we define the absolute error matrix of the decoupled SFM estimate as a function

$$\mathbf{E}_m(t_0, t_1, \ldots, t_m) \stackrel{\text{def}}{=} \mathbf{abs}\left(\mathbf{P}_{t_0}^{t_m} - \mathbf{P}_{t_0}^{t_1}\mathbf{P}_{t_1}^{t_2}\cdots\mathbf{P}_{t_{m-1}}^{t_m}\right), \quad (9)$$

where $t_0, t_1, \ldots, t_m$ represent m ($m \leq 2$) consecutive time spans $(t_0, t_1), (t_1, t_2), \ldots, (t_{m-1}, t_m)$; $\mathbf{abs}(\cdot)$ is a matrix whose elements are the absolute values of the corresponding elements in the input matrix. The dimensionality of $\mathbf{E}_m$ is $n_c \times n_c$, where n_c is the number of cells in the mesh discretization. $E_m(t_0, t_1, \ldots, t_m; i_0, i_m)$—the element at the i_0 row and i_m column of $\mathbf{E}_m$—denotes the absolute error bound of decoupled SFM estimate of $p(0, m, i_0, i_m)$.

We define a matrix $\mathbf{A}$ less than or equal to another matrix $\mathbf{B}$ and write $\mathbf{A} \preccurlyeq \mathbf{B}$, if $\mathbf{A}$ and $\mathbf{B}$ are of the exact same dimension, and $A_{ij} \leq B_{ij}$ for all i and j.

Lemma 1. $\mathbf{abs}(\mathbf{AB}) \preccurlyeq \mathbf{A}\,\mathbf{abs}(\mathbf{B})$ *if* $\mathbf{B}$ *is non-negative.*

Proof. The lemma holds because $|\sum_k a_{ik}b_{kj}| \leq \sum_k |a_{ik}b_{kj}| = \sum_k |a_{ik}|\, b_{kj}$. □

Assumption 1. *The transition probability density function ρ is Lipschitz continuous.*[3]

The rationale is based on our observation that probability densities of SFMs are smooth. In our derivation, we assume there exists a positive real constant L such that for any starting locations $\mathbf{x}_1$ and $\mathbf{x}_2$, we have

$$|\rho(t_0, t_1, \mathbf{x}_1, \mathbf{y}) - \rho(t_0, t_1, \mathbf{x}_2, \mathbf{y})| \leq L\,\|\mathbf{x}_1 - \mathbf{x}_2\|, \quad (10)$$

where t_0, t_1, and $\mathbf{y}$ are the given start time, end time, and end location, respectively; $\|\cdot\|$ is the Euclidean norm in $\mathbb{R}^n$.

The Lipschitz constant L, which is directly related to the error bound in our further derivations, characterizes how fast ρ can change with respect to the starting location $\mathbf{x}$. For example, if ρ is continuously differentiable, L is the supremum of the derivative:

$$L = \sup\left\|\frac{\partial\rho(t_0, t_1, \mathbf{x}, \mathbf{y})}{\partial\mathbf{x}}\right\|. \quad (11)$$

To have an intuition of how large L may be, we compute the derivative of ρ in the synthetic uncertain double gyre case in Figure 5(b), which is estimated using a large number of particles as the ground truth for error evaluation. We use the central difference method to estimate the gradient of ρ with respect to starting location $\mathbf{x}$, and then compute the gradient magnitude with the 2-norm. As a result, the value of L—the supremum of gradient magnitude—is 14.86 in this case.

Rationale of the decoupled SFM estimate

We first review the rationale of Eq. (6), which is based on the Chapman-Kolmogorov equation:

$$p(t_0, t_2, i, k) \stackrel{(3)}{=} \int_{C_k} \rho(t_0, t_2, \bar{C}_i, \mathbf{z})d\mathbf{z}$$
$$= \int_{C_k}\int_{D} \rho(t_0, t_1, \bar{C}_i, \mathbf{y})\rho(t_1, t_2, \mathbf{y}, \mathbf{z})d\mathbf{y}d\mathbf{z}, \quad (\mathbf{y} \in D, \mathbf{z} \in C_k) \quad (12)$$

where D is the data domain (union of all cells in the mesh). We can further transform the integral by decomposing D into individual cells:

3. If we weaken the assumption to only assume ρ is continuous, we can still approximate the bounded error. See footnote 5 for more details.

$$\int_{C_k}\int_{D}\rho(t_0,t_1,\bar{C}_i,\mathbf{y})\rho(t_1,t_2,\mathbf{y},\mathbf{z})d\mathbf{y}d\mathbf{z}$$
$$=\sum_j\int_{C_k}\int_{C_j}\rho(t_0,t_1,\bar{C}_i,\mathbf{y})\rho(t_1,t_2,\mathbf{y},\mathbf{z})d\mathbf{y}d\mathbf{z}. \quad (\mathbf{y}\in C_j) \tag{13}$$

Based on the first mean value theorem for definite integrals,[4] for each cell C_j, there exists a point $\xi_j \in C_j$ such that

$$\sum_j\int_{C_k}\left(\int_{C_j}\rho(t_0,t_1,\bar{C}_i,\mathbf{y})\rho(t_1,t_2,\mathbf{y},\mathbf{z})d\mathbf{y}\right)d\mathbf{z}$$
$$=\sum_j\int_{C_k}\left(\rho(t_1,t_2,\xi_j,\mathbf{z})\int_{C_j}\rho(t_0,t_1,\bar{C}_i,\mathbf{y})d\mathbf{y}\right)d\mathbf{z}. \tag{14}$$

The above equation can be simplified:

$$\sum_j\int_{C_k}\rho(t_1,t_2,\xi_j,\mathbf{z})\int_{C_j}\rho(t_0,t_1,\bar{C}_i,\mathbf{y})d\mathbf{y}d\mathbf{z}$$
$$\overset{(3)}{=}\sum_j\int_{C_k}\rho(t_1,t_2,\xi_j;\mathbf{z})p(t_0,t_1,i,j)d\mathbf{z}$$
$$=\sum_j p(t_0,t_1,i,j)\int_{C_k}\rho(t_1,t_2,\xi_j,\mathbf{z})d\mathbf{z}. \tag{15}$$

Because ξ_j and $\bar{C}_j$ are in the same cell, the two points are very close. We thus approximate the above equation by

$$\sum_j p(t_0,t_1,i,j)\int_{C_k}\rho(t_1,t_2,\xi_j,\mathbf{z})d\mathbf{z} \tag{16}$$
$$\approx\sum_j p(t_0,t_1,i,j)\int_{C_k}\rho(t_1,t_2,\bar{C}_j,\mathbf{z})d\mathbf{z} \tag{17}$$
$$\overset{(3)}{=}\sum_j p(t_0,t_1,i,j)p(t_1,t_2,j,k). \tag{18}$$

The outcome of Eq. (18) above is the same as Eq. (6) in Section 3 of the paper. We further derive the bounded error of this approximation in the following.

Error of Coupling Two Consecutive Time Spans

Based on the above definition, the absolute error of coupling two consecutive time spans is:

$$E(t_0,t_1,t_2;i,k)$$
$$\overset{(9)}{=}\left|p(t_0,t_2,i,k)-\sum_j p(t_0,t_1,i,j)p(t_1,t_2,j,k)\right|$$
$$\underset{(17)}{\overset{(16)}{=}}\left|\sum_j p(t_0,t_1,i,j)\int_{C_k}\rho(t_1,t_2,\xi_j,\mathbf{z})d\mathbf{z}\right.$$
$$\left.-\sum_j p(t_0,t_1,i,j)\int_{C_k}\rho(t_1,t_2,\bar{C}_j,\mathbf{z})d\mathbf{z}\right|$$
$$=\sum_j p(t_0,t_1,i,j)\left|\int_{C_k}\rho(t_1,t_2,\xi_j,\mathbf{z})-\rho(t_1,t_2,\bar{C}_j,\mathbf{z})d\mathbf{z}\right|. \tag{19}$$

Because the absolute value of a definite integral of a function is less than or equal to the definite integral of the absolute value of a function, we have

$$\left|\int_{C_k}\rho(t_1,t_2,\xi_j,\mathbf{z})-\rho(t_1,t_2,\bar{C}_j,\mathbf{z})d\mathbf{z}\right|$$
$$\le\int_{C_k}\left|\rho(t_1,t_2,\xi_j,\mathbf{z})-\rho(t_1,t_2,\bar{C}_j,\mathbf{z})\right|d\mathbf{z}. \tag{20}$$

Based on our Lipschitz continuous assumption[5] in Eq. (10), Eq. (20) can be bounded by

$$\int_{C_k}\left|\rho(t_1,t_2,\xi_j,\mathbf{z})-\rho(t_1,t_2,\bar{C}_j,\mathbf{z})\right|d\mathbf{z}$$
$$\le\int_{C_k}L||\xi_j-\bar{C}_j||d\mathbf{z}. \tag{21}$$

Because ξ_j and $\bar{C}_j$ are in the same cell C_j, the distance must be less than or equal to $\sqrt{n}\delta$, with δ the maximum cell size in any dimension. We then have

$$\int_{C_k}L||\xi_j-\bar{C}_j||d\mathbf{z}\le\int_{C_k}L\cdot\delta d\mathbf{z}=\sqrt{n}L\delta V(C_k), \tag{22}$$

where $V(C_k)$—the volume of cell C_k—is bounded by $O(\delta^n)$. Thus, the error in Eq (19) is hereby bounded by

$$E(t_0,t_1,t_2;i,k)\le\sum_j p(t_0,t_1,i,j)\sqrt{n}L\delta V(C_k), \tag{23}$$

which will be eventually bounded by $O(L\delta^{n+1})$ in the next section. We introduce a non-negative matrix $\mathbf{\Theta}$, whose element at jth row and kth colum is $\theta_{j,k}=\sqrt{n}L\delta V(C_k)$, thus the above equation can be written in the matrix form:

$$\mathbf{E}_2(t_0,t_1,t_2)\preccurlyeq\mathbf{P}_{t_0}^{t_1}\mathbf{\Theta} \tag{24}$$

for any given t_0, t_1, and t_2.

Error of Coupling More Consecutive Time Spans

We next derive the bounded approximation error of coupling three consecutive time spans. We use the triangle inequality by arbitrarily inserting $\mathbf{P}_{t_0}^{t_1}\mathbf{P}_{t_1}^{t_2}\cdots\mathbf{P}_{t_{m-2}}^{t_m}$ into the equation:

$$\mathbf{E}_m(t_0,t_1,\ldots,t_m)\overset{(9)}{=}\mathbf{abs}(\mathbf{P}_{t_0}^{t_m}-\mathbf{P}_{t_0}^{t_1}\mathbf{P}_{t_1}^{t_2}\cdots\mathbf{P}_{t_{m-1}}^{t_m})$$
$$\preccurlyeq\mathbf{abs}(\mathbf{P}_{t_0}^{t_m}-\mathbf{P}_{t_0}^{t_1}\mathbf{P}_{t_1}^{t_2}\cdots\mathbf{P}_{t_{m-2}}^{t_m})$$
$$+\mathbf{abs}(\mathbf{P}_{t_0}^{t_1}\mathbf{P}_{t_1}^{t_2}\cdots\mathbf{P}_{t_{m-2}}^{t_m}-\mathbf{P}_{t_0}^{t_1}\mathbf{P}_{t_1}^{t_2}\cdots\mathbf{P}_{t_{m-1}}^{t_m}), \tag{25}$$

where the first component $\mathbf{abs}(\mathbf{P}_{t_0}^{t_m}-\mathbf{P}_{t_0}^{t_1}\mathbf{P}_{t_1}^{t_2}\cdots\mathbf{P}_{t_{m-2}}^{t_m})$ is $\mathbf{E}_{m-1}(t_0,t_1,\ldots,t_{m-2},t_m)$ by definition. The second component can be further bounded based on associativity of matrix product, non-negative of $\mathbf{P}$ matrices, and Lemma 1:

4. If f is continuous and g is integrable and does not change sign in a closed domain R, there exists $\xi\in R$ such that $\int_R f(x)g(x)dx=f(\xi)\int_R g(x)dx$.

5. If Lipschitz continuity is not satisfied, we can still get similar results. Because the range of a PDF is $[0,1]$, $\int_{C_k}\left|\rho(t_1,t_2,\xi_j,\mathbf{z})-\rho(t_1,t_2,\bar{C}_j,\mathbf{z})\right|d\mathbf{z}\le\int_{C_k}1d\mathbf{z}$. The error of coupling two time spans will be bounded by $O(\delta^n)$ in this case.

$$\begin{aligned}
&\mathbf{abs}\left(\mathbf{P}_{t_0}^{t_1}\mathbf{P}_{t_1}^{t_2}\cdots\mathbf{P}_{t_{m-2}}^{t_m}-\mathbf{P}_{t_0}^{t_1}\mathbf{P}_{t_1}^{t_2}\cdots\mathbf{P}_{t_{m-1}}^{t_m}\right)\\
\preccurlyeq&\mathbf{P}_{t_0}^{t_1}\mathbf{P}_{t_1}^{t_2}\cdots\mathbf{P}_{t_{m-3}}^{t_{m-2}}\mathbf{abs}(\mathbf{P}_{t_{m-2}}^{t_m}-\mathbf{P}_{t_{m-2}}^{t_{m-1}}\mathbf{P}_{t_{m-1}}^{t_m})\\
\overset{(9)}{=}&\mathbf{P}_{t_0}^{t_1}\mathbf{P}_{t_1}^{t_2}\cdots\mathbf{P}_{t_{m-3}}^{t_{m-2}}\mathbf{E}_2(t_{m-2},t_{m-1},t_m)\\
\overset{(24)}{\preccurlyeq}&\mathbf{P}_{t_0}^{t_1}\mathbf{P}_{t_1}^{t_2}\cdots\mathbf{P}_{t_{m-3}}^{t_{m-2}}\mathbf{P}_{t_{m-2}}^{t_{m-1}}\mathbf{\Theta}.
\end{aligned}\tag{26}$$

We further bound the above equation with the triangle inequality by arbitrarily inserting $\mathbf{P}_{t_0}^{t_{m-1}}$:

$$\begin{aligned}
&\mathbf{P}_{t_0}^{t_1}\mathbf{P}_{t_1}^{t_2}\cdots\mathbf{P}_{t_{m-2}}^{t_{m-1}}\mathbf{\Theta}\\
\preccurlyeq&\left(\mathbf{abs}(\mathbf{P}_{t_0}^{t_1}\mathbf{P}_{t_1}^{t_2}\cdots\mathbf{P}_{t_{m-2}}^{t_{m-1}}-\mathbf{P}_{t_0}^{t_{m-1}})+\mathbf{abs}(\mathbf{P}_{t_0}^{t_{m-1}})\right)\mathbf{\Theta}\\
\overset{(9)}{=}&\mathbf{E}_{m-1}(t_0,t_1,\ldots,t_{m-1})\mathbf{\Theta}+\mathbf{P}_{t_0}^{t_{m-1}}\mathbf{\Theta}
\end{aligned}\tag{27}$$

By combining the two components, Eq. (25) can be written as a recursive form:

$$\begin{aligned}
&\mathbf{E}_m(t_0,t_1,\ldots,t_m)\\
\preccurlyeq&\mathbf{E}_{m-1}(t_0,t_1,\ldots,t_{m-2},t_m)+\mathbf{P}_{t_0}^{t_{m-1}}\mathbf{\Theta}\\
&+\mathbf{E}_{m-1}(t_0,t_1,\ldots,t_{m-1})\mathbf{\Theta}.
\end{aligned}\tag{28}$$

We use mathematical induction to derive the non-recursive form of the above equation:

$$\begin{aligned}
&\mathbf{E}_3(t_0,t_1,t_2,t_3)\\
\overset{(28)}{\preccurlyeq}&\mathbf{E}_2(t_0,t_1,t_3)+\mathbf{P}_{t_0}^{t_2}\mathbf{\Theta}+\mathbf{E_2}(t_0,t_1,t_2)\mathbf{\Theta}\\
\overset{(24)}{\preccurlyeq}&(\mathbf{P}_{t_0}^{t_1}+\mathbf{P}_{t_0}^{t_2})\mathbf{\Theta}+\mathbf{P}_{t_0}^{t_1}\mathbf{\Theta}^2
\end{aligned}\tag{29}$$

for any given t_0, t_1, t_2, and t_3. Likewise, for four time spans,

$$\begin{aligned}
&\mathbf{E}_4(t_0,t_1,t_2,t_3,t_4)\\
\overset{(28)}{\preccurlyeq}&\mathbf{E}_3(t_0,t_1,t_2,t_4)+\mathbf{P}_{t_0}^{t_3}\mathbf{\Theta}+\mathbf{E}_3(t_0,t_1,t_2,t_3)\mathbf{\Theta}\\
\overset{(29)}{\preccurlyeq}&\left(\mathbf{P}_{t_0}^{t_1}\mathbf{\Theta}+\mathbf{P}_{t_0}^{t_2}\mathbf{\Theta}\right)+\mathbf{P}_{t_0}^{t_3}\mathbf{\Theta}\\
&+\left(\mathbf{P}_{t_0}^{t_1}\mathbf{\Theta}+\mathbf{P}_{t_0}^{t_2}\mathbf{\Theta}+\mathbf{P}_{t_0}^{t_1}\mathbf{\Theta}^2\right)\mathbf{\Theta}\\
\preccurlyeq&(\mathbf{P}_{t_0}^{t_1}+\mathbf{P}_{t_0}^{t_2}+\mathbf{P}_{t_0}^{t_3})\mathbf{\Theta}+(\mathbf{P}_{t_0}^{t_1}+\mathbf{P}_{t_0}^{t_2})\mathbf{\Theta}^2+\mathbf{P}_{t_0}^{t_1}\mathbf{\Theta}^3.
\end{aligned}\tag{30}$$

We can further recursively bound $\mathbf{E}$ for arbitrary m ($m \geq 2$)

$$\mathbf{E}_m(t_0,t_1,\ldots,t_m)\preccurlyeq\sum_{r=1}^{m-1}\sum_{q=1}^{m-r}\mathbf{P}_{t_0}^{t_q}\mathbf{\Theta}^r.\tag{31}$$

For individual elements in $\mathbf{E}_m$, we can bound the error by safely dropping all higher order infinitesimals of $\mathbf{\Theta}$:

$$\begin{aligned}
&E_m(t_0,t_1,\ldots,t_m;i,j)\\
\overset{(31)}{\leq}&\sum_{q=1}^{m-1}\sum_j p(t_0,t_q,i,j)\sqrt{n}L\delta V(C_j)\\
\leq&\sqrt{n}L\delta\max_j\{V(C_j)\}\sum_{q=1}^{m-1}\sum_j p(t_0,t_q,i,j)\\
=&\sqrt{n}L\delta\max_j\{V(C_j)\}\sum_{q=1}^{m-1}1\\
\leq&\sqrt{n}L\delta(m-2)\delta^n=O(mL\delta^{n+1}).
\end{aligned}\tag{32}$$

Overall, the absolute error of decoupled SFM estimate is no larger than $O(mL\delta^{n+1})$, which is related to the maximum cell size in any dimension δ, number of time steps that are coupled for SFM estimate m, and the Lipschitz constant L that characterizes how fast SFMs can change. We can also evaluate the bound of RMSE:

$$RMSE=\sqrt{\frac{\sum_i\sum_j E_m(t_0,t_1,\ldots,t_m;i,j)^2}{n_c^2}}\leq O(mL\delta^{n+1}).\tag{33}$$

Figures 6 and 14 serve as an empirical error study with respect to δ and m. We can see that fewer coupled time spans (smaller m) and finer discretization (smaller δ) leads to more precise results.

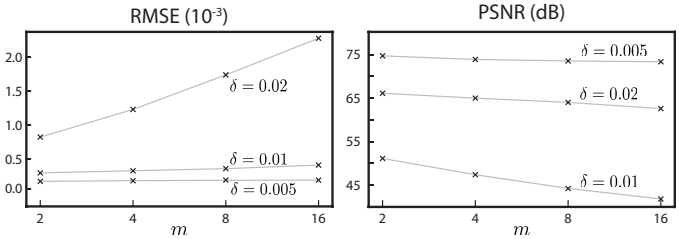


Fig. 14. RMSE and PSNR values of decoupled SFM estimate with respect to different δ and m in the synthetic uncertain double gyre data.

ACKNOWLEDGMENTS

The authors would like to thank Dr. Hong Zhang and Dr. Julie Bessac for useful discussions. Dr. Chunhui Liu is supported by JSPS KAKENHI Grant Number JP17F17730. This material is based upon work supported by the U.S. Department of Energy, Office of Science, under contract number DE-AC02-06CH11357. This work is also supported by the U.S. Department of Energy, Office of Advanced Scientific Computing Research, Scientific Discovery through Advanced Computing (SciDAC) program. This research used resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357. This research also used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

REFERENCES

[1] M. Otto, T. Germer, and H. Theisel, "Closed stream lines in uncertain vector fields," in *SCCG'11: Proceedings of the 27th Spring Conference on Computer Graphics*, 2013, pp. 87–94.

[2] H. Guo, W. He, T. Peterka, H.-W. Shen, S. M. Collis, and J. J. Helmus, "Finite-time Lyapunov exponents and Lagrangian coherent structures in uncertain unsteady flows," *IEEE Trans. Vis. Comput. Graph.*, vol. 22, no. 6, pp. 1672–1682, 2016.

[3] D. Schneider, J. Fuhrmann, W. Reich, and G. Scheuermann, "A variance based FTLE-like method for unsteady uncertain vector fields," in *Topological Methods in Data Analysis and Visualization II*, ser. Mathematics and Visualization, R. Peikert, H. Hauser, H. Carr, and R. Fuchs, Eds. Springer, 2011, pp. 255–268.

[4] M. Otto, T. Germer, H.-C. Hege, and H. Theisel, "Uncertain 2D vector field topology," *Comput. Graph. Forum*, vol. 29, no. 2, pp. 347–356, 2010.

[5] M. Otto, T. Germer, and H. Theisel, "Uncertain topology of 3D vector fields," in *Proceedings of IEEE Pacific Visualization Symposium 2011*, 2011, pp. 67–74.

[6] M. Otto and H. Theisel, "Vortex analysis in uncertain vector fields," *Comput. Graph. Forum*, vol. 31, no. 3, pp. 1035–1044, 2012.

[7] 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 ACM/IEEE Conference on Supercomputing*, 2012, pp. 61:1–61:11.

[8] W. Kendall, J. Wang, M. Allen, T. Peterka, J. Huang, and D. Erickson, "Simplified parallel domain traversal," in *SC'11: Proceedings of the ACM/IEEE Conference on Supercomputing*, 2011, pp. 10:1–10:11.

[9] B. Øksendal, *Stochastic Differential Equations*. Springer-Verlag Berlin Heidelberg, 2014.

[10] F. Sadlo and R. Peikert, "Efficient visualization of Lagrangian coherent structures by filtered AMR ridge extraction," *IEEE Trans. Vis. Comput. Graph.*, vol. 13, no. 6, pp. 1456–1463, 2007.

[11] S. S. Barakat and X. Tricoche, "Adaptive refinement of the flow map using sparse samples," *IEEE Trans. Vis. Comput. Graph.*, vol. 19, no. 12, pp. 2753–2762, 2013.

[12] M. Hlawatsch, F. Sadlo, and D. Weiskopf, "Hierarchical line integration," *IEEE Trans. Vis. Comput. Graph.*, vol. 17, no. 8, pp. 1148–1163, 2011.

[13] C. R. Johnson and A. R. Sanderson, "A next step: Visualizing errors and uncertainty," *IEEE Comput. Graph. Appl.*, vol. 23, no. 5, pp. 6–10, 2003.

[14] K. Brodlie, R. A. Osorio, and A. Lopes, "A review of uncertainty in data visualization," in *Expanding the Frontiers of Visual Analytics and Visualization*, J. Dill, R. Earnshaw, D. Kasik, J. Vince, and P. C. Wong, Eds. Springer London, 2012, pp. 81–109.

[15] R. S. Laramee, H. Hauser, H. Doleisch, B. Vrolijk, F. H. Post, and D. Weiskopf, "The state of the art in flow visualization: Dense and texture-based techniques," *Comput. Graph. Forum*, vol. 23, no. 2, pp. 203–222, 2004.

[16] F. H. Post, B. Vrolijk, H. Hauser, R. S. Laramee, and H. Doleisch, "The state of the art in flow visualization: Feature extraction and tracking," *Comput. Graph. Forum*, vol. 22, no. 4, pp. 1–17, 2003.

[17] A. Pobitzer, R. Peikert, R. Fuchs, B. Schindler, A. Kuhn, H. Theisel, K. Matkovic, and H. Hauser, "The state of the art in topology-based visualization of unsteady flow," *Comput. Graph. Forum*, vol. 30, no. 6, pp. 1789–1811, 2011.

[18] C. M. Wittenbrink, A. Pang, and S. K. Lodha, "Glyphs for visualizing uncertainty in vector fields," *IEEE Trans. Vis. Comput. Graph.*, vol. 2, no. 3, pp. 266–279, 1996.

[19] R. P. Botchen, D. Weiskopf, and T. Ertl, "Texture-based visualization of uncertainty in flow fields," in *Proceedings of IEEE Visualization 2005*, 2005, pp. 647–654.

[20] H. Guo, X. Yuan, J. Huang, and X. Zhu, "Coupled ensemble flow line advection and analysis," *IEEE Trans. Vis. Comput. Graph.*, vol. 19, no. 12, pp. 2733–2742, 2013.

[21] T. Höllt, M. Hadwiger, O. Knio, and I. Hoteit, "Probability Maps for the Visualization of Assimilation Ensemble Flow Data," in *Workshop on Visualisation in Environmental Sciences (EnvirVis)*, A. Middel, K. Rink, and G. H. Weber, Eds. The Eurographics Association, 2015.

[22] R. T. Whitaker, M. Mirzargar, and R. M. Kirby, "Contour boxplots: A method for characterizing uncertainty in feature sets from simulation ensembles," *IEEE Trans. Vis. Comput. Graph.*, vol. 19, no. 12, pp. 2713–2722, 2013.

[23] E. W. Bethel, H. Childs, and C. Hansen, *High Performance Visualization: Enabling Extreme-Scale Scientific Insight*. CRC Press, 2012.

[24] 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*, 2011, pp. 580–591.

[25] B. Nouanesengsy, T.-Y. Lee, and H.-W. Shen, "Load-balanced parallel streamline generation on large scale vector fields," *IEEE Trans. Vis. Comput. Graph.*, vol. 17, no. 12, pp. 1785–1794, 2011.

[26] H. Yu, C. Wang, and K.-L. Ma, "Parallel hierarchical visualization of large time-varying 3D vector fields," in *SC'07: Proceedings of the ACM/IEEE Conference on Supercomputing*, 2007, pp. 24:1–24:12.

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

[28] D. Pugmire, H. Childs, C. Garth, S. Ahern, and G. H. Weber, "Scalable computation of streamlines on very large datasets," in *SC'09: Proceedings of the ACM/IEEE Conference on Supercomputing*, 2009, pp. 16:1–16:12.

[29] 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 Trans. Vis. Comput. Graph.*, vol. 20, no. 12, pp. 2555–2564, 2014.

[30] D. Camp, C. Garth, H. Childs, D. Pugmire, and K. I. Joy, "Parallel stream surface computation for large data sets," in *LDAV'12: Proceedings of IEEE Symposium on Large Data Analysis and Visualization*, 2012, pp. 39–47.

[31] K. Lu, H. Shen, and T. Peterka, "Scalable computation of stream surfaces on large scale vector fields," in *SC'14: Proceedings of the ACM/IEEE Conference on Supercomputing*, 2014, pp. 1008–1019.

[32] C. Mueller, D. Camp, B. Hentschel, and C. Garth, "Distributed parallel particle advection using work requesting," in *LDAV'13: Proceedings of IEEE Symposium on Large Data Analysis and Visualization*, 2013, pp. 109–112.

[33] 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 Trans. Vis. Comput. Graph.*, vol. 1, pp. 954–963, 2018.

[34] D. Camp, C. Garth, H. Childs, D. Pugmire, and K. I. Joy, "Streamline integration using MPI-hybrid parallelism on a large multicore architecture," *IEEE Trans. Vis. Comput. Graph.*, vol. 17, no. 11, pp. 1702–1713, 2011.

[35] D. Camp, H. Krishnan, D. Pugmire, C. Garth, I. Johnson, E. W. Bethel, K. I. Joy, and H. Childs, "GPU acceleration of particle advection workloads in a parallel, distributed memory setting," in *EGPGV'13: Proceedings of Eurographics Parallel Graphics and Visualization Symposium*, 2013, pp. 1–8.

[36] T. Günther, A. Kuhn, and H. Theisel, "MCFTLE: monte carlo rendering of finite-time Lyapunov exponent fields," *Comput. Graph. Forum*, vol. 35, no. 3, pp. 381–390, 2016.

[37] C. Birdsall and D. Fuss, "Cloud-in-cell computer experiments in two and three dimensions," California Univ., Livermore. Lawrence Radiation Lab., Tech. Rep., 1969.

[38] N. Hofmann, T. Mueller-Gronbach, and K. Ritter, "The optimal discretization of stochastic differential equations," *Journal of Complexity*, vol. 17, no. 1, pp. 117–153, 2001.

[39] D. Morozov and T. Peterka, "Block-parallel data analysis with DIY2," in *LDAV'16: Proceedings of IEEE Symposium on Large Data Analysis and Visualization*, 2016, pp. 29–36.

[40] 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*, vol. 31, no. 6, pp. 6–10, 2011.

[41] S. Balay, W. D. Gropp, L. C. McInnes, and B. F. Smith, "Efficient management of parallelism in object oriented numerical software libraries," in *Modern Software Tools in Scientific Computing*, E. Arge, A. M. Bruaset, and H. P. Langtangen, Eds. Birkhäuser Press, 1997, pp. 163–202.

[42] C.-M. Chen, A. Biswas, and H.-W. Shen, "Uncertainty modeling and error reduction for pathline computation in time-varying flow fields," in *Proceedings of IEEE Pacific Visualization Symposium 2015*, 2015, pp. 215–222.

[43] C. Alexander, D. C. Dowell, S. S. Weygandt, S. G. Benjamin, M. Hu, T. G. Smirnova, J. B. Olson, J. M. Brown, E. P. James, and P. Hofmann, "The high-resolution rapid refresh: Recent model and data assimilation development towards an operational implementation in 2014," in *Proceedings of 26th Conference on Weather Analysis and Forecasting / 22nd Conference on Numerical Weather Prediction*. American Meterological Society, 2014.

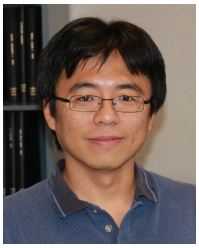
[44] G. Haller, "Distinguished material surfaces and coherent structures in three-dimensional fluid flows," *Physica D: Nonlinear Phenomena*, vol. 149, no. 4, pp. 248–277, 2001.

[45] T. Peterka, R. B. Ross, W. Kendall, A. Gyulassy, V. Pascucci, H.-W. Shen, T.-Y. Lee, and A. Chaudhuri, "Scalable parallel building blocks for custom data analysis," in *LDAV'11: Proceedings of IEEE Symposium on Large Data Analysis and Visualization*, 2011, pp. 105–112.

Hanqi Guo is an assistant computer scientist in the Mathematics and Computer Science Division, Argonne National Laboratory. He received his Ph.D. degree in computer science from Peking University in 2014, and the B.S. degree in mathematics and applied mathematics from Beijing University of Posts and Telecommunications in 2009. His research interests are mainly in flow visualization, uncertainty visualization, and large-scale scientific data visualization.

Wenbin He is a Ph.D. student in computer science and engineering at the Ohio State University. He received his B.S. degree from the Department of Software Engineering at Beijing Institute of Technology in 2012. His research interests include analysis and visualization of large-scale scientific data, uncertainty visualization, and flow visualization.

Sangmin Seo is a software engineer at Ground X, where he develops a next-generation blockchain platform focusing on high-performance, scalability, and service-friendliness. He received the B.S. degree in computer science and engineering and the Ph.D. degree in electrical engineering and computer science from Seoul National University, respectively. After obtaining his Ph.D. degree, he worked as a CEO at ManyCoreSoft Co., Ltd and then joined Argonne National Laboratory where he conducted research on threading models and their interaction with communication runtimes. After that, he moved to Samsung Research and developed a compiler and runtime system specialized for handling neural network models targeting on-device AI. His research interests include high-performance computing, parallel programming models, compilers, runtime systems, artificial intelligence, and blockchains.



Han-Wei Shen is a full professor at the Ohio State University. He received his B.S. degree from Department of Computer Science and Information Engineering at National Taiwan University in 1988, the M.S. degree in computer science from the State University of New York at Stony Brook in 1992, and the Ph.D. degree in computer science from the University of Utah in 1998. From 1996 to 1999, he was a research scientist at NASA Ames Research Center in Mountain View California. His primary research interests are scientific visualization and computer graphics. He is a winner of the National Science Foundation's CAREER award and U.S. Department of Energy's Early Career Principal Investigator Award. He also won the Outstanding Teaching award twice in the Department of Computer Science and Engineering at the Ohio State University.

Emil Mihai Constantinescu is a Computational Mathematician with the Mathematics and Computer Science Division, Argonne National Laboratory. He received his Ph.D. degree in computer science from Virginia Tech, Blacksburg, in 2008. His research interests include numerical analysis of time-stepping and stochastic algorithms and their applications to dynamical systems and high performance computing.

Chunhui Liu is a JSPS international research fellow working in Department of Mathematics, Faculty of Science, Kyoto University. He gets his Ph. D. degree in mathematics from Université Paris Diderot - Paris 7, France.

Tom Peterka is a computer scientist at Argonne National Laboratory, fellow at the Computation Institute of the University of Chicago, adjunct assistant professor at the University of Illinois at Chicago, and fellow at the Northwestern Argonne Institute for Science and Engineering. His research interests are in large-scale parallelism for in situ analysis of scientific data. His work has led to three best paper awards and publications in ACM SIGGRAPH, IEEE VR, IEEE TVCG, and ACM/IEEE SC, among others. Peterka received his Ph.D. in computer science from the University of Illinois at Chicago, and he currently works actively in several DOE- and NSF-funded projects.