Geosci. Model Dev., 10, 4175–4186, 2017
https://doi.org/10.5194/gmd-10-4175-2017

Geoscientific Model Development
Open Access
EGU

Parcels v0.9: prototyping a Lagrangian ocean analysis framework for the petascale age

Michael Lange[1] **and Erik van Sebille**[2,3]

[1]Grantham Institute & Department of Earth Science and Engineering, Imperial College London, London, UK
[2]Institute for Marine and Atmospheric research Utrecht, Utrecht University, Utrecht, the Netherlands
[3]Grantham Institute & Department of Physics, Imperial College London, London, UK

Correspondence to: Erik van Sebille (e.vansebille@uu.nl)

Received: 11 July 2017 – Discussion started: 19 July 2017
Revised: 26 September 2017 – Accepted: 10 October 2017 – Published: 17 November 2017

Abstract. As ocean general circulation models (OGCMs) move into the petascale age, where the output of single simulations exceeds petabytes of storage space, tools to analyse the output of these models will need to scale up too. Lagrangian ocean analysis, where virtual particles are tracked through hydrodynamic fields, is an increasingly popular way to analyse OGCM output, by mapping pathways and connectivity of biotic and abiotic particulates. However, the current software stack of Lagrangian ocean analysis codes is not dynamic enough to cope with the increasing complexity, scale and need for customization of use-cases. Furthermore, most community codes are developed for stand-alone use, making it a nontrivial task to integrate virtual particles at runtime of the OGCM. Here, we introduce the new Parcels code, which was designed from the ground up to be sufficiently scalable to cope with petascale computing. We highlight its API design that combines flexibility and customization with the ability to optimize for HPC workflows, following the paradigm of domain-specific languages. Parcels is primarily written in Python, utilizing the wide range of tools available in the scientific Python ecosystem, while generating low-level C code and using just-in-time compilation for performance-critical computation. We show a worked-out example of its API, and validate the accuracy of the code against seven idealized test cases. This version 0.9 of Parcels is focused on laying out the API, with future work concentrating on support for curvilinear grids, optimization, efficiency and at-runtime coupling with OGCMs.

1 Introduction

Lagrangian ocean analysis, whereby virtual particles are tracked within the flow field of hydrodynamic models, has over the last two decades increasingly been used by physical oceanographers and marine biologists alike (Van Sebille et al., 2018). The particles can represent passive parcels of seawater (e.g. Döös, 1995; Blanke and Raynaud, 1997) or its constituent tracers such as nutrients (e.g. Jönsson et al., 2011; Qin et al., 2016), as well as particulate matter such as microbes (e.g. Hellweger et al., 2014; Doblin and van Sebille, 2016), larvae (e.g. Cowen et al., 2006; Paris et al., 2005; Teske et al., 2015; Cetina-Heredia et al., 2015), pumice (e.g. Jutzeler et al., 2014), plastic litter (e.g. Lebreton et al., 2012), or icebergs (e.g. Marsh et al., 2015). The trajectories of the virtual particles can be used to analyse the flow within ocean general circulation models (OGCMs) and other velocity fields for dispersion characteristics (e.g. Beron-Vera and LaCasce, 2016), Lagrangian coherent structures (e.g. Haller, 2015), water mass pathways and transit times (e.g. Rühs et al., 2013), Lagrangian stream functions (e.g. Döös et al., 2008) and biological connectivity between regions (e.g. Kool et al., 2013). See Van Sebille et al. (2018) for an extensive review on Lagrangian ocean analysis.

There are currently three main community codes available to calculate the trajectories of virtual particles in OGCMs: Ariane (Blanke and Raynaud, 1997), TRACMASS (Döös et al., 2013, 2017), and the Connectivity Modeling System (CMS; Paris et al., 2013). These codes, being open-source and having excellent support teams, have served the wider community very well over the past decades. However, it is not clear that these three codes will be able to scale up easily

to the petascale age of computing, where particle trajectory codes will need to be able to deal with potentially petabytes of hydrodynamic field data and gigabytes of particle trajectory data. Exploring advanced optimization strategies to overcome these big-data challenges, such as coupled (online) execution with the host OGCM or reducing the volume of hydrodynamic data by selectively filtering data regions based on particle locations, will require a flexible execution model that can dynamically be adapted to complement the respective data and execution formats of various host OGCMs.

Furthermore, the current stack of codes is mostly built for the tracking of water parcels or passive particulates. While the CMS and TRACMASS do support the addition of diffusive processes through Markovian stochastic models (e.g. Griffa, 1996), it is non-trivial to incorporate "behaviour" of particulates to these codes. Effortless incorporation of behaviour such as sinking, fragmentation, or even swimming to particulates would simplify exploration of the dynamics of particulates such as fish, icebergs, and marine debris.

Here, we describe a novel framework for computing Lagrangian particle trajectories, named Parcels ("Probably A Really Computationally Efficient Lagrangian Simulator"). Being developed from the ground up with scalability and performance in mind, we hope that this Parcels framework will be able to keep up with OGCM development for the coming decades, particularly by being scalable and efficient at reading in hydrodynamic data. We have furthermore focused on flexibility and customizability of the particle dispersion schemes, so that it is relatively straightforward to add new functionality such as active particle behaviours.

We have decided to brand this version of Parcels as v0.9, signalling that while in principle it is feature-complete, the code is not nearly as fast and efficient as we envision it to be in the future. Improving performance will be the main priority as we work towards v1.0. We invite all interested researchers to contribute to the development by starting to use the code.

While development efforts of Parcels focus on oceanographic applications, the Parcels framework should in principle also be adaptable to atmospheric particle tracking simulations. Models such as FLEXPART (Stohl and James, 2005) and the MetOffice NAME model (Jones et al., 2007) are state of the art and have an excellent track record in the field of atmospheric dispersion modelling, but perhaps some of the ideas presented here could be incorporated or used in these models too.

This paper is structured as follows: in the next section, we will describe the philosophy behind the Parcels code. We then present a worked-out example of an application of Parcels for an actual scientific experiment in Sect. 3. Following that, we evaluate the accuracy of the code in Sect. 4, by comparison to analytical solutions in idealized test. We provide a future outlook in Sect. 5, before concluding in Sect. 6.

2 Prototype design and philosophy

A key contribution of the new Parcels v0.9 framework is to define a set of interfaces and composable abstractions that encapsulate the various processes required to create dynamic and extensible Lagrangian models that feature direct interactions between particles and an associated OGCM grid. The design follows modern scientific software engineering practices, providing high levels of modularity and flexibility with a clear intent to further specialize various sub-components at a later stage. The interfaces provided in Parcels are therefore intended to capture the general domain-specific challenges posed by particle tracking for Lagrangian ocean analysis. The overall design philosophy, as well as the structure of the code, is driven by three major design considerations:

Extensibility While the core algorithm of Lagrangian particle models is concerned with the advection and dispersion of passive particles that constitute infinitely small point parcels, practical oceanographic applications often require more complex behaviour of the particles. Potential extensions towards individual-based modelling of particulates to simulate biological species or marine debris will require extensions to particle data definitions and programmable behavioural customization at a per-particle level.

Compatibility Particle tracking in oceanography requires the close coupling of computational particles to velocity data that define the hydrodynamic flow field. Parcels aims to make as few assumptions about the nature and structure of the hydrodynamic fields as possible, so as to be compatible with various types of OGCMs and data formats. While the focus in this v0.9 is on utilizing offline data, this includes considerations for at-runtime coupling with OGCMs in the future.

Dynamic data Particle data are sparse in nature and can, depending on application context, exhibit very dynamic data access patterns where new particles are inserted and deleted from the active set at runtime. For this reason, structured compile-time performance optimizations and parallelization strategies are insufficient, and just-in-time scheduling is required to handle the amorphous data parallelism inherent in dynamic particle applications (Pingali et al., 2011).

The above list of requirements suggests that a static compile-time approach is likely to provide insufficient flexibility to adjust to the various scientific contexts in which oceanographic particle tracking might be utilized. For this reason Parcels is based on the domain-specific languages paradigm, which aims to decouple the problem definition as defined by the scientific modeller from the implementation that is ultimately executed on a particular hardware architecture. This approach is based on automated code generation at

runtime and creates a separation of concerns between domain scientists and computational experts that allows hardware-specific performance optimization and thus greater flexibility with respect to advances in high-performance computing resources.

Since the prototype of the Parcels framework presented here provides a conceptual blueprint for future versions, we define a clear set of abstractions for the following three software layers:

User-facing API The primary objective of Parcels is to provide a user-friendly, clear, and concise API for scientists to perform oceanographic particle tracking experiments with very little effort, while leaving room for customizations that go beyond traditional configuration files. For this reason Parcels provides a high-level Python API that enables users to define a complete model in a small number of lines of code (see examples in Sect. 3). For more advanced models, the API also provides enough scope to fully control the variable layout of particles in memory, as well as to define custom behaviour via individual kernel operations.

Execution layer The transient nature of Lagrangian particles implies that many practical oceanographic applications rely on particle sets that may grow and shrink dynamically, while also relying on external hydrodynamic field data that might be sampled at a time step much different from the primary particle loop. This complex parameter variability entails that the core loop that updates individual particle states needs to be highly dynamic and flexible, as well as highly optimized for large-scale applications. Parcels aims to encapsulate the core parameters of the particle update loop so as to establish an interface for integration with a variety of external host OGCMs, and leaves enough scope for more advanced performance optimizations in the future.

Data layout The two fundamental types of data involved in Lagrangian particle tracking algorithms constitute field data provided by the external OGCM, as well as data on the particle state. Since the data layout for particle data might change with future performance optimizations, and the memory layout of field data depends on the OGCM implementation, Parcels provides high-level abstractions for both types of data, allowing the actual data layout in memory to change.

The abstractions shown in Fig. 1 comprise the core functionalities provided by the framework. The primary input in the user layer consists of generic definitions of the particle variables for individual types of particles, alongside an interface to define the computation kernels. Parcels' core execution loop uses this information to update particle data given external parameters, such as time-stepping constraints, and interpolated hydrodynamic field data. Thus, given a stable user-level API and a highly modular code structure, it is possible to implement various applications and experiments without committing to a particular implementation, while leaving enough scope for further development and future performance optimization "under the hood".

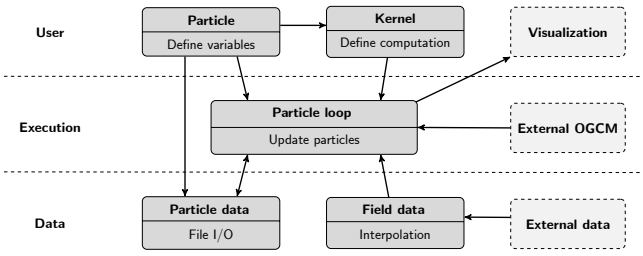

Figure 1. Conceptual abstractions (dark) and functionalities encapsulated in the Parcels prototype in relation to external components (light).

2.1 Programmable user interface

The prototype presented in this paper provides a highly flexible user API that allows users to define complete models via the Python programming language. The user hereby manages creation, execution, and customization of individual sets of particles, as well as combinations of computational kernels to update the particle state. In contrast to traditional configuration files, this approach provides the user with native compatibility with the open-source libraries and tools available in the scientific Python ecosystem.

The key components of Parcels' overall class structure are depicted in Fig. 2. The definition of the variables that constitute a single particle is hereby encapsulated in the `Particle` class, while container objects of type `ParticleSet` provide the runtime handling and management of particle data. Python descriptor objects are used to generically define the compound data type underlying each type of particle, leaving allocation and memory layout choices to the particular implementation of the data container structure.

The computational behaviour of particles is encapsulated through the `Kernel`. Parcels provides a set of pre-defined advection methods, as well as allowing users to define custom behaviour programmatically. Multiple kernels can be concatenated, allowing users to incrementally build complex behaviour from individual components.

2.1.1 Advection algorithm

At its core, computing Lagrangian particle trajectories is equivalent to solving the following equation:

$$\boldsymbol{X}(t+\Delta t) = \boldsymbol{X}(t) + \int_{t}^{t+\Delta t} \boldsymbol{v}(\boldsymbol{x},\tau)\,\mathrm{d}\tau + \Delta \boldsymbol{X}_b(t), \tag{1}$$

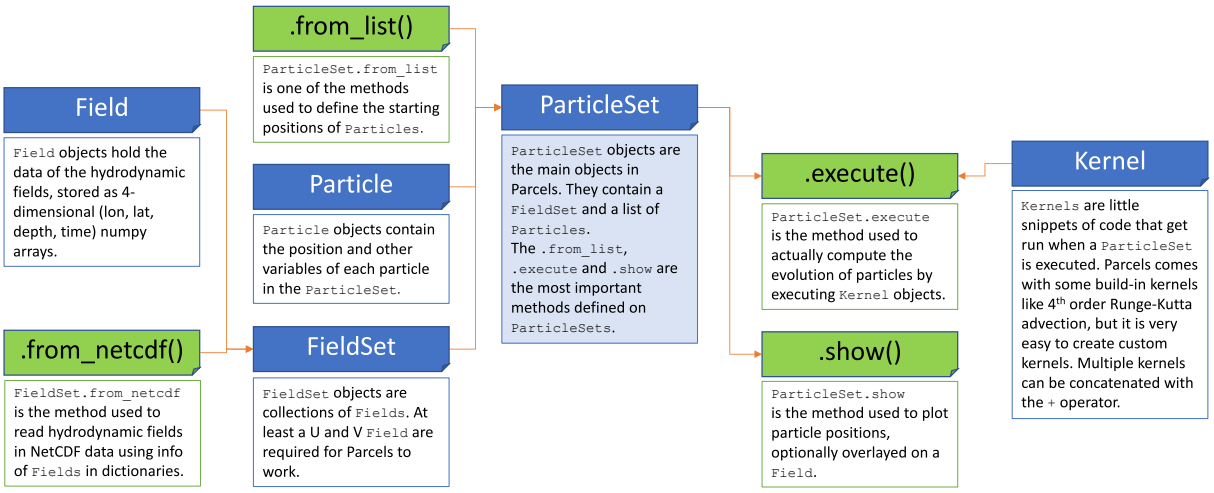

Figure 2. Class diagram of the Parcels v0.9 prototype implementation. Classes are depicted in blue, methods in green. Note that not all methods and classes are shown in this diagram.

where $\boldsymbol{X}$ is the three-dimensional position of a particle, $\boldsymbol{v}(\boldsymbol{x},t)$ is the three-dimensional velocity field at that location from an OGCM, and $\Delta \boldsymbol{X}_b(t)$ is a change in position due to "behaviour". The latter can itself be an integration of a (three-dimensional) velocity field, for example when a particle sinks downward because of a negative buoyancy force.

In Parcels, the trajectory Eq. (1) is by default time-stepped using a fourth-order Runge–Kutta scheme, although schemes for Euler-forward and adaptive Runge–Kutta–Fehlberg integration (RKF45, e.g. Alexander, 1990) are also provided. In principle, the Parcels framework should be flexible enough to also implement integration using the discrete analytical streamtube method (Blanke and Raynaud, 1997; Döös et al., 2017).

2.1.2 Custom kernels

Lagrangian particle tracking in the ocean often involves more complex displacement schemes than simple velocity-driven advection. For example, in the presence of turbulence, a random walk kernel or Brownian motion is required, while ocean ecology models often include active locomotion. Parcels therefore allows users to create generic kernel functions by providing native Python functions that adhere to the function signature `KernelName(particle, fieldset, time, dt)`. Within these kernel functions, users can access built-in particle state variables, such as `particle.lat` and `particle.lon`, or user-defined ones. Access to field data from within kernels is provided through the `fieldset` object, which provides fields as named properties, for example `fieldset.U` for the zonal velocity. Interpolation of field data is implemented via overloaded member access on the field object (square bracket notation), allowing a user to express field sampling as `fieldset.fieldname[time, lon, lat, depth]`.

In addition to kernels that update the internal state of particles, Parcels' execution engine also enables users to customize the behaviour of particles under various error conditions. For this, a similar type of kernel function can be created and passed to the execution call, mapped to a particular error type that might be triggered during the main particle update, for example `OutOfBoundsError`.

2.2 Execution and JIT compilation

The update of the internal state of particles is facilitated by a dynamic loop, which applies a user-defined combination of kernels to each particle in a `ParticleSet`. The primary particle update loop can either be run with a forward time stepping, or in a time-backward mode to enable inverse modelling. For this central update loop, Parcels provides two modes of execution:

Scipy mode A pure Python mode that utilizes `interpolator` objects provided by the Scientific Python package (SciPy) to perform interpolation of field data. This mode is primarily intended as a debug option due to the performance penalty of running kernels in the Python interpreter itself.

JIT mode Runtime code generation and just-in-time compilation (JIT) are utilized to generate low-level C code that performs the particle state update and field data interpolation. The code generation engine hereby primarily translates a restricted subset of the Python language into equivalent C code, while a set of utility modules provides auxiliary functionality such as random number generation or mathematical utilities (`math.h`).

The execution mode of the particle update loop is determined by the type of the particle (`ScipyParticle` or `JITParticle`) used to create the `ParticleSet`. Development of new features in the current Parcels prototype is strongly driven by the fact that both modes are intended to be semantically equivalent. This means that new features can rapidly be developed using the full flexibility of the Python interpreter, providing a template implementation and test case for implementation in the computationally more efficient JIT mode.

Parcels' dynamic update loop also provides an `interval` keyword to impose a secondary sub-time-stepping that allows for direct coupling with a host OGCM in the future. The dynamic composition of multiple time-stepping intervals might also be used for future data and performance optimization strategies, for example directed prefetching of regional field data. Such strategies, as well as requiring a potentially more intricate execution engine, have to be explored carefully to successfully tackle the big-data challenges facing Lagrangian tracking codes in the petascale age.

2.3 Interpolation

The interaction of particles with their enclosing fields is currently limited to interpolating field data onto the current particle position. In the SciPy debug mode this is facilitated by `scipy.interpolate.RegularGridInterpolator` objects and supports linear and nearest-neighbour interpolation. Equivalent low-level C routines are also included in the Parcels source code as macros that can be inlined into the generated C kernel code by the code generation engine. More advanced interpolation methods, such as quadratic, cubic, or spline interpolation, may easily be added in future releases if a fast C implementation can be provided with Parcels' internal header files.

One of key performance advantages of using runtime code generation is the ability to inline bespoke grid interpolation methods with the user-defined kernels in Parcels to avoid the Python interpreter overhead of repeatedly calling native Python interpolation functions. This overhead can be quite significant due to the high frequency at which the associated field data need to be sampled. This can be illustrated using the "Steady-state flow around a peninsula" test case discussed in Sect. 4.2.4, where 100 particles are advected for 20 h with a time-step size of 30 s. While the sequential execution time of the pure Python implementation runs in 305.92 s, the auto-generated JIT kernels can run the same experiment in 1.74 s, a speedup of over 150 times.

2.4 External field data

Parcels v0.9 supports external field data from NetCDF files, with a configurable interface to describe the input data and variable structure. The data are encapsulated in individual `Field` objects, which are accessible from within particle kernels via provided interpolation routines. Individual fields are stored in a `FieldSet` container class, which may also provide global metadata to the kernel execution engine at runtime.

Currently, only linear interpolation schemes are implemented in Parcels, both in space and in time. In space, Parcels can currently only work on regular grids (i.e. where the grid dimensions are functions of only longitude, only latitude, or only depth). However, support for unstructured grids is a priority for the next release of the code, Parcels v1.0.

3 A worked-out example: tracking virtual foraminifera in the Agulhas region

To highlight some of the prototype design and philosophies of the Parcels API, we here present a worked-out example code of a previously published scientific experiment. This example follows the experimental design of Van Sebille et al. (2015), where the goal was to investigate the temperatures that planktic foraminifera experience during their lifespan as they drift with the currents in the upper ocean. In particular, that study looked at the variability of lifespan-averaged temperatures of foraminifera that all end up on one single location on the ocean floor (e.g. Peeters et al., 2004; Katz et al., 2010).

Figure 1b of Van Sebille et al. (2015) depicted the origin of virtual planktic foraminifera that end up on a site just off the coast of Cape Town (17.3° E, 34.7° S), at 2440 m water depth. The virtual particles were released at that site and then tracked in time-backward mode. There were two phases to the experiment: in the sinking phase, the foraminifera were tracked back as they sunk at 200 m per day to the ocean floor, while being advected by the (deep) ocean circulation. In the lifespan phase, the particles were then tracked further backward in time as they were advected by the horizontal circulation at their 50 m dwelling depth. During this last phase, temperature along their trajectory was recorded at daily interval.

While the original experiment was computed with the CMS (Paris et al., 2013), here we have re-coded it using the Parcels API. This experiment setup is a fitting one, as it combines a number of the API highlights of Parcels: custom kernels, NetCDF I/O, and field sampling. The full Python code for this experiment in Parcels is available at https://doi.org/10.5281/zenodo.823994. Below, we emphasize some of the key statements in the Python script.

3.1 Reading the FieldSet

The hydrodynamic fields that carry the foraminifera come from the OFES model (Masumoto et al., 2004) and can be accessed from http://apdrc.soest.hawaii.edu/datadoc/ofes/ncep_0.1_global_3day.php. Three-dimensional veloci-

ties and temperature are available on 1/10° horizontal resolution, on 54 vertical levels, and are stored as 3-day averages. The bash script `get_ofesdata_agulhas.sh` provided at https://doi.org/10.5281/zenodo.823994 was used to download snapshot numbers 3165 to 3289, covering the year 2006, in a subdomain around the core site off Cape Town (note, the total file size is 6 GB).

While the 6 GB file size for this example is not excessively large and could in principle be loaded into memory all at once, this will not be possible for `FieldSets` with larger regional domains or longer time series. Hence, Parcels provides a system to read in hydrodynamic fields during particle integration, at any time storing only three consecutive time slices (e.g. Paris et al., 2013). See also Sect. 3.4.

After the first 3 days of hydrodynamic fields are read in through a call to the user-defined `set_ofes_fieldset` function (see the `example_corefootprintparticles.py` script for the exact formulation of this function, which requires as input a set with filenames, provided as a list of arbitrary length), three global constants are added to the `FieldSet`:

```
fieldset.add_constant('dwellingdepth', 50.)
fieldset.add_constant('sinkspeed', 200./86400)
fieldset.add_constant('maxage', 30.*86400)
```

These constants will be used later in the custom kernels controlling the movement of the particles.

3.2 Defining the ParticleSet

Apart from information on their location and time, the virtual foraminifera particles will need two extra `Variables`: the seawater temperature at their present location, and their age. Therefore, we define a new particle class, which inherits from the standard `JITParticle`:

```
class ForamParticle(JITParticle):
    temp = Variable('temp', dtype=np.float32,
                    initial=np.nan)
    age = Variable('age', dtype=np.float32,
                   initial=0.)
```

We then define a `ParticleSet` containing a single particle as

```
pset = ParticleSet(fieldset=fieldset,
                   pclass=ForamParticle, lon=[17.3],
                   lat=[-34.7], depth=[2440],
                   time=fieldset.U.time[-1])
```

3.3 Defining the custom kernels

We need to define four custom kernels: one that causes the particle to sink after it dies, one that keeps track of its age and deletes it once it reaches its maximum age, one that samples the temperature at its location, and one that deletes the particle when it reaches a boundary of the domain (since we only have hydrodynamic data in a subset of the global OFES domain). Note that while in principle the first three could be written in one kernel, here we write three separate kernels and then concatenate these with the built-in `AdvectionRK4_3D` kernel.

The first kernel, controlling the sinking of the particle after it died (i.e. the first 12 days in our reverse-time experiment), can be written as

```
def Sink(particle, fieldset, time, dt):
    if particle.depth > fieldset.dwellingdepth:
        particle.depth = particle.depth + \
                         fieldset.sinkspeed * dt
    else:
        particle.depth = fieldset.dwellingdepth
```

The second kernel, which keeps track of the age and deletes the particle when it reaches `maxage`, can be written as

```
def Age(particle, fieldset, time, dt):
    if particle.depth <= fieldset.dwellingdepth:
        particle.age = particle.age + math.fabs(dt)
    if particle.age > fieldset.maxage:
        particle.delete()
```

The third kernel, which samples the temperature, can be written as

```
def SampleTemp(particle, fieldset, time, dt):
    particle.temp = fieldset.temp[time, particle.lon,
                                  particle.lat,
                                  particle.depth]
```

These three kernels are then concatenated with the `AdvectionRK4_3D` kernel as

```
kernels = pset.Kernel(AdvectionRK4_3D) + \
          Sink + SampleTemp + Age
```

where at least one of the kernels needs to be cast into a `Kernel` object for the overloading of the + operator as a kernel concatenator to work.

Finally, the kernel that deletes a particle if it reaches one of the lateral boundaries and which will be invoked through the error recovery execution is

```
def DeleteParticle(particle, fieldset, time, dt):
    particle.delete()
```

3.4 Executing the ParticleSet

The `ParticleSet` can now be integrated with a call to `pset.execute()`. This method requires as input the list of kernels, the start time of the execution loop, the runtime of the execution loop, the Runge–Kutta integration time step (here taken to be 5 min), the interval at which output is written (here once per day), and the recovery kernel that gets called when a particle crosses the boundary of the regional domain.

As mentioned in Sect. 3.1, only three time slices are held in memory at any one time. The loading of new fields is controlled by the `fieldset.advancetime()` method, which replaces the oldest time slice with a new one (held in this case in `[snapshots[s]]`). This also means that the executing of the ParticleSet has to be done within a loop:

```
for s in range(len(snapshots)-5, -1, -1):
    pset.execute(kernels, starttime=pset[0].time,
                 runtime=delta(days=3),
                 dt=delta(minutes=-5),
                 interval=delta(days=-1),
                 recovery={ErrorCode.ErrorOutOfBounds:
                           DeleteParticle})
    fieldset.advancetime(set_ofes_fieldset([snapshots[s]]))
```

There is another reason to call the `pset.execute` method within a loop: it allows for a new particle to be released every 3 days (the frequency with which hydrodynamic data is available). This happens within the for-loop through a call to

```
pset.add(ForamParticle(lon=[17.3], lat=[-34.7],
                       depth=[2440],
                       fieldset=fieldset))
```

3.5 Saving and plotting the output

The Parcels framework allows for storing of the locations of the particle to disk on-the-fly in NetCDF files, following the Discrete Sampling Geometries section of http://cfconventions.org/cf-conventions/v1.6.0/cf-conventions.html#discrete-sampling-geometries, and is hence CF-1.6-compliant. Storing of the particle trajectories and properties such as age and along-track temperature happens in the for-loop through calls to `pfile.write(pset, pset[0].time)` Since particles are continually added to and deleted from the ParticleSet, the `ParticleFile` needs to be stored in "indexed" format, where for each variable all particle states are written in one long vector:

```
pfile = ParticleFile(outfile, pset, type="indexed")
```

These long vectors in Indexed format, however, are not very easy to work with, so Parcels provides the utility script `convert_IndexedOutputToArray` to convert an Indexed NetCDF file to array format.

The particle trajectories can then be plotted using the `matplotlib` and `Basemap` libraries – see Fig. 3. This figure shows the temperature recorded on each day during the lifespan of all virtual particles. It highlights that foraminifera that end up on the ocean floor off Cape Town travel hundreds to thousands of kilometres during their lifespan, and that while some originate from the Agulhas Current as far north as 27° S, others originate from the much colder Southern Ocean south of 40° S.

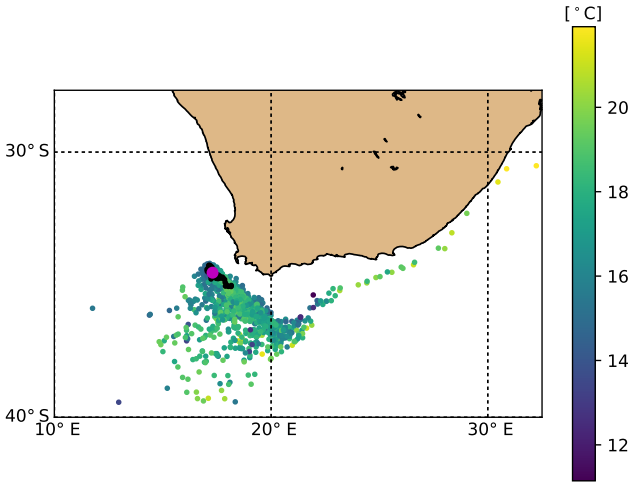

Figure 3. Footprints of virtual foraminifera ending up on the ocean floor just off Cape Town in the Agulhas region. This experiment is a Parcels implementation of the study described in Van Sebille et al. (2015), and this figure can be compared to Fig. 1b in that paper. The magenta dot is the location of the sediment core, from which virtual particles are first tracked back until they reach their 50 m dwelling depth (black dots), and then further tracked back for their 30-day lifespan. Temperatures (in Celsius) are recorded each day throughout their lifespan and shown as colours. The code for this experiment and plotting is available at https://doi.org/10.5281/zenodo.823994.

4 Model evaluation

Evaluation of a codebase's accuracy and performance is a key component of its validation and roll-out. For this Parcels v0.9, performance and speed are not a priority; these will be the focus for the v1.0 release (see also Sect. 5). Instead, in developing Parcels v0.9 we have concentrated on accuracy.

4.1 Unit tests and continuous integration

Following best practices in software engineering, we have incorporated Unit testing and continuous integration into the development cycle of Parcels. Every push of code changes to GitHub automatically triggers a validation of the entire codebase (an important component of the Continuous Integration paradigm), through the www.travis-ci.org web service.

The validation of the codebase is done through so-called unit tests; small snippets of code that test individual components of the codebase. Parcels v0.9 has over 150 of these unit tests, which check the integrity and consistency of the codebase. Where relevant, these unit tests are run in both Scipy and JIT mode, to test both modes of executing the kernels.

The following Python snippet shows a typical example of a unit test for Parcels (as included in the `test_particle_sets.py` file). It performs the test that `Particles` in a `ParticleSet` indeed get their assigned longitudes and latitudes. While this may seem a trivial test, these kinds of unit tests can help prevent bugs:

```
@pytest.mark.parametrize('mode', ['scipy', 'jit'])
def test_pset_create_lon_lat(fieldset, mode,
                             npart=100):
    lon = np.linspace(0, 1, npart, dtype=np.float32)
    lat = np.linspace(1, 0, npart, dtype=np.float32)
    pset = ParticleSet(fieldset, lon=lon, lat=lat,
                       pclass=ptype[mode])
    assert np.allclose([p.lon for p in pset], lon,
                       rtol=1e-12)
    assert np.allclose([p.lat for p in pset], lat,
                       rtol=1e-12)
```

Ideally, the full set of unit tests means that no change of the code can ever break another part of the code, since some of the unit tests would then fail. Of course, in reality the completeness of the unit tests can never be guaranteed, but during Parcels development we have attempted to provide

unit tests for a broad spectrum of the Parcels functionality and code.

4.2 Idealized and analytic test cases

Following the list of standard tests of particle tools, as described in Sec. 6 of Van Sebille et al. (2018), we have validated the accuracy of Parcels v0.9 against seven idealized and analytical test cases. In this section we will describe the results in detail. All test cases are run with Runga–Kutta4 (RK4) integration and in JIT mode. In each case, the hydrodynamic velocities are generated within the Python scripts and converted directly to a `FieldSet` (i.e. without first storing these fields in NetCDF format). The Python code for all test cases is available at https://doi.org/10.5281/zenodo.823994.

4.2.1 Radial rotation with known period

The first test case is that of a simple counter-clockwise solid-body rotation with a period of 24 h. Velocities are defined on a (20×20) km Arakawa A grid centred at the origin with a 100 m horizontal resolution. Solid-body radial velocities $(u, v) = (-\omega r \sin(\phi), \omega r \cos(\phi))$, with r and ϕ the radius and angle from the origin and $\omega = 2\pi/86\,400$ s the angular frequency, are then computed on that grid.

Four particles are started at $x = 0$ km and $y = (1000, 2000, 3000, 4000)$ km and then advected for 24 h, using an RK4 time step of 5 min, and with particle positions stored every hour (Fig. 4a). All four particles indeed follow the flow for the full circle. The maximum distance error after this 24 h advection is less than 3 mm, on path lengths of more than 5 km.

4.2.2 Longitudinal shear flow

The second test case tests the ability of the Parcels code to convert between spherical longitude/latitude space and local flat Euclidian space. When defining a `FieldSet` on a spherical mesh, Parcels automatically performs this conversion under the hood. To test its accuracy, an idealized flow on a sphere at 1° horizontal resolution is created, with a uniform zonal velocity of 1 m s^{-1} and no meridional velocity. A total of 31 particles are then released on a north–south line, with a meridional spacing of 3°. These particles are advected for 57 days, using an RK4 time step of 5 min and output saved every day (Fig. 4b). The main panel shows trajectories in planar projection, with the inset showing the same trajectories in orthographic projection.

At a speed of 1 m s^{-1}, the particles travel 4.9×10^6 m in the 57 days. At the Equator, this amounts to almost 45° of longitude, but because of the cosine dependence of zonal distance with latitude, particles closer to the poles travel farther in degrees (main panel in Fig. 4b). Nevertheless, the inset of Fig. 4b shows that in an orthographic projection, all particles travel the same distance.

4.2.3 Advection due to a time-oscillating zonal flow

The third test case tests the ability of Parcels to cope with simple time-varying flow. The flow in this case is a uniform meridional flow of $v = A = 0.1$ m s^{-1}, and an oscillating zonal flow with $u(t) = A\cos(\omega t)$ where $\omega = 2\pi/T$ and the period is $T = 1$ day. The time resolution of the `FieldSet` is 5 min, and since the flow is constant in space there are only two grid cells in each of the horizontal directions. A total of 20 particles are then released on a zonal line at $y = 0$ km and advected for 4 days, using an RK4 time step of 5 min and storing output every 3 h (Fig. 4c).

The analytical flow for the paths of these particles is $y(t) = At$ and $x(t) = x_0 + A/\omega \sin(\omega t)$ where $\omega = 2\pi/T$ and x_0 is the zonal start location of the particle. Indeed, all particles follow these analytical pathways very closely (Fig. 4c), with largest positional errors after 4 days being 6 cm in the zonal direction and 4 mm in the meridional direction.

4.2.4 Steady-state flow around a peninsula

The test case of steady-state flow around a peninsula follows a description by Ådlandsvik et al. (2009) and was also used as a validation test case in the article describing the CMS (Paris et al., 2013). Starting from the analytical expression for a stream function Ψ of a steady-state flow around a peninsula, analytical expressions of the zonal and meridional component of velocity are solved on a $(1° \times 0.5°)$ Arakawa A grid at $1/100°$ horizontal resolution. A set of 20 particles is seeded just off the western edge of the domain, and then advected with the flow for 24 h using an RK4 time step of 5 min and particle positions stored every hour (Fig. 4d, where the brown semi-circle is the peninsula).

Since the particles should follow streamlines, a comparison of the interpolated stream function value at $t = 24$ h to that at $t = 0$ h gives an estimate of the error. The largest error is 0.008 m^2 s^{-1}, which corresponds to a positional error of 10^{-5}°, or 1 m. Indeed, Fig. 4d shows that the particle trajectories closely follow the dashed streamlines.

4.2.5 Steady-state flow in a Stommel gyre and western boundary current

The test case of the Stommel gyre follows a description in Fabbroni (2009), and provides an analytical solution to the stream function field of a Stommel gyre and western boundary current. Here, we compute the meridional and zonal central derivatives of this stream function field to generate zonal and meridional velocities, respectively, on a $(10\,000 \times 10\,000)$ km Arakawa A grid at 50 km horizontal resolution. A set of four particles is seeded on a line crossing the western boundary, at $y = 5000$ km, and then advected for 50 days with an RK4 time step of 5 min and the particle positions stored every 24 h (Fig. 4e).

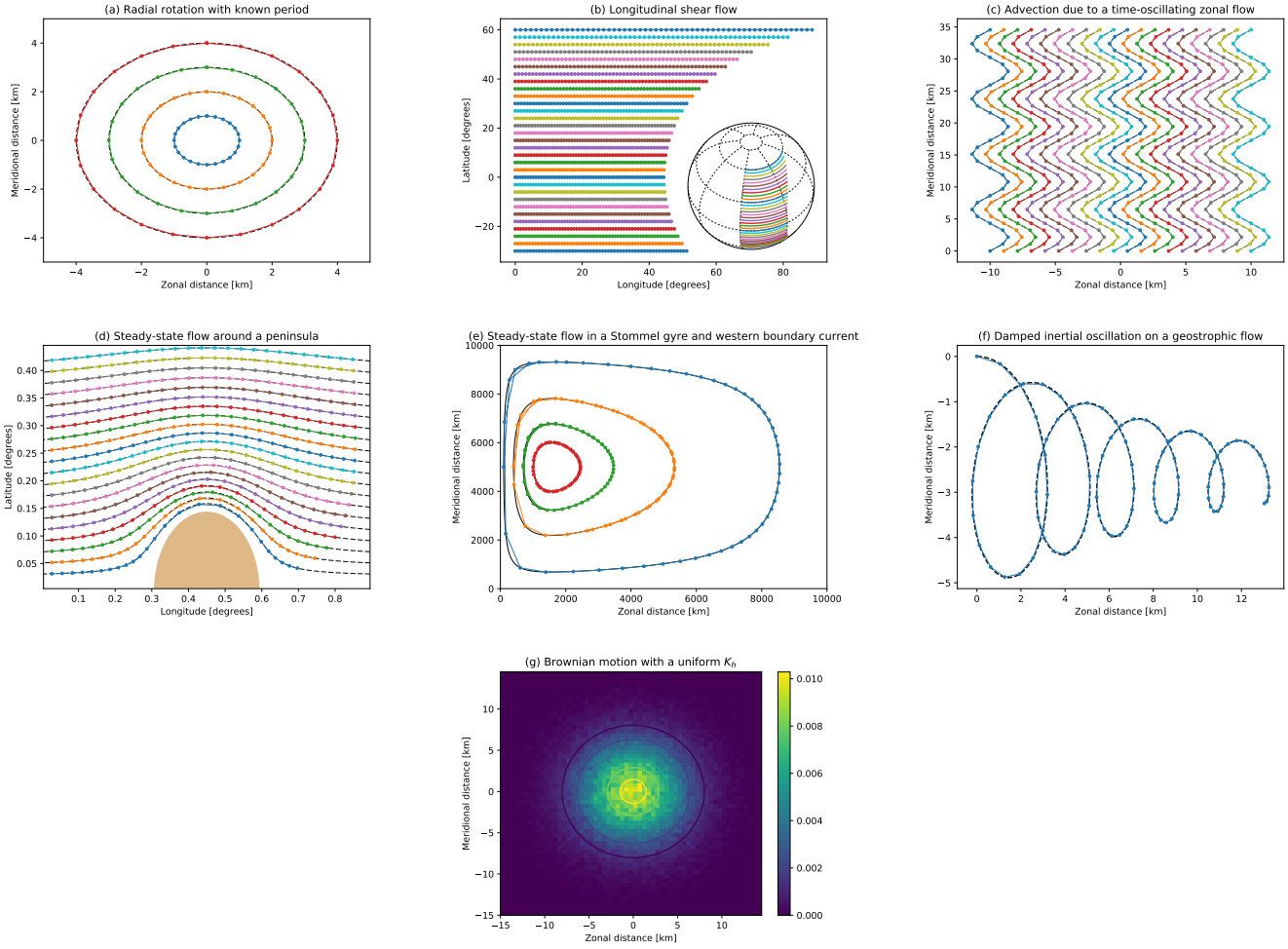

Figure 4. Evaluation of trajectory accuracy in Parcels v0.9, following the seven idealized and analytical test case described in Sect. 6 of Van Sebille et al. (2018): **(a)** radial rotation with known period; **(b)** longitudinal shear flow; **(c)** advection due to a time-oscillating zonal flow; **(d)** steady-state flow around a peninsula; **(e)** steady-state flow in a Stommel gyre and western boundary current; **(f)** damped inertial oscillation on a geostrophic flow; and **(g)** Brownian motion with a uniform K_h. In the upper six panels, the coloured lines are the particle trajectories and the black dashed lines are the analytical solutions. In panel **(g)**, the colouring shows the density of particles, and the contours show the probability density function of the equivalent analytical solution (a two-dimensional Gaussian).

Since the particles should follow streamlines, the deviation of particles from the streamlines is a measure of the accuracy of the method. Figure 4e shows that all four particles stay close to their streamline throughout the 50-day advection period. The largest error is $0.05\,\mathrm{m^2\,s^{-1}}$, which corresponds to a positional error of less than 5 km.

4.2.6 Damped inertial oscillation on a geostrophic flow

The test case of a damped inertial oscillation on a geostrophic flow follows Fabbroni (2009) and Döös et al. (2013). In this test case, the velocity varies over the entire domain, following an analytical time-dependent equation. Here, we use a time resolution of 5 min for the velocity field. A particle is then seeded at the origin and advected for 4 days, with a RK4 time step of 5 min and output stored every hour (Fig. 4f). After four days of advection, the positional error of the particle, as compared to the analytical solution, is less than 5 cm.

4.2.7 Brownian motion with uniform K_h

The test case of Brownian motion with uniform K_h tests for the accuracy and implementation of the random number generator. Here, a total of 100 000 particles are seeded at the origin of a (60×60) km grid centred around the origin with zero velocities, and then diffused using a normal variate random number distribution with $K_h = 100\,\mathrm{m^2\,s^{-1}}$. The particles are diffused for 1 day with a time step of 5 min (Fig. 4g). The two-dimensional normalized histogram agrees very well with the analytical solution of this Brownian motion: a two-dimensional Gaussian with a mean at the origin and standard deviation of $\sigma = \sqrt{2K_h\Delta t} = 4.16\,\mathrm{km}$.

5 Future outlook

As mentioned before, Parcels v0.9 is a prototype. The core contributions of this paper are both the API and the design philosophy which enables a wide range of valuable future improvements of the framework. Below, we discuss some of the conceptual ideas for these planned improvements.

5.1 Performance optimization

The primary performance optimization in version 0.9 of Parcels is the automated generation of C kernel code to allow inlining of field evaluation routines. However, several future optimizations have been planned during the design of the code, based around considerations for irregular data processing. Since dynamic addition and deletion of particles is a common feature of many oceanographic use cases, no assumptions about data layout or iteration protocol have been made in the high-level API of particle sets, allowing more optimized implementations in the future. The use of dynamic code generation at runtime also enables further automated specialization of kernel code, while allowing us to define a clear initial interface for kernel customization.

In addition to optimizing the execution of particle kernels, the extensive interaction with hydrodynamic field data constitutes a considerable cost of the overall computation – a cost that is likely to dominate overall execution if large sets of hydrodynamic field data are to be read from files. Multiple potential approaches can be considered in future versions of Lagrangian particle tracking codes:

- Directly coupled (online) runs within the host OGCM can completely avoid the bandwidth bottlenecks imposed by reading dense field data from disk, at the expense of additional computation. For simulations at local scales with a high particle density, this trade-off might prove beneficial, for example for regional studies on marine ecology.

- For global-scale models that require offline hydrodynamic field data but feature a low particle density with high localization, the total volume of data read from disk might be drastically reduced by explicitly prefetching local subsets of field data based on particle locations. Such a mechanism would require the use of additional geospatial indexing methods, for example via octrees or r-trees (Isaac et al., 2015; Schubert et al., 2013), that decompose the grid into individual sub-regions and provide fast indexing methods. Using explicit prefetch directives in the dynamic execution loop might also enable overlapping of asynchronous file reads with effective computation to further amortize file I/O overheads.

The modularity of Parcels' internal abstractions, as well as the composability of kernels and the flexibility provided by the dynamic execution loop, should facilitate extensive experimentation and exploration with such advanced optimization techniques, without the need for users to change any high-level algorithmic definitions. The use of advanced data handling and task-scheduling libraries, such as Dask (Rocklin, 2015) or Xarray (Hoyer and Hamman, 2017), might also be utilized to quickly achieve efficient out-of-core data management in Parcels.

5.1.1 Towards parallelization

The current version of Parcels is not in itself parallel due to two restrictions:

- The primary input format of field data in the v0.9 prototype is NetCDF-based field data, so that parallelization requires an explicit domain decomposition and a parallel file reader. The current version of the `netcdf` Python package does not provide these features. Alternative implementations of the NetCDF file format, such as Xarray, might be leveraged in future versions of Parcels to provide parallel data management.

- Exchanging particle information between parallel processors is currently not supported, although it is deemed a critical feature for the next release (v1.0).

5.2 Community building and kernel sharing

One of the key ideas between the development of Parcels is for it to be a flexible and extendable codebase, where particle behaviour can easily be customized. The worked-out example in Sect. 3 shows that many types of behaviour (sinking, aging, etc) can be coded in a few lines of Python code.

The customizability of Parcels enables a multitude of oceanographic modelling, from water parcels to plankton to plastic litter to fish. We therefore envision an active community of Parcels users who share and discuss kernel development. We encourage anyone who wishes to share their custom kernels to upload them onto GitHub, and we will provide a properly referenced library of user-contributed kernels for others to reuse on www.oceanparcels.org.

5.3 Towards runtime integration with OGCMs

Although the current version of Parcels primarily uses offline field data, the overall design of the particle execution engine is designed to be compatible with a variety of OGCMs for directly coupled (at-runtime) simulations. In particular, the current `Field` interface can easily be extended to provide interpolation routines for various types of field data, for example based on unstructured meshes, while the primary particle update loop provides a mechanism for host models to dictate a model time-step size that varies from that of the particle update. Moreover, the explicit generation of C code allows Parcels kernel code to be easily injected into existing ocean modelling frameworks, while the provision of error-

recovery kernels can guarantee progression of the coupled model.

6 Conclusions

Here, we have introduced a new framework for Lagrangian ocean analysis that focuses on customizability, flexibility, and ease of use. This v0.9 of Parcels is very much a prototype, providing a proof of concept of the API and showcasing how it can be used to create high-level Python code for fully fledged scientific experiments. We also assess the accuracy of the current implementation, with the idea to provide a benchmark for future versions. Future development will focus on increasing efficiency of the framework, and also towards providing easy tools to port the generated C code of Parcels experiments to at-runtime integration within OGCMs.

Code availability. The code for Parcels is licensed under the MIT licence and is available through GitHub at www.github.com/OceanParcels/parcels. The version 0.9 described here is archived at Zenodo at https://doi.org/10.5281/zenodo.823562. More information is available on the project webpage at www.oceanparcels.org.

Author contributions. ML and EvS developed the code and wrote the paper jointly.

Competing interests. The authors declare that they have no conflict of interest.

Acknowledgements. The initial ideas for the Parcels framework were the result of very fruitful discussions with the attendees of the "Future of Lagrangian Ocean Modelling" workshop, held at Imperial College London, UK, in September 2015. Funding for this workshop was provided through an EPSRC Institutional Sponsorship grant to EvS under reference number EP/N50869X/1. Erik van Sebille is supported through funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement no. 715386). The OFES simulation was conducted on the Earth Simulator under the support of JAMSTEC. We thank Joe Scutt-Phillips, Ronan McAdam, Joel Kronberg, Thomas Stokes, Nathaniel Tarshish, Michael Hart-Davis, Birgit Sutzl, Ben Snowball, Samuel Wetherell, and David Ham for their support in testing and developing aspects of the Parcels code. We thank the reviewers Yu Cheng and Joakim Kjellsson for their very helpful comments.

Edited by: Robert Marsh
Reviewed by: Joakim Kjellsson and Yu Cheng

References

Ådlandsvik, B., Bartsch, J., Brickman, D., Browman, H. I., Edwards, K., Fiksen, Ø., Gallego, A., Hermann, A. J., Hinckley, S., Houde, E., Huret, M., Irisson, J.-O., Lacroix, G., Leis, J. M., McCloghrie, P., Megrey, B. A., Miller, T., Van der Molen, J., Mullon, C., North, E. W., Parada, C., Paris, C. B., Pepin, P., Petitgas, P., Rose, K., Thygesen, U. H., and Werner, C.: Manual of recommended practices for modelling physical – biological interactions during fish early life, Tech. rep., 118 pp., 2009.

Alexander, R.: Solving ordinary differential equations I: Nonstiff problems (E. Hairer, SP Norsett, and G. Wanner), SIAM Rev., 32, 485, 1990.

Beron-Vera, F. J. and LaCasce, J. H.: Statistics of Simulated and Observed Pair Separations in the Gulf of Mexico, J. Phys. Oceanogr., 46, 2183–2199, 2016.

Blanke, B. and Raynaud, S.: Kinematics of the Pacific Equatorial Undercurrent: An Eulerian and Lagrangian approach from GCM results, J. Phys. Oceanogr., 27, 1038–1053, 1997.

Cetina-Heredia, P., Roughan, M., van Sebille, E., Feng, M., and Coleman, M. A.: Strengthened currents override the effect of warming on lobster larval dispersal & survival, Glob. Change Biol., 21, 4377–4386, 2015.

Cowen, R. K., Paris, C. B., and Srinivasan, A.: Scaling of connectivity in marine populations, Science, 311, 522–527, 2006.

Doblin, M. A. and van Sebille, E.: Drift in ocean currents impacts intergenerational microbial exposure to temperature, P. Natl. Acad. Sci. USA, 113, 5700–5705, 2016.

Döös, K.: Interocean Exchange of Water Masses, J. Geophys. Res.-Oceans, 100, 13499–13514, 1995.

Döös, K., Nycander, J., and Coward, A. C.: Lagrangian decomposition of the Deacon Cell, J. Geophys. Res.-Oceans, 113, C07028, https://doi.org/10.1029/2007JC004351, 2008.

Döös, K., Kjellsson, J., and Jonsson, B. F.: TRACMASS-A Lagrangian Trajectory Model, in: Preventive Methods for Coastal Protection, Springer International Publishing, Heidelberg, 225–249, 2013.

Döös, K., Jönsson, B., and Kjellsson, J.: Evaluation of oceanic and atmospheric trajectory schemes in the TRACMASS trajectory model v6.0, Geosci. Model Dev., 10, 1733–1749, https://doi.org/10.5194/gmd-10-1733-2017, 2017.

Fabbroni, N.: Numerical simulations of passive tracers dispersion in the sea, PhD thesis, Universita di Bologna, 2009.

Griffa, A.: Applications of stochastic particle models to oceanographic problems, in: Stochastic modelling in physical oceanography, Springer, 113–140, 1996.

Haller, G.: Lagrangian Coherent Structures, Annu. Rev. Fluid Mech., 47, 137–162, https://doi.org/10.1146/annurev-fluid-010313-141322, 2015.

Hellweger, F. L., van Sebille, E., and Fredrick, N. D.: Biogeographic patterns in ocean microbes emerge in a neutral agent-based model, Science, 345, 1346–1349, 2014.

Hoyer, S. and Hamman, J.: xarray: N-D labeled Arrays and Datasets in Python, Journal of Open Research Software, 5, 10, https://doi.org/10.5334/jors.148, 2017.

Isaac, T., Burstedde, C., Wilcox, L. C., and Ghattas, O.: Recursive Algorithms for Distributed Forests of Octrees, SIAM J. Sci. Comput., 37, C497–C531, https://doi.org/10.1137/140970963, 2015.

Jones, A., Thomson, D., Hort, M., and Devenish, B.: The U.K. Met Office's Next-Generation Atmospheric Dispersion Model, NAME III, in: Air Pollution Modeling and Its Application XVII, edited by: Borrego, C. and Norman, A.-L., Springer US, Boston, MA, 580–589, 2007.

Jönsson, B. F., Salisbury, J. E., and Mahadevan, A.: Large variability in continental shelf production of phytoplankton carbon revealed by satellite, Biogeosciences, 8, 1213–1223, https://doi.org/10.5194/bg-8-1213-2011, 2011.

Jutzeler, M., Marsh, R., Carey, R. J., White, J. D. L., Talling, P. J., and Karlstrom, L.: On the fate of pumice rafts formed during the 2012 Havre submarine eruption, Nat. Commun., 5, 3660, 2014.

Katz, M. E., Cramer, B. S., Franzese, A. M., Hoenisch, B., Miller, K. G., Rosenthal, Y., and Wright, J. D.: Traditional and emerging geochemical proxies in foraminifera, J. Foramin. Res., 40, 165–192, 2010.

Kool, J. T., Moilanen, A., and Treml, E. A.: Population connectivity: recent advances and new perspectives, Landscape Ecol., 28, 165–185, https://doi.org/10.1007/s10980-012-9819-z, 2013.

Lebreton, L. C. M., Greer, S. D., and Borerro, J. C.: Numerical modelling of floating debris in the world's oceans, Mar. Pollut. Bull., 64, 653–661, 2012.

Marsh, R., Ivchenko, V. O., Skliris, N., Alderson, S., Bigg, G. R., Madec, G., Blaker, A. T., Aksenov, Y., Sinha, B., Coward, A. C., Le Sommer, J., Merino, N., and Zalesny, V. B.: NEMO-ICB (v1.0): interactive icebergs in the NEMO ocean model globally configured at eddy-permitting resolution, Geosci. Model Dev., 8, 1547–1562, https://doi.org/10.5194/gmd-8-1547-2015, 2015.

Masumoto, Y., Sasaki, H., Kagimoto, T., Komori, N., Ishida, A., Sasai, Y., Miyama, T., Motoi, T., Mitsudera, H., Takahashi, K., Sakuma, H., and Yamagata, T.: A fifty-year eddy-resolving simulation of the world ocean – Preliminary outcomes of OFES (OGCM for the Earth Simulator), Journal of the Earth Simulator, 1, 35–56, 2004.

Paris, C. B., Cowen, R. K., Claro, R., and Lindeman, K. C.: Larval transport pathways from Cuban snapper (Lutjanidae) spawning aggregations based on biophysical modeling, Mar. Ecol.-Prog. Ser., 296, 93–106, 2005.

Paris, C. B., Helgers, J., van Sebille, E., and Srinivasan, A.: Connectivity Modeling System: A probabilistic modeling tool for the multi-scale tracking of biotic and abiotic variability in the ocean, Environ. Modell. Softw., 42, 47–54, 2013.

Peeters, F. J. C., Acheson, R., Brummer, G.-J. A., de Ruijter, W. P. M., Schneider, R. R., Ganssen, G. M., Ufkes, E., and Kroon, D.: Vigorous exchange between the Indian and Atlantic oceans at the end of the past five glacial periods, Nature, 430, 661–665, 2004.

Pingali, K., Nguyen, D., Kulkarni, M., Burtscher, M., Hassaan, M. A., Kaleem, R., Lee, T.-H., Lenharth, A., Manevich, R., Méndez-Lojo, M., Prountzos, D., and Sui, X.: The Tao of Parallelism in Algorithms, SIGPLAN Not., 46, 12–25, https://doi.org/10.1145/1993316.1993501, 2011.

Qin, X., Menviel, L., Sen Gupta, A., and van Sebille, E.: Iron sources and pathways into the Pacific Equatorial Undercurrent, Geophys. Res. Lett., 43, 9843–9851, https://doi.org/10.1002/2016GL070501, 2016.

Rocklin, M.: Dask: Parallel Computation with Blocked algorithms and Task Scheduling, in: Proceedings of the 14th Python in Science Conference, edited by: Huff, K. and Bergstra, J., 130–136, 2015.

Rühs, S., Durgadoo, J. V., Behrens, E., and Biastoch, A.: Advective timescales and pathways of Agulhas leakage, Geophys. Res. Lett., 40, 3997–4000, https://doi.org/10.1002/grl.50782, 2013.

Schubert, E., Zimek, A., and Kriegel, H.-P.: Geodetic Distance Queries on R-Trees for Indexing Geographic Data, Springer Berlin Heidelberg, Berlin, Heidelberg, 146–164,, https://doi.org/10.1007/978-3-642-40235-7_9, 2013.

Stohl, A. and James, P.: A Lagrangian analysis of the atmospheric branch of the global water cycle. Part II: Moisture transports between earth's ocean basins and river catchments, J. Hydrometeorol., 6, 961–984, 2005.

Teske, P. R., Sandoval-Castillo, J., van Sebille, E., Waters, J., and Beheregaray, L. B.: On-shelf larval retention limits population connectivity in a coastal broadcast spawner, Mar. Ecol.-Prog. Ser., 532, 1–12, 2015.

Van Sebille, E., Scussolini, P., Durgadoo, J. V., Peeters, F. J. C., Biastoch, A., Weijer, W., Turney, C. S. M., Paris, C. B., and Zahn, R.: Ocean currents generate large footprints in marine palaeoclimate proxies, Nat. Commun., 6, 6521, 2015.

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., Deleersnijder, E., Doöös, K., Drake, H., Drijfhout, S., Gary, S. F., Heemink, A. W., Kjellsson, J., Koszalka, I. M., Lange, M., Lique, C., MacGilchrist, G. A., Marsh, R., Mayorga Adame, C. G., McAdam, R., Nencioli, F., Paris, C. B., Piggott, M. D., Polton, J. A., Rühs, S., Shah, S. H. A. M., Thomas, M. D., Wang, J., Wolfram, P. J., Zanna, L., and Zika, J. D.: Lagrangian ocean analysis: fundamentals and practices, Ocean Model., under review, 2018.