Geosci. Model Dev. Discuss.,
https://doi.org/10.5194/gmd-2017-167-RC2, 2017

Interactive comment on "Parcels v0.9: prototyping a Lagrangian Ocean Analysis framework for the petascale age" *by* Michael Lange and Erik van Sebille

Y. Cheng (Referee)

yucheng@rsmas.miami.edu

Received and published: 24 August 2017

The authors introduce a new Lagrangian Ocean Analysis framework, Parcels that is built with flexibility, scalability, and ease-of-use in mind. This manuscript serves well as a proof-of-concept and demonstrates its user-friendly interfaces and the accuracy of the codes. However, the claimed improvement of performance, scalability for large datasets and ease to integrated with different OGCMs are yet to be seen.

The authors' effort to build Parcels following modern software engineering practice is much appreciated. From my experience, none of the existing tools, although gradually migrated to Github, use CI tools, making them difficult to be implemented to and tested on different machines. This could be one of the reasons that each of community only reaches a certain size. Positioning itself as a framework, together with its modular design, Parcels has much more potentials than any existing tools to attract more users from different fields. At its early stage of development, Parcels urgently needs the input from a larger community, and this paper is a well-constructed invitation. I recommend the paper to be published once some of following points are addressed. Also, some of my questions and comments are also listed:

- P.2 line 6-8: Is there a limit that you expect each of three codes will not be computationally efficient anymore? From my own experience, instead of computation, the bottle neck of CMS is in the "output to NetCDF" step. Opening a huge NetCDF file and dumping particle information into it drain the system memory (32G/16core node) and dragged down the whole machine.

- P.2 line 28: It is not clear to me what "generic particle-mesh interaction computations." is.

- P.5 line 29: Is this limited to a particular JIT package? How much faster is using JITParticle than using ScipyParticle? It'd be good to see a benchmark comparison. If I want to use a new Interpolation scheme, how can I implement it into both modes?

- P.8 line 15 and the codes. Please correct me if my understanding is incorrect: The "snapshots" was defined in the code to load the data of initial three time steps. The for-loop was so designed because the particles are back tracked. The pset.execute uses the loaded filedset to advect particles in these three days (runtime) with time step (5min) and output every one day, and then "fldset.advancetime" is called to replace the last snapshot by concatenating the newly loaded snapshot to the front.

- P.7 line 8, and P.8 line 15: It is confusing to me that "Snapshots" was first a list of three members. Within the loop, it got updated to a single member list, which is the condition to trigger advancetime. Also, it might be better to point out that the "set_ofes_fieldset" is a user-defined function, not included in the Parcels package.

- Is there a particular reason that field.py uses the native Netcdf4 API to load in fields, but uses Xarray to write to output files? Performance concerned?

- Section 4.2: How did you generate the idealized flow fields? It might be trivial, but the grid-size information is missing for some cases.

- Section 4.2.7: I am just curious why $$K_{h}=100 m^{2}/s$$? From my experience, this value depends on resolution. Let's say if I want to use Parcels with 1/10 deg OFES, what value should I use? Is there a test case that I can tune this value against some observations?

- P.12 line 16: Not exactly clear what "Spatial indexing methods" indicates. Also, this sentence sounds not clear to me: How will something "impacted" become promising in the future? How are you planning to do the "close integration of such scheduling techniques with the particle loop?"

- Section 5.1.1: How are you planning to proceed on this front? I know there is a "Parallel NetCDF" project, but I don't think its Python API will be available anytime soon. How much of performance gain could be achieved once the second point is addressed in version 1.0?

- Section 5.3: This is exciting! It would be wonderful to be able to couple Parcels to many ocean models. However, to do so, Parcels need to work on their native grids. It is possible to run a remapping layer in between, but I think that will significantly impact the performance. Just curious, could you briefly suggest how to modify the field.py to handle the non-regular grids, for example, the tri-polar grid in POP2, or the unstructured triangular meshes in FESOM? What's your current plan to support multiple grids? Or will that be left to the users?