OceanTracker 0.5: Fast Adaptable Lagrangian Particle Tracking in Structured and Unstructured Grids

Ross Vennell[1], Laurin Steidle[2], Malcolm Smeaton[1], Romain Chaput[1], and Ben Knight[1]

[1]Cawthron Institute, Nelson, New Zealand
[2]University of Hamburg, Germany

Correspondence: L. Steidle (laurin.steidle@cawthron.org.nz)

Abstract. Particle tracking is frequently used to compute particle movements within hydrodynamic ocean models; however, modelling millions of particles is computationally challenging. OceanTracker is designed to reduce the time required to obtain results from particle tracking. Firstly, by being computationally efficient, it enables users to simulate large numbers of particles in complex costal environments, enabling improved statistics or exploring a wider range of cases within acceptable run times. Secondly, OceanTracker can calculate multiple particle statistics during the computational run, eliminating the need to record and post-process large volumes of particle trajectories. The adaptability of OceanTracker's modular computational pipeline allows users to add and modify components which govern particle physics, behaviour, and statistics. The computational pipeline is entirely assembled from user-provided parameters, supplied as a text file or built using helper methods. Coders can easily modify existing components through code inheritance. Currently, OceanTracker supports hydrodynamic model output for unstructured grids (SCHISM, FVCOM, DELFT3D-FM) and structured grids (ROMS, NEMO/GLORYS). Computing the trajectories for more than a million particles with OceanTracker on a single computer core is more than ten times faster than the OpenDrift code and twice as fast as the Ocean Parcels code, despite treating structured grids as unstructured. In addition to its single-core performance, OceanTracker can run computations in parallel across available cores. This can further increase speed by up to an order of magnitude on currently available multicore desktop processors.

1 Introduction

Particle tracking is key to answering many scientific and practical questions about physical and biophysical transport in the ocean (Lynch et al., 2014; Van Sebille et al., 2018; Garnier et al., 2025), such as dispersal of pollutants (Ainsworth et al., 2021), movements of larvae, residence times (Lucas and Deleersnijder, 2020), or quantifying connectivity between regions, (Atalah et al., 2022).

A major challenge is simulating millions of particles to enable statistically significant results when introducing stochastic movements in the models to account for uncertainties in the physical currents or the biological traits (Van Sebille et al., 2018; Chaput et al., 2022). This is particularly important when estimating the bio-physical connectivities of rare events, which have significant consequences, such as for the initial arrival of invasive species. In these cases, it becomes necessary to release large numbers of particles to represent very large numbers of possible trajectories in the system.

Simulating millions of particles creates two challenges. Firstly, computing their trajectories in user-acceptable time frames. Secondly, the time required to post-process the trajectories to obtain the required particle statistics, along with the effort required to manage large volumes of output (see Sec. 2.1). A third challenge is creating code that is easy to use and yet adaptable to many use cases, e.g. for different particle behaviours. OceanTracker addresses all three challenges.

OceanTracker's computational speed addresses the first challenge by computing large numbers of particle trajectories with modest computer hardware. Table 1 shows a range of hardware, all of which can compute one month of particle trajectories for 1 million particles in 4 to 20 minutes, depending on the number of processors and their speed. This computational speed is mainly a result of efficient numerical algorithms (Vennell et al., 2021), and secondarily due to spreading computations across multiple computer cores using threading (see Sec. 4.2).

Having the computational speed to scale to millions of particles creates the second challenge, the time and effort required to analyse the trajectories to yield the required particle statistics, along with the logistical issues of storing and accessing what may be terabytes of data. OceanTracker resolves these issues by recording particle statistics during the computational run, e.g. heat maps and connectivities between regions, as shown in Fig. 1 and Sec. 3.8. On-the-fly statistics address both the post-processing time and storage issues, as no particle trajectories need to be recorded, and the size of the statistical output files is independent of the number of particles released. If required, individual particle trajectories can also be recorded to allow detailed inspection of results.

A modular *computational pipeline* addresses the third challenge of adaptability, by structuring it as a sequence of discrete steps, each responsible for a specific computational task (see Fig. 3). Users configure these tasks by adding *components*, which are implemented as Python classes and dynamically assigned to *roles* at runtime. For example, multiple release group components, each with distinct locations and schedules, can be assigned to the particle release role within a single computational run. Similarly, multiple components can be used concurrently to compute distinct on-the-fly statistics. Other roles in the computational pipeline include modelling dispersion, resuspension, and modifying trajectories following biological behviours (e.g., larvae settling within a defined area). This modular approach allows users to flexibly construct custom simulations tailored to their specific needs.

A challenge inherent in dynamically building the computational pipeline built from components is enabling both coders and non-coders to access the same level of adaptability. To address this, OceanTracker's computational pipeline is entirely constructed from a user-supplied configuration file or Python dictionary. This defines which components are added to roles within the computational pipeline. For each component, the configuration also gives the *parameters* which specify their individual settings, Sec. 3.1.1 Fig. 4 provides a minimal example of this, showing the same run executed either through coding or, or from parameters supplied in one of two standard text file formats. An additional advantage of the use of text file parameters is that it allows web-based on-demand particle tracking services to easily access the same level of computational pipeline flexibility, (Vennell et al., 2019).

A single particle tracking framework that works for both structured and unstructured hydrodymamic model grids simplifies the particle tracking process for users by having a uniform approach regardless of the hydrodynamic model used. Particle trackers for unstructured grids are rare due to the complexity involved in coding the movement and interpolations within the

grid. A review of ocean Lagrangian analysis (Van Sebille et al., 2018) lists 11 particle trackers, of which only one "LIGHT in MPAS-O" is capable of working with unstructured grids. Those compatible with both types of grids are even rarer, e.g OpenDrift. A secondary goal for OceanTracker is to provide a single framework that auto-detects the type of grid from multiple hydrodynamic model formats and identifies useful optional variables, such as bottom stress used in particle re-suspension. This enables users to focus on the outcomes they need from particle tracking, rather than the details of the underlying hydrodynamic model's grid and outputs.

OceanTracker has been applied to a range of studies involving particle-based simulations to understand transport processes in marine environments. These applications typically involve large-scale particle releases, tracking their movement over time, and analysing connectivity patterns. Examples include:

- Backtracking to infer likely locations of mussel larvae parents: Approximately 600 million particles were released over ten years and tracked for their six-week lifetimes to estimate where settled larvae originated (Atalah et al., 2022).
- Investigating phytoplankton retention in an estuary: A simulation with one billion particles — up to one million active at any time — was used to model population growth through particle splitting and retention mechanisms(Steidle and Vennell, 2023).
- Inferring the dispersal area of eDNA: OceanTracker was used to infer the spatial extent within which species' environmental DNA may be detected from water samples (Pastor Rollan et al., 2024).
- Assessing invasive species risk from ballast water discharge: The transport of invasive species was modeled by releasing 65 million particles from ships along coastal shipping routes (Smeaton et al., in prep).
- Modelling disease spread in aquaculture farms: Connectivity between 500 aquaculture farms was analyzed using 150 million particles to determine potential disease transmission pathways (Knight et al., 2024).
- Engaging the public with an online ocean plastics tracker: A web-based tool allows users to drop virtual plastic and instantly visualize where it travels, returning 20 particle trajectories in less than one second (Vennell et al., 2019).

This paper outlines the features and structure of the latest version of OceanTracker (Vennell et al., 2021). An overview of its features is provided in Sec. 2, including some example outputs in Fig. 1. Sec. 3 details its structure and describes how to configure the computational pipeline in Sec. 3.1.1. Additionally, Sec. 3.9 outlines integrated models which combines components within roles to achieve higher-level functionality, such as calculating Lagrangian coherent structures. Sec. 4 explores the features contributing to OceanTracker's speed and compares its performance to OpenDrift and Ocean Parcels.

1.1 Example

Fig. 1 presents some basic examples of OceanTracker outputs from a 3D simulation. particles released from points, polygons and a regular grid, all within the same computational run, see Sec. 3.5. The snapshot in Fig. 1a) displays particles coloured according to their status, moving (blue), stranded by the tide (green) or on the bottom (grey).

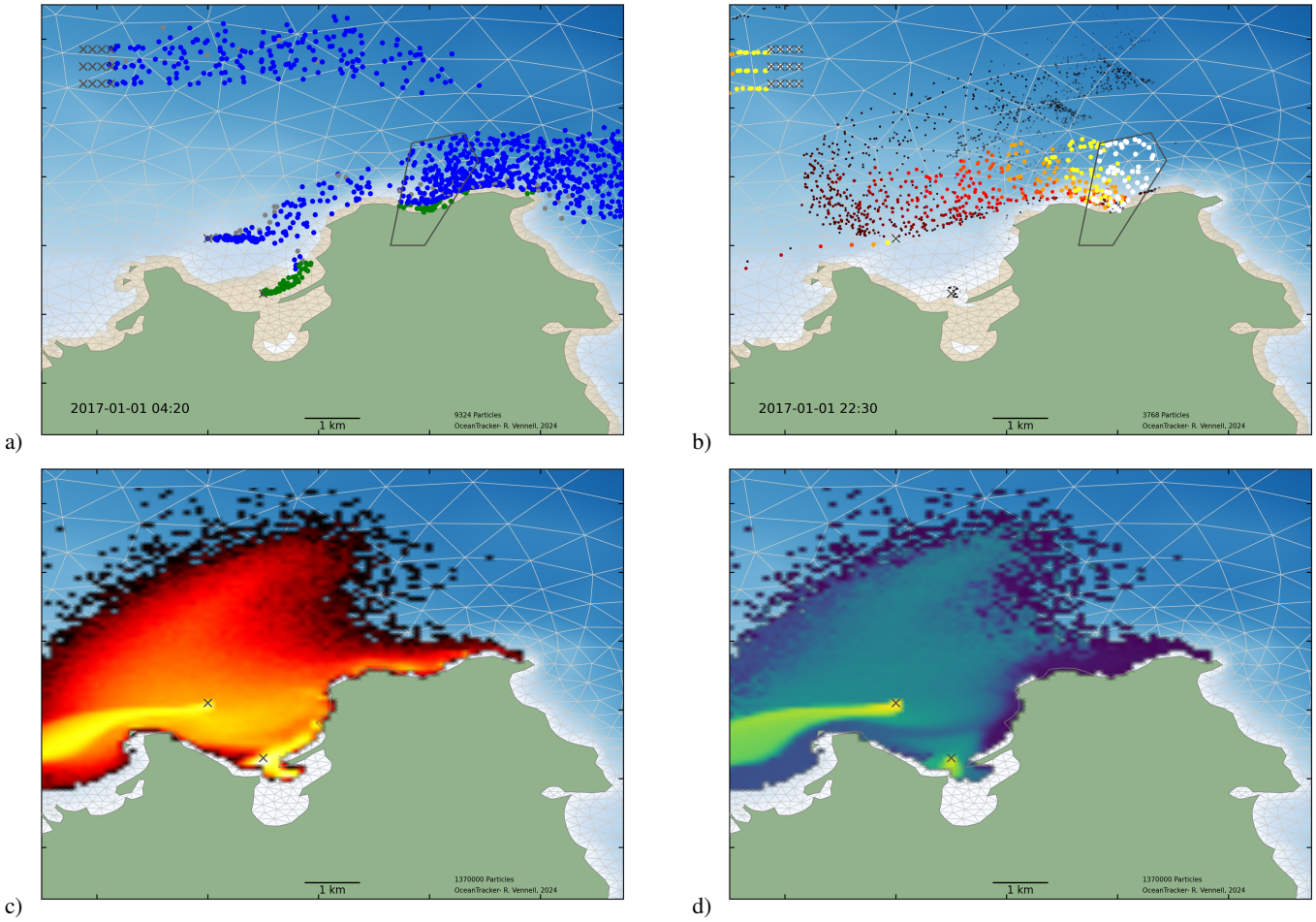


Figure 1. Examples of results from point, polygon and grid particle release groups computed during a single run, Sec. 3.5. a) A snapshot of the particles: blue indicates particles are moving, grey signifies particles on the bottom that may later resuspend and green particles signifies those stranded by an outgoing tide. Dry cells are shown in brown and blue shading indicates water depth. b) Particles sized and coloured according to a decaying particle property Sec. 3.2.2. c) Heat-map of log particle counts from a release of 1.3 million particles from a pair of point sources. d) Heat-map of decaying particle property on a logarithmic scale. The code to run this example is in Appendix A.

		Specifications			Minutes per modelled month		
	CPU Name	Physical cores	RAM Gb	Freq. MHz	Read	Computation	Total
Laptop I	Intel i5-8265U (2018)	4	8	1.6-3.9	4	18	22
Laptop II	Intel i7-1255U (2022)	10	16	1.7-4.7	7	16	21
Desktop I	AMD Ryzen 2700X (2018)	8	32	3.7-4.4	5	8	13
Work station	Dual Intel Xeon 6154 (2017)	36	512	3.0-3.7	4	5	9
Desktop II	AMD Ryzen 7970X (2023)	32	512	4.0-5.3	2	2	4

Table 1. OceanTracker run times for modelling 1 million particles over a period of 1 month. The runs utilised all physical computer cores, completed 2880 time steps of 15 minutes each using Runge-Kutta 4 time integration in 3D. The unstructured hydrodynamic model had 79k nodes and 140k triangles.

Heat maps can be used to illustrate the decay and dispersion of a pollutant from its source. To construct heat maps using particle tracking, enough particles must fall within each grid cell to ensure a spatially smooth representation. The efficiency of OceanTracker allows for the release of millions of particles, enabling the direct computation of heat maps without the need for additional radial smoothing of particle counts. In Fig. 1c, 1.3 million particles were sufficient to produce a detailed heat map. This heat map was generated by counting particles into grid cells on the fly, thereby avoiding the need to record large volumes of particle trajectories. Fig. 1d) illustrates a heat map of the average value of a user-added exponentially decaying particle property.

2 OceanTracker features

This section highlights some of OceanTracker features that minimise user effort and its physics.

2.1 Features reducing user effort

In addition to computational speed, OceanTracker incorporates an number of features that significantly reduce the time and effort required for users to obtain results from the analysis of particle trajectories. The most significant features are:

Release groups: Allows simultaneous release of multiple groups of particles, each with distinct locations and timing. This functionality enables users to obtain more comprehensive results from a single computational run (see Fig. 1, Sec. 3.5 and Appendix A). This eliminates the need for setting up and managing multiple runs to explore different release scenarios.

On-the-fly particle statistics: OceanTracker enables the simultaneous computation of multiple statistics. For instance, with the same run OceanTracker can generate heat maps for moving and settled particles while also calculating the connectivity between different areas defined by polygons, all within the same run and without additional post-processing steps. These statistics are based on counting particles within user specified grids or polygons on-the-fly. On-the-fly statistics are calculated

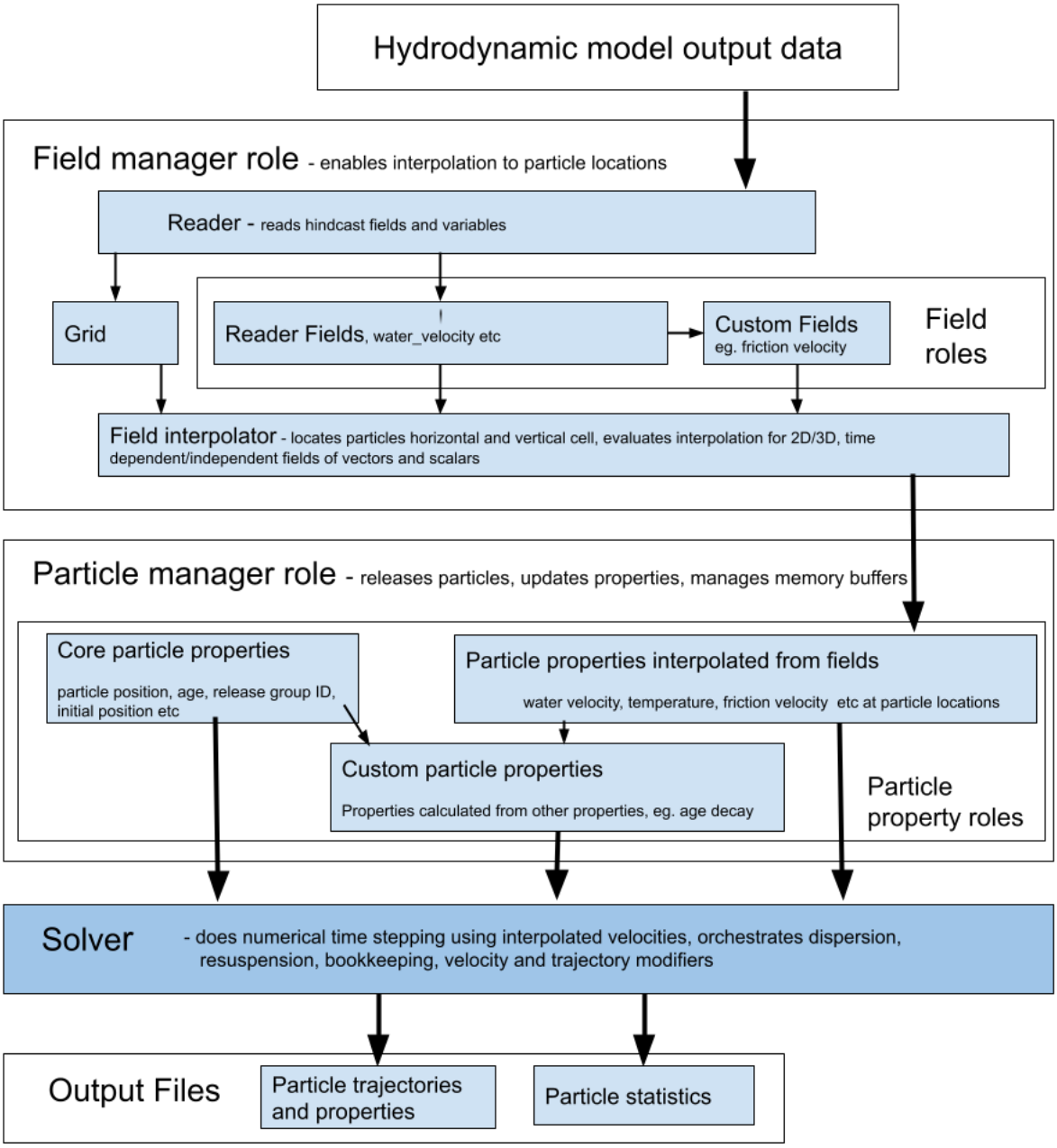


Figure 2. Outline of data flow through the two main data structures, fields and particle properties, from hydrodynamic model files to output. These data structure roles are outlined in Sec. 3.2. Each have "managers" to orchestrate operations on the variants of each data structure, to deliver particle properties to the solver. The steps carried out by the solver are given in Fig. 3.

separately for each release group, e.g. individual heat maps for each release site, or connectivities between releases within multiple polygons and another set of polygons.

Coding productivity New functionalities for tailored applications can be added to OceanTracker using the widely used Python language NumPy package (Harris et al., 2020), without the need for users to learn an additional language or syntax. In addition, user's computationally intensive tasks can often be sped by simply adding a Numba decorator to their functions which use Python and NumPy code (Lam et al., 2015). Thus, avoiding the need to learn an additional language, such as C, to speed intensive operations (more on Numba in Sec. 4).

Self-managing particle buffer: Particle properties such as current location or status are stored in memory buffers, that expand dynamically as more particles are released. This removes the need to specify the maximum number of particles that will be used in advance. The memory buffers are also dynamically consolidated as particles are killed to minimise the total memory required, allowing efficient computation even on memory restricted hardware.

Offline particle tracking Like many other particle trackers, OceanTracker saves users' time by performing particle tracking "offline", based on the recorded output from a hydrodynamic model (e.g. PARTRACK (Knight et al., 2009), LIGHT (Wolfram et al., 2015), OpenDrift (Dagestad et al., 2018), ROMSpath (Hunter et al., 2022) and Ocean Parcels (Delandmeter and Van Sebille, 2019)). In contrast, "online" particle tracking performs computations during the hydrodynamic model run. Hydrodynamic model run times are typically much longer than those of particle tracking. Thus, for online particle trackers any new variations in particle tracking requires re-running a hydrodynamic model. Offline particle tracking enables faster exploration of variations in behaviours and the sensitivity of results to parameters, e.g. fall velocity.

2.2 Physics features

Along with advection by water velocity, the core physics features affecting particle motions are outlined below. A coding user can easily adapt these to create variants to suit their need through class inheritance.

Dispersion: A random walk mechanism simulates dispersion due to sub-hydrodynamic grid scale processes (Lynch et al., 2014). By default, constant turbulent eddy viscosities are applied. However, if vertical 3D turbulent viscosity profiles are available in the hydrodynamic model, these profiles are interpolated to calculate the vertical random walk instead. This, requires the inclusion of an additional vertical velocity equal to the vertical gradient of the turbulent viscosity (Visser, 1998). Note that, the random walk is not applied as a step change in position, but as an equivalent additional velocity applied to each Runge-Kutta (RK) sub-step in the velocity modifiers loop, see Fig. 3 and Sec. 3.6.

Re-suspension: Particles that settle on the seabed can be re-suspended if the flow is strong enough. The likelihood and height of re-suspension depend on the friction velocity, as described by Lynch et al. (equation 9.28 in Lynch et al. (2014)). Stronger flows increase the probability of re-suspension and cause particles to be lifted higher into the water column. The friction velocity is calculated from the near-bed velocity using a logarithmic velocity profile or, if available, from the bottom-stress field in the hydrodynamic model output.

Tidal stranding: Particles within hydrodynamic model cells that become dry as the tide recedes remain stationary until the cell becomes wet again. Dry cells are flagged based on flags in the hydrodynamic model files. If a dry cell flag is not available, a cell is considered dry when the total water depth at its center falls below a user-specified minimum value.

Backtracking: OceanTracker supports reverse time simulations, which can be useful for identifying potential sources of particles arriving at a given location. (Thygesen, 2011). Note that dispersion is not time-reversible, and this operates the same in the forward direction time, producing outputs, like heat maps, that offer a probabilistic view of sources.

Nested grids: To enable particle tracking beyond the open boundaries of a single hydrodynamic model's grid, OceanTracker can nest multiple inner grids within a broader coarser outer grid. Particles exiting the open boundary of an inner grid are transferred to the outer grid, and particles on the outer grid which move inside an inner grid, are transferred to that grid. Each particle is aware of its current grid, allowing field values to be interpolated from the relevant grid. The inner and outer grids may consist of any combination of structured or unstructured grids.

In addition to the above, OceanTracker allows users to add particle behaviour by adding addition physical processes. This is achieved by adding given components to specific computational roles, such as velocity modifiers (e.g., sinking velocities) and trajectory modifiers (e.g., particle splitting or larval behaviors), as described in Sections 3.7 and 3.6.

3 OceanTracker structure

At a high level, particle tracking code takes the Eulerian water velocity field from a hydrodynamic model, interpolates these velocities to provide Lagrangian velocities at particle locations, and then numerically integrates these to give particle trajectories.In addtion, there are multiple other processes and computations that must be done at each time step. Such as the physical processes of dispersion, resuspension and tidal stranding, along with multiple bookkeeping processes. OceanTracker decomposes these processes into a series of components that fulfil specific roles, as part of the *computational pipeline* outlined in Fig. 3 (see Sec. 3).

Fig. 2 illustrates the data flow in OceanTracker from hydrodynamic model files to outputs, via its two main data structures *fields* and *particle properties* (see Sec. 3.2). Fields store data from the hydrodynamic model, such as water velocity, salinity, wind stress, as well as custom fields which are calculated from other fields, such as friction velocity. Particle proprieties hold data for each particle, which could include their current locations, status or temperature. These data structures enable access to, and operations on, their data and are collectively "managed" by their respective manger roles (see Sec. 3.3).

3.1 Computational pipeline

The computational pipeline is constructed from components assigned to specific *roles* within the pipeline which implement required tasks at each time step, see Fig. 3. From a user's perspective, the adaptability of OceanTracker comes from the ability to customise which components are added each role to create a computational pipeline for their needs. For example, a gridded 2D statistic added to the 'particle statistics' role to create heatmaps of particle locations on-the-fly.

Thes components are constructed as Python classes, which are dynamically added to the computational pipeline during setup. Time steping within the computational pipeline proceededs by calling the "update" method of each component within each role. Some roles, such as dispersion and re-suspension, allow only one class to be added, while others allow multiple classes to be added to that role. For exmaple, multiple trajectory modifiers can be combined to give the required particle behaviour.

3.1.1 Building a computational pipeline.

The computational pipeline in OceanTracker can be fully configured using parameters provided in either a text file or a Python dictionary, eliminating the need for direct coding (see Fig. 1b, c). For more complex simulations, parameters can be specified programatically using a helper wrapper. This wrapper provides two key methods"settings" and "add_class", which enable users to construct a parameter dictionary using keyword arguments (see Fig. 1a). The resulting parameter dictionary follows the same structure as the JSON text file format shown in Fig. 1c and is passed to OceanTracker for execution.

Users can configure top-level settings, such as the particle tracking time step and output directories, as well as component-specific settings. This trajectory calculation process constitutes a series of roles performed by a collection of components. Each component's configuration typically includes the name of the Python class assigned to a particular role. Most component settings have default values defined within their respective Python classes, but some must be explicitly provided by the user. During setup, all user-defined settings are automatically validated for type correctness and appropriate value ranges.

3.1.2 Computational steps.

Time stepping in OceanTracker integrates particle velocity to compute trajectories. Within the computational pipeline, time stepping is divided into a series of roles, each performed by one or more components (see Fig. 3). The update of each component is automated within the time stepping of the computational pipeline, with updates grouped by role, see Fig. 3. The order in which the roles are updated reflects their temporal dependence on the data from other roles. For example, custom particle properties may be calculated from field particle properties and are therefore updated after the interpolated field properties. The solver class implements the time stepping by managing the classes within the computational pipeline and the associated bookkeeping functions.

The first step in the computational pipeline involves looping over the release groups and releasing a single pulse of particles if scheduled for the current time step, and then finding each newly released particle's current horizontal and vertical cell. The solver then checks if the reader field time buffers contain the required time steps; if not, the reader fills the buffers and any custom fields are calculated from the newly read time steps. Next, any additional particle velocities are added together to give a total of "velocity modifiers" (e.g. a fall velocity), to this a random walk dispersion is added as an equivalent velocity, see Sec. 2.2. Particles are then advected by RK integration based on the sum of the water velocity and the total of the velocity modifiers. At each sub-step, their current horizontal and vertical cells and barycentric coordinates are updated, so that each particle's velocity can be interpolated to their locations. By default, fourth order RK time integration is used, although first and second order RK are also available.

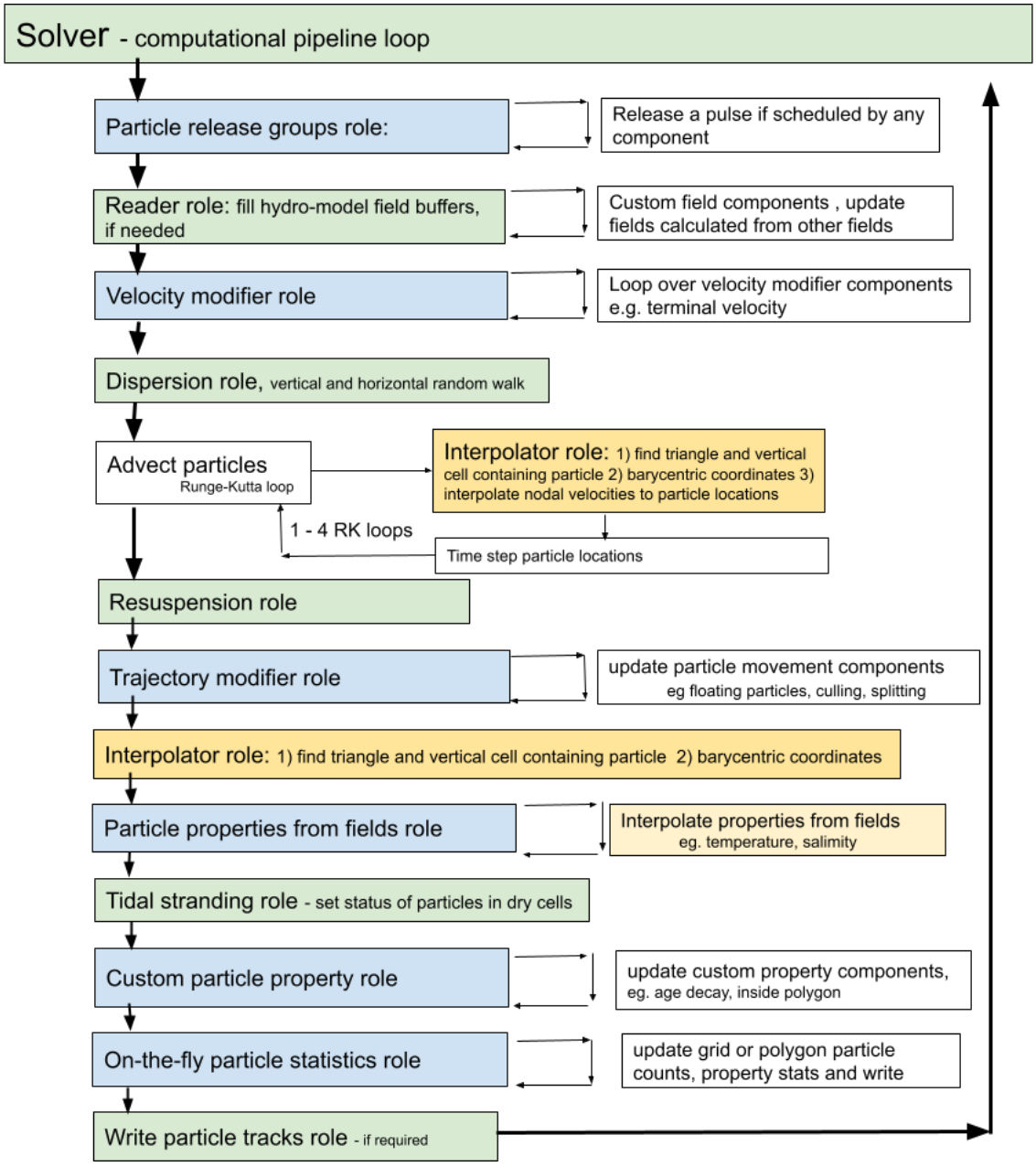


Figure 3. Flow chart of time stepping to advect particles within the solver component in Fig. 2. The figure illustrates the order in which components are updated within their roles in the computational pipeline . These operation roles are outlined in Sec. 3.2. Green are "core roles" which only containing one component, blue roles may contain one or more components which are looped through. For large runs, the most computational expensive steps are 1.) finding the cell containing each particle and 2.) evaluating the field interpolation. These steps are coloured yellow.

a) python

```python
from oceantracker.main import OceanTracker
# make instance of oceantracker
ot = OceanTracker()

# add  settings
ot.settings(output_file_base='minimal_example',
            root_output_dir='output',
            time_step= 120.)
# reader for hindcast files, format is auto detected
ot.add_class('reader',
             input_dir= '..\\demos\\demo_hindcast',
             file_mask=  'demoHindcastSchism*.nc')
# add  (x,y,z) locations where particles are released
# note: can add multiple release groups
ot.add_class('release_groups', name ='my_release_points',
             points= [[1595000., 5482600.,-1.],
                      [1599000., 5486200.,-2.]],
             release_interval= 3600, pulse_size= 10)
# start computation
ot.run()

```

b) yaml

```yaml
output_file_base: minimal_example
root_output_dir: output
time_step: 120.0
reader:
  input_dir: "..\\demos\\demo_hindcast"
  file_mask: "demoHindcastSchism*.nc"
release_groups:
  my_release_point:
  points: [[1595000,5482600],
           [1599000,5486200]]
  release_interval: 3600
  pulse_size: 10
```

c) json

```json
{
  "output_file_base": "minimal_example",
  "root_output_dir": "output",
  "time_step": 120.0,
  "reader": {
    "input_dir": "..\\demos\\demo_hindcast"
    ,
    "file_mask": "demoHindcastSchism*.nc"
    },
  "release_groups": {
    "my_release_point": {
      "points": [[1595000,5482600],
                 [1599000,5486200]],
      "release_interval": 3600,
      "pulse_size": 10}
  }
}
```

Figure 4. Minimal example of building a computational pipeline by a) Coding, using the 'helper' wrapper to build a Python parameter dictionary, or as user supplied parameters within a text file following the b) YAML or c) JSON standard.

Once particles are at their a new locations, additional changes to particle status or positions due to resuspension or specified trajectory modifiers are made. These modifiers add any additional particles movements or changes to their status, e.g. settling in a polygon or culling. To enable interpolation after these movements', particle horizontal and vertical cells and Barycentric coordinates are updated. Then, all field-derived particle properties are looped over and updated by interpolation. Subsequent steps include updating any custom particle properties, calculating any particle statistics that have been added to the computational pipeline, and then writing out time series of particle trajectories and particle properties if this has been requested

3.1.3 Mechanisms enabling computational pipeline adaptability

OceanTracker's computational pipeline pipeline flexibility is implemented through several key mechanisms:

Dynamic importing: Components for all roles are dynamically imported at runtime, allowing users to build custom computational pipelines. To specify a component, users provide a "class name" setting (see Sec. 3.1.1), which follows standard Python class referencing conventions. Core roles have default class names, and all built-in OceanTracker classes can be imported using a shortened version of their full names. Users can also import customized variants of Python classes into any of the roles in the computational pipeline , enabling them to modify core functionality or add optional functionality (e.g. dispersion models, trajectory modifiers or on-the-fly statistics).

Inheritance: New component variants can be created by inheriting from an existing class. Users can modify computations by overwriting specific methods, such as the "update" method, while retaining inherited configuration settings and the infrastructure than manages bookkeeping operations. This approach allows for incremental modifications—settings from a parent class are automatically inherited but can be redefined, removed, or extended. For example, the polygon particle release class inherits most of its functionality from the point release parent class.

Internal naming: To simplify code interaction, OceanTracker provides standardised internal names for instances of the field and particle property components. Additionally, both reader-defined fields and custom user-defined fields automatically generate particle properties of the same name, which are interpolated at each time step and can be accessed directly in the code. Users can also assign names to optional components for easier reference within their scripts. For example, essential fields like "water velocity" and "water depth" are mapped to corresponding variables in the hydrodynamic model files by the reader. Vector fields may require mapping multiple file variables to a single internal name

3.2 Data structures computational pipeline

OceanTracker's two main data structures are fields and particle properties, see Fig. 2. This section outlines these data roles, while Sec. 3.3 describes their management.

3.2.1 Field role

The spatial fields stored within the hydrodynamic model's files are the foundation of offline particle tracking. These are stored and accessed using the "field" data structure, see Fig. 2. The water velocity field is a crucial spatial field for particle tracking,

but there may be many other relevant fields, such as water depth, tide or wind-stress. OceanTracker automatically detects whether fields are 2D or 3D, time-dependent or independent, scalar or vector, and manages they appropriately. There are also grid variables associated with the fields, including nodal locations, triangulation and cell adjacency.

Fields are read from hydrodynamic model files as described in Sec. 3.4. Users can also integrate custom fields into the computational pipeline by deriving new values from existing fields. For example, a friction velocity field can be computed from the 3D velocity field near the seabed to determine whether a particle can be re-suspended by the flow.

3.2.2 Particle property role

There are three types of particle properties which store the values for each particle and enable high level operations on these values as shown in Fig. 2b. These properties can different types depending on how they are updated at each time step, and may be integer, bookkeeping information, scalars or vectors.

Field particle properties store field values at each particle location, and are updated by interpolating a field data structure, eg. water velocity.

Custom particle properties store values calculated from other particle properties. For example, the "inside-polygons" class uses the location particle property to determine whether any user-given polygon contains each particle. These are updated by calling their classes's update method.

Core particle properties are updated within the core code, i.e. outside of their class update method. Examples include particle location and bookkeeping particle properties, such as particleID and release groupID numbers.

Some examples of currently available custom particle properties are:

Age decay which models an exponentially decaying particle load, such as bacteria, based on the core "age" particle property

Inside polygon records the polygons, from a given set, containing a particle. This information is used to compute polygon connectivity statistics or to determine events such as larval settlement when a particle drifts over a reef.

Total water depth which represents the sum of tidal elevation and water depth. This property is useful for particles whose behaviour differs in different water depths, such as larvae that only settle in shallow water, even if a cell is not completely dry .

3.3 Manager roles

Instances of field and particle properties store, update, and manage access to individual data values. Higher level operations on all the individual instances are orchestrated by "managers" which automate key processes (see Fig. 2).

Fields group manager role This orchestrates the setup, reading, updating and interpolation of fields, along with setting up the required grid variables, such as nodal locations, triangulation and the adjacency matrix. It also manages the process of finding each particle's current horizontal and vertical cell, and updates the status of dry cells. By default, the fields group manager automatically adds a particle property with the same internal name as the field to the computational pipeline, to be interpolated at each time step. When nesting small fine-scale grids within a larger outer grid, a special fields group manager, creates a Fields group manager for each grid. It associates each particle with the appropriate grid, so that their water velocity and other particle properties can be interpolated from the fields of that grid. At each time step, any particles crossing the open

boundaries of the inner grid are associated with the outer grid. Conversely, any particles on the outer grid that move inside an inner grid are then associated with that grid.

Particle group manager role This manages the release of particles and the updating of all three types of particle properties. It also manages the dynamic memory buffers which hold the individual particle property values, expanding them as needed when particle numbers grow and culling computationally dead particles that are no longer of interest. Additionally, if required, this manager handles the writing of time series of particle trajectories and properties.

3.4 Reader role

The primary function of the reader is to convert hydrodynamic model file variables into standardised internal formats, which may be stored in different files. The reader builds a catalogue of all file variables and which files contain them. It then maps the file variables to standard internal variable names. Field variables are categorised as time varying, vector or 3D. For vector variables, e.g. water velocity, several file variable names can be mapped to a single internal vector variable, allowing vector fields to be treated as a single variable within computations.

To ensure files for time-varying variables are read in the correct order, files are sorted into time order, using the hindcast's time variable. Thus OceanTracker is not reliant on a file naming convention to determine file order. Each variable has its own list of files, which enables the reader to seamlessly accommodate hydrodynamic models where variables are split between files, as is the case for SCHISM version 5 output files and NEMO/GLORYS output files.

The reader loads fields into memory buffers as they are needed. For time-dependent variables it reads multiple time steps into buffers, if the next time step is not already in the buffer. By default, the buffer maintains 24 hydrodynamic model time steps in memory.

If needed, the reader converts the non-nodal values to values at the nodes of the triangles through interpolation. Readers and interpolators based on a hydrodynamic model's native grid, that do not need to do this conversion, could be developed in the future. To facilitate automation, the reader stores all fields in 4D arrays with dimensions (corresponding to time, node, z-depth and vector components).

Currently, OceanTracker supports hydrdynamic model formats SCHISM, FVCOM, DEFLT3D-FM, NEMO/GLORYS and ROMS (Zhang et al., 2016; Lai et al., 2010; Deltares, 2014; Moore et al., 2011). For the unstructured SCHISM and DEFLT3D-FM grids, which can have a mixture of triangular and quad cells, quad cells are divided into triangles.

3.4.1 Interpolator role

The interpolator serves as the link between the hydrodynamic model's fields held by the reader and the corresponding particle properties. The interpolator has two functions , 1) determining each particle's current horizontal and vertical cell, 2) the interpolating of field values at the triangles' vertices to each particle's location, which are then stored as particle properties. In the horizontal, OceanTracker currently uses linear interpolation in unstructured triangular grids from nodal values, utilizing a particles' Barycentric coordinates within their current triangle. Linear interpolation is applied within vertical layers and between

time steps, except for the water velocity within the seabed layer. Here, vertical interpolation is based on a logarithmic layer, ensuring that particles near the seabed experience more realistic horizontal velocities.

The interpolator supports linear interpolation for several vertical grids, Sigma grid (ROMS, Deft-3D FM) and fixed z levels (Deft-3D FM, NEMO/GLORYS), which apply the same vertical grid at all locations. It also supports grids with spatially varying layer thicknesses (SCHISM, FVCOM). This includes SCHISM's LSC vertical grid, where the number of vertical layers also varies spatially.

3.5 Particle release groups role

A "particle release group" introduces new particles into the computational pipeline. Each release group generates new particles at specified locations, which are released at designated times, in specified pulses sizes. Multiple release groups can be added, each with their specific release locations and release schedules. Release group can be:

Point release spawns particle from a set of specified locations and depths. An optional radius setting allows particle releases randomly within a circular area around each point.

Polygon release: Particles are spawned at random locations within a user-defined given 2D polygon.

Grid release: spawns particles from points of a regular grid.

Particles will not be released on land, e.g. where a release polygon overlaps land as in Fig. 1a. By default, particles are not released within cells that are currently dry due to the tide. For all types it is possible to restrict releases to be randomly distributed within a given vertical layer, or to locations with water depth in a given range.

3.6 Velocity modifiers role

Additional bio-physical processes can be added to the water velocity experienced by each particle. These are incorporated into the computational pipeline as "velocity modifiers", which are added to the water velocity for use in the time integration (Fig. 3). An example of an in-built modifier is:

Terminal velocity: The modifier adds the terminal sinking or buoyant velocity to the ambient water velocity, either as a uniform value or a particle specific value drawn from a normal distribution.

3.7 Trajectory modifiers role

"Trajectory modifiers" are bio-physical processes that alter the movement of particles at each time step or their status. Examples of in-built modifiers include:

Settlement: Allows particles to settle within user defined polygons, e.g. larvae to settle on reefs. The trajectory is modified by changing its status to "stationary".

Floating: Sets each particle's vertical position to that of the free-surface height at its current location.

Culling: Sets the status of a random fraction of particles for a given status (e.g. on the bottom) to be dead, allowing them to be removed from subsequent computations and particle statistics.

Splitting: Splits particles in two to simulate reproduction, at a set rate or probability. This can rapidly generate very large numbers of particles, which may need to be contained by a culling mechanism (Steidle and Vennell, 2023).

3.8 On-the-fly particle statistics roles

OceanTracker employs an on-the-fly particle counting approach to produce particle statistics during the computational run. This produces a data output volume that is independent of the total number of released particles. Currently available spatial particle statistics are:

Gridded: : Count particles inside the cells of a regular grid at user-specified time intervals. These are commonly used to generate heat maps (see Fig. 1).

Polygon: Compute the physical connectivity matrix between each release group and user-defined polygonal regions.

Both gridded and polygon-based statistics can be recorded using two approaches:

- Time Series: Counts particles at specified time intervals, producing time-based heat maps or connectivity data.
- Age Bin Series: Categorises particles into age bins within each spatial region, enabling the generation of age-based heat maps or connectivity matrices.

Users can filter which particles are counted, such as restricting statistics to particles within a specific vertical range or to particles resting on the seabed. Multiple on-the-fly statistics components can be combined, allowing, for example, separate particle counts for different depth layers.

Beyond counting particles, average values of particle properties can also be computed within the spatial bins. For instance, users can track average water temperature of particles within a given grid cell or polygon.

3.9 Integrated models

Users may need to combine multiple roles to create higher level functionality. OceanTracker supports this aggregation of roles into a single component, facilitating wider use of that functionality and collaboration. These integrated models only require the parameters essential for executing their higher-level function; the model manages the intricacies of assigning classes to appropriate roles to complete the overall function. Currently, there is one integrated model:

On-the-fly Lagrangian Coherent Structures: These structures identify regions of convergence or divergence within a fluid flow over time. OceanTracker calculates time series of Finite-Time Lyapunov Exponents (FTLE) (Haller, 2015; Harrison and Glatzmaier, 2012), which can be used to derive characteristics of Lagrangian Coherent Structures. The FTLE calculation uses the largest Eigenvalue of the strain tensor after specified lag times. This process involves releasing particles on a regular grid and calculating the distances between adjacent released particles. The user only needs to designate one or more grid locations and the required time lags. The integrated model sets up new regular grid releases at regular time intervals and calculates the FTLE on-the-fly for each grid and lag, eliminating the need to record and post process large volumes of particle trajectories.

4 Computational speed

The primary features contributing to computational speed of OceanTracker are:

Finding cells containing particles: For unstructured grids, significant computational time is spent determining the horizontal and vertical cells containing each particle. OceanTracker uses the particle history to improve the speed of its cell search. Short triangle and vertical walk algorithms improve the search speed by an order of magnitude, as outlined in (Vennell et al., 2021).

Calculate once, use many times: To avoid repetition in performing key computational tasks, a particle's current triangle, barycentric coordinates, vertical cell and fraction of the vertical cell are recorded at each time step. These values are then used repeatedly to interpolate multiple fields to the particle's location. Adopting this, 'calculate once, use many times' methodology significantly improves the efficiency of interpolations (Vennell et al., 2021).

Dynamic particle buffer: Particle proprieties are stored in memory buffers, which are expanded as needed to accommodate newly released particles. System performance and memory utilisation can be optimised by periodically culling dead particles. When more than 20% are flagged dead, particle memory buffers are compacted by removing dead particles, which can significantly improve performance.

Uniform sigma-grid: For 3D SCHSIM and FVCOM models layer thicknesses vary temporally and spatially, making the search for each particle's vertical cell the most time-consuming step. Optionally, for these models, the vertical cell search can be made 5 times faster by vertically interpolating to spatially uniform sigma-layer fractional thicknesses when filling the reader's buffers. With uniform fractional thicknesses, finding the vertical cell is significantly quicker using rounding within a pre-calculated layer map, followed by a single correction step. For large numbers of particles, the time spent re-interpolating to this near-native vertical grid when reading the hydrodynamic model's files, is small compared to the time saved by the faster vertical cell search. Models which use a sigma vertical grid can directly exploit the faster approach to vertical cell search at no additional effort.

Numba: Particle tracking involves complex operations with nested loops, requiring per-particle decisions at each time step. To significantly accelerate these computations—by hundreds of times OceanTracker leverages Numba, a Python extension for Just-In-Time (JIT) compilation (Lam et al., 2015). Numba optimises performance by compiling functions at runtime using a simple function decorator. Numba achieves execution speeds comparable to C or Cython while maintaining Python's ease of use. Once a function is compiled, the optimised code is reused for subsequent calls, eliminating further compilation overhead. However, a drawback of using Numba is that compiling the many required functions adds 20–30 seconds to OceanTracker's startup time. This delay can be minimised by enabling optional caching, which stores the compiled Numba code on disk for reuse between runs.

Parallelisation: OceanTracker spreads most demanding computations across all available physical computer cores, exploiting Numba's multithreading capabilities. Specifically, when iterating over particles OceanTracker uses Numba's parallel for-loop, to automatically distribute particle computations across multiple physical cores. All threads access the same field vari-

able memory; thus each hydrodynamic model time step is only read once during the run. This makes OceanTracker suitable for running on a single computer or single node of a super-computer where cores share the same memory.

4.1 Speed comparisons

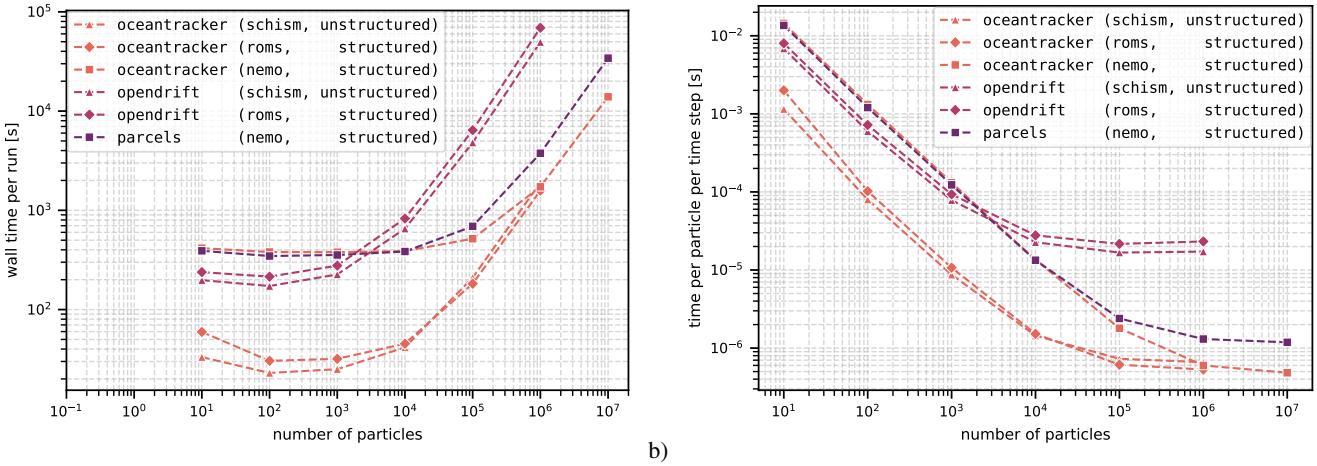
The computational performance of OceanTracker and its closest alternative, OpenDrift (version 1.1.1) (Dagestad et al., 2018) is compared in Fig. 6 for structured and unstructured grids. A similar comparison has been reported in Vennell et al. (2021). Since that publication, the base code has undergone a significant overhaul, from using NumPy and SciPy for critical computational steps,to using Numba. This transition has facilitated the implementation of far more particle-specific decisions processes that enhances the realism of their movement. A speed comparison with Ocean Parcels (version 3.0.6) (Delandmeter and Van Sebille, 2019) is also included in Fig. 6. Both Parcels and OceanTracker can exploit parallelisation using threads OpenDrift does not natively support parallelisation, but can be used to run multiple independent cases, each separately reading the hydrodynamic model, i.e. "dumb parallelisation". Thus, the comparisons presented here show performance using only a single computer core. Parcels currently only supports structured grids, so could only be compared with OceanTracker for this type of grid.

All comparisons use 3D hydro-dynamic models. The first is a high-resolution unstructured mesh estuarine model described in Steidle and Vennell (2023), built using SCHISM (Zhang et al., 2016). This model features 32,000 horizontal nodes and employs terrain-following coordinates for the vertical grid, with up to 20 levels. Its spatial resolution varies from 5m and 1400m. The second model is a regular grid ROMS model presented in López et al. (2020), consisting of about 250,000 nodes and 40 z-grid type vertical levels with a uniform horizontal resolution of 7 km. The third model is large NEMO model of the Baltic Sea (Kärnä et al., 2021), which uses GLORYS for its hydrodynamics. This contains about 600,000 horizontal nodes and 56 z*-grid type vertical levels. All hydrodynamic models had a temporal resolution of 1 h.

OceanTracker and OpenDrift were compared using dispersion, while the comparison to Parcels using the NEMO data set are performed using advection only, due to a lack of a native 3D dispersion method in Parcels. The comparisons were performed with OpenDrift version 1.11.1 and Parcels 3.0.6. All particle trackers used RK4 time integration. The computations covered 10 model days using 5 minute time steps, resulting in a total of 2880 time steps. Particles were released at 30 locations, with total particle numbers varying from 1000 to one million particles per model run. Note, that OpenDrift by default uses a different time step size for its vertical diffusion calculations of 1 minute. We chose not to override this default behavior as OpenDrift's internal timing information shows that the computational cost did not dominate the total runtime (<10%) and is similar to what we expect based on our experience with OceanTracker .

The computational experiments were performed on the desktop computer "imf20-l01", which is equipped with an Intel(R) Core(TM) i5-10500 CPU @ 3.10GHz and 16 GB RAM. The source code that has been used to perform this speed tests is publicly available (see section Code Availability).

Figure 5a) displays the total run time across OceanTracker and OpenDrift (structured and unstructured grids) and Parcels (structured grid). Figure 5b) illustrates the scaling of both models in terms of time per particle per RK4 time step. This demonstrates that OceanTracker can process nearly over a million particles per second per time step on a single desktop core.



a) b)

Figure 5. Computational comparison speed for OceanTracker with OpenDrift for structured (ROMS) and unstructured (SCHISM) grids, and with Parcels for structured (NEMO/GLORYS) grids. a) Compares the total run time measured as wall time. b) shows the normalized computational time time per particle per RK4 time step.

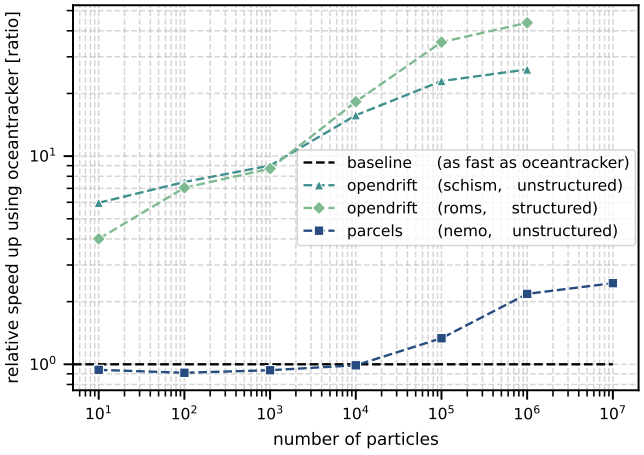


Figure 6. Comparison of the relative speed up from OceanTracker compared to OpenDrift and Parcels for the different data sets and particle sizes. The black baseline represents performance equal to OceanTracker where ratio equals 1

For the SCHISM and ROMS data set, model setup times are the main contributor to OceanTracker's run time for particle counts up to 10000 particles. For these small numbers, OpenDrift's minimum run time was approximately 2 minutes, while OceanTracker's completed these runs in 30 seconds. For particle counts exceeding 10,000 setup times become negligible for both models. In 1 million particle simulations OceanTracker is up to 35 times faster than OpenDrift as shown in figure 6, resulting in total run times of 15 hours for OpenDrift and half an hour for OceanTracker.

OceanTracker treats structured grids as unstructured grids by triangulating them, whereas OpenDrift uses a native reader for the structured ROMS grids. Surprisingly, the native ROMs reader of OpenDrift does not significantly enhance its performance with structured grids.

For the NEMO/GLORYS case, which uses a large data set with over 3 million spatial nodes, model setup times and reading the data remains the dominant cost for cases with particle counts of below 100,000. For the larger particle counts over 1,000,000 ocean tracker performs approximately twice as fast as Parcels, making it the fastest general-purpose particle tracker for both structured and unstructured grids to our knowledge.

4.2 Multi-processing scaling.

Fig. 7 shows the multi-processing scaling for two machines (note logarithmic x-axis). The first is the high-performance computer "Levante" based at the DKRZ (German Climate Computing Center). It is equipped with two AMD 7763 CPUs @ 2.45-3.5 GHz, containing 128 physical cores in total and 256 GB of RAM. The second is "imf20-l01", a desktop computer which was also used for the single processor speed tests. Scaling quickly becomes sub-linear for threads counts greater than two, presumably due to bandwidth limitations between the CPU memory caches and the much larger Random Access Memory (RAM) when accessing particle or hindcast data. This bottleneck becomes particularly limiting on Levante, where there are no significant speed-up gains beyond 32 threads. This data access bottleneck suggested that restructuring the code to slice up the hydrodynamic hindcast into sub-domains distributed across threads or redesigning how particle data is ordered in memory (Kehl et al., 2023) could improve how speed scales with the number of processors.

5 Discussion

Vennell et al. (2021) found that OceanTracker performs significantly faster than OpenDrift when working with unstructured grids. Here,Fig. 5 demonstrates that OceanTracker maintains a speed advantage on structured grids, despite treating them as unstructured. This result is somewhat surprising since, in structured grids, cell identification is typically much faster, as it only requires simple coordinate rounding. The observed performance difference suggests that the advantage is not due to differences in cell-finding algorithms, but rather other aspects of how the two models handle computations. Note, that for the speed comparisons we focused excursively on computational time as our metric. Because all models internally show significant difference in their implementations it was not possible to configure them to perform the exact same computation. Hence, for future comparisons it would be desirable to compare computational cost while normalizing for model accuracy.

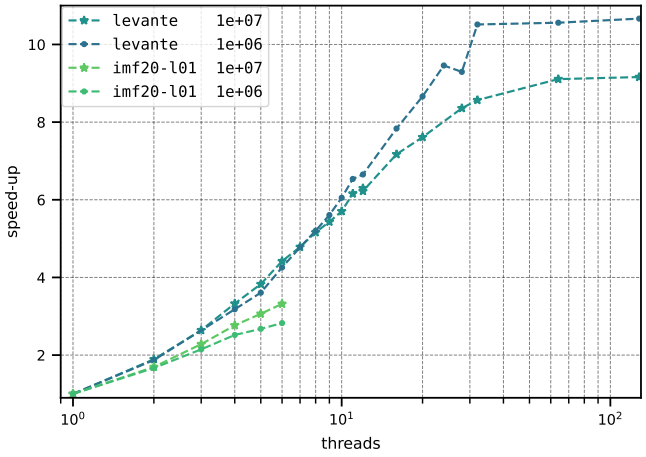


Figure 7. Scaling of computation speed when run in parallel mode. The vertical axis is the speed-up relative to single thread runs. Comparisons were performed on "imf20-l01", a standard office desktop machine, and on node of Levante, a high-performance computer, with 128 CPU cores for particle counts of 1 and 10 million (1e6 and 1e7). Note the logarithmic scaling on the x-axis

For unstructured grids, the original OceanTracker was over two orders of magnitude faster than OpenDrift (Vennell et al., 2021). The latest version of OceanTracker is an order of magnitude slower than its predecessor. However, this trade-off comes with significantly improved physics Sec. 2.2. The enhancements require more computationally intensive, particle-by-particle decisions regarding movement. While these decisions introduce slower branching code, they result in more realistic particle behaviour and improved simulation accuracy.

Implementing an adaptable computational pipeline built from user-added components can sometimes conflict with optimizing code for speed. For instance, the fastest approach would be to consolidate all particle operations into a single loop, where all necessary computations for each particle are performed in one pass. However, this rigid structure would limit flexibility, making it difficult to dynamically add or modify components within the pipeline. This approach increases efficiency, as the data required for each particle is likely to be in the faster chip memory cache if used more than once. It also avoids the need to create, read or write arrays in main memory of intermediate results. However, it is much harder to adapt a single loop over all particles when there are many operations with different variants. OceanTracker takes a middle ground, by breaking up the computational pipeline into a series of modular operations on all particles assembled into a sequence of interchangeable operations within the roles illustrated in Fig. 3. This modular approach compromises speed but significantly enhances adaptability.

To validate OceanTracker's accuracy, we performed a standard test using a circular flow with a known period, as recommended by (Van Sebille et al., 2018). A 2D synthetic eddy test was conducted, featuring a peak flow of 1 m/s at a 10 km radius with a 12-hour period. With 15-minute RK4 time steps, particles released 10 km from the centre deviated by less than 0.3 m from their initial radius after 10 days, demonstrating high accuracy.

6 Summary

OceanTracker provides a comprehensive ocean particle tracking framework compatible with both structured and unstructured grids. Its speed allows users to scale to larger numbers of particles on modest computer hardware within acceptable run times. This capability facilitates enhanced particle statistics and broader exploration of variations in particle behaviours. Integrating the calculation of particle statistics directly within computational runs significantly reduces the time required to derive needed statistics. In addition, on-the-fly-statistics produce more manageable data volumes, as their size does not depend on the number of particles released. The ability to add multiple release groups and statistics also reduces user efforts, by enabling multiple outcomes within the same computational run. Adaptability in building a computational pipeline is enabled by a modular approach to roles within the computational pipeline .

OceanTracker's speed primarily stems from its algorithmic efficiency in locating each particle's current triangle and interpolating field values at its position. This efficiency allows it to significantly outperform current versions of OpenDrift and Ocean Parcels on a single computer core. When using more than 30 cores, OceanTracker's performance increases by an additional factor, further surpassing models that do not leverage multithreading or parallel computation.

For small particle counts, reading the hydrodynamic model data will dominate the run time, making the actual trajectory calculations negligibly short. For the Laptop I and Desktop II benchmarks in Table 1, the tipping point between time spent reading and time spent calculating trajectories is around 100,000 and 1,000,000 particles respectively. Trajectory computation for significantly smaller particle numbers is almost free. This tipping point will be higher for hindcasts with more nodes or a larger particle tracking time step than those used for the examples in Table 1.

Future improvements to enhance OceanTracker's speed might include greater use of "Single Instruction Multiple Data instructions" on conventional CPUs (SIMD). Additional optimisation may be possible by adapting expensive computational kernels to take advantage of GPUs. Speed could may also be improved by chunking or restructuring the data in memory (Kehl et al., 2023). An asynchronous reader could fill field buffers in a separate parallel process and share memory with the particle computational threads. This could make reading the hydrodynamic model almost "free" in terms of overall run times for numbers above the tipping point, or the trajectory computations free for numbers below this point.

Code availability. OceanTrackers source code is available at https://github.com/.oceantracker/oceantracker/, Demonstrations and user guide at https://oceantracker.github.io/oceantracker/. The code is released under the MIT license. The code to perform the speed tests and their output are available at https://doi.org/10.25592/uhhfdm.14172

Author contributions. Vennell and Steidle are core code developers and primarily responsible for writing this paper. Smeaton, Chaput and Knight contributed to development of specific components of the code, extensive code testing and the structure of the paper.

Competing interests. The contact author has declared that neither of the authors has any competing interests.

Acknowledgements. This study was supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) within Research Training Group 2530: "Biota-mediated effects on carbon cycling in es- tuaries" (project number 407270017). Also, New Zealand Ministry of Business, Innovation and Employment science grants- Data Science for Aquaculture (RTVU1914), Reimagining Aquaculture (C11X1903) and the Sustainable Seas National Science Challenge.

Appendix A: Code for Fig. 1

```
# code to run particle tracking for Fig 1

from oceantracker.main import OceanTracker # load
ot= OceanTracker() # create an instance to build parameter dictionary= ot.params

# add settings
ot.settings(output_file_base='OTpaper_exmaple_A',  root_output_dir='output', time_step= 600.)

 # add reader to acces the hyrdo-model
ot.add_class('reader',  input_dir=r'oceantracker\demos\demo_hindcast',
                 file_mask=  'demoHindcastSchism*.nc')  # file mask to search for
# add a point release
ot.add_class('release_groups',name = 'my_point_release', class_name='PointRelease',
        points= [[1595000, 5482600, -2], [1594000, 5484200, -2]], # (x,y,z) of release points
        release_interval= 600, pulse_size= 5000)
# add polygon release at random depths between two z values
ot.add_class('release_groups',  name = 'my_polygon_release',  class_name='PolygonRelease',
        points=[[1597682., 5486972], [1598604, 5487275], [1598886, 5486464],
                [1597917., 5484000], [1597300, 5484000], [1597682, 5486972]],
        release_interval= 600,  pulse_size= 50, z_min= -2., z_max = 0.5)
# add grid releasing at random depths between two z values
ot.add_class('release_groups',  name = 'my_grid_release', class_name='GridRelease',
        grid_center=[1592000, 5489200], grid_span=[500, 1000], grid_size=[3, 4],
        release_interval= 1800,  pulse_size= 2, z_min= -2, z_max = -0.5)
# add a decaying particle property,# with exponential decay based on age
ot.add_class('particle_properties', name ='a_pollutant', class_name='AgeDecay',
            initial_value= 1000, decay_time_scale = 7200.) # exponential decay time scale 2hours
# add a gridded particle statistic to use as heat map
ot.add_class('particle_statistics', name = 'my_heatmap', class_name= 'GriddedStats2D_timeBased',
        grid_size=[120, 121],  release_group_centered_grids = True, update_interval = 600,
        particle_property_list = ['a_pollutant'],  status_min ='moving', z_min =-10.)
ot.add_class('resuspension', critical_friction_velocity=0.01) #set value for particle resupension
```

```
33:
34: # run OT and return file name useful in plotting
35: case_info_file= ot.run()
```

References

Ainsworth, C., Chassignet, E. P., French-McCay, D., Beegle-Krause, C. J., Berenshtein, I., Englehardt, J., Fiddaman, T., Huang, H., Huettel, M., Justic, D., et al.: Ten years of modeling the Deepwater Horizon oil spill, Environmental Modelling & Software, 142, 105 070, 2021.

Atalah, J., South, P. M., Briscoe, D. K., and Vennell, R.: Inferring parental areas of juvenile mussels using hydrodynamic modelling, Aquaculture, 555, 738 227, 2022.

Chaput, R., Sochala, P., Miron, P., Kourafalou, V. H., and Iskandarani, M.: Quantitative uncertainty estimation in biophysical models of fish larval connectivity in the Florida Keys, ICES journal of marine science, 79, 609–632, 2022.

Dagestad, K.-F., Röhrs, J., Breivik, Ø., and Ådlandsvik, B.: OpenDrift v1. 0: a generic framework for trajectory modelling, Geoscience Model Development, 2018.

Delandmeter, P. and Van Sebille, E.: The Parcels v2. 0 Lagrangian framework: new field interpolation schemes, Geoscientific Model Development, 12, 3571–3584, 2019.

Deltares, D.: Delft3D-FLOW simulation of multi-dimensional hydrodynamic flows and transport phenomena including sediments, user manual, Deltares Delft, The Netherlands, 2014.

Garnier, S., Murray, R. O., Gillibrand, P. A., Gallego, A., Robins, P., and Moriarty, M.: Particle tracking modelling in coastal marine environments: Recommended practices and performance limitations, Ecological Modelling, 501, 110 999, 2025.

Haller, G.: Lagrangian coherent structures, Annual review of fluid mechanics, 47, 137–162, 2015.

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., and Oliphant, T. E.: Array programming with NumPy, Nature, 585, 357–362, https://doi.org/10.1038/s41586-020-2649-2, 2020.

Harrison, C. S. and Glatzmaier, G. A.: Lagrangian coherent structures in the California Current System–sensitivities and limitations, Geophysical & Astrophysical Fluid Dynamics, 106, 22–44, 2012.

Hunter, E. J., Fuchs, H. L., Wilkin, J. L., Gerbi, G. P., Chant, R. J., and Garwood, J. C.: ROMSPath v1. 0: offline particle tracking for the Regional Ocean Modeling System (ROMS), Geoscientific Model Development, 15, 4297–4311, 2022.

Kehl, C., Nooteboom, P. D., Kaandorp, M. L. A., and van Sebille, E.: Efficiently simulating Lagrangian particles in large-scale ocean flows — Data structures and their impact on geophysical applications, Computers & Geosciences, 175, 105 322, https://doi.org/10.1016/j.cageo.2023.105322, 2023.

Knight, B., Treml, E., Waddington, Z., Vennell, R., and Hutson, K.: Hindcasting farmed salmon mortality to improve future health and production outcomes, Journal of Fish Diseases, in press, 2024.

Knight, B. R., Zyngfogel, R., Forrest, B., et al.: PartTracker-a fate analysis tool for marine particles, Coasts and Ports 2009: In a Dynamic Environment, p. 186, 2009.

Kärnä, T., Ljungemyr, P., Falahat, S., Ringgaard, I., Axell, L., Korabel, V., Murawski, J., Maljutenko, I., Lindenthal, A., Jandt-Scheelke, S., Verjovkina, S., Lorkowski, I., Lagemaa, P., She, J., Tuomi, L., Nord, A., and Huess, V.: Nemo-Nordic 2.0: operational marine forecast model for the Baltic Sea, Geoscientific Model Development, 14, 5731–5749, https://doi.org/10.5194/gmd-14-5731-2021, publisher: Copernicus GmbH, 2021.

Lai, Z., Chen, C., Cowles, G. W., and Beardsley, R. C.: A nonhydrostatic version of FVCOM: 1. Validation experiments, Journal of Geophysical Research: Oceans, 115, 2010.

Lam, S. K., Pitrou, A., and Seibert, S.: Numba: A llvm-based python jit compiler, in: Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC, pp. 1–6, 2015.

López, A. G., Wilkin, J. L., and Levin, J. C.: Doppio-a ROMS (v3.6)-based circulation model for the Mid-Atlantic Bight and Gulf of Maine: Configuration and comparison to integrated coastal observing network observations, Geoscientific Model Development, 13, 3709–3729, https://doi.org/10.5194/gmd-13-3709-2020, 2020.

Lucas, L. V. and Deleersnijder, E.: Timescale methods for simplifying, understanding and modeling biophysical and water quality processes in coastal aquatic ecosystems: A review, Water, 12, 2717, 2020.

Lynch, D. R., Greenberg, D. A., Bilgili, A., McGillicuddy Jr, D. J., Manning, J. P., and Aretxabaleta, A. L.: Particles in the coastal ocean: Theory and applications, Cambridge University Press, 2014.

Moore, A. M., Arango, H. G., Broquet, G., Powell, B. S., Weaver, A. T., and Zavala-Garay, J.: The Regional Ocean Modeling System (ROMS) 4-dimensional variational data assimilation systems: Part I–System overview and formulation, Progress in Oceanography, 91, 34–49, 2011.

Pastor Rollan, A., Sherman, C. D., Ellis, M. R., Tuohey, K., Vennell, R., Foster-Thorpe, C., and Treml, E. A.: Current Trends in Biophysical Modeling of eDNA Dynamics for the Detection of Marine Species, Environmental DNA, 6, e70 021, 2024.

Smeaton, M., Atalah1, J., and Lauren M. Fletcher1, Ian C. Davidson1, L. F. K. H. E. T. R. V. E. G. D. K.: Hydrodynamic modelling of ballast water exchange for domestic voyage routes, Journal of Fish Diseases, in prep.

Steidle, L. and Vennell, R.: Phytoplankton Retention Mechanisms in Estuaries: A Case Study of the Elbe Estuary, EGUsphere, 2023, 1–16, 2023.

Thygesen, U. H.: How to reverse time in stochastic particle tracking models, Journal of Marine Systems, 88, 159–168, 2011.

Van Sebille, E., Griffies, S. M., Abernathey, R., Adams, T. P., Berloff, P., Biastoch, A., Blanke, B., Chassignet, E. P., Cheng, Y., Cotter, C. J., et al.: Lagrangian ocean analysis: Fundamentals and practices, Ocean Modelling, 121, 49–75, 2018.

Vennell, R., Unwin, H., and Scheel, M.: Where's Our Plastic Going?, https://ocean-plastic-simulator.cawthron.org.nz/, 2019.

Vennell, R., Scheel, M., Weppe, S., Knight, B., and Smeaton, M.: Fast *L*agrangian particle tracking in unstructured ocean model grids, Ocean Dynamics, 71, 423–437, 2021.

Visser, A.: Using random walk models to simulate the vertical distribution of particles in a turbulent water column, Oceanographic Literature Review, 6, 1086, 1998.

Wolfram, P. J., Ringler, T. D., Maltrud, M. E., Jacobsen, D. W., and Petersen, M. R.: Diagnosing isopycnal diffusivity in an eddying, idealized midlatitude ocean basin via Lagrangian, in Situ, Global, High-Performance Particle Tracking (LIGHT), Journal of Physical Oceanography, 45, 2114–2133, 2015.

Zhang, Y. J., Ye, F., Stanev, E. V., and Grashorn, S.: Seamless cross-scale modeling with SCHISM, Ocean Modelling, 102, 64–81, 2016.