Implementation of a GPU-Accelerated Lagrangian Particle Dispersion Model for Atmospheric Transport of Radioactive Nuclides

Authors: Li, Mr. Qing-Yun, He, Mr. Tao, Li, Dr. MingYe, Zhang, Miss Jun-Fang, Lian, Mr. Bing, Liu, Prof. Liye, QIU, Dr. RUI 邱睿, LI, Prof. JUNLI, Li, Mr. Qing-Yun

Date: 2026-01-22T19:35:31+00:00

Abstract

Large-scale atmospheric dispersion modeling for emergency response to nuclear accidents requires both high computational efficiency and numerical reliability. A GPU-oriented Lagrangian particle dispersion modeling was developed within the FLEXPART framework to address these demands. Core transport processes—including advection, turbulent diffusion, convective mixing, and dry and wet deposition—were restructured for GPU parallel execution, achieving efficient operation on modern accelerator architectures. Further incorporation of fast arithmetic operators and multi-level parallelization strategies substantially improved overall computational performance while preserving physical accuracy. Additional MPI-based parallel meteorological data decoupling and preprocessing tool has been developed, which alleviates data-handling bottlenecks. While multi-GPU execution and a load-balancing strategy enable efficient scaling in heterogeneous computing environments. Based on the European Tracer Experiment' s first trial (ETEX-I), the computational accuracy and acceleration performance of the developed GPU program were rigorously evaluated. The results demonstrate that the program achieves simulation accuracy comparable to the benchmark reference while exhibiting modest improvements in the representation of turbulent mixing under certain conditions. Performance assessments indicate an overall speedup of approximately 40.45 on a single-GPU platform, which can be further increased to about 52.05 in high-performance application scenarios where meteorological background fields are reusable. Moreover, multi-GPU experiments reveal favorable parallel scalability across configurations ranging from one to four GPUs, and confirm that the proposed load-balancing strategy effectively enhances computational efficiency in heterogeneous GPU environments.

Full Text

Preamble

Implementation of a GPU-Accelerated Lagrangian Particle Dispersion Model for Atmospheric Transport of Radioactive Nuclides Qing-Yun Li,1, 2, 3, * Tao He,1, * Ming-Ye Li,1 Jun-Fang Zhang,1 Bing Lian,1 Li-Ye Liu,1 Rui Qiu,2, 3, † and Jun-Li Li2, 3 1China Institute of Radiation Protection, Taiyuan 030006, China 2Department of Engineering Physics, Tsinghua University, Beijing 100084, China 3Key Laboratory of Particle & Radiation Imaging, Tsinghua University, Ministry of Education, Beijing 100084, China Large-scale atmospheric dispersion modeling for emergency response to nuclear accidents requires both high computational efficiency and numerical reliability. A GPU-oriented Lagrangian particle dispersion modeling was developed within the FLEXPART framework to address these demands. Core transport processes—including advection, turbulent diffusion, convective mixing, and dry and wet deposition—were restruc- tured for GPU parallel execution, achieving efficient operation on modern accelerator architectures. Further in- corporation of fast arithmetic operators and multi-level parallelization strategies substantially improved overall computational performance while preserving physical accuracy. Additional MPI-based parallel meteorological data decoupling and preprocessing tool has been developed, which alleviates data-handling bottlenecks. While multi-GPU execution and a load-balancing strategy enable efficient scaling in heterogeneous computing envi- ronments. Based on the European Tracer Experiment' s first trial (ETEX-I), the computational accuracy and acceleration performance of the developed GPU program were rigorously evaluated. The results demonstrate that the program achieves simulation accuracy comparable to the benchmark reference while exhibiting mod- est improvements in the representation of turbulent mixing under certain conditions. Performance assessments indicate an overall speedup of approximately 40.45 on a single-GPU platform, which can be further increased to about 52.05 in high-performance application scenarios where meteorological background fields are reusable.

Moreover, multi-GPU experiments reveal favorable parallel scalability across configurations ranging from one to four GPUs, and confirm that the proposed load-balancing strategy effectively enhances computational effi- ciency in heterogeneous GPU environments.

Keywords: GPU Acceleration, Lagrangian Dispersion Model, Emergency Response

INTRODUCTION

Driven by the ongoing transformation of the global energy structure and low-carbon development objectives, nuclear en- ergy continues to expand as a major source of low-emission power, accompanied by an elevated potential risk of nu-

clear accidents and inadvertent releases of radioactive materials. In the event of an accident, substantial quantities of radionu- clides may be emitted into the atmosphere, posing severe threats to public health and ecological security[1]. Numer- ical atmospheric dispersion models constitute a cornerstone for consequence assessment and emergency decision support in such scenarios, with their computational speed and numer- ical reliability directly governing the timeliness and accuracy of response actions[2–5]. processes dispersion Atmospheric intrinsically complex[6]. As a result, contemporary dispersion models necessarily rely on low-order approximations or empirically than first-principles parameterized representations rather transport and formulations when simulating pollutant removal[7, 8].

Previous studies have demonstrated that simulated concentration fields are highly sensitive to the parameterization of dry and wet deposition[9–12], turbulent mixing[9–11], and boundary-layer dynamics[10, 11, 13].

Accurately resolving plume evolution under complex mete- orological regimes—particularly within unstable convective * These authors contributed equally to this work. † Corresponding author, qiurui@tsinghua.edu.cn.

Moreover, boundary layers—often requires higher-order and more sophisticated parameterization schemes[14, 15]. Although such schemes can substantially reduce simulation errors, they incur a pronounced increase in computational cost[16].

Under nuclear-emergency conditions, the adoption of high- fidelity physical representations therefore entails a trade-off between numerical accuracy and computational efficiency, intensifying constraints on model responsiveness[4, 17, 18]. the Chernobyl experiences of Fukushima that during emer- accidents demonstrate gency situations, power outages caused by the incidents make it difficult to obtain accurate source terms[19–21].

Concurrently, meteorological forecast products also carry significant uncertainties[6, 22, 23], leading to a lack of comprehensive conditions for dispersion simulations. Un- reliable estimation of atmospheric der these conditions, radionuclide distributions commonly depends on auxiliary approaches such as source-term inversion[24–28] and data assimilation[29–32] both of which require a large number of dispersion simulations[9, 20, 33]. Source-term inversion is particularly computationally demanding, as the accuracy the inferred release parameters critically determines the credibility of subsequent predictions[34–36], while prevailing inversion algorithms often involve thousands of sequential dispersion runs, as shown in table 1 . This computational burden constitutes a major bottleneck for time-critical decision making during emergencies[37–40].

The simultaneous demand for high timeliness and high nu- merical accuracy in atmospheric dispersion simulations has increasingly exposed the computational limitations of con- In contrast, the rapid ad- ventional CPU-based platforms.

Table 1. The number of iterations used in typical inversion algorithm studies.

Author (Year) Convergence iterations 3000 Zhao et al. 2400 Li et al.

Xu et al. vancement of graphics processing units (GPUs), character- ized by massive parallelism and high throughput, has opened new technical avenues for alleviating computational burdens and enhancing overall simulation efficiency[44–47]. Among existing dispersion modeling approaches, Lagrangian particle models are widely employed[48, 49] because of their abil- ity to maintain robust numerical accuracy under large-scale, heterogeneous, and highly complex turbulent conditions. By explicitly tracking the trajectories of a large ensemble of representative particles, these models describe pollutant trans- port and diffusion in a statistically consistent manner. creasing the particle number improves the statistical stability of the simulation results—where the associated error scales inversely with the square root of the particle count—while also leading to a proportional increase in computational cost.

Importantly, the mutual independence of particles in La- grangian formulations aligns naturally with the "high paral- lelism, weak dependency" paradigm of GPU architectures, making such models intrinsically well suited for large-scale GPU acceleration[46, 50].

Recent studies have explored the potential of accelerat- ing Lagrangian particle models on heterogeneous computing platforms. Existing efforts have demonstrated that paralleliz- ing core particle-update loops can yield measurable perfor- mance gains on both multi-core CPUs and GPUs. However, reported speedups typically range from 10 to 15 times[44, 46, 50] and often saturate as particle counts approach hard- ware concurrency limit[50]. Moreover, most implementa- tions focus on partial offloading of selected computational kernels[46, 50], leaving substantial portions of the physical processes, data management, and input–output operations on the CPU.As a result, the overall acceleration is constrained by residual serial components and data-transfer overheads.

In addition, the majority of current GPU-based implementa- tions are restricted to single-device execution, lacking scal- able multi-GPU or cross-node capabilities required for large- ensemble simulations. More fundamentally, these studies pri- marily pursue technical porting and generic performance im- provement, without explicitly accounting for the operational characteristics and constraints of nuclear-emergency response scenarios. Consequently, despite demonstrated benefits, ex- isting GPU-accelerated Lagrangian particle models remain insufficient to simultaneously satisfy the stringent require- ments for accuracy and real-time responsiveness under emer- gency conditions.

The present study builds upon the internationally estab- lished and extensively validated FLEXPART framework[14, 15] to develop a GPU-oriented Lagrangian dispersion model tailored to the demands of emergency-response applications.

First, focusing on the most computationally intensive physi- cal processes of pollutant dispersion—including particle ad- vection, turbulent random displacement, convective mixing, and wet deposition—a fine-grained data parallel strategy was systematically implemented to port these processes entirely to the GPU.

With tailored memory configurations and com- putational flow optimizations specific to each physical mech- anism, the approach overcomes previous constraints of par- tial parallelization limited to core loops without covering the complete physical processes. Subsequently, by integrating fast arithmetic instructions, the execution latency of funda- mental operators was effectively reduced, further enhanc- ing the computational throughput of GPU kernels. Furthermore, leveraging the architectural characteristics of GPUs, optimizations were carried out from both the parallel gran- ularity and resource utilization perspectives. On one hand, a coarse-grained parallel strategy was introduced to appropri- ately adjust the mapping of computational tasks onto Stream- ing Multiprocessors (SMs), aligning with GPU asynchronous scheduling and compute–memory access overlapping mech- anisms, thereby boosting overall parallel throughput. On the other hand, through precise control of the number of registers per thread, register spilling was minimized and thread concur- rency was increased, significantly improving GPU resource utilization efficiency. These synergistic optimization strate- gies collectively enhanced GPU computational performance.

Considering the high reusability of background meteorolog- ical fields in emergency simulations, this study structurally decoupled the reading and computational processes of such fields. Employing an MPI-based preprocessing and decoding strategy for meteorological data enabled efficient parallel exe- cution during the preprocessing phase, substantially reducing I/O overhead while improving the efficiency of emergency technologies such as source inversion and data assimilation that rely on reusable background fields. Finally, a multi-GPU particle domain decomposition and load-balancing method was introduced to achieve scalable parallel capability across multiple GPUs. The accelerated performance and simula- tion accuracy of the developed program were systematically and comprehensively validated using the release scenario and measurement data from the European Tracer Experiment' s first trial (ETEX-I)[51].

II. MATERIALS AND METHODS A. Lagrangian Particle Model and the FLEXPART Framework The atmospheric transport of radioactive contaminants can be described from a Lagrangian perspective by a system of partial differential equations (PDEs): $= \nabla \cdot (K\nabla c) - vdc\delta(z) - \Lambda c - \lambda c + q(x, y, z, t)$ (1) Here, c denotes the material (Lagrangian-following) concen- tration of the contaminant, K represents the turbulent diffu- sion coefficient tensor, vd, Λ and λ denote the dry deposi- tion velocity, the wet scavenging coefficient, and the radioac- tive decay constant, respectively. The Dirac delta function, $\delta(z)$, constrains dry deposition to occur only within the near- surface layer[52, 53]. The term q(x, y, z, t) represents the source release-rate density as a function of spatial position (x, y, z) and time t.

In the Lagrangian particle approach, an ensemble of com- putational particles is used to represent the contaminant plume. By releasing and tracking these particles through- out the atmosphere, key processes such as advective trans-

port, turbulent diffusion, and depositional removal are explic- itly resolved[54, 55]. The motion of each particle is assumed to be independent and governed solely by the local meteoro- logical conditions at its instantaneous position, with particle states updated at each time step. Under this assumption, the diffusive behavior described by eq. 2 can be simplified for an individual particle as follows:

X t+Δt = X t + U Δt − vg Δt ˆz, µt+Δt = µt e−(Λ+Λd+λ)Δt, 2href z < 2href , z ≥ 2href .

Here, X = (x, y, z) denotes the particle position vec- tor, and µ represents the mass or activity of radioactive con- taminants carried by an individual particle. U = ¯U + U is the wind velocity vector at the particle location, where ¯U = (¯u, ¯v, ¯w) denotes the mean wind velocity interpolated) represents from meteorological fields, and U the stochastic turbulent velocity fluctuations in each spatial direction. The statistical properties of U are estimated based on boundary-layer parameters such as friction velocity and are generated using a Langevin stochastic differential equa- tion or related random-process models to characterize the ran- dom perturbations induced by atmospheric turbulence. vg de- notes the gravitational settling coefficient, and ˆz = (0, 0, 1) specifies the direction of gravitational settling. The parame- ters Λd and 2href represent the dry deposition removal coef- ficient and the effective thickness of the dry deposition layer, respectively. The term Δt denotes the minimum time step used in the simulation.

Based on the discrete particle update formulation de- scribed above, the FLEX-PART model adopts the computa- tional workflow illustrated in Fig. 1 [Figure 1: see original paper]. At each time step ti, the model first performs advection and turbulent diffusion calcu- lations using the meteorological fields and particle distribu- tions from the previous time step ti−1, thereby updating par- ticle positions. Subsequently, dry-deposition–induced mass loss is evaluated according to particle height, and the radioac- tive decay term is applied. After these operations, the me- teorological fields are updated to the current time level and used to compute wet deposition removal, while the pollutant source emissions corresponding to the current time step are read in parallel. The convective process is then executed to update the vertical positions of the particles. Finally, following the application of radioactive decay to the deposited mass, the model outputs all state variables for the current time step, including concentration fields, particle trajectories, and depo- sition fluxes1.

Fig. 2 [Figure 2: see original paper] illustrates the relative computational time contribu- tions of the major modules involved in the dispersion sim- ulation. As shown, the dominant computational cost arises from the advection–diffusion module, which, in addition to advective transport and stochastic turbulent displacement, in- cludes key operations such as aerosol gravitational settling and dry-deposition probability calculations. The convective mixing and wet deposition modules together account for most of the remaining computational workload. Because all of these processes require particle-by-particle updates at every time step, the overall computational complexity scales lin- early with both the

number of particles and the number of time steps. This characteristic endows the dispersion model with pronounced data parallelism, making it inherently well suited for large-scale parallelization on GPU architectures. In addition, the Get fields module is responsible for reading and decoding multi-level meteorological fields from GRIB and related encoded formats, while also performing the neces- sary coordinate transformations and physical variable deriva- tions to generate all meteorological drivers required for the dispersion calculations. This procedure involves sequential decoding of meteorological data and intensive access to grid- ded fields during time stepping, and therefore accounts for a non-negligible fraction of the total runtime. In particular, under scenarios with relatively small particle counts and long time steps, the overhead associated with background-field de- coding and preprocessing may exceed half of the total prop- agation cost, thereby becoming a critical factor limiting the overall computational efficiency of the model.

B. Fine-Grained Parallel Acceleration Architecture To fully exploit the parallel computing capability of GPUs and achieve maximal performance gains, a full offloading strategy was adopted, in which all core subroutines originally executed on the CPU were restructured and migrated to run entirely on the GPU device. This approach eliminates fre- quent host–device data transfers and the associated commu- nication overhead, thereby enabling end-to-end acceleration of the computational workflow. Building on this design, and leveraging the inherent data-parallel nature of the Lagrangian particle model, a fine-grained GPU parallelization scheme was employed in which each particle is mapped to a single GPU thread.

1 The above description presents the workflow with reference to the beginning

of each time step. In the actual FLEXPART program structure, however, output generation and certain diagnostic operations are performed at intermediate points within the time step, resulting in a temporal shift in the execution order. The exact implementation sequence is illustrated in Fig. 1.

Fig. 1. Schematic overview of the FLEXPART computational framework.

1. Memory Allocation and Data Transfer

The GPU employs a hierarchical memory architecture comprising registers, L1/L2 caches, and global memory. In the Lagrangian particle dispersion model, each particle is processed by an independent thread, which requires effi- cient access to meteorological fields, control parameters, and particle-specific variables. Consequently, data layout and memory-management strategies play a decisive role in re- ducing memory-access latency and improving overall parallel performance.

Because particle positions evolve dynamically and cannot be predetermined, the

model must support fast, random ac- cess to background-field data over the full spatial domain.

All relevant background data are therefore resident on the GPU. To mitigate the memory-bandwidth pressure caused by concurrent global-memory access, time-invariant quanti- ties—including particle attributes, simulation control parameters, and meteorological-field dimensions—are stored in GPU constant memory, leveraging warp-level broadcast to reduce bandwidth consumption and improve throughput.

To satisfy the GPU execution model, all state updates are Intermediate variables were accordingly re- thread-private.

Fig. 2. Relative computational time distribution among the major FLEXPART modules for a simulation with 100,000 particles. structured: performance-critical quantities were assigned as thread-local variables to exploit registers and caches, while less frequently accessed variables were organized in global-memory arrays indexed by particle ID. This design avoids race conditions, lim- its register pressure, and achieves an ef- fective balance between correctness, performance, and re- source utilization.

2. Parallelization of Particle Grid Reordering and the Two-Level

Loop Structure In Lagrangian particle dispersion models, the update of particle states typically depends on the background condi- tions of the grid cell in which each particle resides. For cer- tain physical processes, such as convective mixing, the com- putation of within-grid particle redistribution relationships is relatively complex, while particles located in the same grid cell can share identical transformation relationships. As il- lustrated in Fig. 3 Figure 3: see original paper, traditional CPU implementations com- monly employ a grid-binning strategy, in which particles are reordered according to their associated grid cells and their states are updated using a two-level nested loop structure that first iterates over grid cells and then over particles within each grid cell. When ported to a GPU parallel architecture, how- ever, this grid-dominated, nested-loop computation pattern cannot be directly mapped onto a kernel execution model in which particles serve as the fundamental parallel units. If the reuse of within-grid transformation relationships is preserved, grid cells containing only a small number of particles can lead to low thread utilization and insufficient parallelism, resulting in typical GPU load-imbalance issues. Conversely, recomputing grid-level transformation relationships independently for each particle would substantially increase the computational burden per thread, thereby diminishing the effectiveness of parallel acceleration. Based on these considerations, the com- putational procedures involving particle reordering and two- level loop structures were structurally redesigned to accom- modate a particle-centric GPU execution model. As shown in Fig. 3(b), because the computation of grid identifiers for individual surviving particles is free of inter-particle data de- pendencies,

this step was implemented as a GPU-side paral- lel kernel. Moreover, leveraging the stable parallel reorder- ing operators provided by CUDA Fortran, particle reordering can be efficiently performed entirely on the GPU device. In contrast, the computation of redistribution matrices for active grid cells involves complex logic but is required for only a relatively small number of grid cells. Therefore, the original grid-first, particle-second nested-loop structure was function- ally decoupled: the outer, grid-based computation of redis- tribution matrices was retained on the CPU and the result- ing matrices were written to GPU global memory. Subse- quently, in the particle-level parallel kernels, each thread directly indexes the corresponding redistribution matrix based on the grid identifier of the particle, enabling fully parallel updates of all particles. This strategy effectively avoids the insufficient parallelism and GPU load imbalance caused by nonuniform particle counts per grid cell in the original grid- based traversal scheme. It should be noted that this workflow introduces an additional device-to-host data transfer during the redistribution-matrix generation stage. However, because the data transferred in each instance consist only of a one- dimensional array with a size proportional to the number of surviving particles, the associated data volume is relatively small, and its impact on overall computational performance is negligible.

C. Fast Arithmetic Instruction Optimization In GPU parallel computing, although the overall floating- point throughput is high, complex arithmetic operations such as division and square root still exhibit significantly higher instruction latency and lower execution throughput in single- precision arithmetic compared with simple operations such as addition and multiplication. When these operators are in- voked frequently during particle updates, they can substan- tially limit kernel execution efficiency and thus become major performance bottlenecks that constrain the overall accelera- tion achieved.

1. Fast Division Operations

On CUDA architectures, the standard division operator "/" follows IEEE 754 floating-point semantics and is typically translated by the compiler into a single-precision floating- point division instruction (e.g., div.rn.f 32 with round-to-nearest mode). At the hardware level, this instruction is usu- ally implemented through a sequence of high-latency micro- operations, resulting in substantially higher execution latency and lower throughput than basic arithmetic instructions such as addition and multiplication. In Lagrangian particle disper- sion simulations, division operations are frequently invoked in key numerical procedures, including turbulence-intensity calculations, gradient normalization, and scaling of stochas- tic increments, and therefore constitute a primary arithmetic bottleneck limiting kernel performance. To reduce this com- putational overhead, standard division operations were refor- mulated using a reciprocal–multiplication scheme: a ⋅approx(Here, approx(⋅) denotes an approximate reciprocal op- erator. The reciprocal operation can be implemented us- ing the reciprocal approximation mechanism provided by the GPU multi-function units

(MFUs), which is typically based on hardware lookup tables combined with interpola- tion. Compared with the iterative logic employed by stan- dard division instructions, this mechanism exhibits substan- tially lower instruction latency and higher throughput. Un- der this approximation, the execution efficiency of the cor- responding arithmetic operations can be several times higher than that of standard division. In terms of numerical accu- racy, for single-precision floating-point numbers satisfying Fig. 3. Computational workflow of the convective mixing process: (a) CPU-based implementation and (b) optimized GPU-adapted imple- mentation. The red boxes indicate procedures executed on the GPU. |b| (cid:2)2−126, 2126(cid:3), the maximum error of the reciprocal ap- proximation does not exceed 2 units in the last place (ULP).

This accuracy level is sufficient to maintain numerical stabil- ity and precision requirements while significantly improving overall computational performance. instruction latency is substantially lower than that of the standard square-root operation, while offering higher execu- tion throughput. Within the single-precision floating-point range, the maximum error of this approximation is likewise bounded by 2 units in the last place (ULP).

2. Fast Square-Root Operations

In calculations related to turbulence parameterization, gravitational settling corrections, and normalization of stochastic increments, a large number of single-precision floating-point square-root operations are required. Similar to floating-point division, standard square-root operations that conform to IEEE 754 semantics impose strict rounding rules and boundary-condition handling (e.g., N aN/Inf and sub- normal values). Compilers typically map these operations to single-precision square-root instructions (e.g., sqrt.rn.f 32), which exhibit relatively high execution latency and low throughput at the hardware level and can therefore constitute a significant arithmetic bottleneck for kernel performance. To mitigate this overhead, standard square-root operations were reformulated as a combination of a single multiplication and a reciprocal square-root operation: a a ·approx(reciprocal Here, approximation square-root approx(1√ a) is likewise executed by the GPU multi- function units (MFUs) and relies on hardware lookup tables and interpolation to obtain a fast approximate value.

3. Newton–Raphson Iterative Refinement

The introduction of approximate operators results in a max- imum numerical error of up to 2 ULP. For most atmospheric dispersion simulations, this error magnitude is substantially smaller than the uncertainties associated with stochastic tur- bulent perturbations and meteorological-field interpolation.

However, along certain computational pathways—such as the calculation of vertical diffusion coefficients, normalization of turbulence time scales, and gravitational settling correc- tions—these errors may still accumulate over long-term integrations. To preserve numerical consistency between the GPU-based computa-

tions and the original implementation to the greatest extent possible, Newton–Raphson iterative re- finement was applied to correct the approximate results.

The Newton–Raphson method is used to solve equations of the form f (x) = 0 through the following iterative formula- tion: xk+1 = xk − f (xk) f ′(xk) For division operations, the numerical uncertainty origi- b . Therefore, nates from the fast reciprocal approximation of 1 Newton–Raphson refinement is applied only to this compo- nent. Let f (x) = x−1 − b , after one Newton–Raphson itera- tion, the refined expression can be written as:

D. GPU Parallel Execution Strategy and Resource Utilization Optimization xk+1 = xk − k − b = xk ·(2 − bxk) For square-root computations, Newton–Raphson refine- ment is likewise applied only to the reciprocal square-root term of the form 1√ a . Accordingly, by defining f (x) = x−2 − a, the expression after a single Newton–Raphson it- eration can be written as: xk+1 = xk − k − a −2x−3 = xk(1.5 − 0.5 ax2 As noted above, the maximum initial error introduced by the reciprocal and reciprocal square-root approximation oper- ators does not exceed 2 ULP, corresponding to approximately 2.4 × 10−7. Owing to the quadratic convergence property of the Newton–Raphson method, once the error is sufficiently small, each iteration reduces the relative error to approxi- mately the square of its previous magnitude. Accordingly, for an initial relative error on the order of 10−7, a single Newton–Raphson refinement step can suppress the error to the order of 10−14, which is well below the effective repre- sentational precision of single-precision floating-point arith- metic. Given that rounding errors in single-precision floating- point operations are typically bounded by 0.5 ULP (corre- sponding to a relative error of approximately 10−8), the in- clusion of a single Newton–Raphson iteration is sufficient to ensure that the final result attains the maximum effective pre- It should be empha- cision achievable in single precision. sized that both types of Newton–Raphson refinements involve only a small number of multiplication and addition opera- tions. Compared with the full sqrt.rn.f 32 or div.rn.f 32 instructions, their computational cost is substantially lower, and they can be efficiently executed by the GPU fused multiply–add (FMA) units without imposing a noticeable burden on overall kernel performance.

In addition, to further mitigate the potential accumulation of numerical errors introduced by fast division approxima- tions, algebraic reformulation strategies were applied in se- lected computations to reduce the number of division operations. For example, expressions of the form d = a/b/c were consistently rewritten as d = a/(bc), and expressions of the form d = a/(b/c) were reformulated as d = (ac)/b, thereby effectively decreasing the frequency of division operations.

Similarly, power expressions such as a1.5 were explicitly de- composed and expressed as a ·a, enabling efficient evalu- ation in combination with the fast square-root approximation operator.

After the parallel restructuring of the particle–grid compu- tation workflow and the optimization of arithmetic operators, the overall performance of the GPU

kernels is further con- strained by the design of the parallel execution model and the efficiency of hardware resource utilization. To fully ex- ploit the parallel computing potential of GPUs for large-scale particle simulations, this study systematically optimized ker- nel execution from two complementary perspectives: parallel granularity design and hardware resource allocation.

1. Parallel Granularity and Thread Organization Strategy

After implementing fine-grained particle-level parallelism, further improvement in GPU resource utilization primarily depends on reducing idle periods between successive kernel launches and data-preparation stages, as well as increasing the degree of overlap between computation and associated data-access operations. To this end, a coarse-grained paral- lelization strategy based on CUDA streams was introduced.

Through stream-based task scheduling, multiple computa- tional tasks are executed in a temporally interleaved manner, thereby constructing a higher-throughput parallel execution pipeline.

CUDA streams are execution channels that support asyn- chronous scheduling. Operations submitted within the same stream are executed sequentially in the order of submission, whereas operations issued to different streams may execute concurrently, subject to hardware resource availability. As illustrated in Fig. 4 Figure 4: see original paper, when parallel granularity is not opti- mized, particle simulations typically follow a sequential exe- cution pattern in which kernel execution is initiated only after all relevant data have been read and prepared. As a result, computational resources remain idle during data-preparation stages, increasing overall execution latency. In contrast, as shown in Fig. 4(b), by appropriately partitioning the parti- cle ensemble and assigning multiple CUDA streams, data- reading and kernel-execution operations for different parti- cle subsets can be temporally interleaved. This approach ef- fectively overlaps computation with data preparation, signif- icantly reducing idle waiting periods and thereby improving overall computational throughput.

2. Register Usage Control Optimization

During GPU kernel execution, registers constitute the lowest-latency and highest-throughput storage resource, and their allocation strategy directly affects instruction-level par- allelism within individual threads as well as the number of threads that can concurrently reside on a streaming multipro- cessor (SM). Excessive register allocation, while beneficial for reducing memory-access overhead associated with inter- mediate variables, may exhaust available register resources and trigger register spilling, forcing some variables to be F. Multi-GPU Scalability and Load Balancing To overcome the limitations on particle scale and computa- tional resources under a single-GPU configuration, a multi- GPU parallel computing architecture based on particle-set partitioning was designed and implemented. Distinct particle subsets are assigned to differ-

ent GPU devices, on which par- ticle advection and associated physical process calculations are performed independently. During computation, frequent exchange of particle state information between GPUs is not required; instead, simulation results from all devices are ag- gregated only at the output stage, thereby enabling efficient and scalable parallel computation under multi-GPU configu- rations.

In multi-GPU systems, however, individual devices may differ in computational capability. To achieve optimal overall performance, it is necessary to ensure load balancing across GPUs as much as possible. Given that, in mainstream GPU architectures, the configuration of SMs is typically uniform within a single device, the number of SMs was adopted as the primary metric for assessing relative computational capacity across devices. On this basis, a particle subset partitioning strategy proportional to SM count was designed as follows:

Ni = N ⋅ Here, N denotes the total number of particles, SM i and Ni represent the number of streaming multiprocessors available on the i-th GPU and the corresponding number of particles as- signed to that device, respectively, and S = (cid:80) SMi denotes the total number of SMs across all GPUs. After task parti- tioning, particle subsets are scheduled to the corresponding GPUs under host-side OpenMP management. Upon comple- tion of all computations, the simulation results are collected and integrated by the master thread.

G. Validation

1. Validation Experiments and Reference Benchmarks

To systematically evaluate the simulation accuracy and ac- celeration performance of the GPU-accelerated Lagrangian particle dispersion model developed in this study, ETEX-I was selected as a unified test scenario for validation. During the experiment, a total of 340kg of the tracer PMCH was released into the atmosphere at Monterfil, Brittany, France (48.058 ∘ N, 2.0083 ∘ W). A network of 168 ground-based monitoring stations was deployed across Europe, and continuous observations were conducted starting at 15:00 (UTC) on 23 October. The experiment yielded a total of 3,104 valid concentration measurements. The location of the re- lease source and the spatial distribution of the ground monitoring stations are shown in Fig. 5 [Figure 5: see original paper].

The serial version of FLEXPART v11.04 was adopted as the reference benchmark for both simulation accuracy and acceleration performance. The meteorological driving fields Fig. 4. Schematic illustration of the coarse-grained parallel execu- tion mechanism: (a) single CUDA stream execution and (b) multi-CUDA stream execution. stored in local memory and thereby significantly increasing access latency. Conversely, overly restrictive limits on reg- ister usage can constrain the compiler' s optimization space and degrade single-thread execution efficiency. In this work, the maximum number of registers available per thread was explicitly constrained to avoid register spilling while substan- tially increasing thread occupancy.

E. Decoupling of Background-Field Preprocessing Lagrangian particle dispersion calculations rely on mete- orological fields stored in encoded formats such as GRIB.

These meteorological datasets are typically decoded and pre- processed sequentially in time, including the computation of intermediate physical variables and coordinate-system trans- formations. However, in high-demand emergency application scenarios—such as source-term inversion and data assimila- tion—the background meteorological fields often remain un- changed across multiple simulation runs. Repeated decod- ing and preprocessing of meteorological data in such cases inevitably introduce substantial computational redundancy and significantly increase I/O overhead, thereby constraining overall computational efficiency. To address this issue, the meteorological-field preprocessing module was structurally decoupled from the main FLEXPART computational workflow, and an independent meteorological preprocessing pro- gram, gribtobinmpi, was designed and implemented. This program executes in parallel on multicore CPU platforms us- ing the Message Passing Interface (MPI), enabling efficient decoding of GRIB and related encoded meteorological files, derivation of physical variables, and coordinate transforma- tions. The processed meteorological fields are then stored in binary format. Correspondingly, the FLEXPART main pro- gram was adapted, and the meteorological data input inter- face was reimplemented to ensure efficient and stable access to the preprocessed meteorological fields during subsequent GPU-accelerated computations. The design, computational details, and stored parameters of the preprocessing program are provided in Appendix A.

3. Computational Performance Evaluation

In the computational performance evaluation, it is noted that the PMCH tracer used in the ETEX-I experiment does not involve wet deposition processes and therefore cannot fully reflect the performance gains of GPU acceleration in complex physical modules, particularly wet deposition calcula- tions. Accordingly, in all performance evaluation cases, the released substance was replaced with Cs − 137 to activate wet deposition and related processes, ensuring that the acceleration effects of all key computational components could be comprehensively assessed. The performance evaluation com- prised three aspects:

1. Single-GPU acceleration assessment. This evaluation

was conducted on a computing platform equipped with an AMD Ryzen Threadripper 7970X CPU (32 cores) and an NVIDIA GeForce RTX 5080 GPU (hereafter referred to as Platform A). The runtime performance of the developed GPU-accelerated program was com- pared with that of the reference benchmark. The analy- sis focused on quantifying performance improvements achieved through the successive introduction of the fine-grained parallel architecture, fast arithmetic in- struction optimization, and parallel execution and

re- source utilization strategies, thereby enabling a quan- titative assessment of the relative contribution of each optimization to the overall speedup.

2. Multi-GPU scalability evaluation. By progressively

increasing the number of GPUs, the variation in to- tal execution time and speedup with respect to GPU count was systematically analyzed to assess the paral- lel scalability of the developed program in multi-device environments. Because such scalability tests require substantial GPU resources to adequately characterize performance trends, this evaluation was performed on a server platform equipped with an Intel(R) Xeon(R) Platinum 8160 CPU @ 2.10 GHz and eight NVIDIA Tesla V100 GPUs (hereafter referred to as Platform B).

3. Heterogeneous GPU load-balancing evaluation. Based

on Platform A, an additional NVIDIA GeForce RTX

5070 GPU was introduced to construct a heterogeneous

multi-GPU environment (hereafter referred to as Plat- form C) for evaluating the proposed load-balancing strategy. By comparing the distribution of computation time across GPUs with and without the load-balancing strategy enabled, the effectiveness of the strategy in im- proving task allocation balance and overall parallel effi- ciency under heterogeneous computing conditions was analyzed.

Fig. 5. Location of the ETEX-I release (red star) and spatial distri- bution of monitoring stations (yellow circles).

Table 2 . Statistical metrics used for accuracy validation. and de- note the model-predicted and observed concentrations at sample i, respectively; $^{-}$P and $^{-}$M are the corresponding mean values; N is the total number of samples.

Metric Mathematical definition F B = 2($^{-}$P − $^{-}$M) $^{-}$P + $^{-}$M (cid:115) RMSE RM SE = (Pi − Mi)2 (cid:80) F A2 = 1 F A5 = 1 (cid:80) (cid:80) I(0.5 ≤ Pi I(0.2 ≤ Pi used for validation were obtained from the Climate Forecast System Reanalysis (CFSR) dataset provided by the National Centers for Environmental Prediction (NCEP). In the simu- lations, the total number of particles was set to 1.0 × 106, and the horizontal grid resolution of the output concentration fields was configured as 0.1 ◦ × 0.1 ◦ .

2. Accuracy Validation

Under identical simulation parameter settings, the GPU- accelerated dispersion model developed in this study and the reference benchmark model were executed independently.

The simulation results from the two models were then com- pared, and the statistical differences between the simulated values and the corresponding ob-

servational data were further analyzed to evaluate the numerical consistency and simula- tion accuracy of the GPU-based implementation. The statis- tical evaluation metrics adopted in this analysis are listed in Table 2.

III. RESULTS AND DISCUSSION A. Accuracy Validation Fig. 6 Figure 6: see original paper and Fig. 6(b) present the mean concentration fields at 48 h after release simulated by the reference CPU code and the developed GPU implementation, respectively.

The two fields exhibit highly consistent large-scale spatial patterns, indicating that the GPU implementation reproduces the overall advection and dispersion structure of the radioac- tive plume with comparable fidelity. The relative difference field shown in Fig. 6(c) indicates that, within the high- concentration core of the plume, concentrations simulated by the GPU code are marginally lower than those of the ref- erence code, whereas slightly higher values occur along the low-concentration plume margins. This pattern suggests that, relative to the reference implementation, the GPU code may produce moderately enhanced turbulent mixing effects.

Despite these differences, the absolute deviation between the two simulations (Fig. 6(d)) remains consistently low throughout the simulation period, with magnitudes approxi- mately two orders of magnitude lower than the local mean concentrations at each corresponding time step. This indi- cates that the discrepancies have a limited influence on the overall concentration field.

To further assess their practical relevance, Fig. 6(e) shows the spatial distribution of differences between observations and the reference simulation at the same time. In the main plume region, observed concentrations are generally lower than simulated values, while higher relative values appear near the plume edges. This error pattern is consistent with the direction of the differences observed between the GPU and reference simulations, suggesting that the GPU implemen- tation maintains comparable accuracy in high-concentration regions. The observed behavior may be related to the accu- mulation characteristics of numerical corrections introduced by the Newton-iteration scheme used in the GPU code. Al- though the theoretical magnitude of such numerical errors is on the order of $10-14$ and therefore not directly resolv- able under single-precision floating-point accuracy (approx- imately $10-8$), minor local perturbations may still influence concentration distributions after time integration over multi- ple steps in large-scale parallel computations. Table 3 sum- marizes the statistical performance metrics of the reference CPU code and the GPU implementation based on the com- plete ETEX-I observational dataset. The results indicate that the two implementations achieve very similar overall simula- tion accuracy. However, for the FB metric, which is more sen- sitive to systematic bias, and the NMSE metric, which places greater emphasis on extreme errors, the GPU implementation yields slightly lower error values. This behavior suggests a more accurate reconstruction of the overall concentration dis- tribution, as well as improved representation of peak concen- trations, thereby reducing large deviations in localized high- concentration regions. These findings are consistent

with the error distribution pattern observed in Fig. 6(e). Overall, the GPU implementation exhibits accuracy comparable to that of the CPU reference model and shows modest improvements Table 3. Statistical performance metrics of the reference CPU model and the GPU implementation relative to observed measurements.

Metric CPU reference implementation GPU implementation RMSE 0.58 for selected statistical indicators and peak-concentration rep- resentation, supporting the reliability of the proposed GPU- based approach while maintaining numerical consistency.

B. Computational Performance Evaluation

1. Single-GPU Acceleration Performance

Fig. 7 Figure 7: see original paper compares the runtime performance of individual computational components between the GPU implementation with fine-grained parallelization and the reference CPU code on validation platform A. After adopting the fine-grained "one-particle–one-thread" mapping strategy, the overall com- putational performance is substantially improved. Among the major physical processes, the advection–diffusion module ex- hibits the highest acceleration, with a speedup of 38.50, while the convective mixing and wet deposition modules achieve speedups of 10.79 and 12.53, respectively. These differ- ences are primarily associated with the computational char- acteristics of each process and their inherent parallel scalabil- ity. During the advection–diffusion stage, particle updates are mutually independent and involve minimal data dependency, allowing efficient utilization of large-scale GPU parallelism and resulting in high parallel efficiency. In contrast, the ac- celeration of the convective mixing process is more limited, as this stage involves particle–grid rearrangement and nested loop structures; after GPU porting, the construction and up- date of the corresponding transition matrices remain on the CPU, which constrains the achievable speedup. For the wet deposition process, precipitation events during the ETEX-I experiment occur infrequently, leading to relatively low ac- tivation frequency and computational density. Under these conditions, the proportion of memory-access operations is comparatively high, while the effective arithmetic workload is limited, resulting in lower acceleration than that observed for the advection–diffusion process. Nevertheless, at the overall model level, the fine-grained parallelization strategy provides a clear performance benefit. Even when time-consuming se- quential procedures such as meteorological data decoding and preprocessing are included, the total speedup reaches 9.75.

Given that computationally expensive arithmetic opera- tions, such as division and square-root calculations, are primarily concentrated in the advection–diffusion stage, Fig. 7(b) further presents the performance obtained after in- troducing fast arithmetic instruction optimizations for this module. The results show that the execution time of the ad- vection–diffusion process is reduced from 152.3 s to 110.0 s, corresponding to an overall reduction of approximately Fig. 6.

Comparative validation of numerical simulations, illustrating the performance of the GPU and CPU models and their discrepancies relative to observations. (a) Spatial distribution of the plume at 48 h simulated by the reference CPU model; (b) spatial distribution of the plume at 48 h simulated by the GPU implementation; (c) concentration difference between the GPU and CPU simulations at 48 h (|GPU − CPU|); (d) temporal evolution of the domain-averaged concentration simulated by the CPU and GPU models, together with the corresponding time series of their mean absolute difference (|GPU − CPU|); (e) relative differences between observed concentrations at monitoring stations and the simulated concentrations. 38.4%. This outcome indicates that, without modifying the underlying algorithmic structure, the use of fast arithmetic instructions can effectively reduce arithmetic overhead and further increase computational throughput for this stage.

After introducing the coarse-grained parallelization strat- egy, differences in data-transfer patterns and computational workflows among the physical processes lead to distinct task- partitioning schemes. Fig. 8 [Figure 8: see original paper] summarizes the acceleration performance obtained for different physical processes un- der varying numbers of CUDA streams. For the advec- tion–diffusion and wet deposition processes, the computa- tional time generally decreases as the number of streams in- creases, while the performance gain approaches saturation when the stream count reaches eight. In principle, increasing the number of streams enhances the overlap between com- putation and data transfer, thereby improving execution ef- ficiency.

In practice, however, excessive task subdivision shortens individual kernel execution times, increasing the rel- ative contribution of kernel launch and scheduling overhead to the total runtime. At the same time, splitting contiguous data into an excessive number of sub-blocks reduces the ef- ficiency of each data transfer. As a result, once the number of streams exceeds a certain threshold, further refinement of task granularity yields diminishing returns and may even lead to increased overall runtime due to additional scheduling and transfer overheads.

In contrast, when coarse-grained parallelization is applied to the convective mixing process, the computational time in- creases. This behavior is primarily associated with the parti- cle–grid rearrangement mechanism used in this stage. Because particle grid locations cannot be determined a priori during task partitioning, parameters related to active grid cells cannot be effectively divided across streams, which limits the achievable overlap between data transfer and kernel execu- tion. Consequently, the potential benefits of coarse-grained parallelism are not fully realized. In addition, the additional scheduling and management overhead introduced by coarse- grained partitioning becomes more pronounced for this pro- cess, offsetting the gains expected from parallel execution and ultimately resulting in degraded performance.

Based on these observations, process-specific coarse- grained configurations are adopted in this study. eight CUDA streams are selected for the advection–diffusion and wet de- position processes, while coarse-grained parallelization is

not applied to the convective mixing process.

Fig. 9 [Figure 9: see original paper] illustrates the impact of limiting the maximum num- ber of registers per thread on the performance of individual computational modules. As the register limit is progressively reduced, the execution time of the advection–diffusion pro- cess decreases markedly from 256 to 128 registers, then de- clines more gradually from 128 to 72, and finally exhibits a tendency toward performance degradation when the limit is further reduced to 64. This behavior is primarily attributable to the influence of register usage on the number of warps that can reside concurrently on a SM. With a high register count, the number of resident warps per SM is constrained; when active warps stall due to memory accesses, the lack of alter- native warps for scheduling leads to SM idle cycles and re- duced computational efficiency. As the register limit is low- ered, the number of resident warps increases substantially, Fig. 7. Effects of fine-grained parallelization (a) and fast arithmetic instruction optimization (b) on computational time. striction of register usage has only a minor impact on per- formance. For the wet deposition process, the initial register usage is relatively high (approximately 250), comparable to that of the advection–diffusion process. However, its memory throughput is only about 5–7%, which is substantially lower than that of the advection–diffusion and convective mixing processes, indicating that it is not predominantly limited by memory bandwidth or latency.

In this case, increasing oc- cupancy does not readily translate into tangible performance gains, while the risks associated with register spilling and ad- ditional instruction overhead introduced by register compres- sion tend to dominate, leading to increased execution time.

Because the maximum register constraint applies globally to all kernels, an overall balance across different computational modules is required. Based on the combined performance characteristics, the maximum number of registers per thread is set to 72 in this study.

After completing the GPU implementation and the asso- ciated optimizations, the runtime statistics of the individual computational modules are summarized in Table 4 . The re- sults indicate that, at this stage, the reading and preprocessing of the meteorological background fields account for 78.56% of the total execution time, thereby constituting the dominant bottleneck limiting further improvements in overall computa- tional efficiency.

Table 5 reports the runtime of the independently developed MPI-based meteorological data decoupling and preprocess- ing program, gribtobinmpi, under different numbers of pro- cesses, and compares it with the corresponding procedure in the reference CPU implementation. The results indicate that, under a typical modern CPU configuration with 18 threads, the meteorological field preprocessing achieves a speedup of approximately 11.63. This substantially reduces the propor- Fig. 8. Variation of computational time for individual model com- ponents as a function of the number of CUDA streams. enabling more

effective warp switching and improved hid- ing of memory latency, which results in a rapid performance improvement. Once the resident warp count becomes suffi- cient to mask most memory stalls, further increases yield di- minishing returns. Moreover, excessive register compression may induce register spilling or increase instruction counts, ul- timately leading to performance degradation. For the con- vective mixing process, the core computational kernel uses approximately 116 registers per thread in the absence of any imposed limit, as summarized in Appendix Table B.1, cor- responding to an occupancy of about 33.3%. Under these conditions, the available number of resident warps is already sufficient to effectively hide memory latency, and further re- Table 6 . Speedup ratio reported for recent GPU-accelerated imple- mentations of Lagrangian particle dispersion models.

Original Model Speedup Ratio Author(s) HYSPLIT Yu et al.[**?**] Zeng et al.[**?**] FLEXPART Harvey et al.[**?**] FLEXPART FLEXPART implementation further increases to approximately 52.05. As summarized in Table 6, this performance substantially ex- ceeds that reported for existing GPU-accelerated Lagrangian particle dispersion approaches.

2. Multi-GPU Scalability Assessment

Multi-GPU scalability tests of the developed GPU program were conducted on validation platform B, and the results are shown in Fig. 10 [Figure 10: see original paper]. As the number of GPUs increases from 1 to 4, the advection–diffusion process exhibits near-linear speedup, indicating favorable parallel scalability for this com- ponent. When the GPU count is increased beyond this range, the incremental performance gains gradually diminish. This behavior arises because, under a fixed total problem size, the computational workload assigned to each GPU decreases as more devices are introduced, while the relative contribution of communication and synchronization overhead to the total runtime correspondingly increases, thereby constraining fur- ther speedup. In contrast, the convective mixing and wet de- position processes do not show substantial acceleration with increasing GPU count. On the one hand, the proportion of kernel execution time associated with these processes is rela- tively small within the overall computation, limiting the im- pact of additional GPUs on their runtime. On the other hand, as discussed in Section III B 1, the convective mixing process involves relatively frequent data transfers; under fixed CPU- side communication bandwidth, introducing additional GPUs further amplifies communication and synchronization over- heads and may even lead to increased execution time. Despite these limitations in the scalability of certain components, the overall GPU program maintains satisfactory parallel scalability within a moderate GPU count, supporting its feasibility and effectiveness in multi-GPU environments.

3. Load Balancing across Heterogeneous GPUs

The acceleration effects of the load-balancing strategy in a heterogeneous GPU environment were evaluated on valida- tion platform C, with the results shown in Fig. 11 [Figure 11: see original paper]. Using the single NVIDIA GeForce RTX 5080 configuration as the refer- ence, the dual-GPU setup (RTX 5080 + RTX 5070) achieves a speedup of only about 12.52% for the advection–diffusion process when no load-balancing strategy is applied. After in- troducing the proposed load-balancing scheme, the speedup increases to approximately 31.94%, representing an abso- lute improvement of about 19.42 percentage points rela- tive Fig. 9. Variation of computational time as a function of the maxi- mum number of registers available per thread.

Table 4. Percentage contribution of individual computational com- ponents to the total runtime of the GPU program prior to decoupling the meteorological field preprocessing.

Computational Module Execution Time (s) Time Fraction (%) Advection-Diffusion Get fields Convective mixing Wet deposition Statistics & Output Others tion of time spent in sequential meteorological data process- ing and mitigates its impact on overall computational perfor- mance.

Based on the above developments, the total simulation time is reduced from 6598.00s in the original implementa- tion to 163.12s, corresponding to an overall speedup of ap- proximately 40.45. In practical application scenarios involving high computational demand, such as source-term inver- sion, multiple simulation tasks typically share the same me- teorological background fields. Once the preprocessing step is completed, the meteorological data can be repeatedly ac- cessed in binary form, with an associated I/O time of only about 2.85s. Under these conditions, the overall speedup of the developed GPU program relative to the reference CPU Table 5. teorological gribtobinmpiasaf unctionof thenumberof parallelthreads. computational time of preprocessing Variation of field decoding the me- program Number of Processes Execution Time (s) Speedup CPU baseline Fig. 11. time.

Impact of the load-balancing strategy on computational tension to multi-GPU execution, together with a heteroge- neous load-balancing strategy tailored to mixed-performance devices, demonstrates robust scalability and efficient resource utilization across diverse hardware configurations.

Validation against the ETEX-I tracer experiment confirms that the GPU-based model remains consistent with estab- lished reference simulations, with minor numerical deviations being confined and physically reasonable. Compared to the benchmark verification program, it achieves a speedup ra- tio of approximately 40.45–52.05, significantly outperform- ing previously developed GPU-accelerated Lagrangian parti- cle programs. Furthermore, the model demonstrates favorable multi-GPU scalability and can optimize execution efficiency across heterogeneous GPU devices through load-balancing strategies. Overall,

the developed framework provides a prac- tical and extensible solution for computationally intensive at- mospheric dispersion applications—particularly those requir- ing rapid turnaround and repeated simulations, such as emer- gency response assessment and source-term analysis—within the domain of large-scale transport modeling.

V. DATA AND SOFTWARE AVAILABILITY To enhance the reproducibility of this study and to facil- itate direct replication of the reported validation results by subsequent researchers and end users, the developed GPU- accelerated Lagrangian particle dispersion program, together with all validation test cases, has been packaged as a Docker image. This image integrates the GPU executable, the com- plete dependency and runtime environment, and all validation scenarios and parameter files presented in this work, enabling users to reproduce the experiments through minimal configu- ration on CUDA-enabled computing platforms.

Docker image: https://doi.org/10.5281/zenodo.18164030 Fig. 10. Variation of GPU program execution time as a function of the number of GPU devices. to the unbalanced configuration. This result indicates that the load-balancing strategy effectively mitigates the adverse impact of performance heterogeneity among GPUs on par- allel efficiency, thereby enhancing overall acceleration. The load-balancing strategy also exerts an indirect influence on the execution efficiency of the convective mixing process. By allocating computational tasks according to the relative capa- bilities of the GPUs and avoiding excessive workloads on the less capable device, delays associated with data transfer and synchronization are reduced. This scheduling approach par- tially suppresses the increase in convective mixing runtime observed under heterogeneous multi-GPU configurations. In contrast, the impact of load balancing on the wet deposition process is limited. Because the kernel workload of this pro- cess is relatively small, both GPUs complete the computa- tions within a short time, making its performance less sensi- tive to the task distribution. As a result, the overall execu- tion time of the wet deposition process remains largely un- changed.

IV. CONCLUSIONS This study establishes a GPU-oriented Lagrangian particle dispersion models within the FLEXPART framework, achieving a comprehensive migration of core transport and deposition processes to modern accelerator architectures.

Through coordinated fine-grained parallelization and archi- tectural adaptations, the framework enables efficient large- scale dispersion calculations while preserving numerical fidelity across key physical processes.

Complementary arithmetic optimizations and concurrency-oriented execution strategies further unlock GPU computational capacity, result- ing in a balanced and scalable implementation suited to high- throughput operational use.

Beyond kernel-level acceleration, the development of a parallelized meteorological data preparation program allevi- ates a major upstream bottleneck, enabling end-to-end per- formance gains in realistic application workflows. The

ex- VI. ACKNOWLEDGEMENTS This work is supported by the Innovative Team Project of the China Institute for Radiation Protection. The authors gratefully acknowledge the funding support, which made this work possible. [1] X. Zhang, J. Wang, Atmospheric dispersion of chemical, bio- logical, and radiological hazardous pollutants: Informing risk assessment for public safety. J. Saf. Sci. Resil. 3, 372-397 (2022). doi:10.1016/j.jnlssr.2022.09.001 [2] R. Yao, Atmospheric dispersion of radioactive material in radi- ological risk assessment and emergency response. Prog. Nucl.

Sci. Technol. 1, 7-13 (2011). doi:10.15669/pnst.1.7 [3] G. Sugiyama, J. Nasstrom, B. Pobanz et al., Atmo- spheric Dispersion Modeling: Challenges of the Fukushima Daiichi Response. Health Phys. 102, 493-508 (2012). doi:10.1097/HP.0b013e31824c7bc9 [4] M.A. Hernández-Ceballos, M. Sangiorgi, B. García-Puerta et al., Dispersion and ground deposition of radioactive material according to airflow patterns for enhancing the preparedness to N/R emergencies. J. Environ. Radioact. 216, 106178 (2020). doi:10.1016/j.jenvrad.2020.106178 skills [5] M. Ulimoen, E. Berge, H. Klein et al., Comparing model for deterministic versus ensemble disper- sion modelling: The Fukushima Daiichi NPP accident as a case study. Sci. Total Environ. 806, 150128 (2022). doi:10.1016/j.scitotenv.2021.150128 atmospheric [6] Y. Xu, X. Li, H. Luo et al., Source reconstruction vances in decoding information from atmospheric trans- port physics. 139534 (2025). doi:10.1016/j.jhazmat.2025.139534 J. Hazard. Mater. 497, radionuclide leakage:

Recent [7] P.T. Rakesh, R. Venkatesan, C.V. Srinivas, Formulation of TKE based empirical diffusivity relations from turbulence measurements and incorporation in a Lagrangian particle dis- persion model. Environ. Fluid Mech. 13, 353-369 (2013). doi:10.1007/s10652-013-9273-8 [8] X. Zhang, G. Efthimiou, Y. Wang et al., Comparisons between a new point kernel-based scheme and the infinite plane source assumption method for radiation cal- culation of deposited airborne radionuclides from nuclear power plants. J. Environ. Radioact. 184-185, 32-45 (2018). doi:10.1016/j.jenvrad.2018.01.002 [9] J. Pudykiewicz, Simulation of the Chernobyl dispersion with a 3-D hemispheric tracer model. Tellus B. 41, 391-412 (1989). [10] Á. Leel″ossy, R. Mészáros, I. Lagzi, Short and long term dis- persion patterns of radionuclides in the atmosphere around the Fukushima Nuclear Power Plant. J. Environ. Radioact. 102, 1117-1121 (2011). doi:10.1016/j.jenvrad.2011.07.010 [11] T. Christoudias, Y. Proestos, J. Lelieveld, Atmospheric Dis- persion of Radioactivity from Nuclear Power Plant Accidents:

Global Assessment and Case Study for the Eastern Mediter- ranean and Middle East. Energies. 7, 8338-8354 (2014). doi:10.3390/en7128338 [12] X. Hu, D. Li, H. Huang et al., Modeling and sensitivity anal- ysis of transport and deposition of radionuclides from the Fukushima Dai-ichi accident. Atmos. Chem. Phys. 14, 11065- 11092 (2014). doi:10.5194/acp-14-11065-2014 [13] T. Christoudias, J. Lelieveld, Modelling the global atmospheric transport and deposition of radionuclides from the Fukushima Dai-ichi nuclear accident. Atmos. Chem. Phys. 13, 1425-1438 (2013). doi:10.5194/acp-13-1425-2013 [14] I. Pisso, E. Sollum, H.

Grythe et al., The Lagrangian parti- cle dispersion model FLEXPART version 10.4. Geosci. Model Dev. 12, 4955-4997 (2019). doi:10.5194/gmd-12-4955-2019 [15] L. Bakels, D. Tatsii, A. Tipka et al., FLEXPART version 11: improved accuracy, efficiency, and flexibility. Geosci. Model Dev. 17, 7595-7627 (2024). doi:10.5194/gmd-17-7595-2024 [16] M. Cassiani, A. Stohl, J. Brioude, Lagrangian stochastic mod- elling of dispersion in the convective boundary layer with skewed turbulence conditions and a vertical density gradient:

Formulation and implementation in the FLEXPART model.

BoLMe. 154, 367-390 (2015). doi:10.1007/s10546-014-9976- [17] S. Van Thielen, C. Turcanu, J. Camps et al., Op- timizing the calculation grid for atmospheric dispersion J. Environ. Radioact. 142, 103-112 (2015). modelling. doi:10.1016/j.jenvrad.2014.12.014 [18] J.H. Sørensen, J. Bartnicki, A.M. Blixt Buhr et al., Un- certainties in atmospheric dispersion modelling during nu- clear accidents. J. Environ. Radioact. 222, 106356 (2020). doi:10.1016/j.jenvrad.2020.106356 [19] INSA Group, The Chernobyl Accident: Updating of INSAG-1. (Intenational Atomic Energy Agency, Vienna, 1992). [20] O. Saunier, A. Mathieu, D. Didier et al., An inverse modeling method to assess the source term of the Fukushima Nuclear Power Plant accident using gamma dose rate observations. At- mos. Chem. Phys. 13, 11403-11421 (2013). doi:10.5194/acp- [21] R. Jammal, P. Vincze, M. Heitsch et al., The Fukushima Daiichi Accident. (Intenational Atomic Energy Agency, Vienna, 2015). [22] H. Snoun, G. Bellakhal, H. Kanfoudi et al., One-way coupling of WRF with a Gaussian dispersion model: a focused fine-scale air pollution assessment on southern Mediterranean. Environ.

Sci. Pollut. Res. 26, 22892-22906 (2019). doi:10.1007/s11356- [23] J. He, M. Lyu, Z. Qiu et al., Physics-informed optimization for emergency radiation assessment with temporal correction under meteorological uncertainty. J. Environ. Radioact. 291, 107817 (2026). doi:10.1016/j.jenvrad.2025.107817 [24] X. Zhang, W. Raskob, C. Landman et al., Sequen- tial multi-nuclide emission rate estimation method based on gamma dose rate measurement for nuclear emer- gency management. J. Hazard. Mater. 325, 288-300 (2017). doi:10.1016/j.jhazmat.2016.10.072 [25] X. Dong, S. Fang, S. Zhuang et al., Objective inversion of the continuous atmospheric 137Cs release following the Fukushima accident. J. Hazard. Mater. 447, 130786 (2023). doi:10.1016/j.jhazmat.2023.130786 [26] X. Dong, S. Zhuang, Y. Xu et al., Multi-scenario vali- dation of the robust inversion method with biased plume range and values. J. Environ. Radioact. 272, 107363 (2024). doi:10.1016/j.jenvrad.2023.107363 [27] Y. Xu, S. Fang, X. Dong et al., A spatiotemporally separated framework for reconstructing the sources of atmospheric ra- dionuclide releases. Geosci. Model Dev. 17, 4961-4982 (2024). doi:10.5194/gmd-17-4961-2024 [28] Y. Xu, X. Dong, H. Luo et al., Robust source recon- struction of atmospheric radionuclides from observations of different sparsity with spatial preselection and non- smooth constraints. J. Hazard. Mater. 486, 136919 (2025). doi:10.1016/j.jhazmat.2024.136919 [29] X.L.

Zhang, G.F. Su, H.Y. Yuan et al., Modified ensemble Kalman filter for nuclear accident atmospheric dispersion: Pre- diction improved and source estimated. J. Hazard. Mater. 280, 143-155 (2014). doi:10.1016/j.jhazmat.2014.07.064 [30] X.L. Zhang, Q.B. Li, G.F. Su et al., Ensemble-based simul- taneous emission estimates and improved forecast of radioac- [45] M.C. Santos, A. Pinheiro, R. Schirru et al., GPU-based im- plementation of a real-time model for atmospheric dispersion of radionuclides. Prog. Nuclear Energy. 110, 245-259 (2019). doi:10.1016/j.pnucene.2018.09.015 [46] F. Yu, P. Strazdins, J. Henrichs et al., Shared Memory and GPU Parallelization of an Operational Atmospheric Transport and Dispersion Application. Paper presented at the 2019 IEEE International Parallel and Distributed Processing Symposium Workshops, IEEE, Rio de Janeiro, Brazil, 20-24 May 2019 [47] D.B. Kong, D.T. Dai, D.L. Xu et al., CPU-GPU concurrent computing algorithm of particle transport using discontinu- ous finite element discrete ordinates with unstructured grids. doi:10.12074/202504.00216 [48] A. Stohl, M. Hittenberger, G. Wotawa, Validation of the la- grangian particle dispersion model FLEXPART against large- scale tracer experiment data. Atmos. Environ. 32, 4245-4264 (1998). doi:10.1016/S1352-2310(98)00184-8 [49] H. Muhammad, W. Xuan, M. Wang et al., Review of spatial scale dispersion models (ATDMs) to simulate en- vironmental dispersion and deposition of radionuclides and the overview of GIS coupling with dispersion models. Int.

J. Adv. Nucl. React. Des. Technol. 6, 256-280 (2024). doi:10.1016/j.jandt.2025.03.004 [50] J. Zeng, T. Matsunaga, H. Mukai, Using nvidia gpu for mod- elling the lagrangian particle dispersion in the atmosphere. Pa- per presented at the 5th International Congress on Environmen- tal Modelling and Software, International Environmental Mod- elling and Software Society, Ottawa, Ontario, Canada, 5-8 July [51] H. Van dop, R. Addis, G. Fraser et al., ETEX: A Euro- pean tracer experiment; observations, dispersion modelling and emergency response. Atmos. Environ. 32, 4089-4094 (1998). doi:10.1016/S1352-2310(98)00248-9 [52] K.A. Emanuel, M. Živkovi´c-Rothman, Development and Eval- uation of a Convection Scheme for Use in Climate Mod- els. J. Atmos. Sci. 56, 1766-1782 (1999). doi:10.1175/1520- 0469(1999)056<1766:DAEOAC>2.0.CO;2 [53] J.H. Seinfeld, S.N. Pandis, Atmospheric chemistry and physics: from air pollution to climate change. (John Wiley Sons, 2016). [54] J.C. Lin, C. Gerbig, S.C. Wofsy et al., A near-field tool for simulating the upstream influence of atmospheric ob- servations: The Stochastic Time-Inverted Lagrangian Trans- port (STILT) model. J. Geophys. Res.:Atmos. 108, (2003). doi:10.1029/2002JD003161 [55] C.-K. Song, C.-H. Kim, S.-H. Lee et al., A 3-D Lagrangian particle dispersion model with photochemi- reactions. Atmos. Environ. 37, 4607-4623 (2003). doi:10.1016/j.atmosenv.2003.08.001 tive pollution from nuclear power plant accidents: application to ETEX tracer experiment. J. Environ. Radioact. 142, 78-86 (2015). doi:10.1016/j.jenvrad.2015.01.013 [31] X.L. Zhang, G.F. Su, J.G. Chen et al., Iterative ensemble Kalman filter for atmospheric dispersion in nuclear accidents:

An application to Kincaid tracer experiment. J. Hazard. Mater. 297, 329-339

(2015). doi:10.1016/j.jhazmat.2015.05.035 [32] Y. Jianyao, H. Yuan, G. Su et al., Machine learning-enhanced high-resolution exposure assessment of ultrafine particles. Nat.

Commun. 16, 1209 (2025). doi:10.1038/s41467-025-56581-8 J.-P. Zhang, W.-D. Yang et al., Predict- ing and Controlling Nuclear Accident Hazards:

Issues and Challenges. Aerosol Air Qual. Res. 16, 417-429 (2016). doi:10.4209/aaqr.2014.12.0320 [33] S.-X. Huang, [34] S. Sun, H. Li, S. Fang, A forward-backward coupled source term estimation for nuclear power plant accident: A case study of loss of coolant accident scenario. Ann. Nucl. Energy. 104, 64-74 (2017). doi:10.1016/j.anucene.2017.01.039 [35] O. Saunier, I. Korsakissok, D. Didier et al., Real-time use of in- verse modeling techniques to assess the atmospheric accidental release of a nuclear power plant. Radioprotection. 55, 107-115 (2020). doi:10.1051/radiopro/2020044 [36] S. Fang, X. Dong, S. Zhuang et al., Oscillation-free source term inversion of atmospheric radionuclide releases with joint model bias corrections and non-smooth com- J. Hazard. Mater. 440, 129806 (2022). peting priors. doi:10.1016/j.jhazmat.2022.129806 [37] S. Andronopoulos, I.V. Kovalets, Method of Source Identifica- tion Following an Accidental Release at an Unknown Location Using a Lagrangian Atmospheric Dispersion Model. Atmos. 12, 1305 (2021). doi:10.3390/atmos12101305 [38] W. Cui, B. Cao, Q. Fan et al., Source term inver- sion of nuclear accident based on deep feedforward neu- ral network. Ann. Nucl. Energy. 175, 109257 (2022). doi:10.1016/j.anucene.2022.109257 [39] L. Hoffmann, K. Haghighi Mood, A. Herten et al., Accelerating Lagrangian transport simulations on graphics processing units: performance optimizations of Massive-Parallel Trajectory Cal- culations (MPTRAC) v2.6. Geosci. Model Dev. 17, 4077-4094 (2024). doi:10.5194/gmd-17-4077-2024 [40] Y. Ling, C. Liu, Q. Shan et al., Source term inversion of short- lived nuclides in complex nuclear accidents based on machine learning using off-site gamma dose rate. J. Hazard. Mater. 465, 133388 (2024). doi:10.1016/j.jhazmat.2023.133388 [41] Y. Zhao, Y. Liu, L. Wang et al., Source Reconstruction of At- mospheric Releases by Bayesian Inference and the Backward Atmospheric Dispersion Model: An Application to ETEX- I Data. Sci. Technol. Nucl. Install. 2021, 5558825 (2021). doi:10.1155/2021/5558825 [42] Q. Li, J. Zhang, B. Lian et al., A Bayesian Source Term in- version Method Based on Spatiotemporal Trajectory Prior and Joint Adaptive MCMC Sampling. doi:10.12074/202504.00011 [43] Y. Xu, X. Dong, S. Fang, Efficient Bayesian source re- construction and uncertainty quantification of atmospheric radionuclide releases by replacing release rate sampling with Maximum-A-Posteriori time-varying estimation of J. Hazard. Mater. 492, 138171 (2025). release rates. doi:10.1016/j.jhazmat.2025.138171 [44] P. Harvey, S. Hameed, W. Vanderbauwhede, Accelerating La- grangian particle dispersion in the atmosphere with OpenCL across multiple platforms. Paper presented at the Proceedings of the International Workshop on OpenCL 2013 2014, Asso- ciation for Computing Machinery, Bristol, United Kingdom, Appendix A: Design, Computational Details, and Intermediate Variables of the Preprocessing Program gribtobinmpi The core idea of the decoupling strategy

is to transform meteorological field processing into a one-time precomputation step. Specifically, tasks are distributed through an MPI parallel framework, with each process responsible for a subset of GRIB files. Required variables are extracted and computed independently by each process and subsequently stored as standalone binary files. As a result, the FLEXPART main program no longer needs to repeatedly read GRIB files; instead, it directly loads the preprocessed binary data, thereby substantially reducing I/O overhead and redundant computations.

In parallel, the preprocessing program generates an AVAILABLE_{bin} file that records the names of the available binary wind-field files, ensuring compatibility with the original AVAILABLE mechanism. This design leverages MPI broadcast and synchronization to maintain global consistency, while load-balanced task allocation enables efficient utilization of multicore resources.

The main workflow of the preprocessing program is summarized as follows: 1. MPI initialization and process management. Upon startup, the program calls MPI_{Init} to initialize the MPI environment and retrieves the total number of processes (nprocs) and the process rank (myrank). The root process (myrank = 0) is designated with lroot = .true. and is responsible for overall coordination. 2. Configuration file reading and format detection. The root process reads the COMMAND, RELEASES, and AVAILABLE configuration files to obtain simulation parameters (e.g., ldirect, ind_{receptor}, ipin). The meteorological data format (ECMWF or NCEP) is identified using detectf ormat(). Depending on the detected format, gridcheck_{ecmwf} () or gridcheck_{gf} s() is invoked to analyze grid dimensions (e.g., nxmax, nymax, nuvzmax). The resulting metadata are written to header.t for subsequent use by FLEXPART. 3. Global broadcast and memory allocation. The identified format, grid dimensions, and related parameters (e.g., nxshif t, nx, ny, nz, metdata_f ormat) are broadcast to all processes via MPI_{Bcast}. Each process then allocates memory for the required data structures (such as the three-dimensional arrays uuh, vvh, and wwh), ensuring consistency across all ranks. 4. Parallel file processing. GRIB files are distributed among processes according to the numbwf and wf name arrays. Each process reads its assigned files and invokes readwind_{ecmwf} () or readwind_{gf} s() to extract wind-field data. After completing parameter calculations and necessary transformations, the results are written to .bin files. An MPI_{Barrier} is used to synchronize all processes, after which the root process generates the AVAILABLE_{bin} file listing the produced binary files.

5. Memory deallocation and program termination.

Corresponding modifications are also required in the FLEXPART main program. Specifically, gridcheck() is adapted to obtain meteorological format and grid-dimension information directly from header.t, and getf ields() is modified to read data from the .bin files rather than from GRIB input. These changes replace the original GRIB-based workflow and ensure seamless integration of the preprocessing scheme into the overall model framework.

Table A.1. Parameters contained in the binary meteorological field data files. Superscript 1 denotes spatial two-dimensional variables, superscript 2 denotes vertical-level (one-dimensional) variables, and superscript 3 denotes scalar variables; all remaining variables represent spatial three-dimensional fields. Variables with the suffix eta correspond to their counterparts defined in the -coordinate system.

Description Variable short name temperature data on half model levels specific humidity data on half model levels surface pressure total cloud cover 2 meter temperature 2 meter dew point lsprec1 large scale total precipitation convprec1 convective precipitation orography land sea mask logical, indicating whether clwc is available (see ctwc1) ustar1 friction velocity wstar1 convective velocity scale hmix1 mixing height tropopause1 altitude of thermal tropopause inverse Obukhov length (1/L) vdep1 deposition velocity wind components in x uu (uueta) wind components in y vv (vveta) wind components in z ww (wweta) uupol (uupoleta) wind components in polar stereographic projection vvpol (vvpoleta) wind components in polar stereographic projection tt (tteta) temperature data on internal model levels specific humidity data on internal model levels pv (pveta) potential vorticity rho (rhoeta) air density drhodz (drhodzeta) vertical air density gradient ctwc1 total cloud water content (= liquid clwc + ice ciwc) icloudbot1 cloud bottom height icloudtop1 cloud top height2 heights of all levels wheight2 (etawheight2) model level heights uvheigh2 (etauvheight2) half-model level heights nmixz3 pplev number of levels up to maximum PBL height (3500 m) pressure on half model levels air pressure RLT Table A.2. List of general information produced by the preprocessing program and stored in header.t.

Description Variable name metdata_f ormat storing the input data type (ECMWF/NCEP) nxmax nymax nuvzmax nwzmax nzmax nxmin1 nymin1 nxf ield nlev_{ec} xglobal sglobal nglobal dxconst dyconst xlon0 ylat0 southpolemap northpolemap switchnorthg switchsouthg nconvlevmax Size of windfield Size of windfield Size of windfield Size of windfield Size of windfield actual dimensions of wind fields in x, y and z direction actual dimensions of wind fields in x, y and z direction actual dimensions of wind fields in x, y and z direction nx − 1 ny − 1 same as nx for limited area fields, but for global fields nx = nxf ield + 1 vertical dimension of original data (u,v components(staggered grid)) vertical dimension of original data (w component) number of levels ECMWF model T for global fields, F for limited area fields T for global fields, F for limited area fields T for global fields, F for limited area fields coefficients which regulate vertical discretization coefficients which regulate vertical discretization model discretization coefficients at the centre of the layers model discretization coefficients at the centre of the layers grid distance in x direction grid distance in y direction auxiliary variables for utransform auxiliary variables for utransform geographical longitude of lower left grid point geographical latitude of lower left grid point define stereographic projections at the two poles define stereographic projections at the two poles use polar stereographic threshold in grid units use polar stereographic threshold in grid

units maximum number of levels for convection parameter used in Emanuel' s convect subroutine Appendix B: Resource utilization of individual components without imposing a per-thread maximum register limit.

Table B.1. Resource utilization of the GPU model prior to optimization. Computational throughput and memory throughput are expressed as percentages of the device theoretical peak values.

Computational Module Registers per Thread Occupancy Computational Throughput Memory Throughput Advection-Diffusion Convective mixing Wet deposition 16.67% 33.33% 16.67% 4.5%

Note: Figure translations are in progress. See original paper for figures.