

# A fast algorithm for neutrally-buoyant Lagrangian particles in numerical ocean modeling

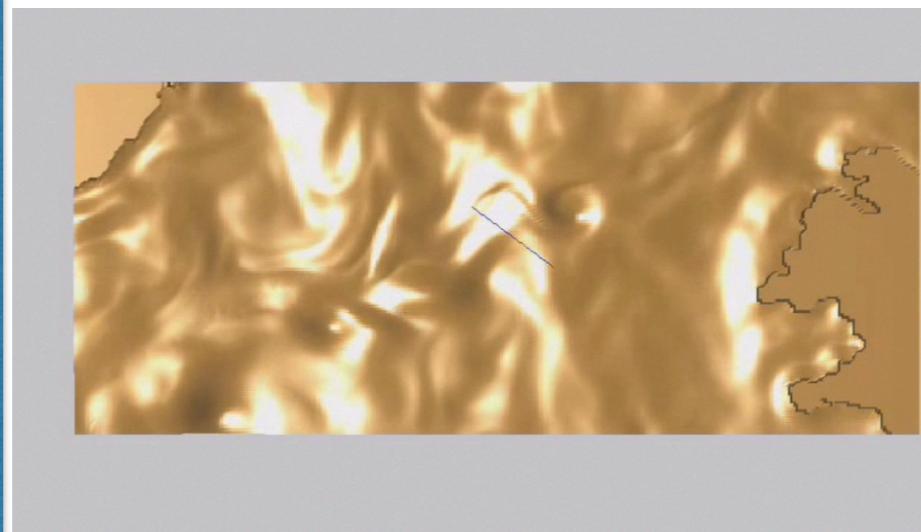
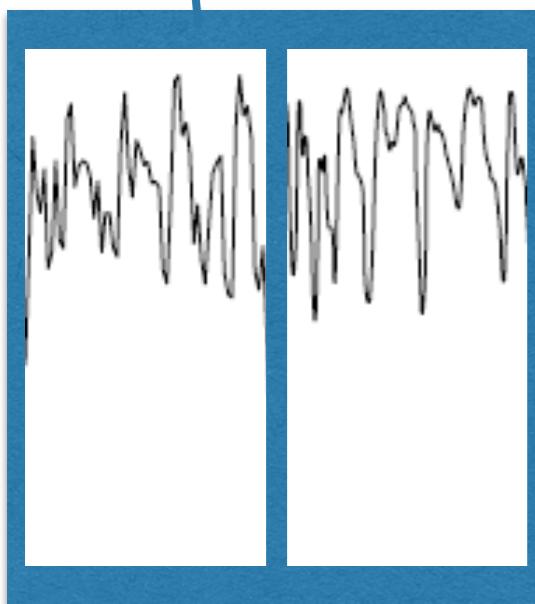
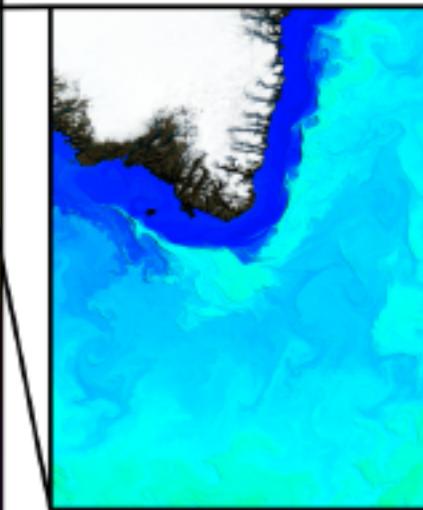
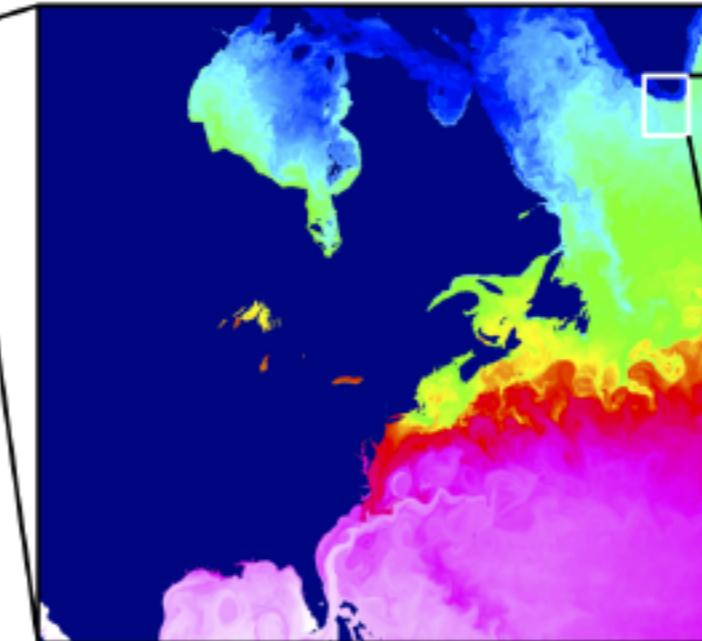
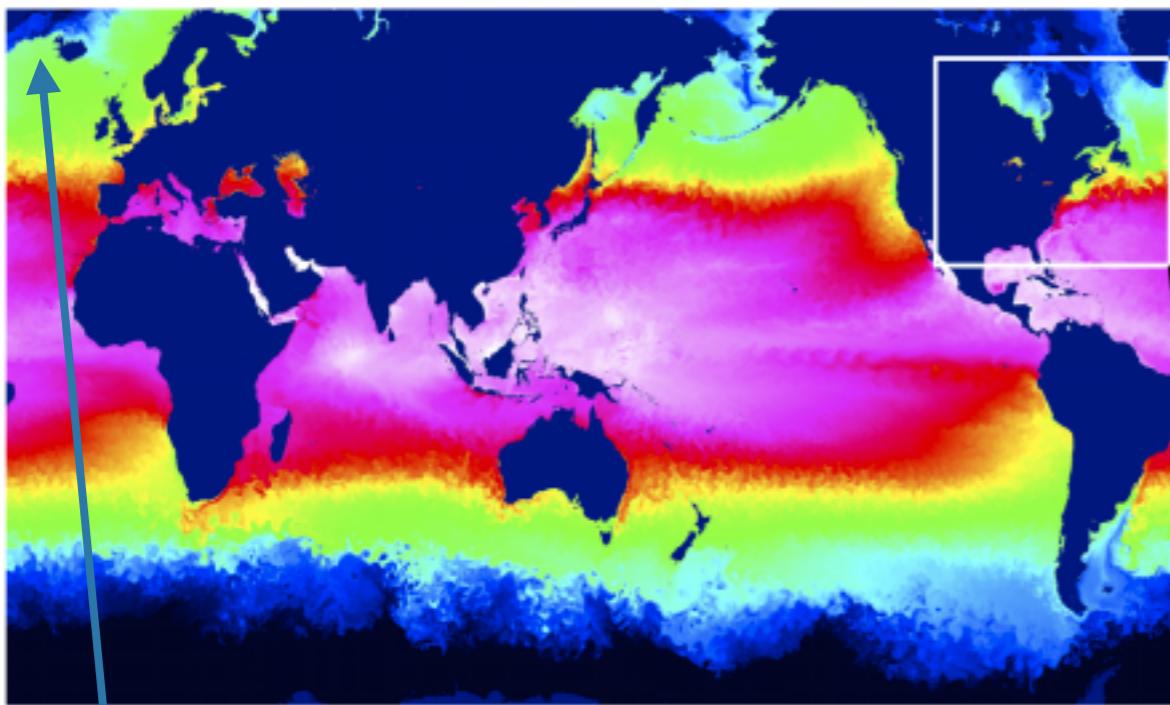
**Thomas Haine**

Renske Gelderloos

Alex Szalay

Gerard Lemson

# Ultra-high-res. ocean models are now highly realistic, revealing,



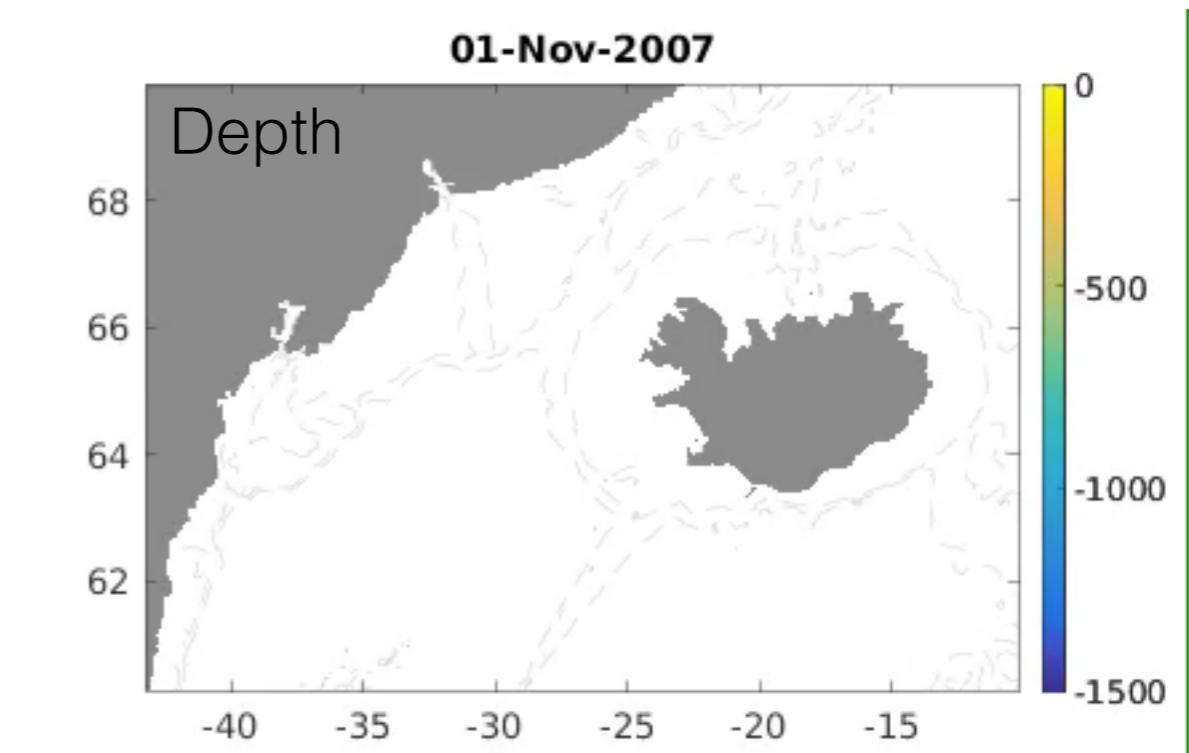
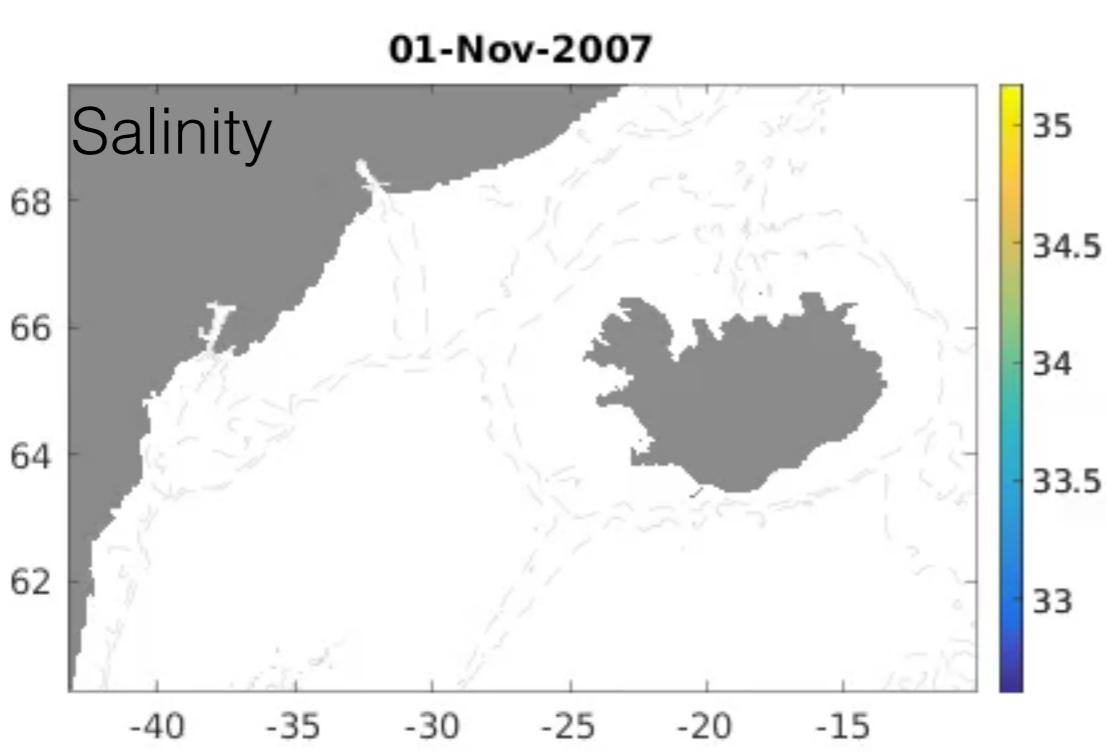
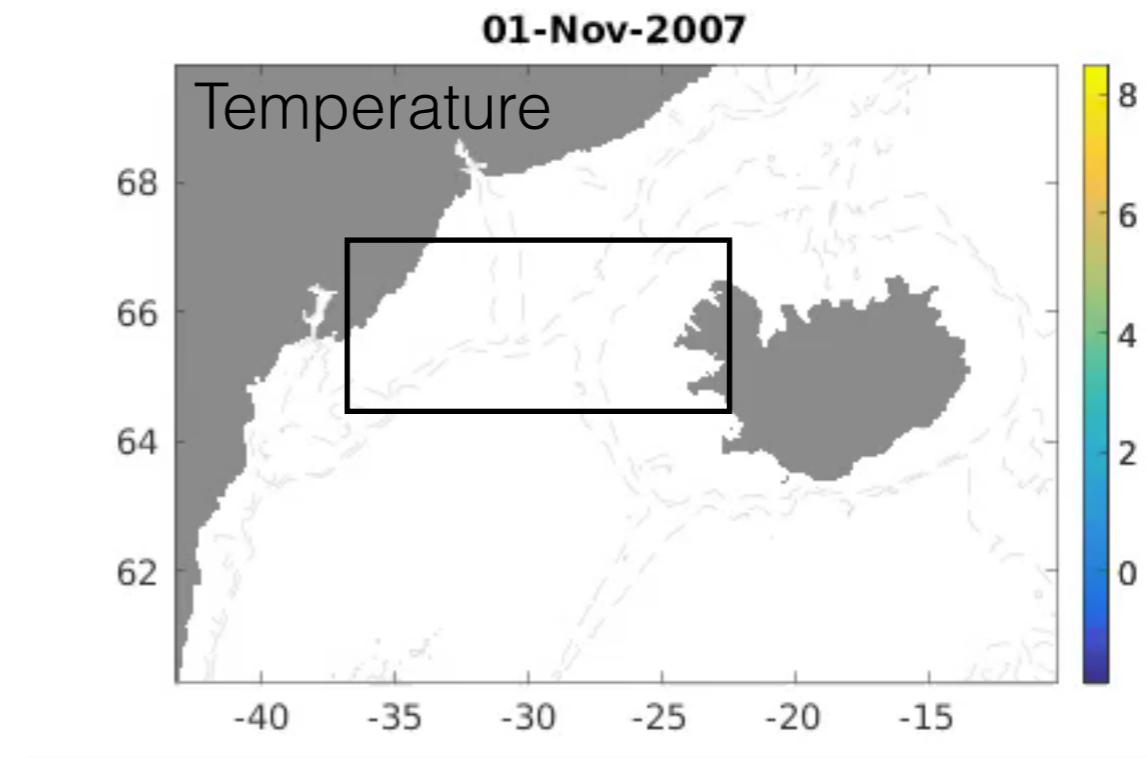
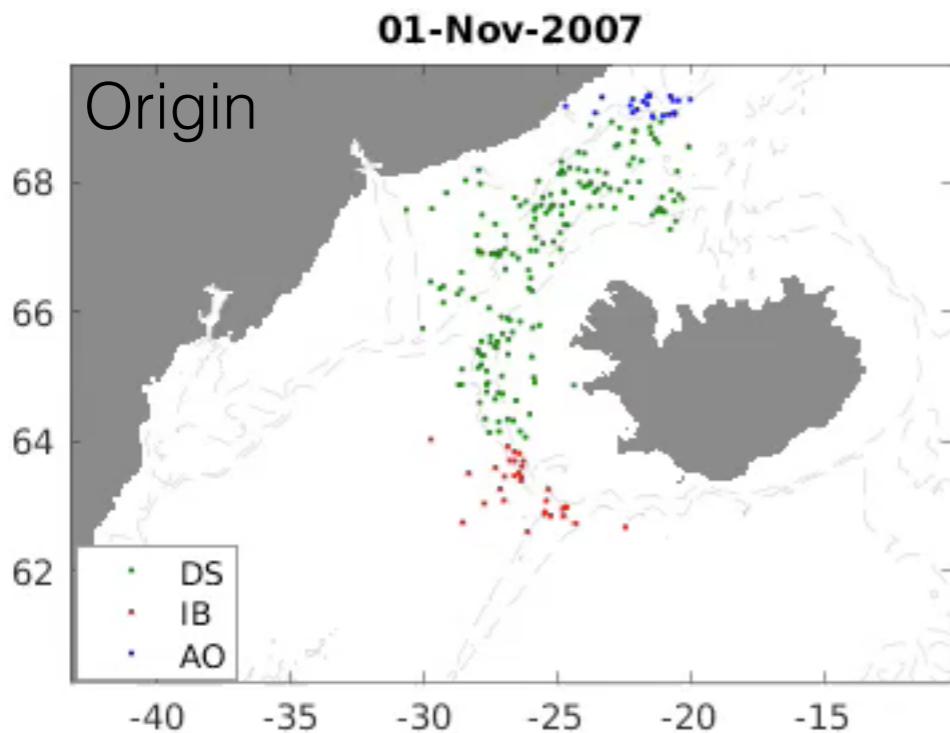
About  $10^{10-11}$  numbers per snapshot  
 $10^{3-6}$  snapshots stored per run  
=  $10^{13-17}$  nos. per run

Source: Chris Hill, MIT, <http://mitgcm.org>, Haine, 2010.



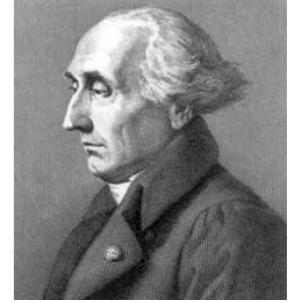
# and need good tools for post processing

Gelderloos et al., 2016b



# Lagrangian particle models

$$\begin{aligned}\frac{d\mathbf{x}_i(t)}{dt} &= \mathbf{u}(\mathbf{x}_i, t) \\ T_i(t) &= T(\mathbf{x}_i, t)\end{aligned}$$



Two types of offline Lagrangian particle-tracking models available:

1. **Analytical models**: Very fast, but assume stationarity between model samples —> inaccurate
2. **Numerical model**: CMS most widely used example, needs Unix and unphysical solid boundary conditions

Our code is numerical with correct boundary-sliding conditions, but very slow —> needs speeding up

# Old implementation

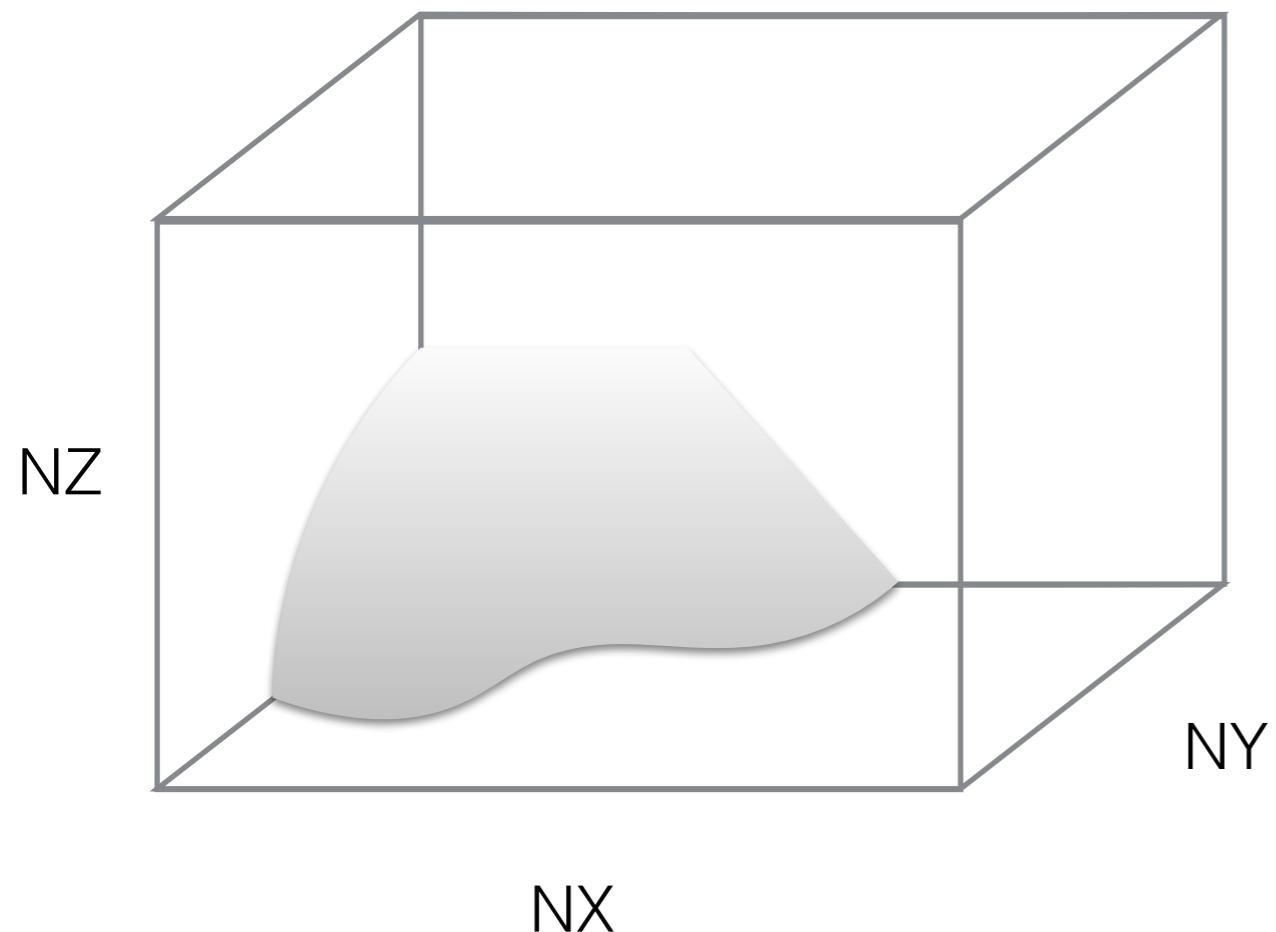
Koszalka et al., 2013

# Old implementation

Four stages:

1. **Initialization**: model domain, grid, bathymetry and initial particle positions

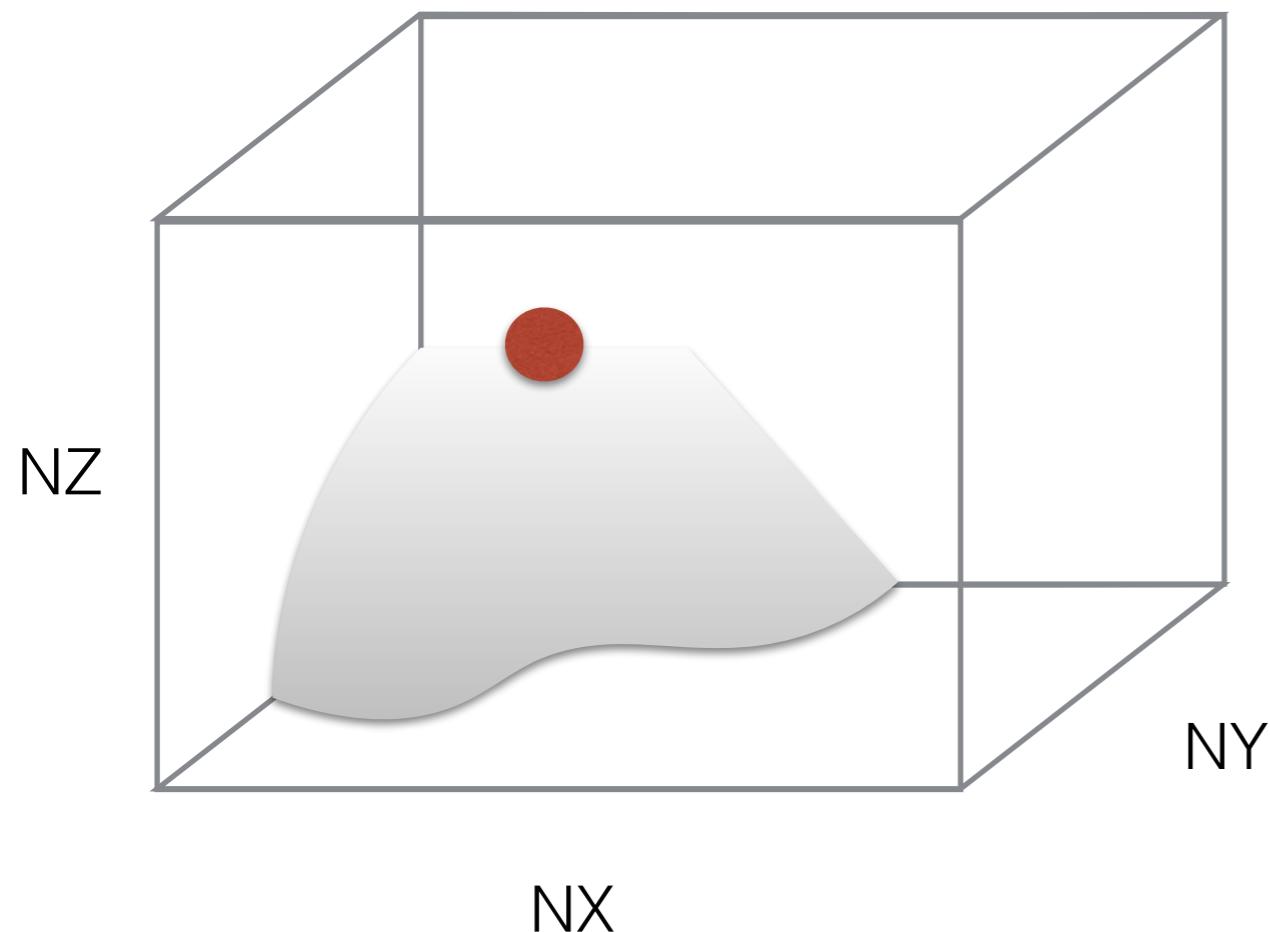
# Old implementation: Step 1



Load grid and bathymetry info

$$H(\mathbf{x})$$

# Old implementation: Step 1



Load grid and bathymetry info

Seed particles in the domain

$$\mathbf{x}_i(t = 0)$$

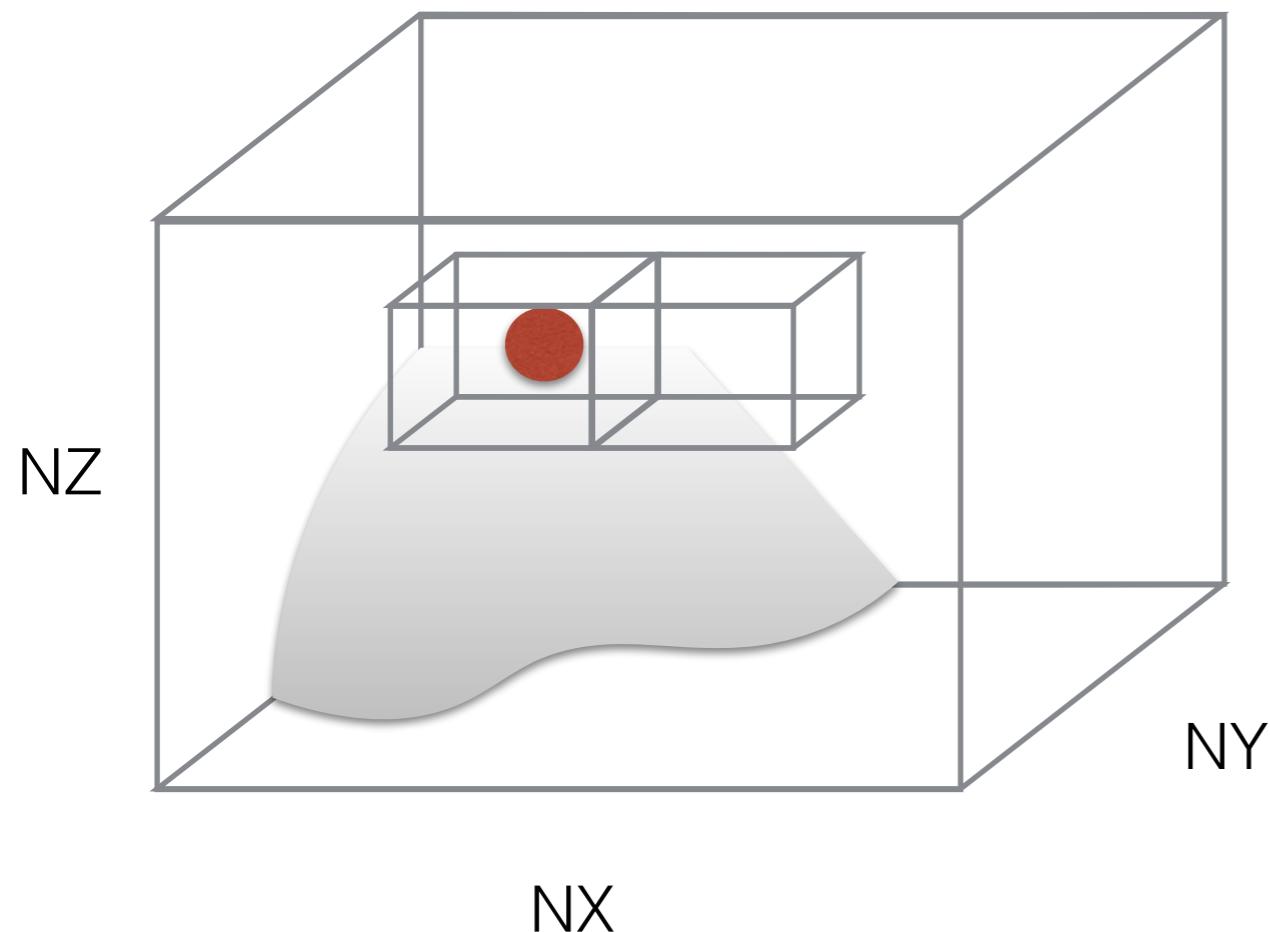
# Old implementation

Four stages:

1. **Initialization**: model domain, grid, bathymetry and initial particle positions
2. Calculate **particle trajectories**: for predetermined length of time, trajectory is calculated based on ocean model velocity fields

$$\frac{d\mathbf{x}_i(t)}{dt} = \mathbf{u}(\mathbf{x}_i, t)$$

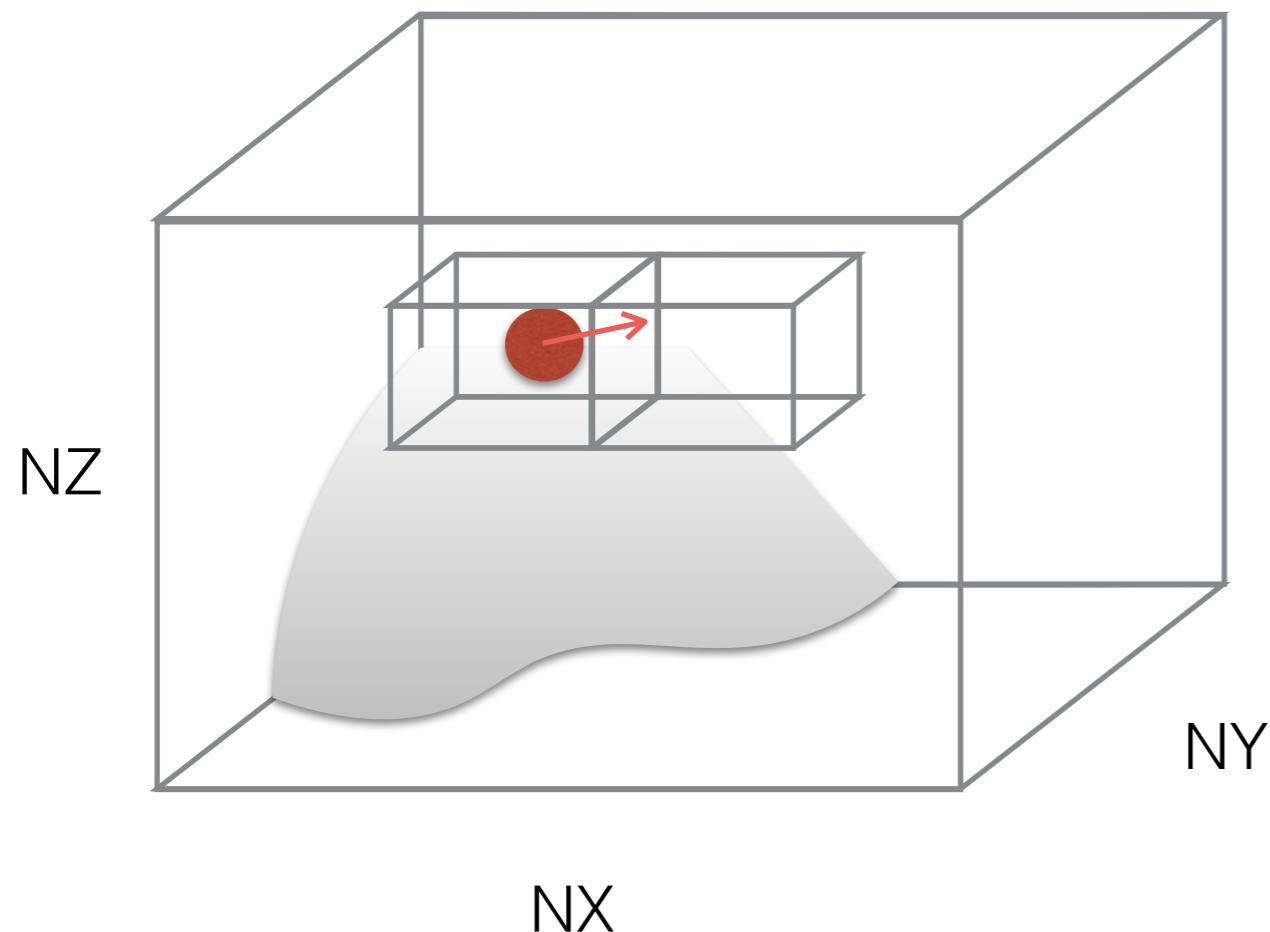
# Old implementation: Step 2



Load 2 sequential  
3D velocity fields

Create a local environment  
around the particle (necessary  
for old `interpn` function)

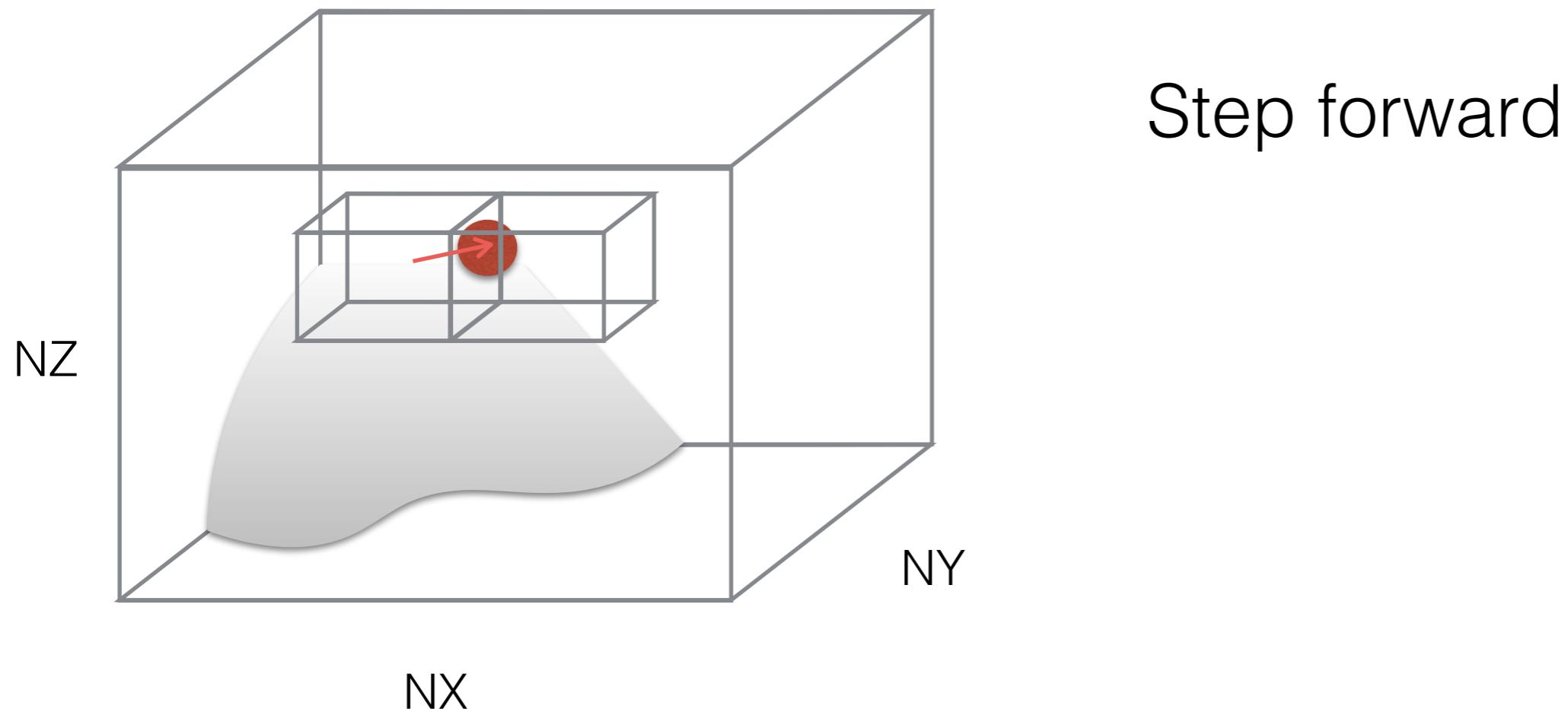
# Old implementation: Step 2



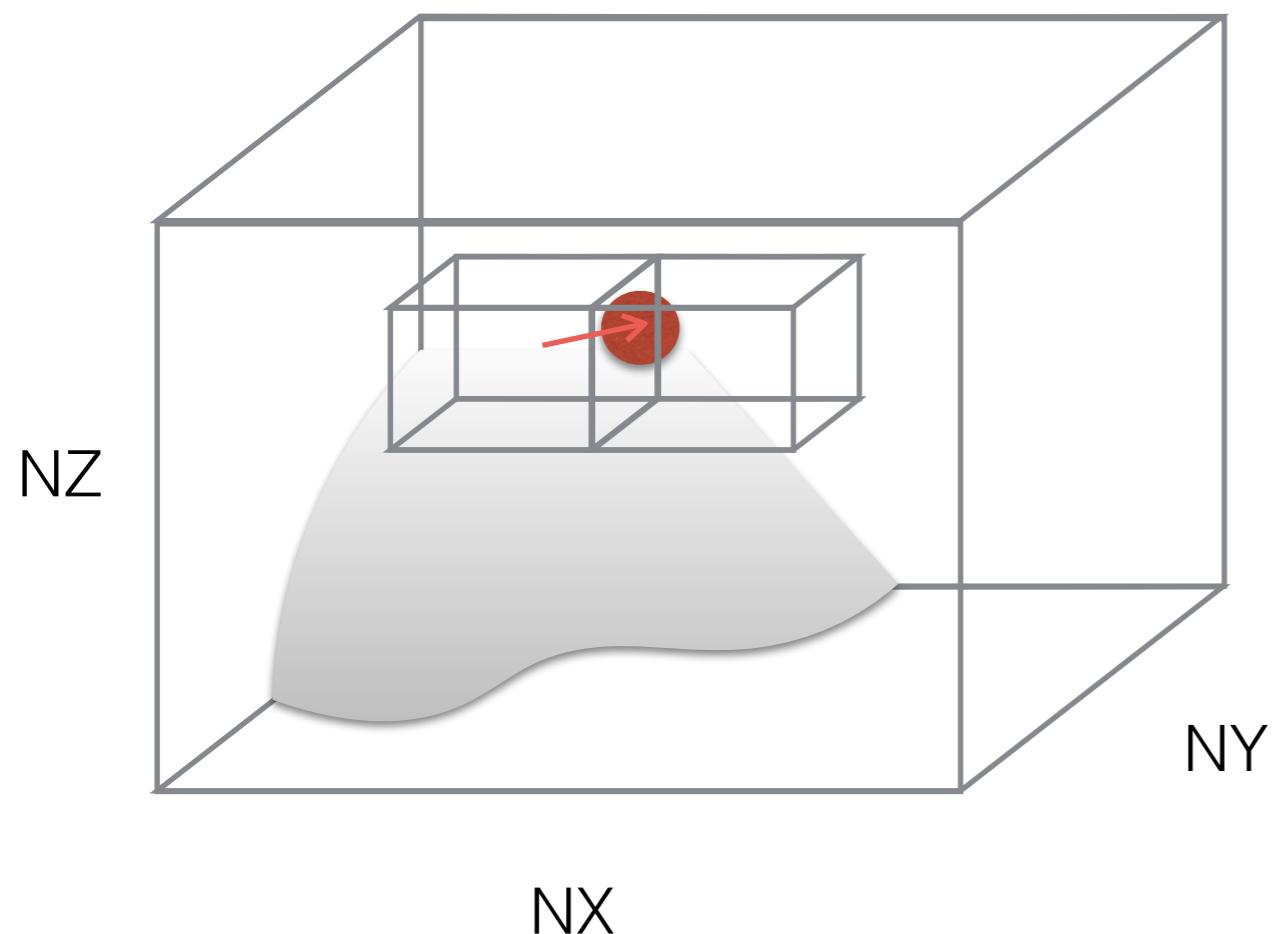
Calculate next piece  
of the trajectory

Explicit Runge-Kutta (2,3)-pair ODE  
solver for moderately stiff problems

# Old implementation: Step 2



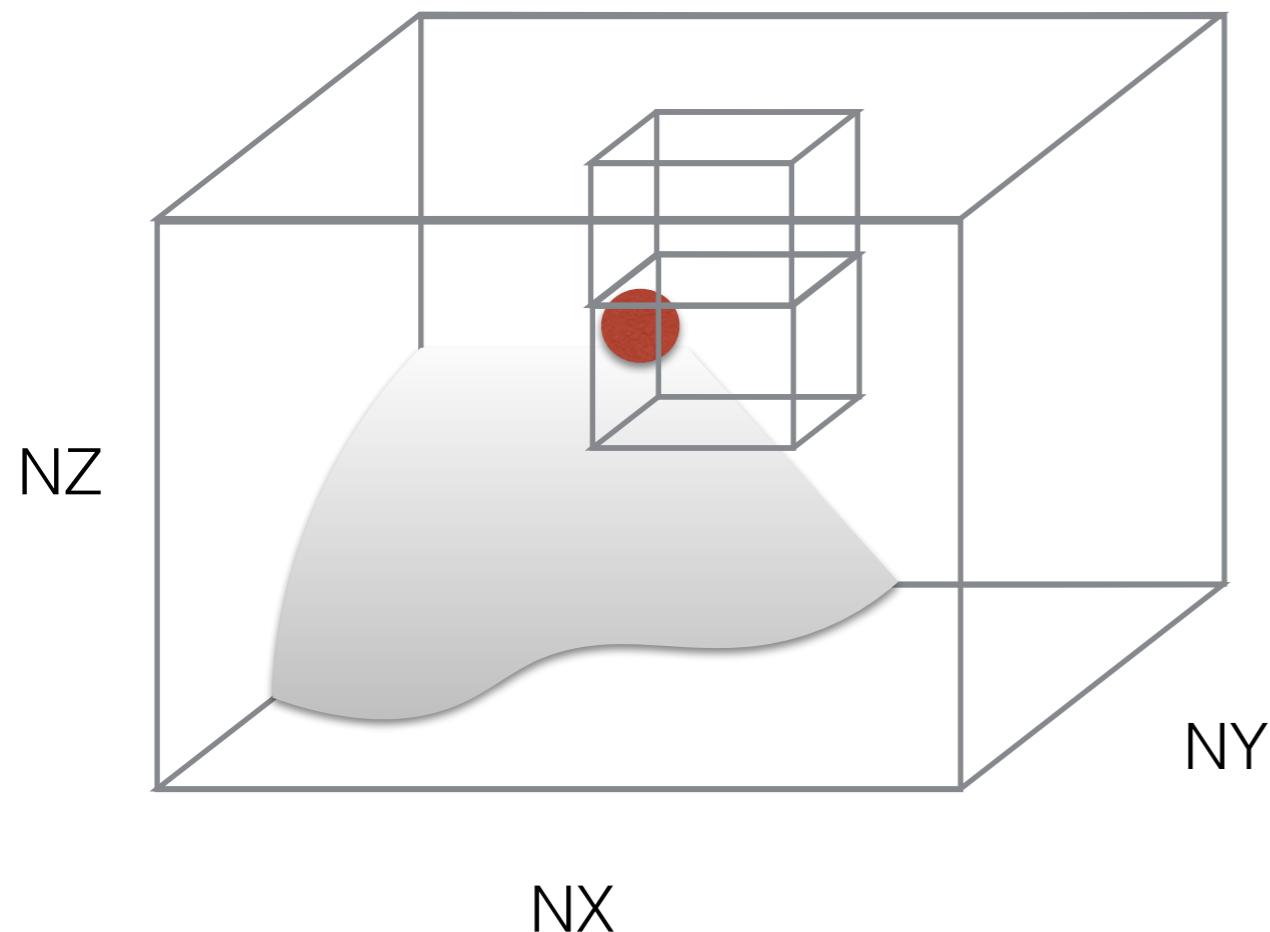
# Old implementation: Step 2



Interrupt if:

- 1) particle moves to another cell
- 2) particle hits the bathymetry

# Old implementation: Step 2

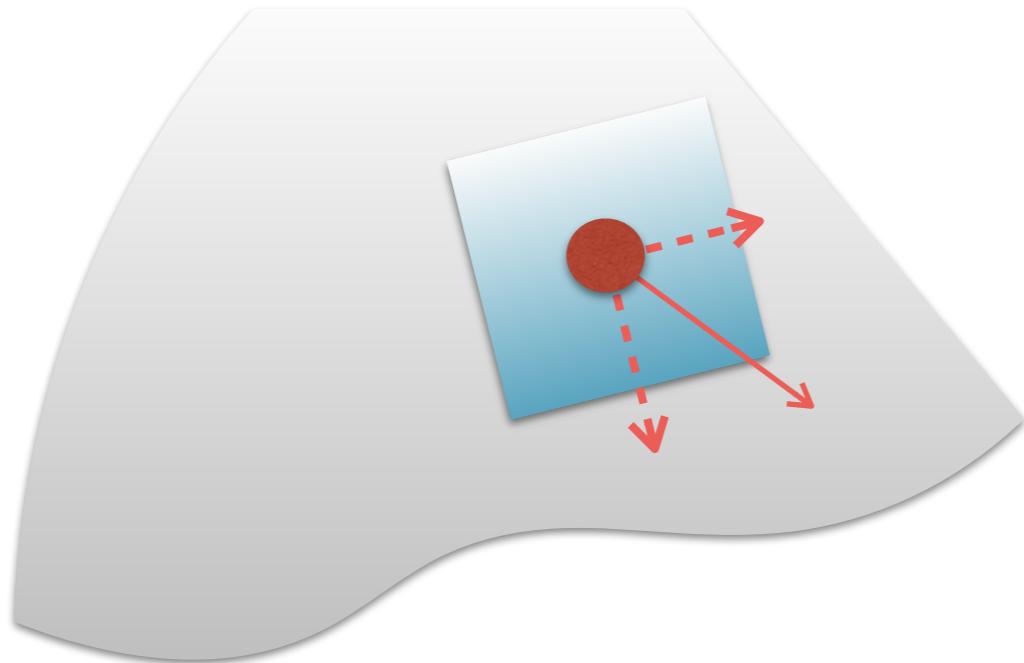


Interrupt if:  
1) particle moves to another cell

Get new local  
neighborhood

# Old implementation: Step 2

Interrupt if:  
2) particle hits the  
bathymetry



Switch to along-bathymetry  
sliding mode

# Old implementation: Step 2

*Sequential implementation* because:

- Required to handle small neighborhoods (because `interpn` used to be very sensitive to cutout size)
- Timings of interrupts are unknown *a priori*

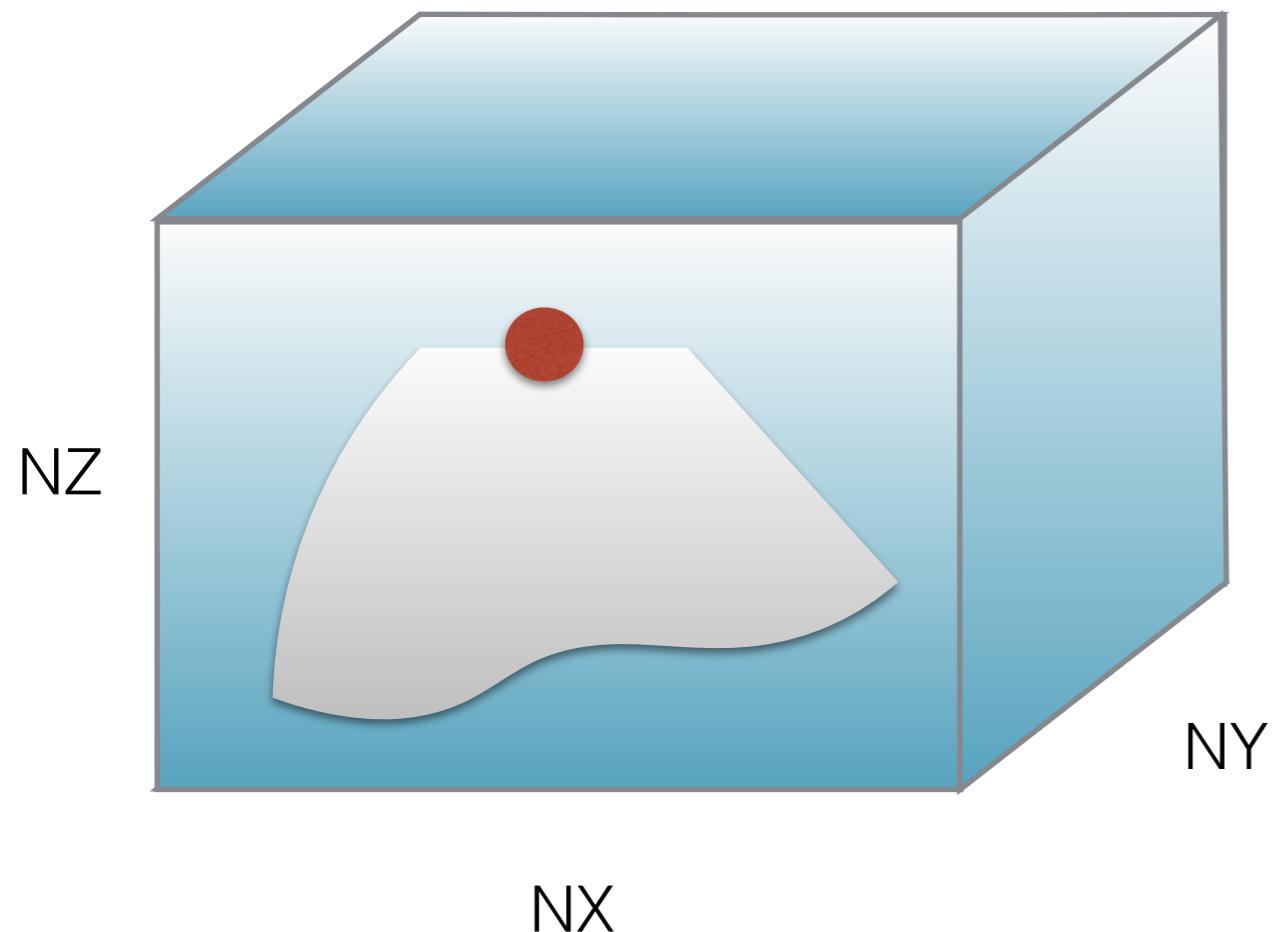
# Old implementation

Four stages:

1. **Initialization**: model domain, grid, bathymetry and initial particle positions
2. Calculate **particle trajectories**: for predetermined length of time, trajectory is calculated based on ocean model velocity fields
3. Particle **property extraction**: Temperature/salinity along trajectory are found by interpolation of model T/S fields

$$T_i(t) = T(\mathbf{x}_i, t)$$

# Old implementation: Step 3



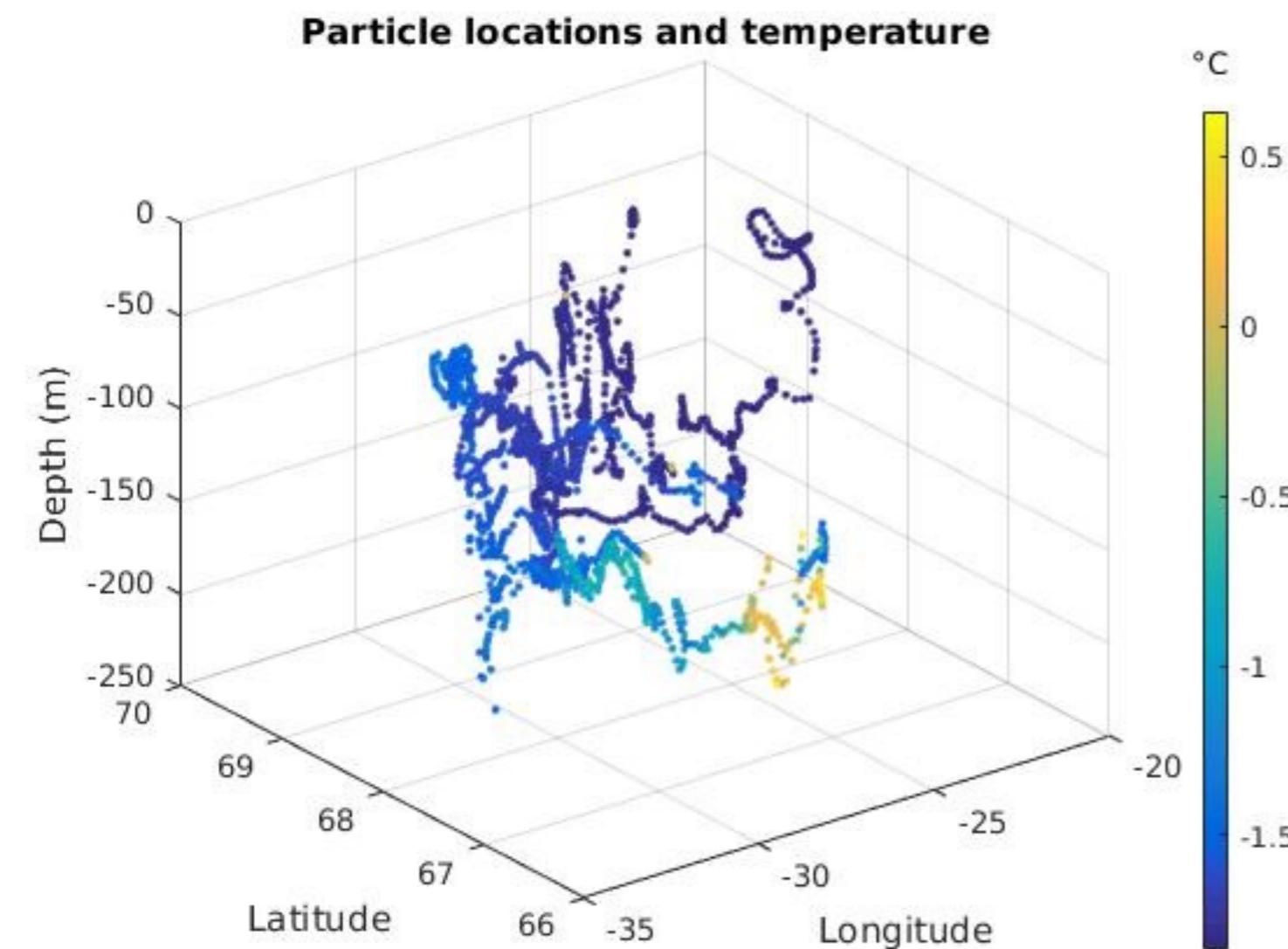
Load sequential property fields and interpolate

# Old implementation

Four stages:

1. **Initialization**: model domain, grid, bathymetry and initial particle positions
2. Calculate **particle trajectories**: for predetermined length of time, trajectory is calculated based on ocean model velocity fields
3. Particle **property extraction**: Temperature/salinity along trajectory are found by interpolation of model T/S fields
4. Particles positions and T/S info **saved** for further analysis

# Old implementation: Step 4



Save time series of particle locations and properties

# Old implementation

Four stages:

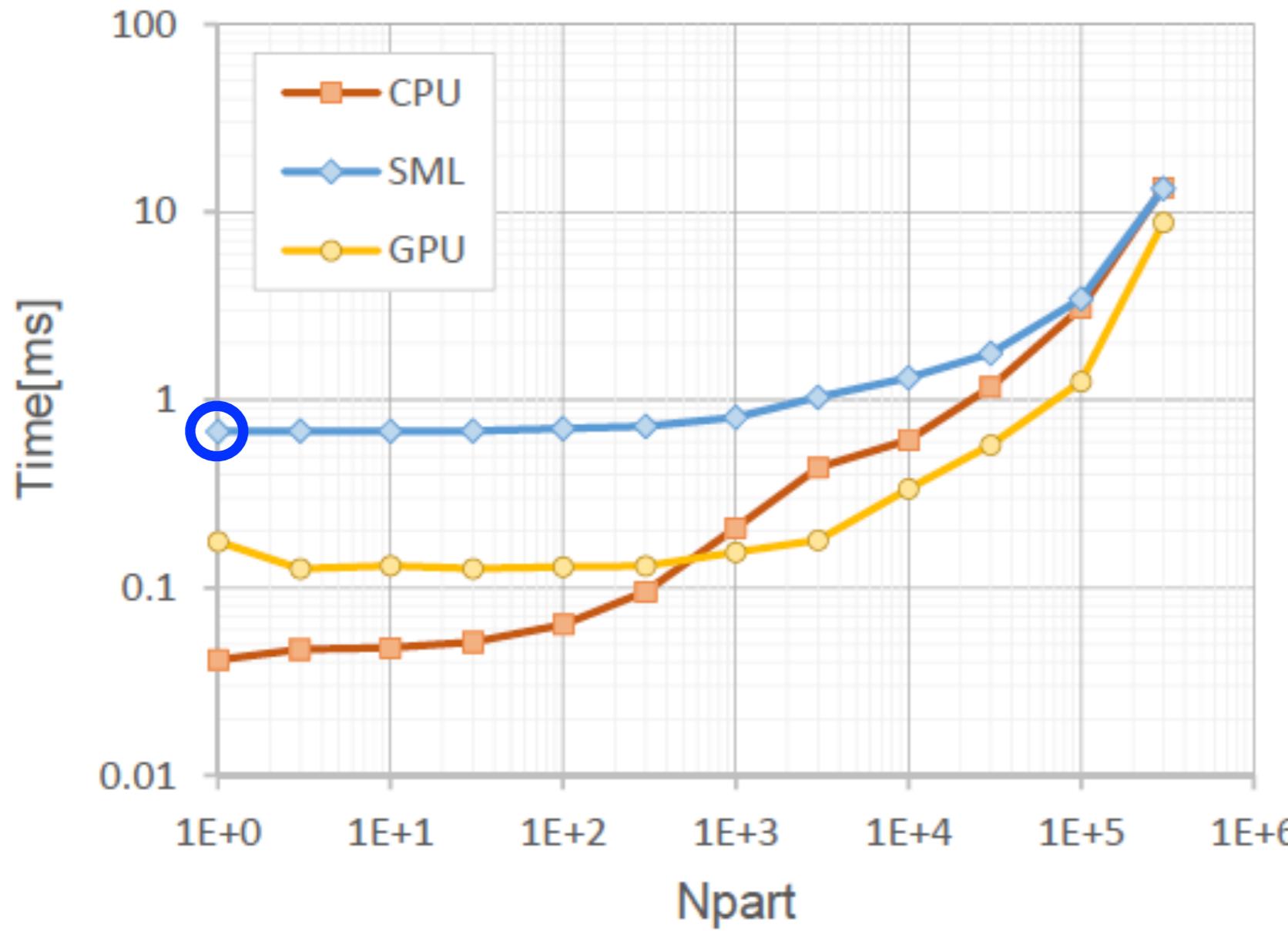
1. **Initialization**: model domain, grid, bathymetry and initial particle positions
2. Calculate **particle trajectories**: for predetermined length of time, trajectory is calculated based on ocean model velocity fields
3. Particle **property extraction**: Temperature/salinity along trajectory are found by interpolation of model T/S fields
4. Particles positions and T/S info **saved** for further analysis

Large potential gain by vectorization & parallelization of steps 2 and 3

# New implementation

Gelderloos et al., 2016a

# Interpolation improvements (I)

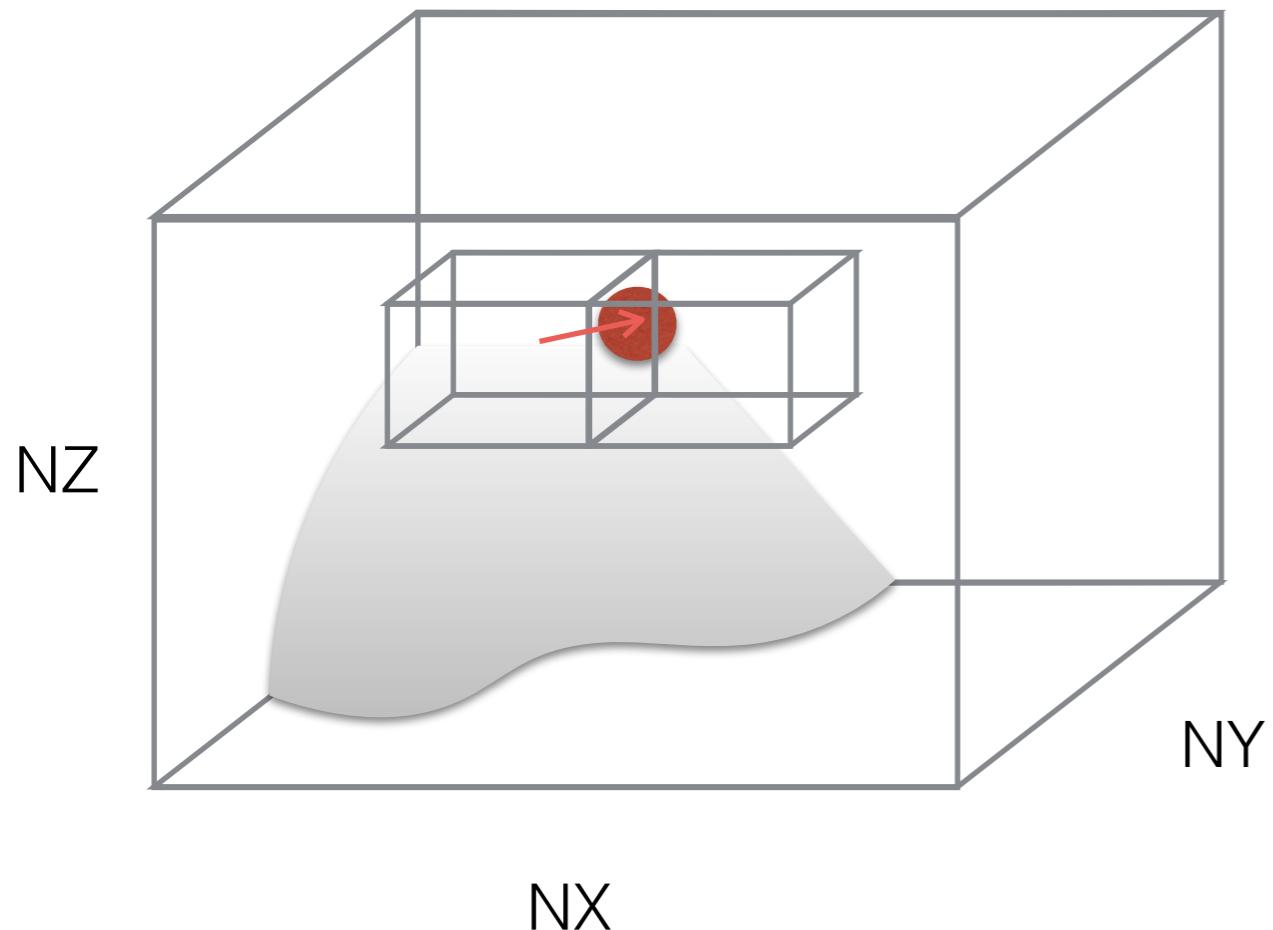


**CPU**: local small array no longer necessary

**GPU**: requires loading data into GPU memory —> faster for >1000 particles

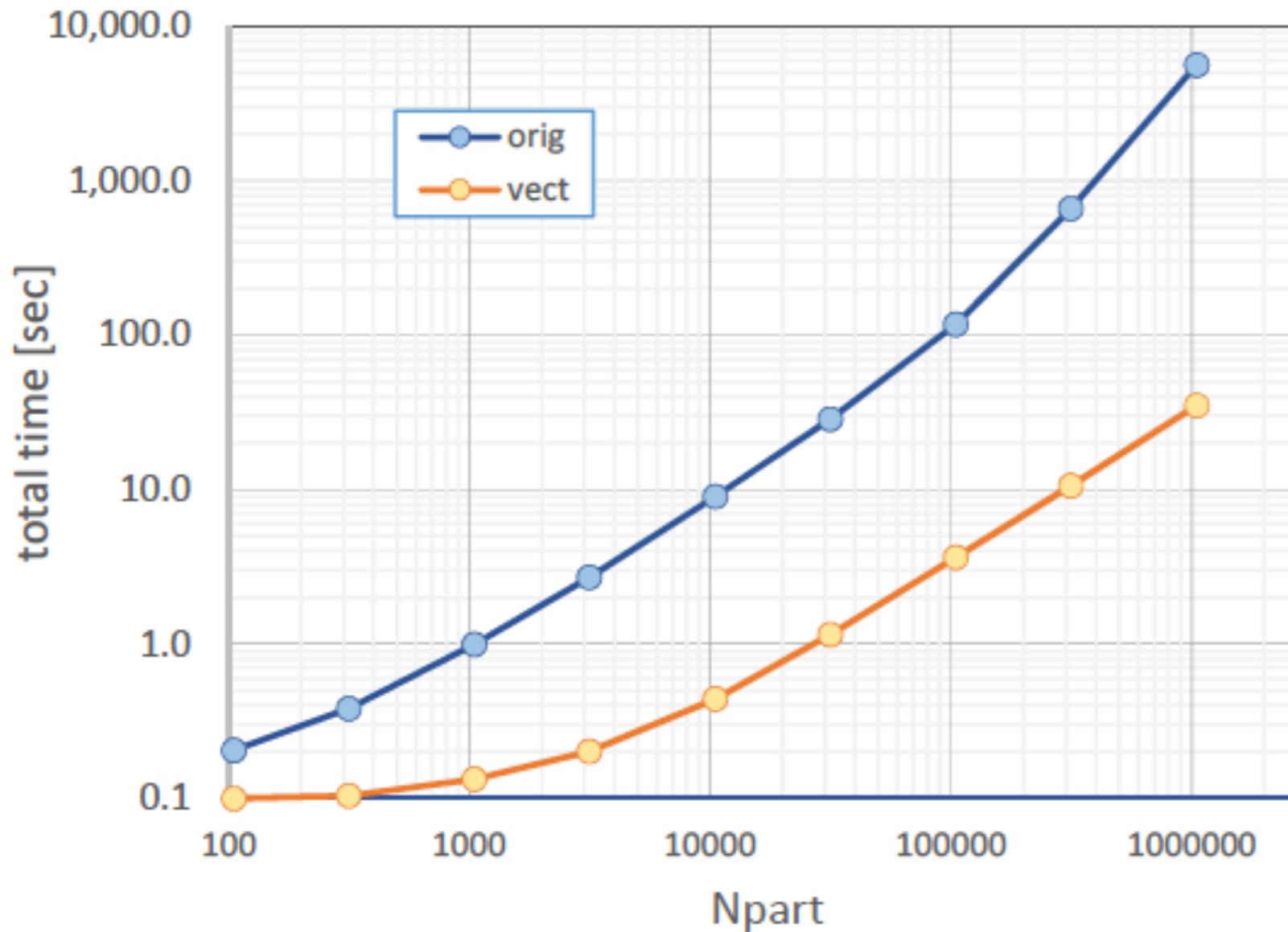
$$\frac{d\mathbf{x}_i(t)}{dt} = \mathbf{u}(\mathbf{x}_i, t)$$

# Interpolation improvements (II)



Removed cell boundary  
interrupts:  
(1) Faster  
(2) More accurate  
(3) Simpler code

# Interpolation improvements (III)

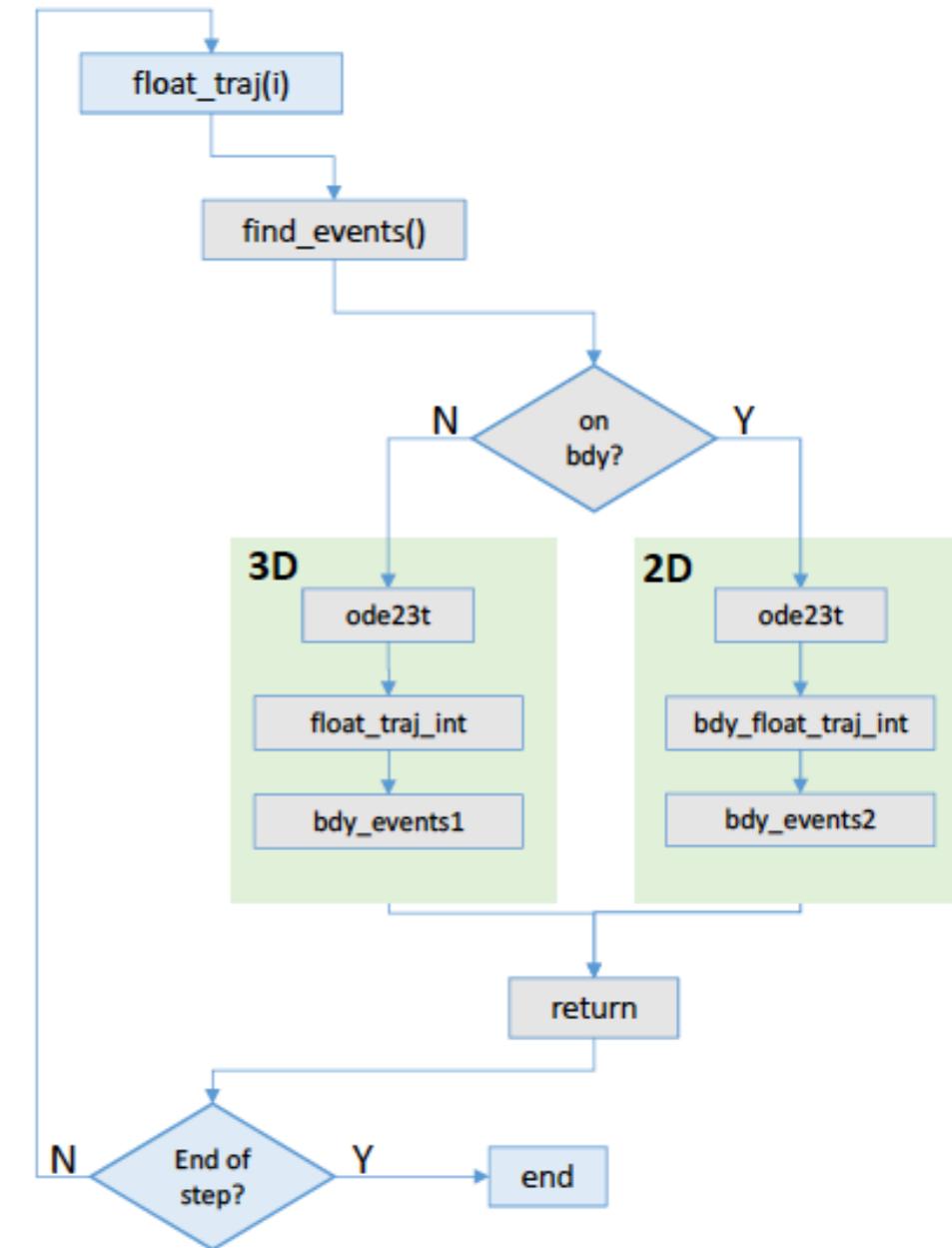
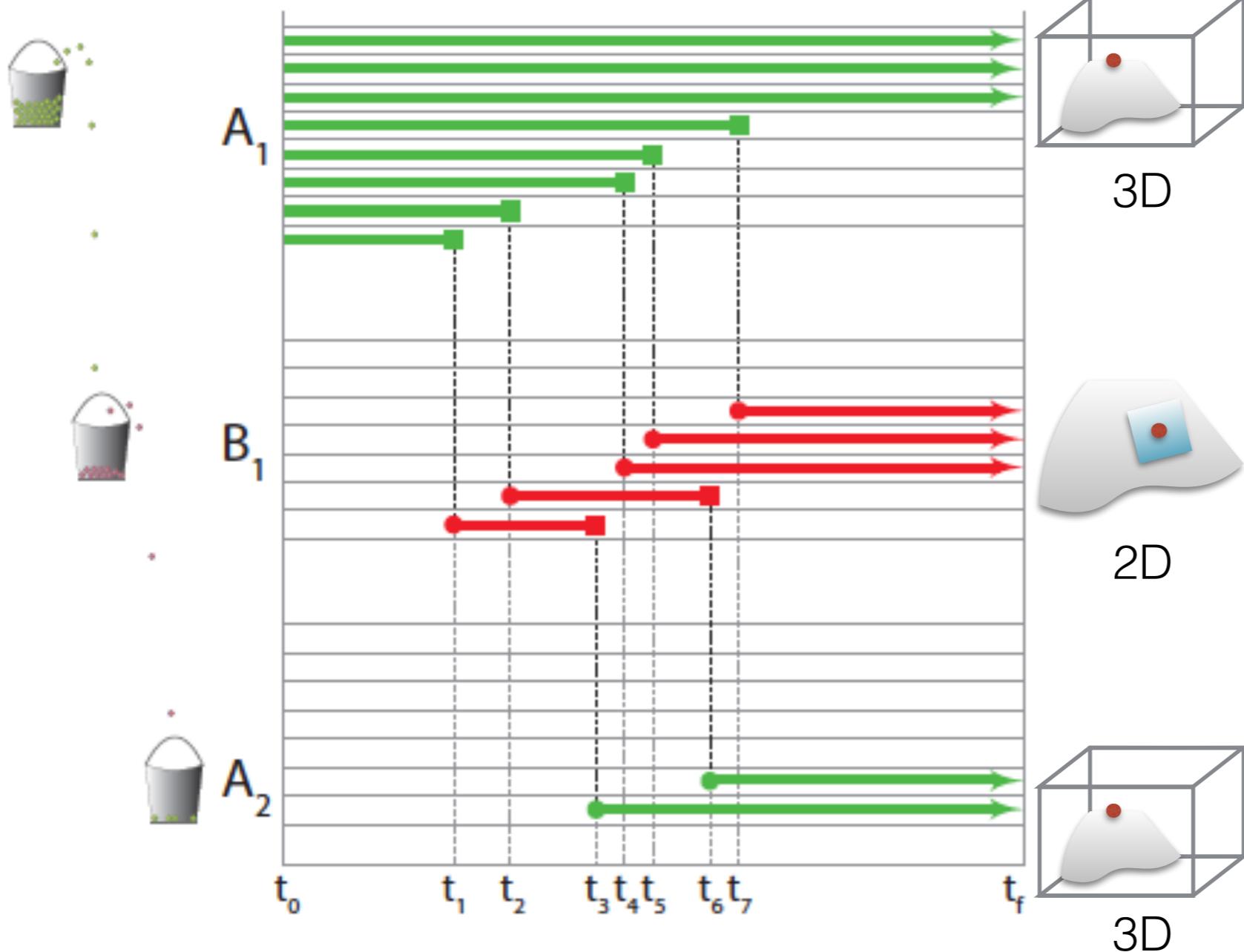


**Vectorization** of ocean-floor impingement interpolations yields a **160x** speedup for  $10^6$  particles

(Cubic interpolation only possible in CPU)

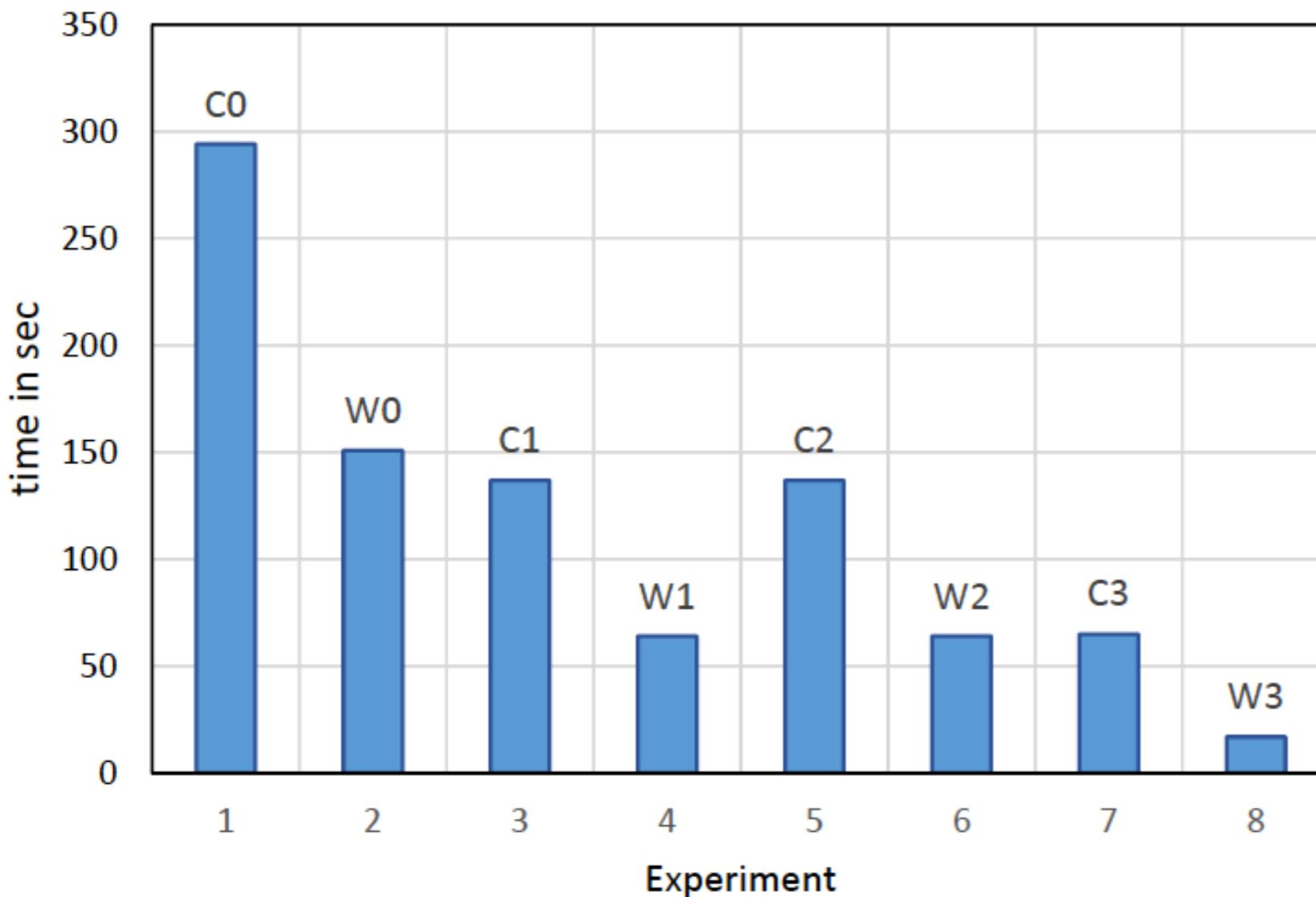
$$\operatorname{sgn} [H(\mathbf{x}_i) + z_i]$$

# Bucket-List Algorithm



# Parallelizing part 3

Vectorization and parallelization (with parfor)



$$T_i(t) = T(\mathbf{x}_i, t)$$

0: original

1: native little endian + fread

2: vectorized

3: parallelized

C/W: cold/warm cache

40 particles test case  
yields **10x** speedup

# Planned improvements

## **Speedup:**

1. Implement Bucket-List algorithm
2. Use sparse array (disregard grid cells under the ocean floor)
3. Use databases and space-filling curves

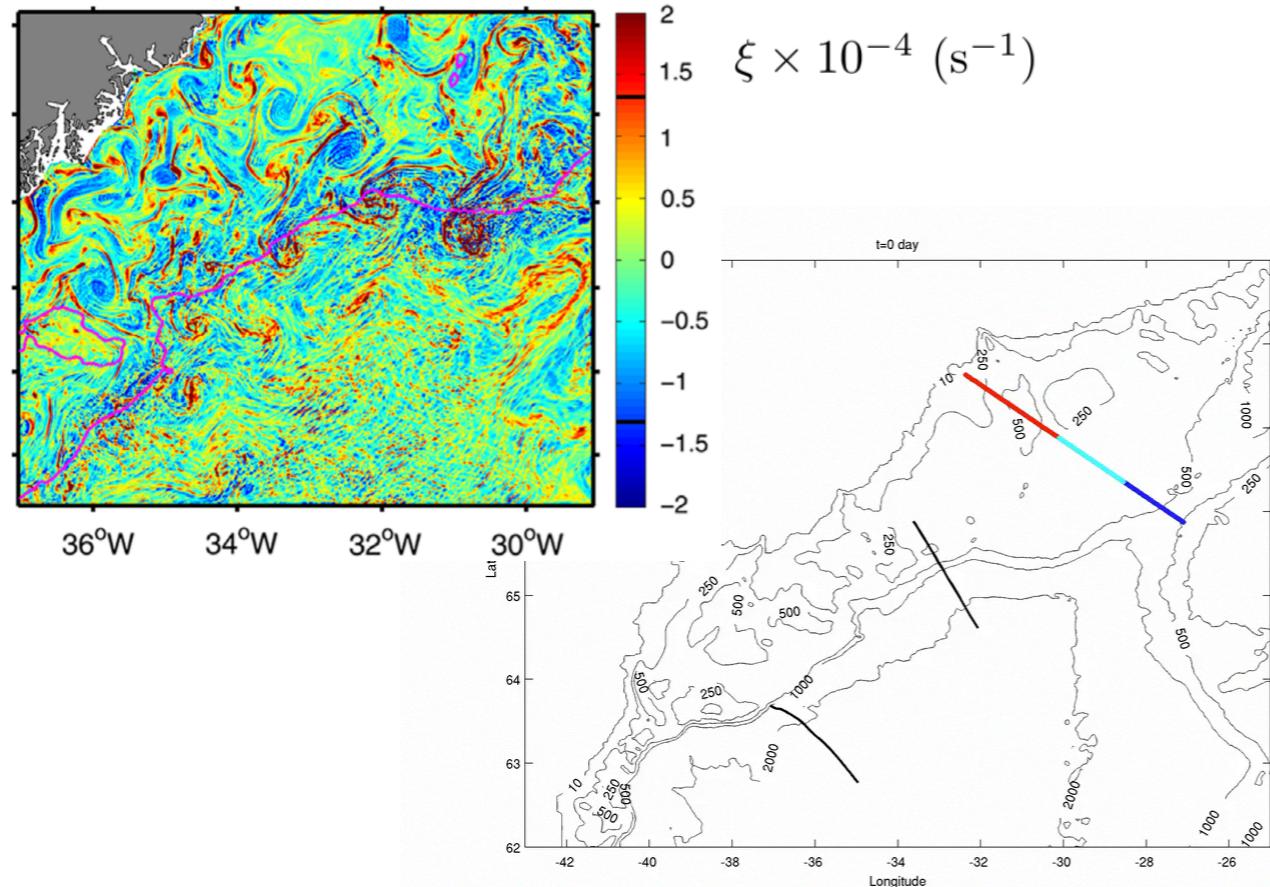
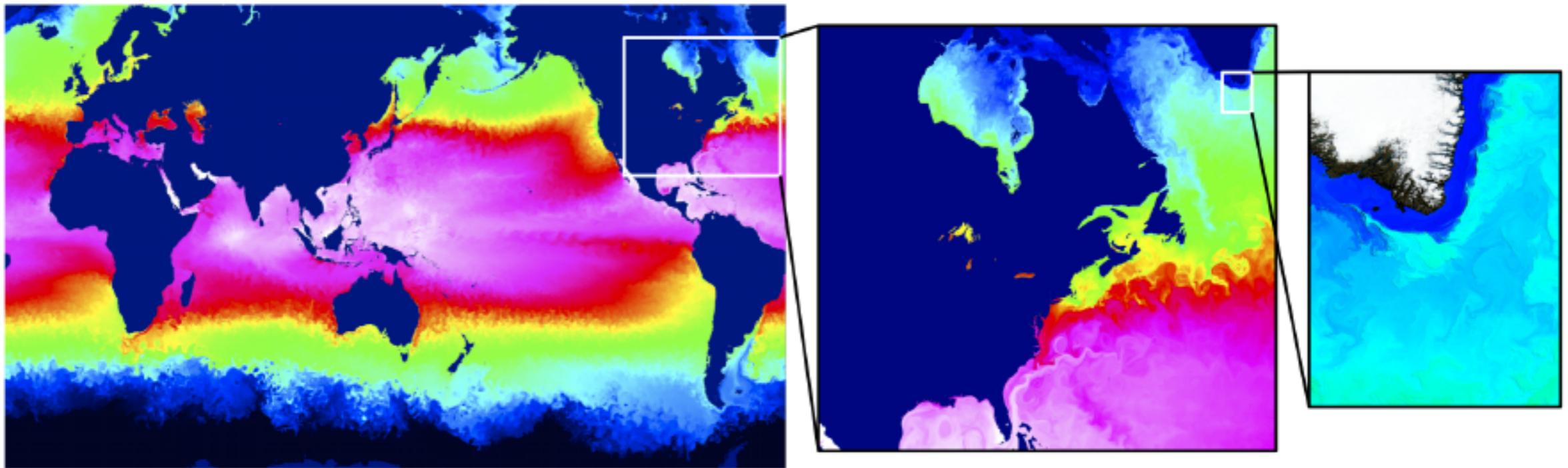
## **Accuracy:**

1. Use different ODE solver for interior and boundary-sliding particle trajectories (with different tolerance settings)

# Conclusions

- We built a particle-tracking code with accurate boundary-sliding representation
- Vectorization of the particle-tracking part of the algorithm has yielded a 3500 times speedup for  $10^3$  particles (5000 for  $10^6$  particles)
- Vectorization of ocean-floor impingement has yielded an additional factor 6 for  $10^3$  particles (160 for  $10^6$  particles)
- Vectorization/parallelization of the property-extraction part of the algorithm has yielded a 10x speedup for 40 particles

# Future Work



- Make the particle-tracking algorithm publicly available
- Improve the back-end database environment
- Enhance post-processing functionality
- Scale up to benchmark global ocean circulation solution

Koszalka et al., 2013; Magaldi & Haine, 2016