Ocean Dynamics
https://doi.org/10.1007/s10236-020-01436-7

Fast lagrangian particle tracking in unstructured ocean model grids

Ross Vennell[1] · Max Scheel[1] · Simon Weppe[2] · Ben Knight[1] · Malcolm Smeaton[1]

Received: 6 May 2020 / Accepted: 23 December 2020
© Springer-Verlag GmbH Germany, part of Springer Nature 2021

Abstract
Lagrangian particle tracking, based on currents derived from hydrodynamic models, is an important tool in quantifying bio-physical transports in the ocean. Particle tracking in the unstructured grids typically used in coastal regions is computationally slow, limiting the number of particles and ranges of behaviours that can be modeled. Techniques used in a new offline particle tracker "OceanTracker" are shown to be two orders of magnitude faster than those used in an existing ocean particle tracker for unstructured grids when run on a single computer core. More significantly, its computational speed can exceed that achieved when particle tracking on a regular grid. The techniques for unstructured grids make it possible to routinely calculate the trajectories of millions of particles. This large number of particles allows much better estimates of dispersion and transport statistics, particularly when the probability of connection is low but the consequences are significant, e.g. the spread of invasive species. It also enables wider exploration of parameter sensitivity and particles' bio-physical behaviours to provide more robust results. The speed increases result largely from exploiting history and reuse within the spatial interpolation of the hydrodynamic model's output. Using multiple computer cores further increased the speed to track a given number of particles.

Keywords Ocean transport · Lagrangian · Particle tracking · Biophysical

1 Introduction

Particle tracking is central to answering many scientific and practical problems about bio-physical transport in the ocean. Applications might involve tracking the movement of larvae or the spread of pollution (Lynch et al. 2014; Van Sebille et al. 2018). The number of physical drifters deployed in the ocean is currently limited by budget and logistics. Tracking large numbers of virtual particles advected by currents obtained from hydrodynamic ocean models is commonly used to quantify connectivity between regions. Tracking virtual particles in structured model grids is relatively easy to code and computationally fast. However, many coastal ocean models use unstructured grids of triangles, in order to make it computationally feasible to model at high spatial resolution in the near-shore areas of interest, while also using much coarser resolution offshore, Fig. 1. Particle tracking in unstructured grids is computationally much slower than in structured grids. Here we present a particle tracker that is much faster at calculating particle trajectories when using unstructured model grids.

The aim of this work is to make it routinely possible to calculate the trajectories of millions of virtual particles in unstructured grids using only desktop computer hardware, and to track even greater numbers on high performance computers (HPC). The techniques presented in this work make a significant step towards a major challenge in particle tracking: modeling billions of particles (Van Sebille et al. 2018), offline in unstructured grids. These large numbers of particles enable much better estimates of physical properties like dispersion (Petton et al. 2020) and improved statistics of bio-physical connectivity to be calculated (Treml et al. 2015; Swearer et al. 2019), particularly when the probability of connection is low but the consequences are significant, e.g. the spread of marine diseases or invasive species. Where particles have biological behaviours which affect how individuals move, e.g. larvae settling when flows fall below a critical value, large numbers allow many more variations of this behaviour to be investigated. Large numbers also allow behaviours to be assigned values from probability distributions, e.g. a distribution of sediment particle fall velocities to give better probabilistic distributions as

Responsible Editor: Eric Deleersnijder

✉ Ross Vennell
ross.vennell@cawthron.org.nz

1 Cawthron Institute, Nelson, New Zealand

2 MetOcean Solutions, Raglan, New Zealand

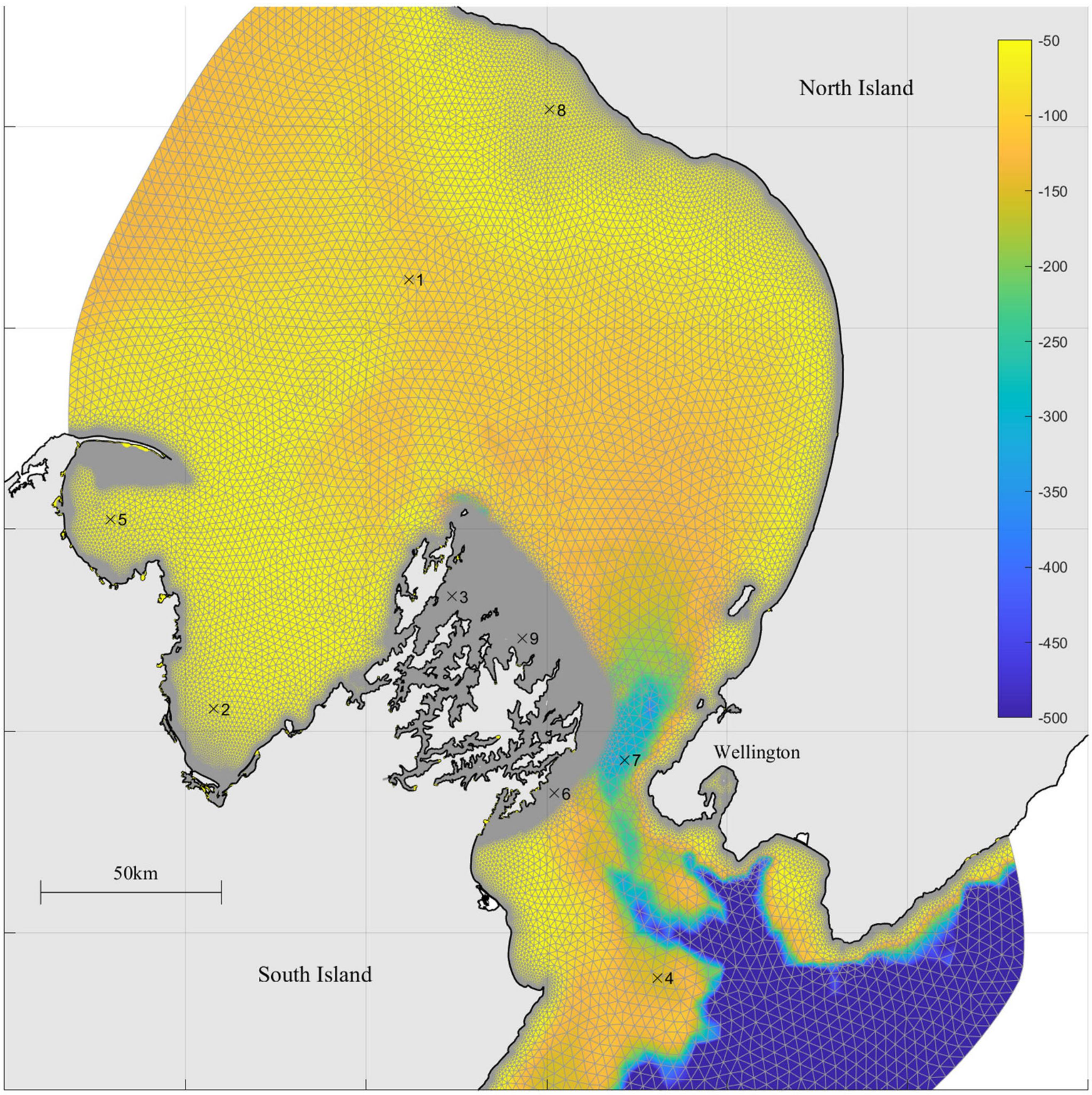


Fig. 1 Unstructured grid of Cook Strait, New Zealand, one of the example ocean model grids used in this work. Resolution for the 149,000 node grid ranges between 4500 m offshore and 36 m nearshore. Colour shows water depth in metres. The 9 example particle release locations used for this work are shown by the "x"

answers to questions about ocean transport. In addition, releasing large numbers makes it much easier to carry out more experiments to test the sensitivity of results to parameter values, a key part of modeling research. The techniques presented here are developed within a new particle tracking code OceanTracker (OT), but they could also be used to speed up existing codes.

Numerically, the time integration required to calculate a particle's trajectory is both simple and fast. The time-consuming part in unstructured ocean model grids is spatial interpolation of the hydrodynamic model's output. This interpolation gives the water velocity and other values required at each particle's current location. Thus, computationally particle tracking in unstructured grids is mainly a spatial interpolation problem. Here we outline a number of approaches that can significantly increase the speed of spatial interpolation by exploiting particle history and reuse of interpolation weights to increase computational speed.

The recent review of ocean Lagrangian analysis (Van Sebille et al. 2018) lists 11 Lagrangian codes, e.g. Delandmeter and Van Sebille (2019). Of these all but one are for regular Arakawa-type grids. The only unstructured Lagrangian code listed, Light in MPAS-O (Wolfram et al. 2015b; Wolfram 2015a), must be run within the hydrodynamic model code so cannot be used offline. Thus, any additional particle tracking incurs the significant computational cost of re-running the hydrodynamic model.

A search for offline ocean particle tracking codes for unstructured grids revealed very few options. Of the five found, three are open-source (OpenDrift (Dagestad et al. 2018), PartTracker (Knight et al. 2009), SCHISM Particle Tracking (Zhang et al. 2016, 2020)). Two were commercial codes with limited details on their methodology (DHI PT module (DHI 2020), SMS-PTM Surface-water Modeling System 2020). To interpret the speed comparisons made in this paper, we needed to know details of the codes' interpolation techniques; however, these details were difficult to access. Of the open source codes, we chose OpenDrift (OD) to make the speed comparisons in this work as it was easy to see what interpolation techniques were used. These techniques use Python's fast C code based module SciPy. PartTracker which is written in Matlab, runs much slower than the other codes and uses opaque internal routines for interpolation. In addition to a comparison with OD, we also compare OceanTracker's speed using unstructured and structured grids on a synthetic data-set.

This short, preliminary results paper presents an initial step to create a fast flexible framework for particle tracking in structured and unstructured grids, which can easily be customized for bio-physical particle behaviour using high level code. See the Appendix for a description of the framework. The following sections test the speed of OceanTracker for a wide range of particle numbers, hydrodynamic model grid sizes, numbers of interpolated fields, and numbers of computer cores. These tests are carried out using both synthetic data and a coastal ocean grid for Cook Strait New Zealand, Fig. 1 (Heath 1978; Vennell 1998).

1.1 Particle tracking and interpolation

Ocean particle tracking is often done offline after running an ocean hydrodynamic model to create a hindcast data-set of velocities and other required variables. Lagrangian particle trackers calculate trajectories by numerically integrating a particle's velocity found by spatially interpolating the hydrodynamic model's velocity to each particle's location.

Spatial interpolation has three main steps: (1) find the grid cell the particle sits within, (2) calculate the interpolation weights required at the particle's location within the cell, and (3) apply these weights to the velocity values at the surrounding cell's nodes to give the velocity at the particle location. Significant speed gains are possible by explicitly separating these steps to avoid repetition.

In regular grids, step (1) is easily done by rounding. However, in unstructured grids, finding the triangular cell containing each particle can account for most of computation time, as demonstrated on the left of Table 1, which is discussed in detail below.

Table 1 Particle tracking run times in seconds for one hydrodynamic model hindcast day

	SciPy-2D		SciPy-3D		OT-2D		OT-3D	
	90k	250k	90k	250k	90k	250k	90k	250k
Total run time	$2190_{100\%}$	$4124_{100\%}$	$5842_{100\%}$	$9952_{100\%}$	$36_{100\%}$	$49_{100\%}$	$74_{100\%}$	$104_{100\%}$
Interpolation total	$2179_{99\%}$	$4095_{99\%}$	$5824_{99\%}$	$9892_{99\%}$	$29_{79\%}$	$40_{76\%}$	$58_{73\%}$	$83_{74\%}$
Find triangle and weights	*$2168_{99\%}$	*$4064_{99\%}$	*$4696_{80\%}$	$7958_{80\%}$	$22_{61\%}$	$31_{63\%}$	$23_{31\%}$	$37_{36\%}$
Apply weights	*$6_{0\%}$	*$8_{0\%}$	*$22_{0\%}$	*$28_{0\%}$	$6_{17\%}$	$8_{15\%}$	$22_{23\%}$	$28_{18\%}$
Find depth cell	-	-	$1105_{19\%}$	$1899_{19\%}$	-	-	$9_{12\%}$	$11_{11\%}$
Read hindcast	$1_{0\%}$	$1_{0\%}$	$3_{0\%}$	$8_{0\%}$	$1_{3\%}$	$1_{2\%}$	$2_{3\%}$	$7_{7\%}$
Other, e.g. integration	$10_{0\%}$	$28_{1\%}$	$15_{0\%}$	$52_{1\%}$	$6_{17\%}$	$8_{16\%}$	$14_{19\%}$	$14_{13\%}$

Particle tracking was done within a synthetic 2D or 3D eddy on two model grid resolutions at 5-min time steps, i.e. 288 RK4 steps per hindcast day. *Values are inferred from total interpolation time and likely similar times to apply weights. "Other" includes initialization, bookkeeping, and Runge-Kutta integration. Values given are for 10^5 particles run on a single computer core. Note OT's weak sensitivity of interpolation times to the size of the grid

In triangular grids, there are a number of techniques used to search for the triangle containing each particle. For example, KD-tree methods use a pre-calculated tree. KD-tree methods are a faster way to find the containing triangle when you cannot make a good guess at which triangle contains the particle; however, using triangle walk methods can be made very fast by exploiting particle history to make a good guess.

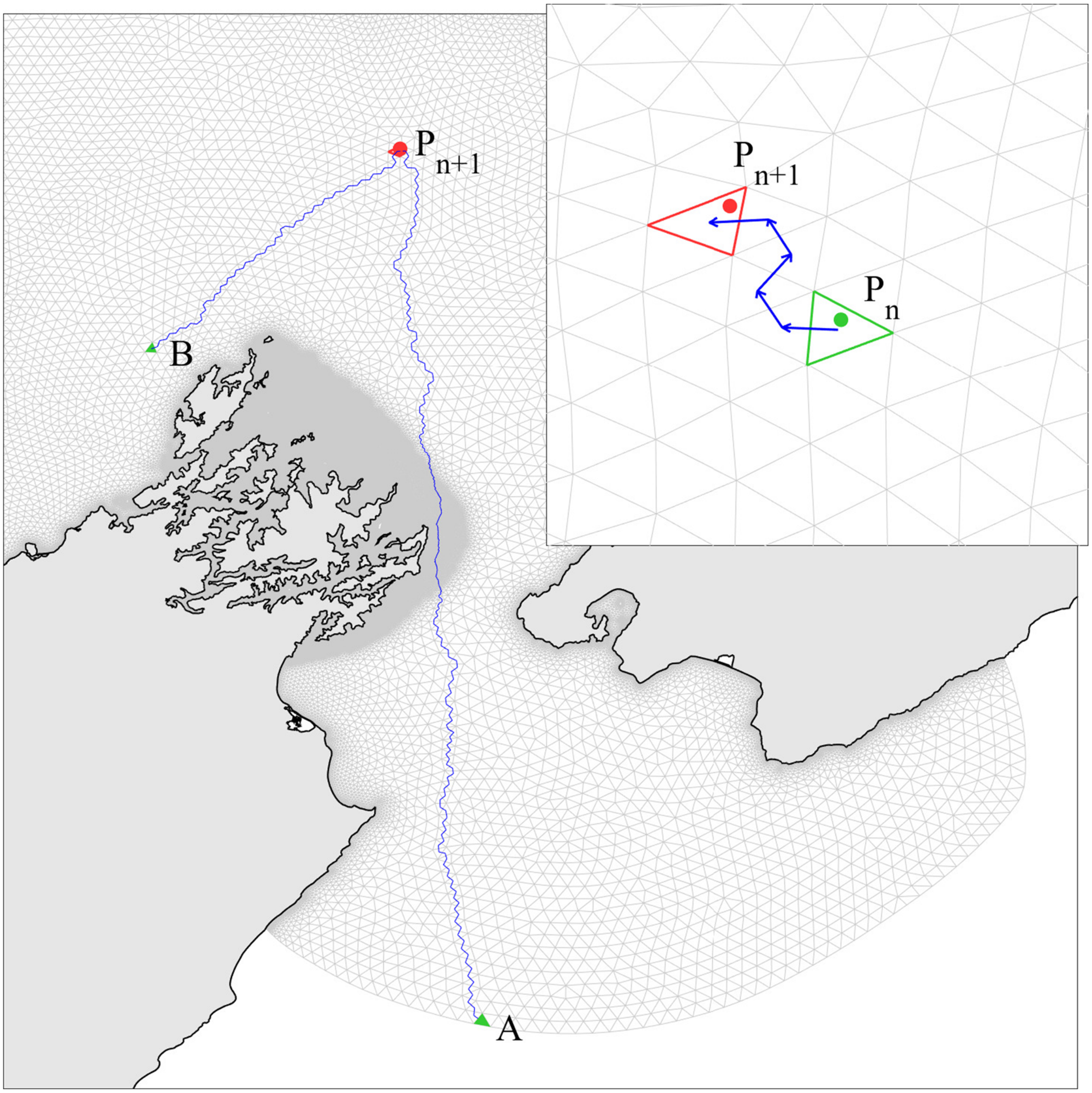


Fig. 2 Illustration of the advantage of the short triangle walk (STW) in searching for the triangle containing the particle at the next time step P_{n+1}, given by red dot. Walks begin with an initial guess of the triangle. A naive long triangle walk (LTW) might always start at the same green triangle, e.g. “A”. A semi-naive LTW, as used within OD, uses the triangle containing the current position of last particle searched for, e.g. “B”. This previous particle is unlikely to be close to P_{n+1}. Inset shows the STW, which starts at a much closer triangle, that containing the particle at the current time step P_n, the green dot. This results in a much lower computational effort in finding the triangle containing P_{n+1}

2 OceanTracker—computational

The structure of the Lagrangian code OceanTracker (OT) is outlined in the Appendix [1]. OceanTracker uses a range of techniques to speed up particle tracking, outlined below.

2.1 Long triangle walk versus short triangle walk

Triangle walk methods start with an initial guess of which triangle a particle is in, green triangles in Fig. 2. If the particle is not in this triangle, the method chooses an adjacent triangle which is a step closer to the particle's new location, P_{n+1}. It continues this walk across the grid until the particle lies within the current triangle. However, computation times will strongly depend on how close the initial triangle is to the required one containing P_{n+1}.

Naive triangle walk algorithms are those which make a poor initial guess of the triangle, requiring a long triangle walk (LTW) through many triangles to find the one containing a particle. A naive starting point would be to always start the walk from the first triangle in the grid, e.g. 370 steps from A to P_{n+1} in Fig. 2. The LTW SciPy interpolator is used by Lagrangian particle tracking code OpenDrift for unstructured grids (Dagestad et al. 2018). When repeatably used, SciPy's LTW retains the triangle found for the previous particle as an initial guess for the next particle. For a single pulse of particles, which initially lie close together, using any other particle's triangle as a guess for the starting triangle for another particle will produce a relatively short walk, 56 steps from B to P_{n+1} in Fig. 2. But for particles released from many locations and different times, this semi-naive guess will result in a poor initial starting point, which will get worse as particle locations diverge over time. A LTW may fail if a coastline lies between the initial and required triangle, or if it takes too many steps. This requires a fall back search method, e.g. SciPy's LTW falls back on a brute force search.

A particle's history can easily provide an excellent guess for the start of a triangle walk. The triangle containing P_n at the current time step is likely very close to the triangle containing its location P_{n+1} at the next time step, Fig. 2. In many cases, P_{n+1} already lies within the triangle containing P_n, or the triangle immediately adjacent to it, resulting in a very short walk. For the first time step, the STW also requires a slow search method to find the triangle containing the particle's release location, P_0. Here, a KD-tree search is used.

[1]An online ocean plastic public education tool which uses OceanTracker's computational speed to calculate trajectories within user attention spans is at https://www.oceanplasticsimulator.nz.

We show here that exploiting this history to enable a short triangle walk (STW) can reduce particle tracking computations by orders of magnitude, when compared to LTW SciPy. We are not aware of another unstructured grid ocean particle tracker that exploits the computational advantage of a STW. The STW algorithm used here is a modified version of the SciPy's Delaunay triangulation code. A future "native" grid version will improve the speed advantage, while also enabling better application of lateral boundary conditions, when particles fall within triangles bordering the coast or an open boundary. A STW starts at the physically sensible current location of a particle; thus should a coastline be encountered on the walk, it triggers the application of a lateral boundary condition.

2.2 Reuse of horizontal interpolation weights

A second computational advantage exploited here is reuse of previously calculated interpolation weights. When performing step (1) of interpolation, finding the triangle is expensive, but separating out step (3) can speed computations. This allows the weights from (2) to be reused to interpolate other variables from their nodal values at a particle's current location. Here we use linear horizontal interpolation within a triangle. For this case, the interpolation weights are each particle's barycentric coordinates within the triangle containing it. Barycentric coordinates are the areas of the three sub-triangles formed by a point within a triangle and its nodes, expressed as a fraction of the triangle's area (Lynch et al. 2014). Barycentric coordinates can also be used to determine whether a particle lies within the current triangle in the triangle walk algorithm.

The first place to implement reuse of already calculated values is in spatially interpolating the velocity at each sub-step of the Runge-Kutta 4 (RK4) method used for time integration. At each sub-step, linear temporal interpolation requires two spatial interpolations of the velocity field at the hindcast time steps immediately before and after the time at which the algorithm needs to calculate the particle velocity. These two interpolations occur at the same spatial location, so that the weights calculated for the first time step can be applied to the nodal values of the second time step. Reusing the weights eliminates at least one triangle search per RK4 sub-step.

2.3 3D short vertical walk

Along with finding the horizontal triangle a particle lies within, 3D particle tracking must determine which depth cell or layer each particle is in, in order to vertically interpolate the hindcast's fields at a particle's 3D location.

Many hydrodynamic models use layer thicknesses that vary across the model domain to provide similar vertical resolution in both the deep and shallow regions made up of triangular prisms (Lynch et al. 2014). For example, SCHISM has layers at fractions of the distance between the free-surface and the bottom as a hybrid “s–z” vertical coordinate (Zhang et al. 2016). This requirement for interpolation at each RK4 sub-step to define the spatial and time dependent vertical grid adds to both computational time and complexity. The current version of OpenDrift (ver. 1.2.0) (Dagestad et al. 2018) avoids these issues by re-gridding the hindcast to space and time invariant depth levels. Here, we preserve the hydrodynamic model’s vertical resolution by treating the layer depths, given at the triangles nodes, as another 2D vector field that must be interpolated to find the layer containing each particle. This approach can flexibly deal with hindcasts from ocean models using the σ and ρ type vertical coordinate systems, which are described in Song and Hou (2006), and other s-level type coordinates (Delandmeter and Van Sebille 2019).

To find a particle’s layer, firstly, the hindcast’s M layer depths are treated as a 2D M-dimensional vector field that is interpolated from its nodal values to create a vector of all the layer depths at each particle’s location. Secondly, these vectors are searched for the layer containing each particle. This approach is made faster here by reusing the horizontal interpolation weights for each layer. However, it still requires M weight reuses before the layer vectors can be searched.

The search time is significantly further reduced by implementing a short vertical walk (SVW) to find the layer containing each particle. SVW uses the vertical layer containing the particle at the previous time step as an initial guess for the current layer. It tests whether the current particle depth is above or below this layer, then steps up or down the layers until it finds the layer containing the particle. Time is saved by doing horizontal interpolation by weight reuse, only when required at each vertical step in the search. Many particles will likely be in the same layer after the time step; these particles only require 2 weight reuses per particle to verify the new layer. SVW typically reduced the computational time required to find the depth layer by a factor of 4 in the 12 layer hindcast (i.e. M=12) examples presented in Section 4.

3 Synthetic eddy data

To enable comparison of particle tracking speeds for a range of particle numbers, grid sizes, and additional fields, a synthetic data-set was created. The hindcast data used was the velocity of a 3D circular eddy with a linearly increasing tangential velocity, which varies over a semi-diurnal tidal cycle. The eddy has a peak velocity of 1 m/s at 10 km from its centre. The hindcast covers a 30-km rectangular domain, with varying numbers of grid points. The regular domain was treated as a unstructured grid for the particle tracking tests presented here (except in Section 4.3, which compares performance for regular and unstructured grids). The water depth was 30 m over most of the domain but included a 5-km wide, 15-m-deep east-west ridge. The vertical grid consisted of 12 s-layers, spaced at 10% of the water depth, or 5% near the bottom and surface. The hindcast included vertical velocities calculated to ensure flows were parallel to the topography near the bottom and zero at the surface. The hindcast files contained the 3D velocity vectors, the layer depths, and the free surface elevation at 30-min intervals for 10 days. The 2D synthetic hindcast also used here, only contained the horizontal velocity components from the uppermost depth bin of the 3D hindcast.

The synthetic tests released a given number of particles in a single pulse at time zero, at random locations within a 2-km-wide annulus centred at 10 km from the centre of the eddy. Particle tracking was done at 5-min time steps. All tests were carried out on a workstation (dual 14 core Intel Xeon Gold 6154 CPUs, 3 GHz, 24.75 MB L3 cache). Slightly faster results were obtained using a desktop computer (Intel 10 core I9-7900x, 3.3 Ghz, 13.75 MB L3 cache), but the workstation results are presented here as it allowed exploration of scalability at higher core numbers.

The particle tracking speed comparisons use two versions of the same base code of OT, one using the LTW SciPy interpolator LinearNDInterpolator, the other using a STW and weight reuse. The Lagrangian particle tracking code OD (Dagestad et al. 2018) has a well-developed regular grid interpolator and for unstructured grids it uses the LTW Scipy interpolator. Thus, the presented speed comparisons are essentially between OT and OD for unstructured grids.

3.1 A direct comparison with OpenDrift?

The current version of OpenDrift (1.0.7) uses LinearNDInterpolator indirectly to interpolate fields on an unstructured grid. This makes a direct speed comparison of OT and OD difficult. OD first regrids the unstructured hindcast to a regular grid over a rectangle which bounds the spatial extent of the particles at each time step, then does particle tracking by interpolating fields on this regular grid. This approach will be fast for small numbers of particles confined to a small region. However, computationally this approach scales poorly once the number of particles exceeds the number of nodes of the unstructured grid within the bounding rectangle. For particles released from widely spaced locations, which then spread out over time, the bounding box will approach the size of the full grid, e.g. Fig. 3. Thus, for the grid in Fig. 1, OD would be slower than using

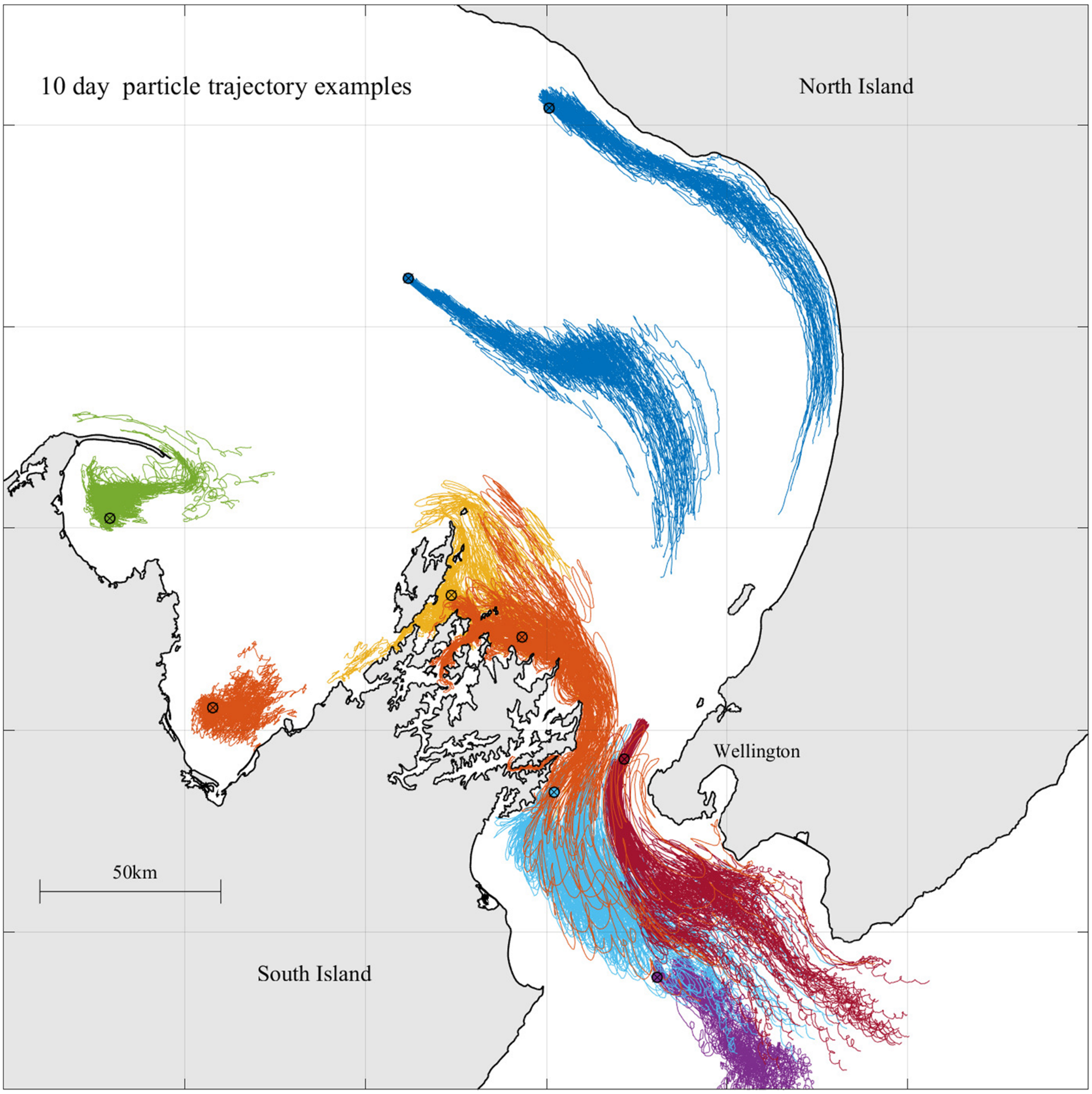


Fig. 3 Example of 10 day particle trajectories for 50 particles released from each of the 9 sites. Colours indicate the particle's release site

LinearNDInterpolator directly for particle numbers above 140,000. In addition, with this large bounding box, to match the underlying high resolution of the unstructured grid, the regular grid will require an unmanageable number of nodes. For example, to match the 36-m minimum triangle size in Fig. 1 would require OD to generate an manageable 65×10^6 node regular grid of values at each time step for each field. Thus, to compare OD with OT for the large number of particles which are the focus of this work, we simulated the performance of OD as if it used LinearNDInterpolator to do particle tracking directly on the unstructured grid (we labeled this simulation SciPy). As a consequence, the computational speed advantages of OT over the current release of OD at large numbers of particles presented in this work are conservative. It is expected that a future release of OD (ver. 1.5.0) will use a KD-tree approach to do "native" interpolation for unstructured grids. Section 6.1 gives a direct speed comparison with this future release.

4 Speed comparisons—synthetic data

4.1 Particle numbers and grid size

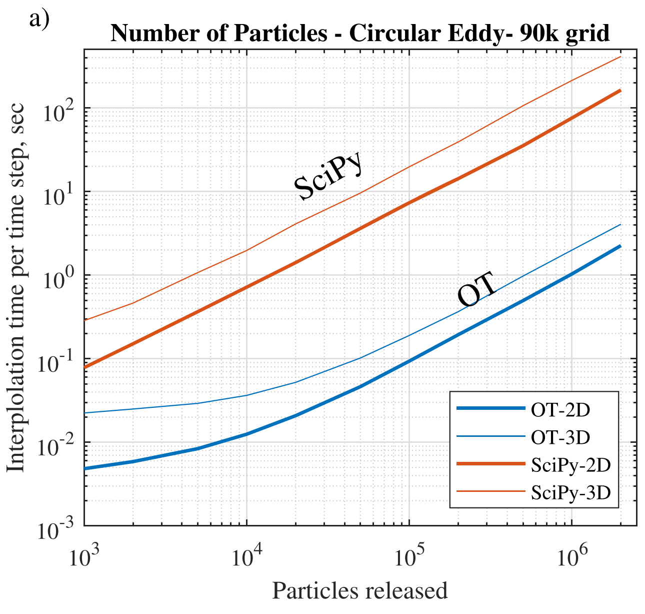
Table 1 shows run times for the SciPy and OT for 2D and 3D synthetic eddies with two hydrodynamic grid sizes. For SciPy, 99% of the time is spent doing the interpolation. Thus, techniques to speed interpolation are the main focus of this work. Table 1 demonstrates that OT is much faster at interpolation: in 2D, 75 times faster for the 90k node hydrodynamic grid and 110 times faster on the 250k node grid. The relative speeds in 3D are even higher. It was not possible to partition the time used within SciPy's LinearNDInterpolator between finding the cell and applying the weights. However, both approaches use similar C++ based code to apply the interpolation weights, so it was assumed that weight application times for both were the same. With this assumption, SciPy expends almost all its computation time on finding the cells and weights.

The significant speed advantage of OT-2D over SciPy-2D derives roughly equally from the STW approach and the reuse of interpolation weights between the two time steps which must be interpolated for each sub-step of the RK4 integration, see Section 4.2. The time spent finding the vertical cell or layer is also significant for SciPy-3D. OT-3D has the additional advantage of using a SVW, Section 2.3.

Figure 4a plots interpolation time versus the number of particles released. Both interpolation techniques show near-linear scaling of computation time with particle numbers, though the times for OT-2D are much lower. Figure 4b plots the time taken to interpolate by SciPy relative to OT. On the 90k grid with 10^3 particles, OT-2D is only 20 times faster, presumably due to fixed overheads of code interpretation, setting up function calls etc., but peaks out at 75 times faster for more than 10^5 particles. For the 250k grid, the relative speed advantage is almost 25% higher.

Figure 4b also shows the relative speed for 3D particle tracking. It demonstrates that the SVW increases the speed advantage of OT by 50% for large numbers of particles. The largest advantage shown is 150 times faster for 10^6 particles on the 250k node 3D grid.

Table 1 indicates that the time taken by OT-2D for the 250k node grid is only marginally longer than the time for the 90k node grid. This illustrates one of the significant advantages of OT-2D that interpolation times are insensitive to grid resolution. As a result, the speed advantage of OT-2D over SciPy-2D grows nearly linearly with hydrodynamic model grid size, as seen in Fig. 5. This insensitivity is a consequence of using an STW, where the triangle at beginning of the time step is likely less than a couple of triangles away from the triangle containing the particle at the end of the time step. Increasing the hydrodynamic model grid resolution may not increase the number of triangles the



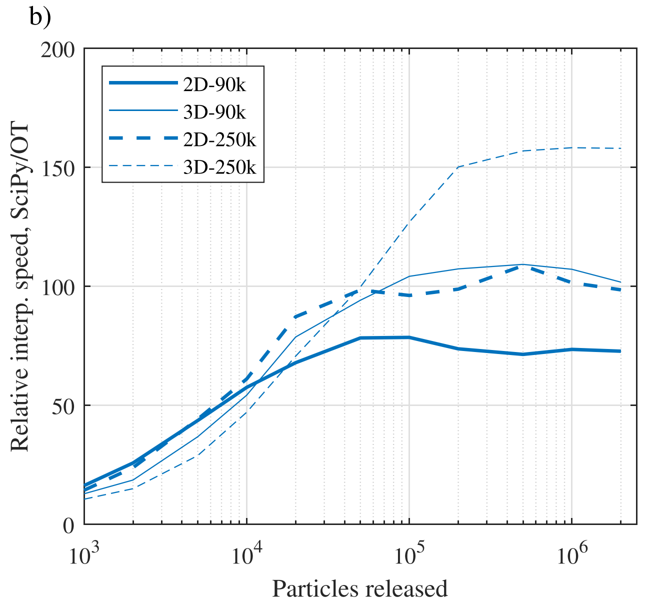


Fig. 4 (a) Comparison of two interpolators for range of particle numbers for synthetic eddy data run on a single computer core. (b) OT-2D is up to 80 times faster than SciPy2D for this example, but up 150 times faster in 3D for the 250k node grid

algorithm must walk through in the short distance between the old position and the new position. For example, if there is a one step walk to the adjacent triangle, the grid size would have to halve in order to increase the number of triangles walked. This would require a hydrodynamic model grid with four times the number of nodes, which would likely make running the hydrodynamic model infeasible. In contrast, for the long walk of SciPy2D, a small increase in grid resolution would likely result in a proportionate

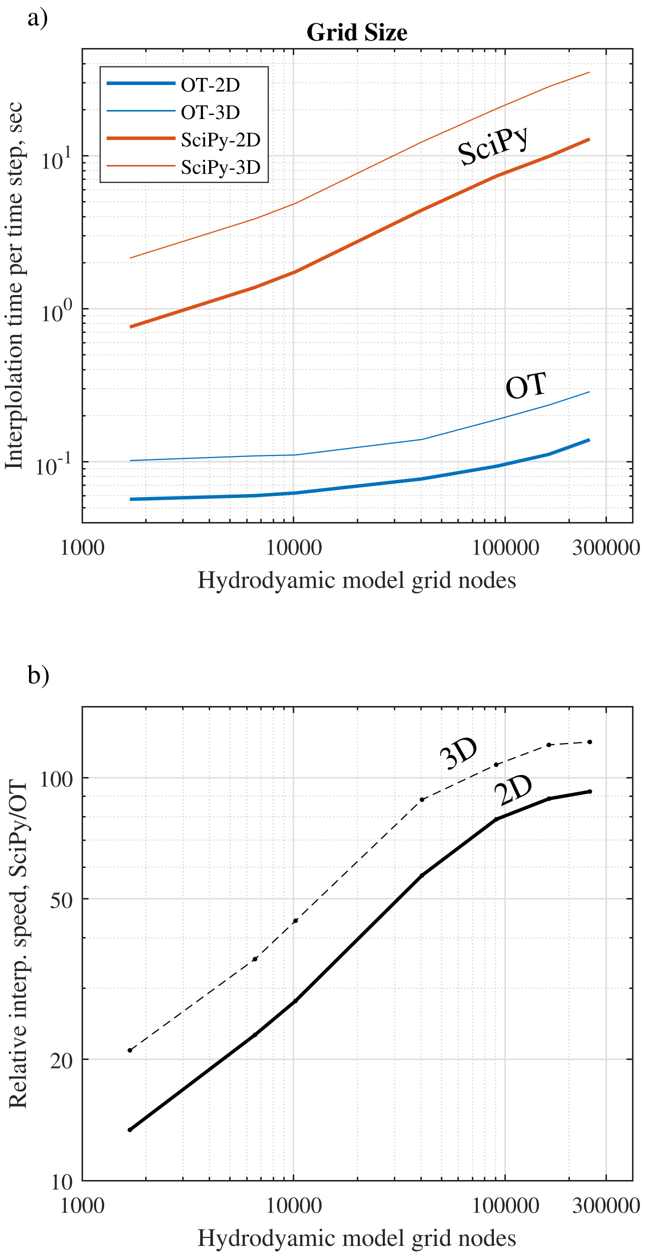


Fig. 5 (a) Comparison of two interpolators for a range of hydrodynamic model grid sizes for 10^5 particles released within the synthetic eddy run on a single computer core. (b) The ratio of computational times of SciPy over OT. OT's speed advantage shows a strong almost linear increase with hydrodynamic model grid size

increase in the number of triangles walked. This makes the long-walk computation times of SciPy more sensitive to grid resolution over realistic ranges of the average grid resolution.

4.2 Reuse of interpolation weights

As noted earlier, weight reuse eliminates one spatial interpolation of the velocity field. Thus, weight reuse explains half of the speed improvement of OT over SciPy in Table 1 and subsequent figures.

All the spatial fields that need to be interpolated can be split into two types. The first are dynamically "active fields", those that affect the velocity of the particle and must be interpolated to find the particle velocity. The active fields always include water velocity but may also include other fields, such as Stokes drift. Dynamically active fields must be interpolated at every sub-step of the RK4 time integration, so are interpolated four times per time step.

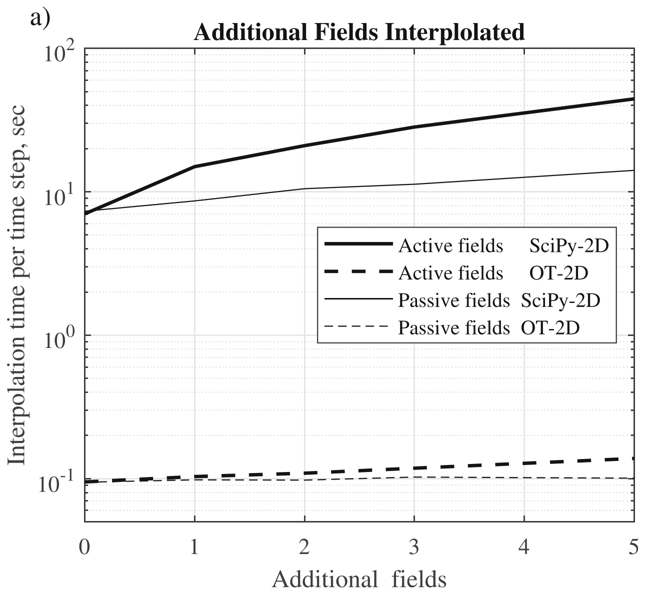
The second type, dynamically "passive fields", do not affect the particle velocity, but may affect how a particle behaves, or must be tracked over time. For example, tidal elevation affects behaviour as it is needed to determine whether a particle has been stranded by the receding tide and needs to remain stationary during the next time step. A tracked example might be the temperature, which may affect how the concentration of a compound within a particle decays with time. Dynamically passive fields only need to be interpolated at the end of each RK4 step, so required only a quarter of the computation effort required to interpolate active fields.

Figure 6a shows much lower computational time per RK4 time step for OT, when compared to SciPy. Though distorted by the log scale vertical axis, the figure also demonstrates that interpolating additional active or passive fields with OT is essentially free, when compared to the cost incurred interpolating additional fields when using SciPy.

Figure 6b shows that for 10^5 particles on the 90k grid OT-2D is 70 times faster than SciPy-2D, when no additional fields require interpolation. The figure also shows that using weight reuse to eliminate a SciPy interpolation increases the relative speed advantage by around a factor of 50 per additional active field and 14 per additional passive field. This roughly 4:1 scaling of the relative advantage for additional active and passive fields agrees well with expectations based on the discussion above.

4.3 Comparison to regular grid

Particle tracking in regular grids is typically much faster, because finding the horizontal cell containing each particle can quickly be done by rounding coordinates down. Figure 7a compares the interpolation speed of OT for the unstructured grid interpolator with weight reuse and a regular grid interpolator without weight reuse. Surprisingly, in 2D at higher particle numbers, OT's triangular grid interpolator is marginally faster than the regular grid interpolator on the 90k node grid. This suggests that the benefits of weight reuse outweighed the additional computational cost of the STW, over coordinate rounding in regular grid.



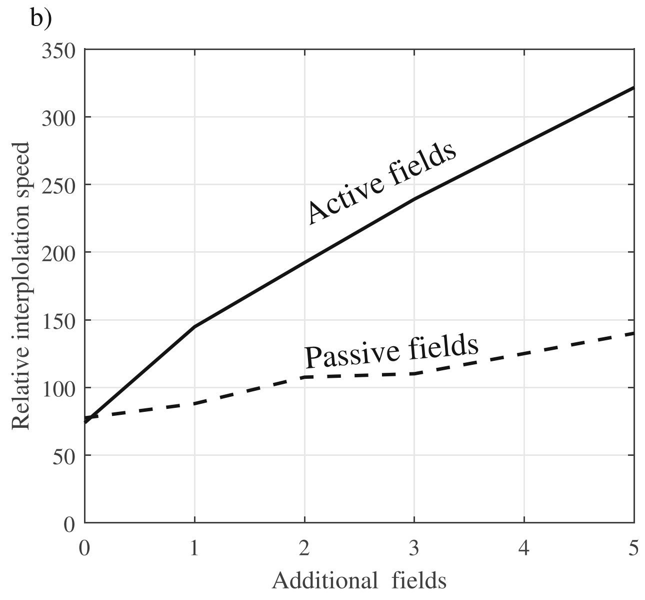
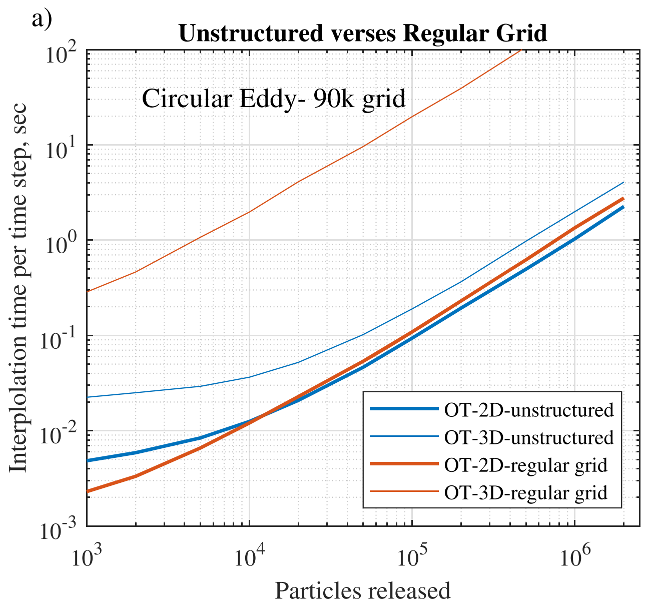


Fig. 6 (a) Benefits of reuse of interpolation weights for 10^5 particles released within the 2D synthetic eddy with a 90k node grid run on a single computer core for dynamically active and passive fields. (b) Weight reuse increases the interpolation speed of OT-2D relative to SciPy- 2D by 50 per dynamically active field and 14 per dynamically passive field



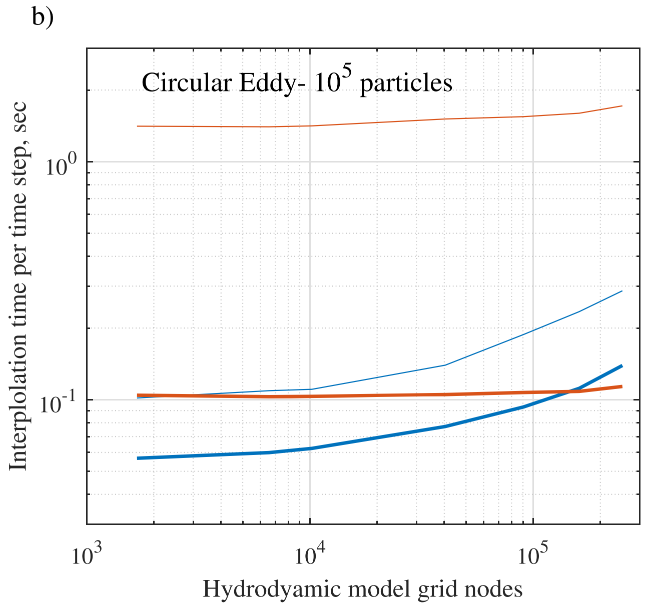


Fig. 7 Comparison of interpolation times for synthetic eddy treated as an unstructured grid and as a regular grid, when run on a single computer core

Figure 7a demonstrates that OT-3D for structured grids is an order of magnitude slower than OT-3D-unstructured. This difference is mainly the result of OT's SVW.

Figure 7b shows that for most hydrodynamic model grid sizes OT's triangular grid interpolator is faster than the regular grid interpolator when releasing 10^5 particles. Only for 2D interpolation of models with more than 10^5 grid nodes does the regular grid have a marginal speed advantage in the examples shown.

5 Multi-core performance

The results above are for OT running on a single computer core. Running on more cores allows more particle trajectories to be calculated at the same time. This was done by using a case-master code to run a specified number of copies of OT. Each copy ran independently on a separate core with its own set of particles and random starting locations. Figure 8 compares computational efficiency for a range of cases based on the total elapsed time to run all of

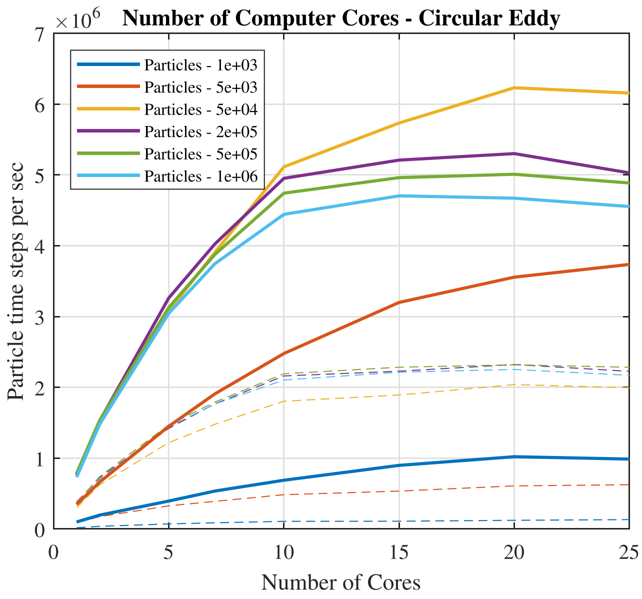


Fig. 8 Effect of number of processor cores on computational speed for the synthetic eddy. Solid lines are for OT-2D, with different lines for different numbers of particles per core as indicated by legend. Dashed lines are for OT-3D. *Y*-axis is the speed is measured by the total number of particle time steps carried out across all cores per second. This includes the time to read the hindcast file. These curves are for a 90k node grid, but curves for 250k grid are similar

the copies (excluding the time to read the hindcast). Each copy released between 10^3 and 3×10^6 particles, resulting in up to 7.5×10^7 particles being released.

The total number of particle time steps processed per second in Fig. 8 demonstrates that more cores are faster, with speeds initially increasing linearly with the core count. There is no speed advantage above 10 cores. Overall, Fig. 8 demonstrates that using more cores to run multiple cases is up to 5–8 times faster. For a given number of cores, times are longer for small numbers of particles per core, as the time to read the hindcast is significant. Above 10^5 particles per core, read time is insignificant compared to the time spent on computations; thus, the curves become similar.

For a given total number of particles, using more than 10 cores did not increase speeds on the hardware used. This limit to the benefit of additional cores may result from limitations of the resources they share. Cores share “L3” memory cache and bandwidth to main memory. Thus beyond 10 cores on the hardware used here, the capacity of the cores to do the computations may exceed the ability of the system to supply them with the data they need. This suggests further that work to reduce the volume of the memory that must be transferred to the CPU may lead to further speed improvements by enabling more cores to exploited. In addition, using HPCs with a large number of physical CPUs with dedicated memory access may also significantly increase the number of trajectories which could be calculated in a given time. One detail apparent in Fig. 8, is that at high core numbers using 5×10^4 particles per core runs slightly faster than running more particles per core. This apparently optimal number of particles per core may be due to caching making it more likely that a previously accessed value is already in the cache than at higher particle numbers.

6 Coastal grid example—Cook Strait

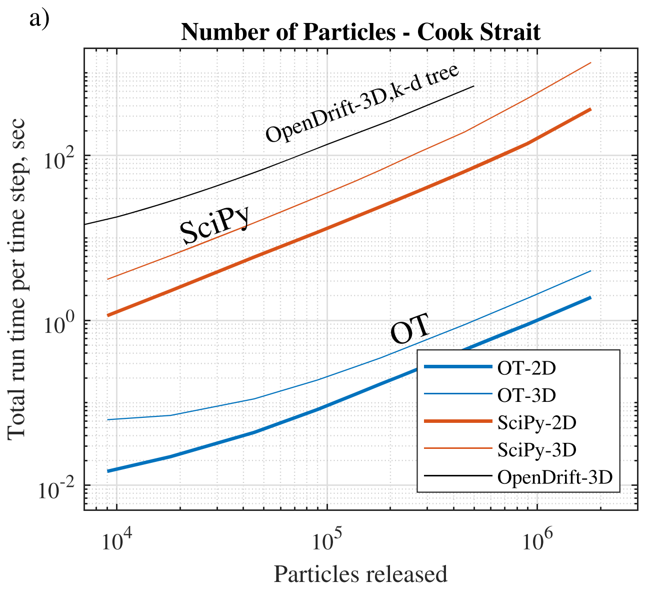
To see if the speed improvements evident in the synthetic hindcast tests also occur with more realistic model domains, particle tracking was done with a hindcast produced by SCHISM (Zhang et al. 2008, 2016) for Cook Strait, New Zealand, Fig. 1. Figure 9 shows example particle tracks from the 9 release locations used in the tests. Unlike the synthetic data tests, the Cook Strait example includes horizontal and vertical random walks to simulate dispersion. These tests also included stranding by the tide, which requires interpolation of two additional passive fields (tide and water depth) to determine whether a particle has stranded or can be re-floated. The simulations were run at a 5-min time step.

The relative computational times for Cook Strait are consistent with those based on the synthetic data, with additional passive fields. Figure 9b shows OT is typically 150 times faster in 2D and 200 times faster in 3D. Unlike previous comparisons, this speed comparison includes the time to read the hindcast. At low particle numbers, read time is a significant fraction of the total time: this read time dilutes the speed advantage of OT’s interpolation techniques on the left of Fig. 9b. Also, the larger files that must be read for the 3D cases dilutes the speed advantage of OT-3D to a point that the gains are less than for OT-2D.

Multi-core performance also has similar computational speed as the synthetic data tests and again there are speed benefits up to 10 cores and little gain beyond this, Fig. 10.

6.1 Comparison with future release of OpenDrift

Previous speed comparisons with OD were indirect, done with OT using SciPy to simulate OD’s interpolator as if it used LinearNDInterpolator directly, see Section 6.1. It is expected that a future release of OD will uses a KD-tree based interpolation. Like OT, this future release “native” grid interpolator preserves the horizontal resolution of the hindcast’s unstructured grid and its time-varying s-level vertical grid. We carried out a direct comparison by running a simulation with a preliminary version of the unstructured grid reader to be included in a future OD release, for the 3D hindcast of the Cook Strait grid.



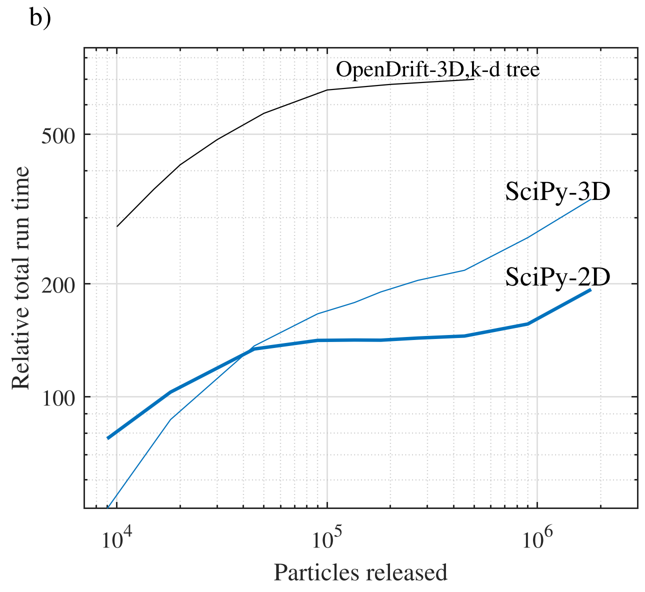


Fig. 9 (a) Speed comparison for particles released at the 9 locations within the Cook Strait 149k node model grid in Fig. 1. Plot gives the total run time (i.e. including time to read the hindcast), run on single core. (b) Ratio of the time taken by SciPy to that for OT. The black lines are a comparison with an expected future release of OpenDrift, discussed in Section 6.1

The black lines in Fig. 9 compare the speed of running OD 3D KD-tree with that of OT for up to 5×10^5particles. For unstructured grids, OT-3D is up to 700 times faster than OD. The OD-3D KD-tree is also 5 times slower than OT's simulation of OpenDrift using SciPy. This is likely due to OD-3D rebuilding the KD-tree at each time step as the vertical grid varies over time. We plan to incorporate some of the speed advantages of OT in future versions of OD's unstructured grid reader.

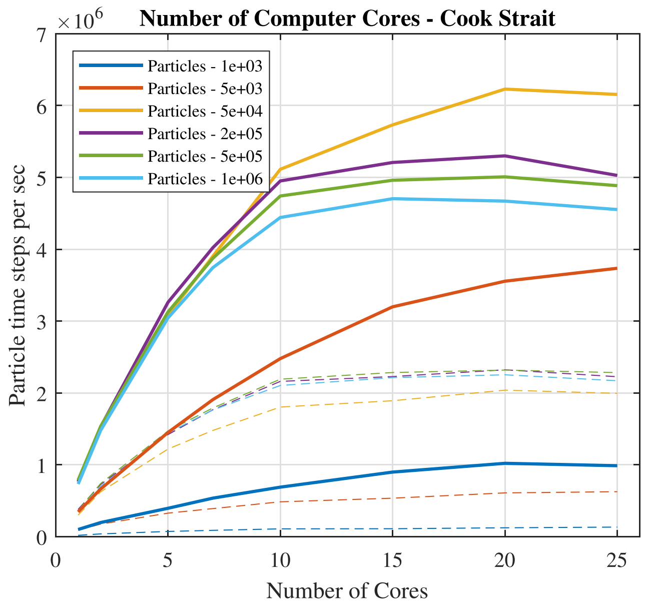


Fig. 10 Benefits of additional computer cores for particles released in Cook Strait from the 9 locations in Fig. 1. Solid lines are for OT-2D, with different lines for different numbers of particles per core as indicated by legend. Dashed lines are for OT-3D. *Y*-axis is the speed which is measured by the total number of particle time steps carried out across all cores per second. This includes the time to read the hindcast file

7 Discussion: future improvements

The multi-processing results indicate that the limiting step for large numbers of particles is not doing the required calculations, but getting the field and particle data from memory to the CPU to perform those calculations, i.e. limitations on memory bandwidth. Thus, reducing the volume of data that must be transferred may provide the best avenue for further speed improvements. Much of the current version uses Python's NumPy module to do basic arithmetic operations using arrays. This means that large arrays must be moved into memory for each arithmetic operation. There are many places where these arithmetic operations could be combined within a for-loop of Numba-based code (Numba compiles flagged sections of Python code into fast running low level code). Tests showed that a combining a matrix addition with an element-wise multiplication within the same for-loop took only 70% of the time to do a matrix addition, followed by a matrix multiplication. Combining more than two operations within a Numba for-loop was even faster.

For 3D s-level hindcasts, the time searching for each particle's depth cell could be significantly reduced by using a particle's vertical velocity to decide the search direction. This would reduce the minimum number of interpolations required to determine the vertical cell from two to one. As many particles likely lie within in the same cell as the previous time step, this approach would significantly reduce the time taken to confirm this.

Running cases as Python processes in parallel to increase the number of particles that could be processed at the same time meant each case was independently reading the hindcast's time steps into separate read buffers within OT. This could be improved by creating a shared hindcast read buffer, which would also require the development of coordination between processes to ensure all have processed the time steps in the buffer before data is discarded. Doing so would significantly reduce both read time and the main memory required to run parallel cases.

The numerical approach used was a RK4 numerical step followed by a random walk. This could be improved by using stochastic differential equations (Kloeden and Platen 2013). Their use has not attracted much attention in ocean modeling (Shah et al. 2013); however, they would improve the physics of the particle tracking when there is random motion.

8 Conclusions

The short triangle walk (STW) and weight reuse, along with other developments, conservatively provides a speed advantage of two orders of magnitude over OpenDrift's interpolation techniques, when particle tracking with the unstructured grids, Fig. 4. A direct comparison with a preliminary version of an expected future release of OpenDrift (ver. 1.5.0) shows OceanTracker (OT) is up to 700 times faster, Fig. 9. More significantly, the speed improvements allowed computation to be marginally faster when using regular horizontal grids in 2D, and much faster in 3D, Fig. 7. Though presented as part of a new code, OT, the fast interpolation techniques could be incorporated into existing unstructured grid particle tracking codes. The techniques remove one of the barriers to using unstructured grid particle tracking: their previous poor performance when compared to structured grids. In addition, Fig. 9 suggests that incorporating weight reuse may improve the performance of particle tracking in structured grids.

OT's speed advantage makes it routinely possible to simulate millions of particles when using unstructured grids, enabling more robust estimates of dispersion and bio-physical ocean connectivity for a wider range of particle parameters and behaviours. For example, the 149k node Cook Strait model required 2 s per RK4 time step to calculate the 3D trajectories of 10^6 particles on a single core, Fig. 9a. This equates to 5 h to calculate one million month-long trajectories at the 5-min intervals that might be required in a tidally-dominated coastal region. Thus, 2–3 million trajectories could be calculated overnight on a single core.

OT's STW makes computational times insensitive to the hydrodynamic model's horizontal grid resolution for realistic variations in average triangle size, Figs. 5 and 7b. OT's SVW preserves a hydrodynamic model's spatially and temporally varying vertical grid and allows computation times in 3D to be typically only 2–3 times longer than those for 2D tracking. Computational efficiency also scales well when additional fields are required to be interpolated, as weight reuse enables additional active and passive fields to be interpolated at low computational cost, Fig. 6.

Using multiple cores to run cases in parallel made processing 5–8 times faster on the hardware used, enabling the trajectories of a given number of particles to be calculated in less time, Figs. 8 and 10. There was no improvement above 10 cores, suggesting that access speeds to shared memory resources is limiting computational speeds at higher core numbers. This indicates that OT could be improved by reducing the volume of memory that must be accessed and optimizing the order in which it is accessed in order to take advantage of hardware caching, see Section 7. Modifications to the code structure would also allow future versions to exploit other hardware types, such as GPUs, or to split regions of the model domain across multiple processors on HPCs (Wolfram 2015a). All of these modifications could provide benefits in addition to the performance improvements outlined here. In addition, the fast techniques could easily be extended to used higher order spatial interpolation schemes or use adaptive time stepping RK45 integration.

Acknowledgements This work was funded by a New Zealand National Science Challenges MBIE grant within the Sustainable Seas Programs 5.1.4 and 4.2.2. We also appreciate an anonymous reviewer's suggestion on a further speed improvement to finding the depth cell, which is outlined in Section 7 for inclusion in subsequent version of OceanTracker.

Appendix. OceanTracker structure (version 0.1)

To make it more accessible, OceanTracker (OT) is written in the modern language Python. Key computational tasks used Numba.

The OT framework uses an explicit family tree of Python classes to connect major parts of the code, Fig. 11. This allows the user to easily insert different versions of each family member into the tree, customized to their usage. The OceanTrackerSim class sets up the particle tracking, which can be on one or more computer cores. Solver does the time integration to calculate the particle trajectories. Spatial and temporal interpolation is managed by the FieldGroup which contains the hindcast file reader and an appropriate spatial interpolator. The file reader fills memory buffers

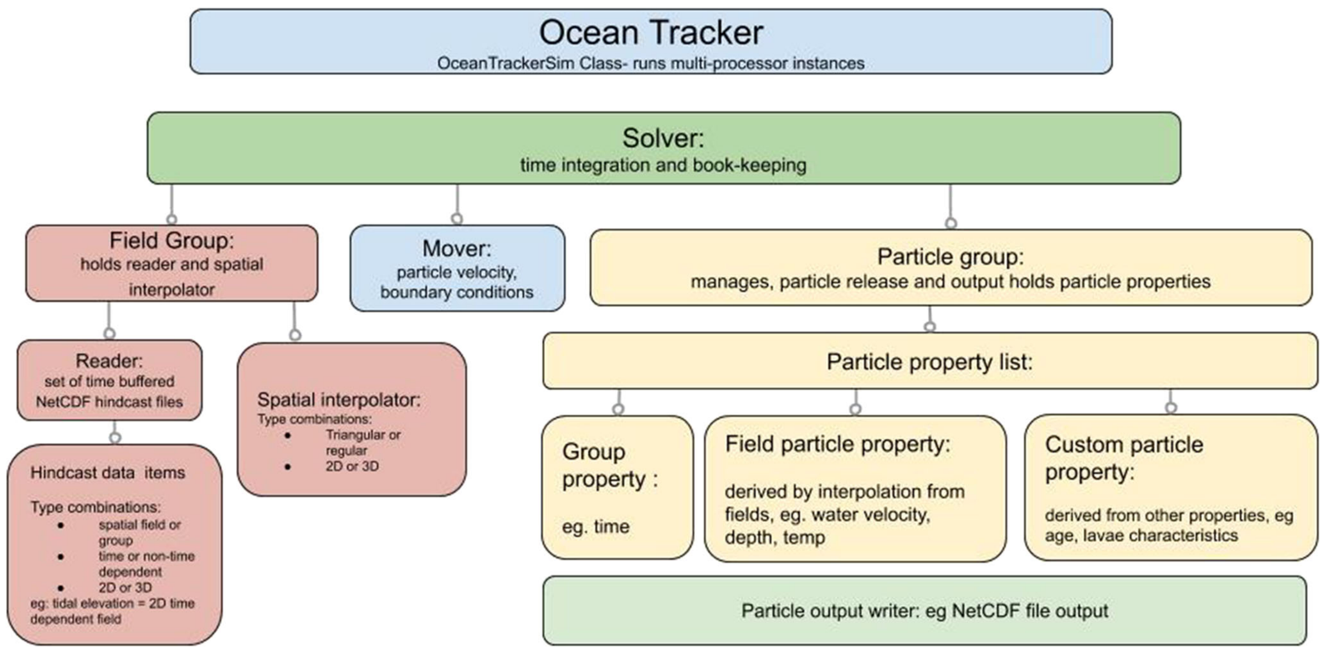

Ocean Tracker
OceanTrackerSim Class- runs multi-processor instances
Solver:
time integration and book-keeping
Field Group:
holds reader and spatial interpolator
Mover:
particle velocity, boundary conditions
Particle group:
manages, particle release and output holds particle properties
Reader:
set of time buffered NetCDF hindcast files
Spatial interpolator:
Type combinations:
Triangular or regular
2D or 3D
Particle property list:
Hindcast data items
Type combinations:
spatial field or group
time or non-time dependent
2D or 3D
eg: tidal elevation = 2D time dependent field
Group property :
eg. time
Field particle property:
derived by interpolation from fields, eg. water velocity, depth, temp
Custom particle property:
derived from other properties, eg age, lavae characteristics
Particle output writer: eg NetCDF file output

Fig. 11 Structure of parent-child relationships of classes within Lagrangian code OceanTracker v0.1

with multiple time steps of the hindcast, which are then used for particle tracking. At each time step of the Solver, the Mover class returns the water velocity at each particle's location, plus any modifications to the velocity due to particle behaviours, such as a fall velocity. The Solver integrates this velocity using RK1, RK2, or RK4 numerical integration.

Data for individual particles are held within a Particle Group, which holds several types of particle properties. Group properties are relevant to the whole set of particles, e.g. the current time. Individual particle properties which derive from spatial interpolation of hindcast fields are held as Field Particle properties, e.g. particle location, velocity, water temperature, and water depth. Finally, Custom properties are those added by a user, which can be calculated from the other properties; for example, the decaying concentration of a compound within each particle, where the decay rate depends on the water temperature experienced by the particle. Finally, the Particle Writer underpins the Particle Group to write output as required by user. This grouping of particle properties into groups allows many aspects of property creation and use to be automated, e.g. requesting a temperature to be loaded from the hindcast file causes it to be added to a list of particles properties which can then be automatically read, interpolated to particle locations and written to output files, without any changes to the code. All hindcast variables are classified in types of 2D/3D, time-dependent/time-independent, vector/scalars, to further aid this automation.

OT's explicit family tree allows members to easily obtain information from other members, e.g. the Mover requests the interpolated water velocity from its sibling "fields" using text strings (written in Python as v = self.sibling("fields").interp("water_velocity")); similarly a custom particle property could request the interpolated water temperature using code self.greataunt("fields").interp("water_temperature").

All family members are driven by parameters held in Python dictionaries, which can have specified default values. The parameters contain particle release locations, integration time step etc., appropriate to each family member. This was done so that the OT could be entirely driven by a set of parameter dictionaries, making it easier to support web-based requests for calculated trajectories. The parameter dictionary approach also means that, as users modify or extend a family member through code inheritance, previous parameters are acquired by its descendants. The most likely member to be extended are the Mover, in order to model how a user's bio-physical particle moves in response to its environment, or the Reader to customize it for the user's file format.

Another feature to make adaption and extensions easier to implement is to use internal property names as text strings (e.g."water_temp"), which are mapped to file variable names by the user. These internal names are then used as keys of a Python dictionary to access and update variables stored as matrices using high level code, e.g. particle.set_property("x", x_new) to update all particle locations with new vectors. These properties hold matrix data of dimensions determined at start-up from the dimensions of the hindcast data or model parameters. This abstraction of particle properties to dictionary keys also enables automation of operations like reading data files and writing output, which simply loop through the set of keys of the property dictionaries.

Some other features of OT include the ability for particles to be stranded/re-floated by the tide and be re-suspended based on critical shear velocity. A basic lateral boundary condition is implemented in this first version of the code. Particles which leave the hydrodynamic model's domain are flagged as bad. There is an option to return these bad particles to their last good position, where they are allowed to move at the next time step.

The numerical accuracy of OT's RK4 time integration was tested by releasing particles in a synthetic hindcast with the 90k nodes spaced at 100-m intervals over a 30-km-wide domain. Particles were released at a 10-km radius from the centre of the eddy, where peak flows were 1 m/s. The numerical drift from the starting radius was 0.1 m per month, for particle tracking done at 5-min time steps.

References

Dagestad K-F, Röhrs J, Breivik O, Ådlandsvik B (2018) Opendrift v1. 0: a generic framework for trajectory modelling. Geoscience Model Development

Delandmeter P, Van Sebille E (2019) The parcels v2. 0 Lagrangian framework: new field interpolation schemes. Geosci Model Dev 12(8):3571–3584

DHI (2020) Mike21 particle tracking module. https://www.mikepoweredbydhi.com/products/mike-21/sediments/particle-tracking

Heath RA (1978) Semidiurnal tides in Cook Strait. NZJMFW 12:87–97

Kloeden PE, Platen E (2013) Numerical solution of stochastic differential equations, vol 23. Springer Science & Business Media

Knight BR, Zyngfogel R, Forrest B et al (2009) Parttracker-a fate analysis tool for marine particles. Coasts and Ports 2009: In a Dynamic Environment, pp 186

Lynch DR, Greenberg DA, Bilgili A, McGillicuddy Jr D J, Manning JP, Aretxabaleta AL (2014) Particles in the coastal ocean: theory and applications. Cambridge University Press

Petton S, Pouvreau S, Dumas F (2020) Intensive use of Lagrangian trajectories to quantify coastal area dispersion. Ocean Dyn:1–19

Shah SHAM, Heemink AW, Gräwe U, Deleersnijder E (2013) Adaptive time stepping algorithm for Lagrangian transport models: theory and idealised test cases. Ocean Model 68:9–21

Song YT, Hou TY (2006) Parametric vertical coordinate formulation for multiscale, boussinesq, and non-boussinesq ocean modeling. Ocean Model 11(3-4):298–332

Surface-water Modeling System (2020) PTM Lagrangian particle tracking with SMS

Swearer SE, Treml EA, Shima JS (2019) H7 a review of biophysical models of marine larval dispersal. In: Oceanography and marine biology. Taylor & Francis

Treml EA, Ford JR, Black KP, Swearer SE (2015) Identifying the key biophysical drivers, connectivity outcomes, and metapopulation consequences of larval dispersal in the sea. Movement Ecol 3(1):17

Van Sebille E, Griffies SM, Abernathey R, Adams TP, Berloff P, Biastoch A, Blanke B, Chassignet EP, Cheng Y, Cotter CJ et al (2018) Lagrangian ocean analysis: fundamentals and practices. Ocean Model 121:49–75

Vennell R (1998) Observations of the phase of tidal currents along a strait. J Phys Oceanogr 28(8):1570–1577. https://doi.org/10.1175/1520-0485(1998)028<1570:OOTPOT>2.0.CO;2

Wolfram (2015a) https://github.com/MPAS-Dev/mpas-dev.github.com

Wolfram PJ, Ringler TD, Maltrud ME, Jacobsen DW, Petersen MR (2015b) Diagnosing isopycnal diffusivity in an eddying, idealized midlatitude ocean basin via Lagrangian, in situ, global, high-performance particle tracking (LIGHT). J Phys Oceanogr 45(8):2114–2133

Zhang YJ, Ye F, Stanev EV, Grashorn S (2016) Seamless cross-scale modeling with SCHISM. Ocean Model 102:64–81

Zhang Y, Baptista AM (2008) SELFE: a semi-implicit eulerian–lagrangian finite-element model for cross-scale ocean circulation. Ocean Model 21(3-4):71–96

Zhang YJ (2020) Schism post processing particle tracking. http://www.stccmop.org/CORIE/modeling/selfe/utilities.html