NEPTUNE: PARTICLES

Will Saunders

UKAEA

NEPTUNE Workshop

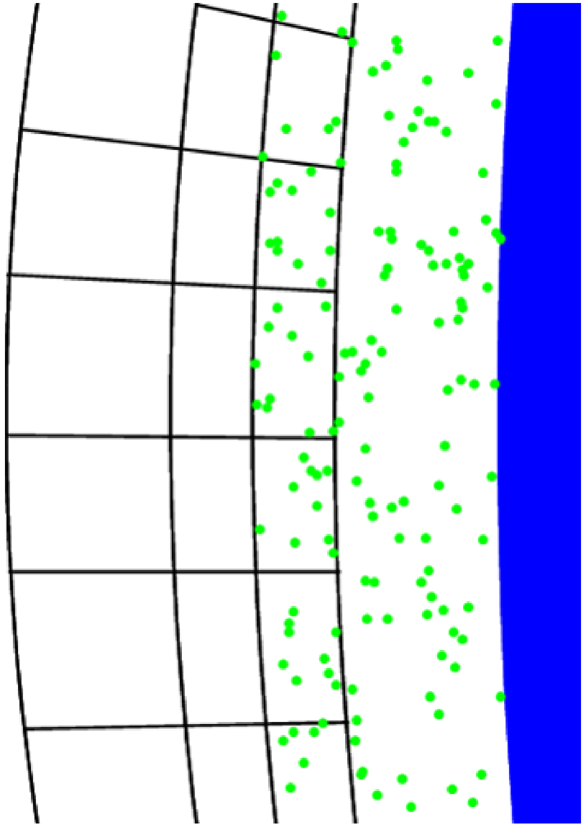
5-6 September 2022

UK Atomic Energy Authority

NEPTUNE Particle Use Cases

- Kinetic representations (I.e. distribution functions are not Maxwellian).
- Plasma
  1. Tritium, Deuterium fuel
  2. Alpha particles
  3. Ionised impurities
- Neutrals
  1. Injected (diagnostics)
  2. Recombined plasma (cooler regions)
  3. Sputtering from wall (molecules ejected from wall)
  4. Impurities
- Boundary conditions

- Less interested in molecular dynamics style operations – e.g. pairwise interactions.
- Small quantities of impurities important due to strong localised radiation.



NEPTUNE Particle Usage

- Domain Specialists desire a high-level interface
    1. Varying levels of interface to match desired level of control.
    2. e.g. Create a set of particles from an existing species and distribution.
    3. e.g. Per-particle control of properties and particle creation/deletion.

- Computational scientists
    1. Abstraction for particle data
    2. Abstraction for particle operations
    3. Works for both plasma and neutral species

- Both parties want efficiency across architectures without re-write (performance-portability).

Core Components

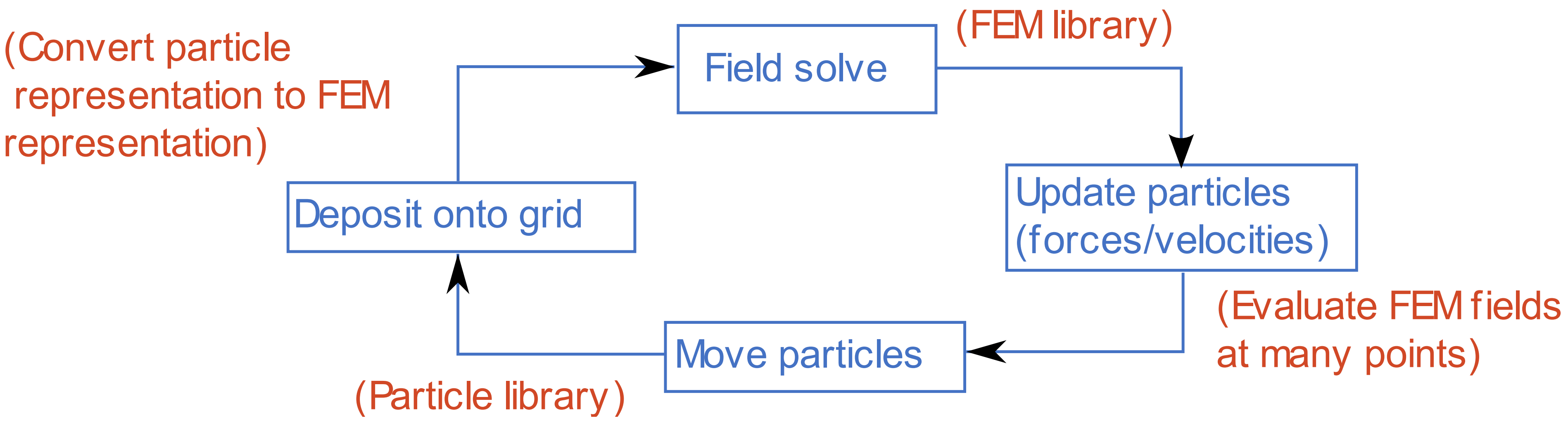
- Particle data communication
  1. Highly directional plasma flow (along field lines)
  2. Fast neutral flow (typically global and omnidirectional)
  3. Unstructured high-order mesh

- Particle current deposition / field evaluation
  1. Compute FEM fields for the deposition stage
  2. Evaluate FEM fields for particle push

- Particle Based Operations/Data structures
  1. Particle properties – position, velocity, charge, id...
  2. Loops over particles
  3. Degrees of Freedom (DOF) – Particle Loops
  4. Particle – Particle Loops

PIC Loop

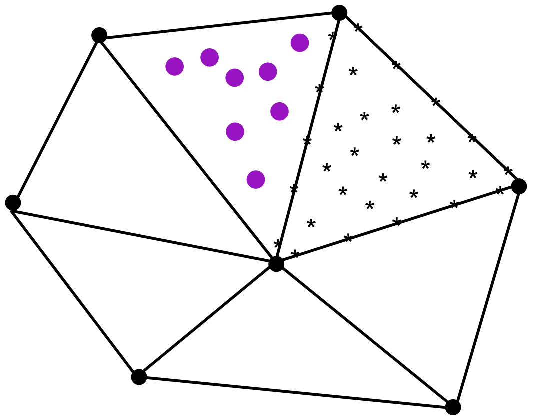
Overview



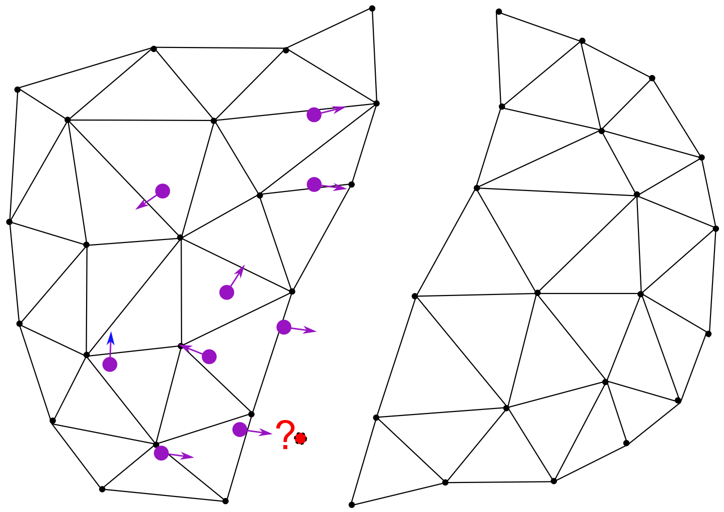
- Rough overview – More involved (and useful) schemes may combine steps.
- PIC schemes which exist to conserve quantities of interest, e.g. charge(mass), energy and momentum.
- Loop till convergence/end time.

Efficient Particle Implementation

Functional Requirements



- Enable efficient particle – grid operations.
  1. Hybrid/dual representations as particle and continuum.
  2. Continuum evaluation at many points
  3. Motivates close coupling between mesh and particles.

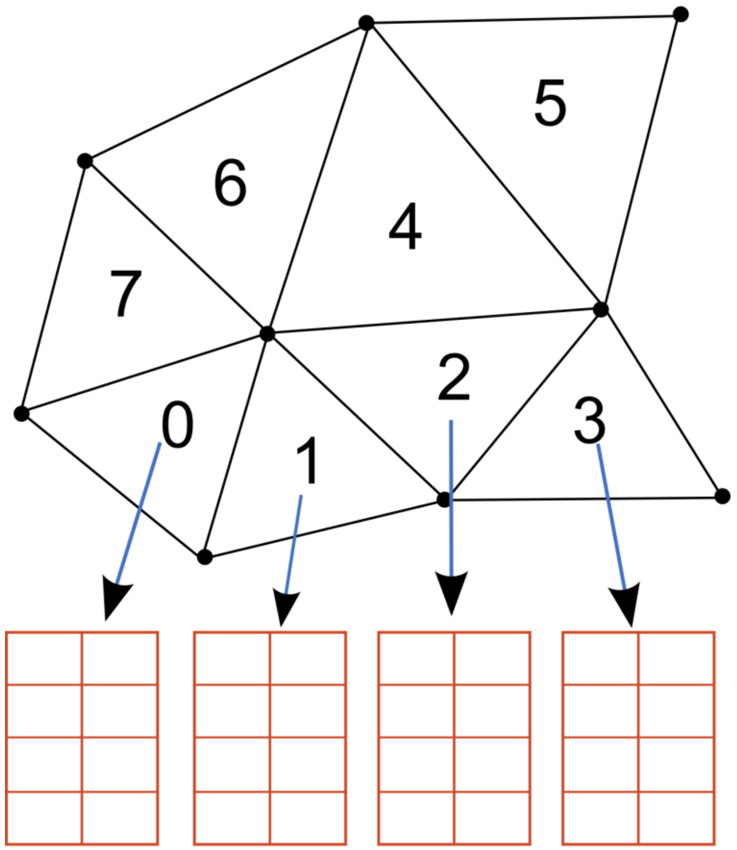


- Efficient and scalable particle movement.
  1. Fast (essentially global) movement of neutrals
  2. Anisotropic flow

- Target implementation for a DSL.
  1. Generate looping operations/tasks

Data – Each mesh cell

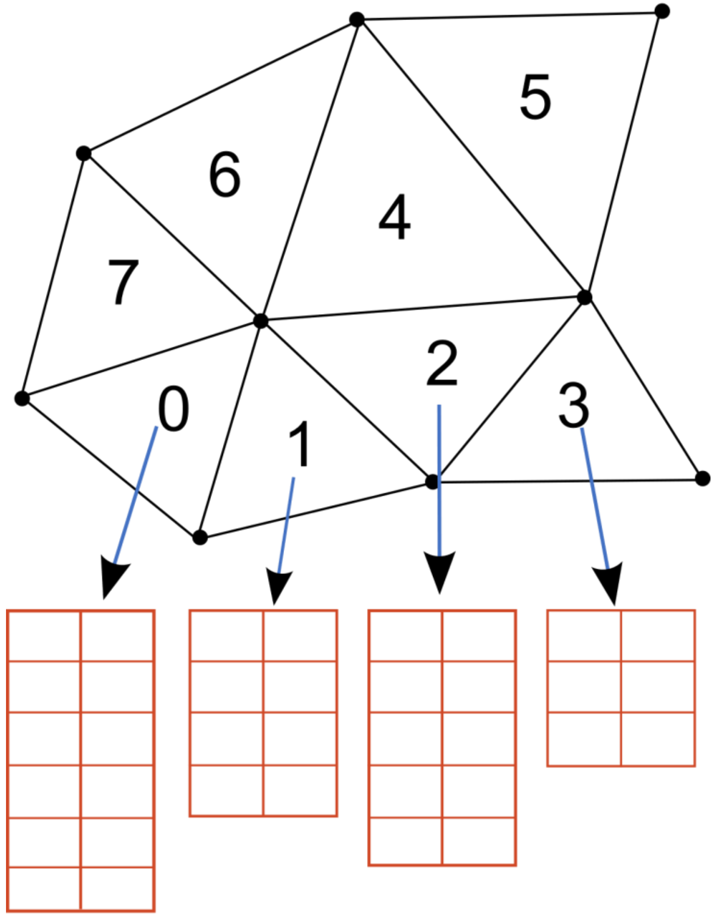
CellDatConst



- Datatype, Column count and Row count (per cell) fixed at construction
- Device allocated (*sycl*::malloc_device)
- DOF Data
- Expansion coefficients
- Geometry/domain information
- Lookup indices

Data – Variable Row Count

CellDat

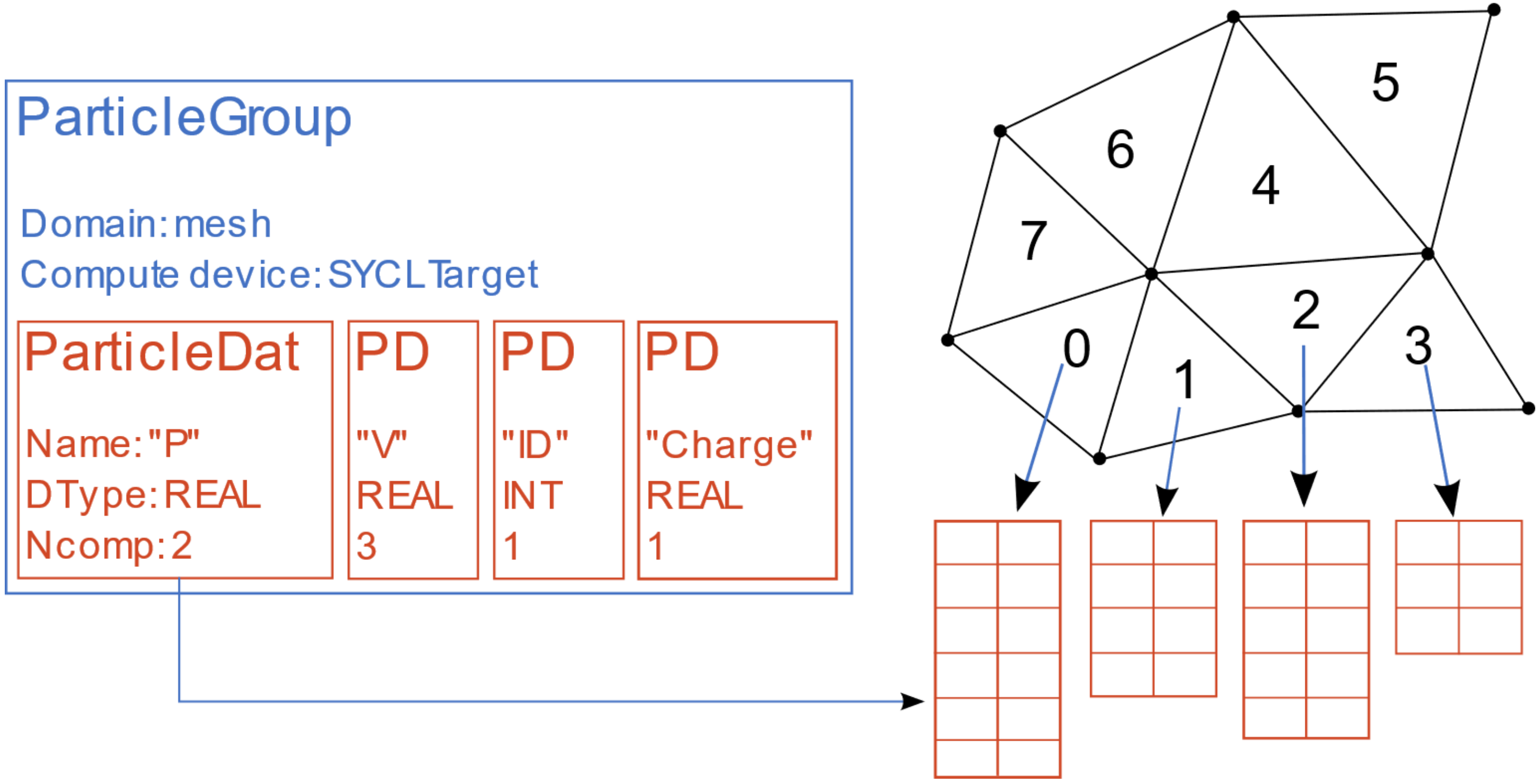


- Datatype and Column count (per cell) fixed at construction
- Variable number of rows (per cell)
- Device allocated (*sycl*::malloc_device)
- Base container for particle data, e.g. A floating point CellDat with 2 columns could store 2D positions.
- Per cell storage is advantageous for particle – grid operations

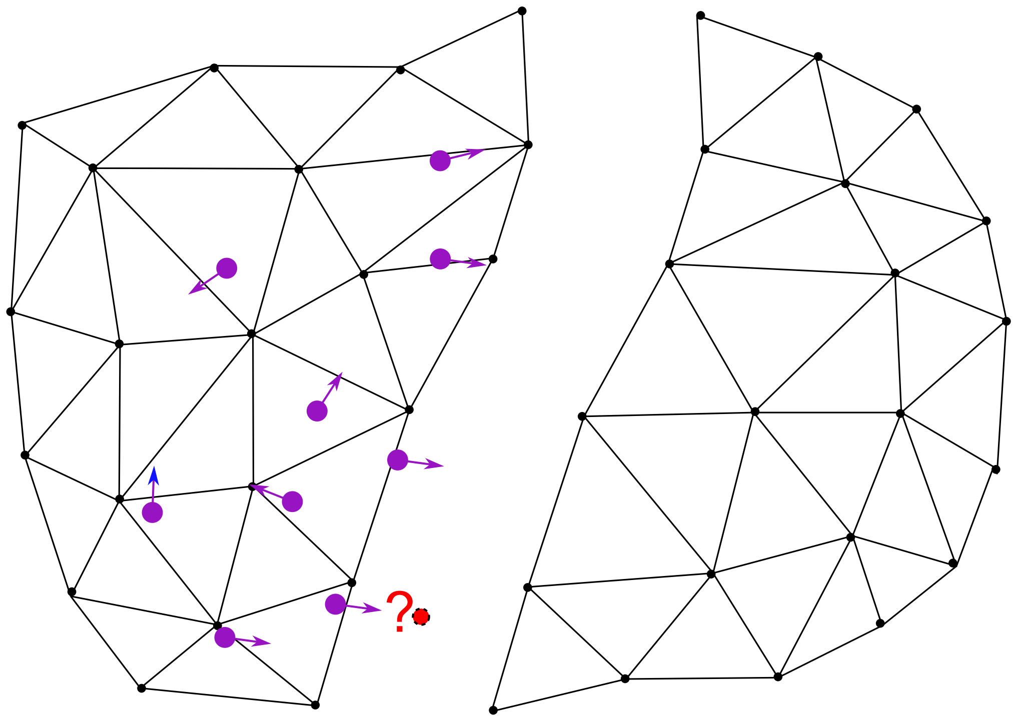
Particle Data

ParticleGroup, ParticleDat

- Combines the: mesh, compute device and particle data.
- Implements particle bookkeeping – cells and MPI ranks.
- General particle properties, e.g. charge, mass, weight, velocity.



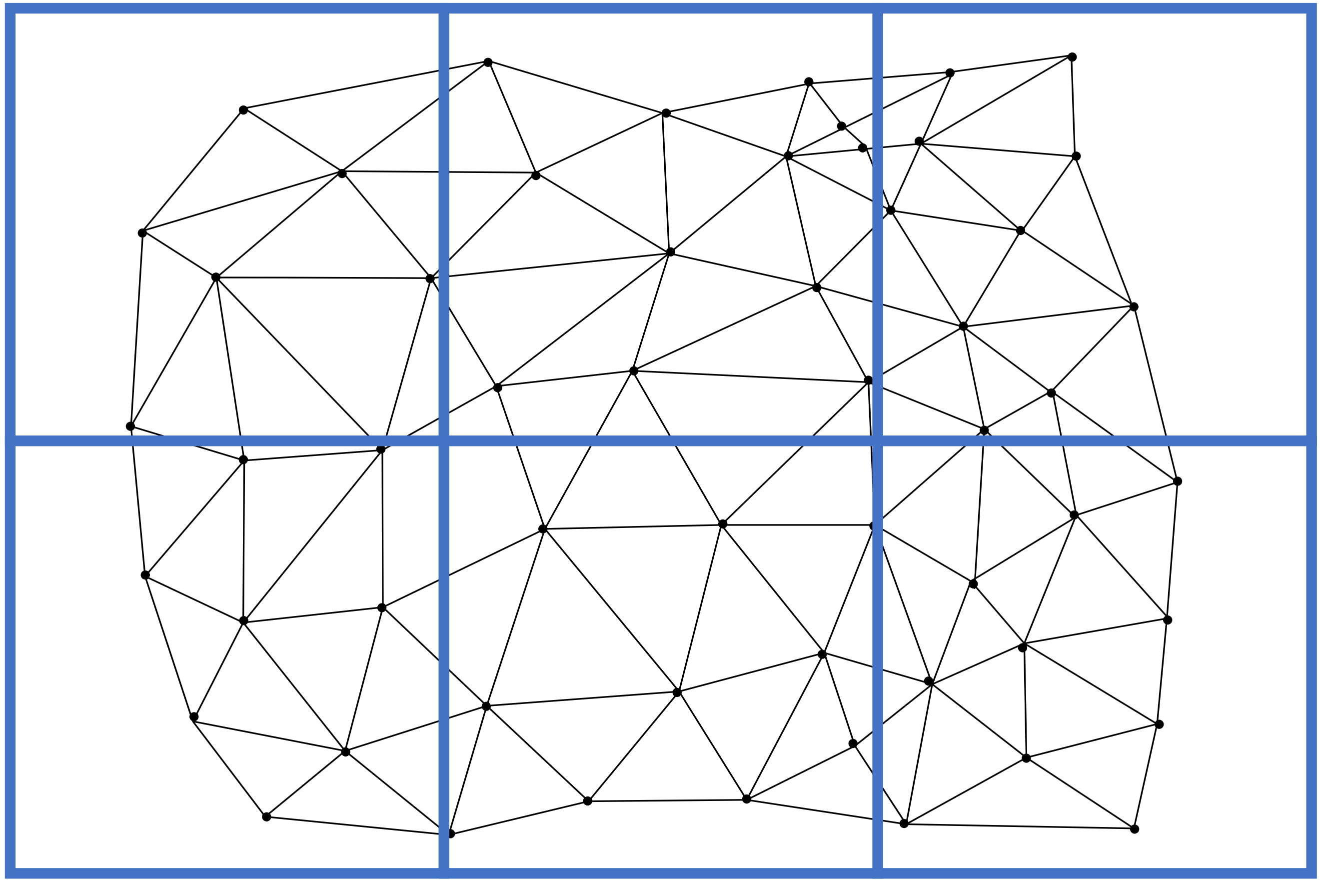
Global Particle Movement



- Efficiently transfer particle ownership – domain decomposition
- Fast moving particles – essentially global
- Want local communication patterns where possible
- Minimise number of non-linear solves (high-order mesh)

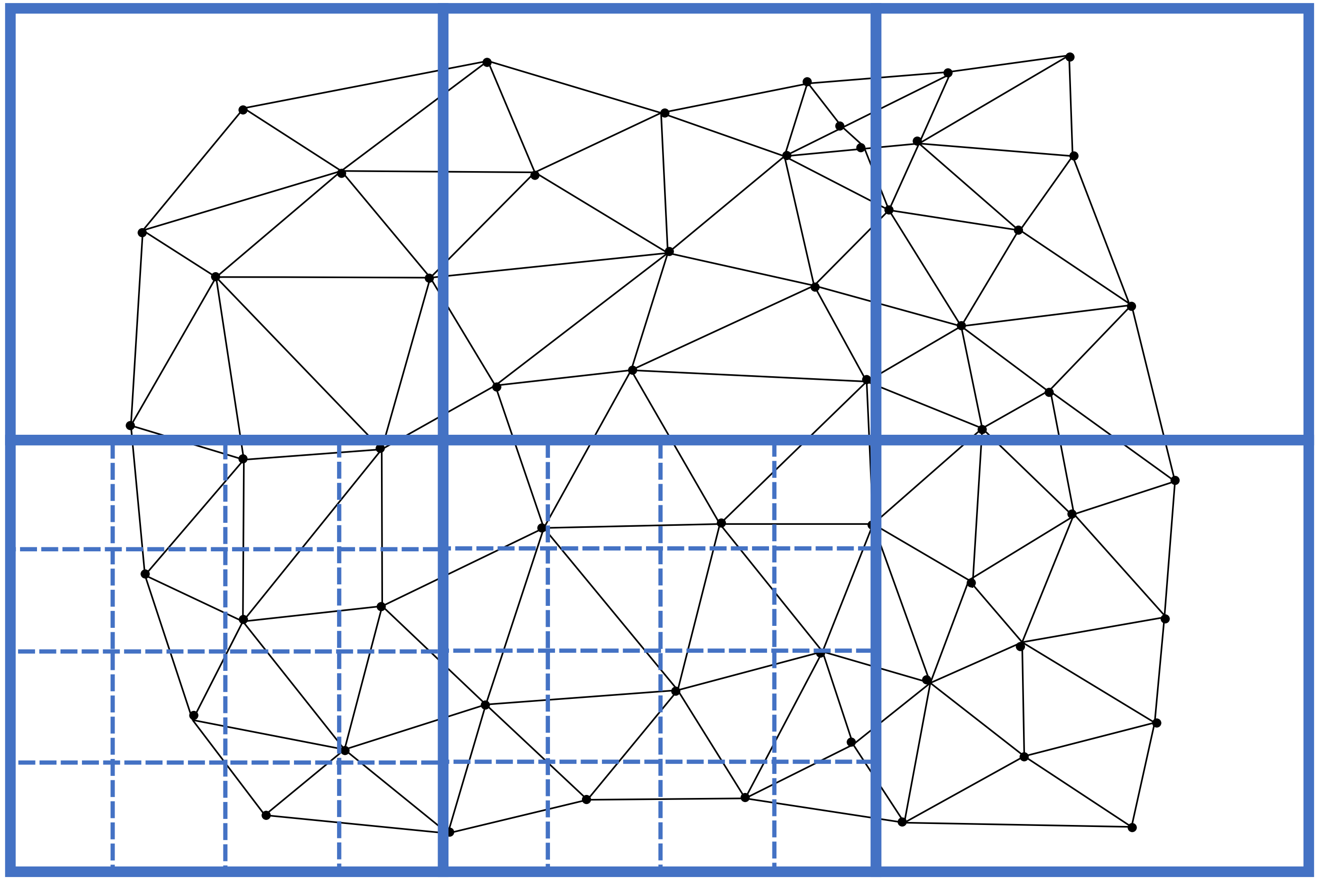
Solution

Overlay Coarse Grid of squares/cubes



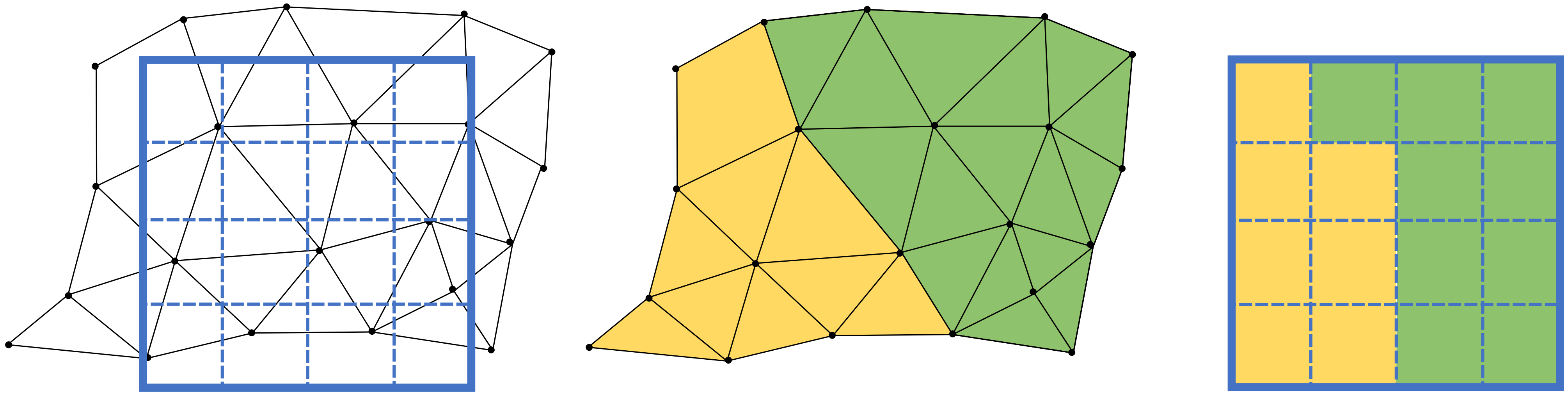
Solution

Subdivide coarse mesh cells



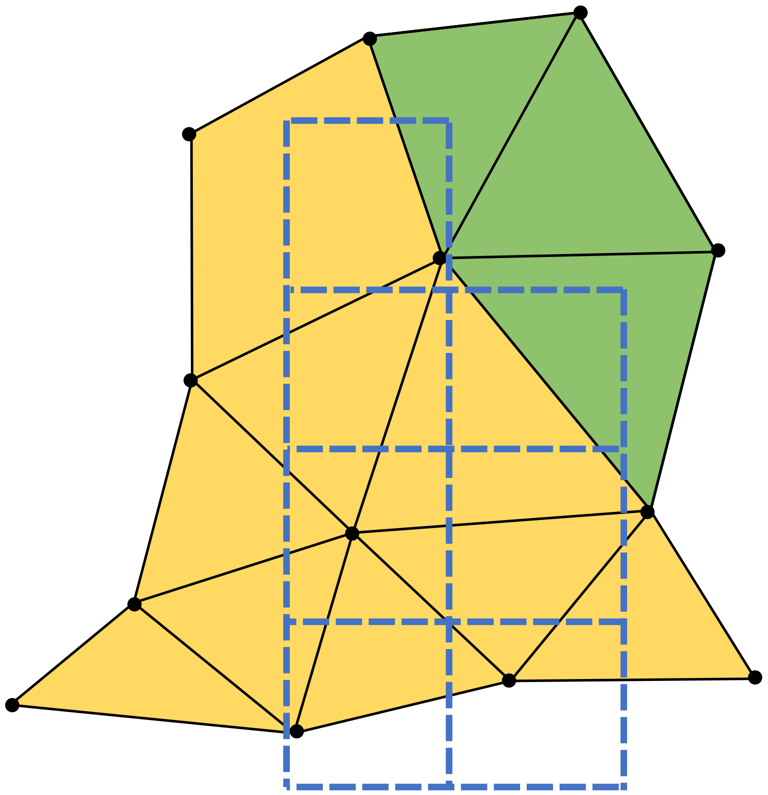
Decompose Fine Mesh

Assign MPI owners to fine mesh cells



Build Halo Regions

Duplicate Geometry Objects



- Duplicate remotely owned geometry to cover owned coarse mesh cells.
- Store owning rank and local id of copied geometry.
- Particles in the coarse mesh cells can be mapped to geometry objects and owning ranks.
- Setup point-to-point communication patterns between neighbours.
- Halo width is tuneable – increase local communication.

Hybrid Particle Transfer

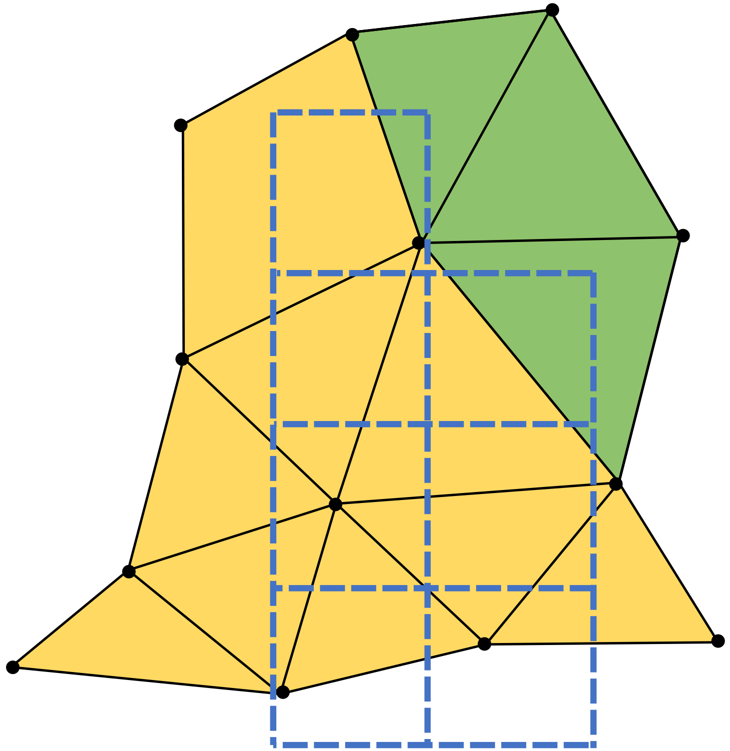
Global + Local Transfer

For each particle:
 Attempt to bin into local mesh cell (either owned or halo)

For "far moving" particles (not binned into owned or halo cells):
 Transfer to MPI rank owning overlayed cartesian cell
 (global transfer)

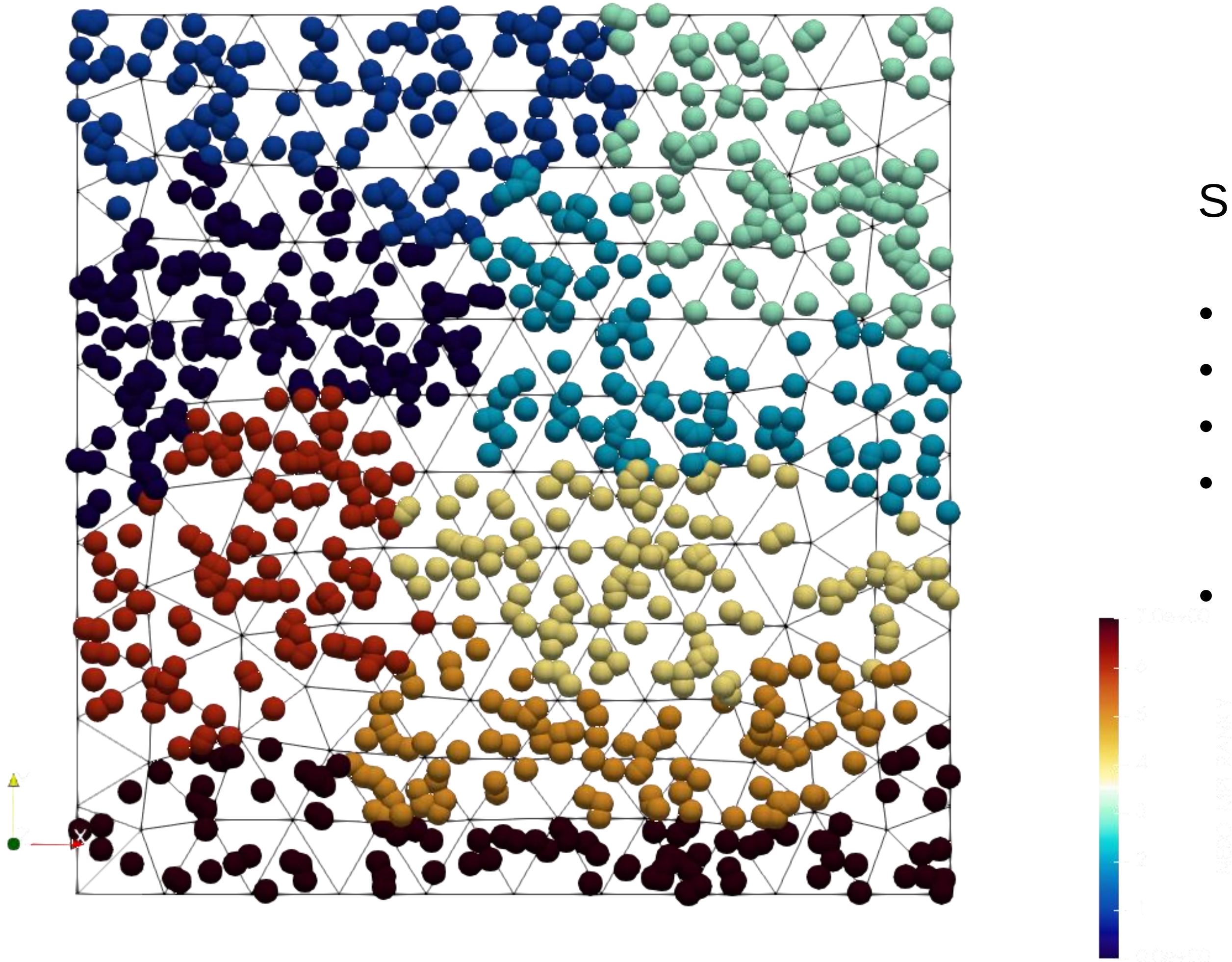
For each particle received in global transfer:
 Bin into local mesh cell (either owned or halo)

For "locally moving" particles:
 Transfer to neighbour using local communication pattern.



Trajectory Snapshot

Colour – owning MPI rank



Summary:

- On device particle state and computation[1].
- Performance oriented data structures.
- Base implementation for DSL.
- MPI domain decomposition with hybrid (global + local) move.
- Particle transport on 2D linear Nektar++ meshes[2].

[1] https://github.com/ExCALIBUR-NEPTUNE/NESO-Particles
[2] https://github.com/ExCALIBUR-NEPTUNE/NESO

The End

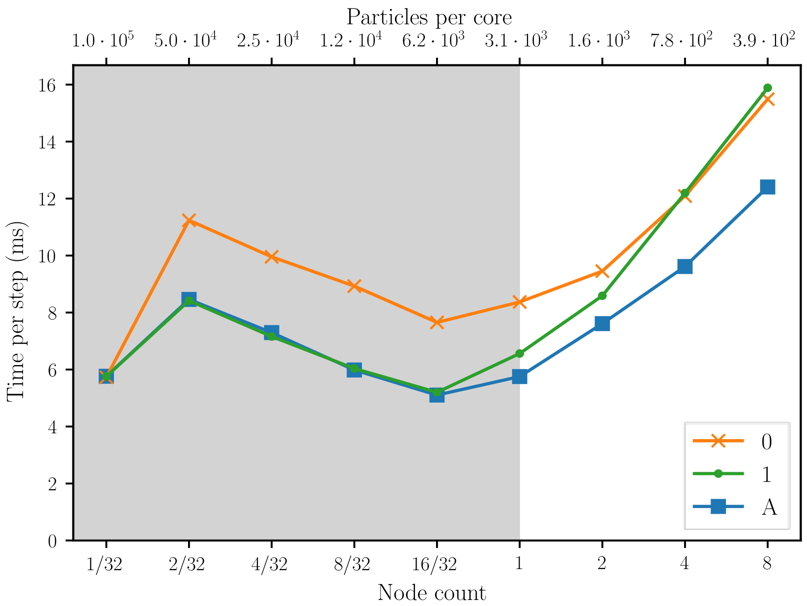
UKAEA NEPTUNE:

Rob Akers
Wayne Arter
Matthew Barton
James Cook
Joseph Parker
Owen Parry
Will Saunders
Ed Threlfall

The support of the UK Meteorological Office and Strategic Priorities Fund is acknowledged.

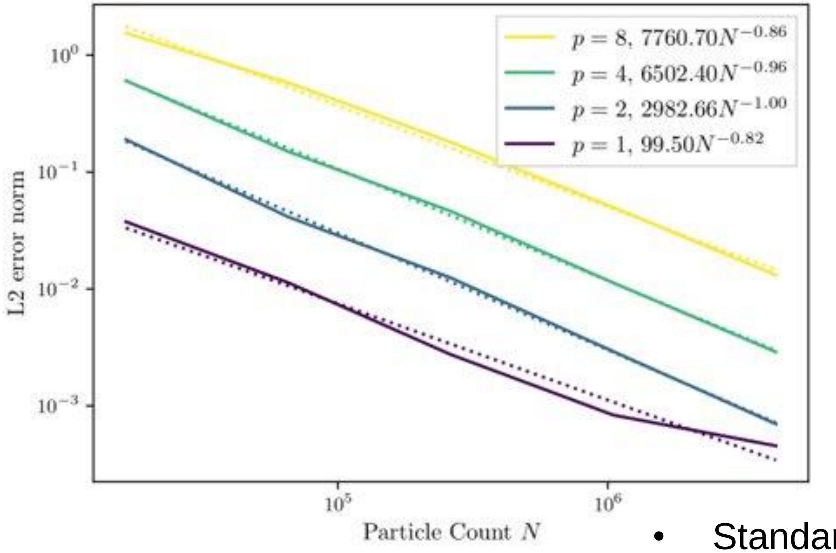
Core Components – Exploratory Ideas and Progress

- Particle data communication
  1. Combine coarse grid/octree with halo cells
  2. Majority of transport using local communication patterns through halos
  3. Global communication where needed (MPI RMA)



0: Halo width 0
1: Halo width 1
A: Adaptive halo width

- Particle current deposition / field evaluation
  1. L2 Galerkin projection inspired (for deposition)
  2. Cell-wise data representation: DOFs/Coefficients
  3. Cell-wise loops between particles and DOFs



- Standard L2 error computed against reference 2D Gaussian
- Higher FEM order captures more particle noise

Particle DSL - Exploratory Ideas and Progress

- User/Developer facing
- Separation of Concerns

- Abstraction:
    - Data Structures: Particle/FEM DOFs
    - Looping operations: Iteration set + kernel + access descriptors
        1. Loops over particles
        2. Cell-wise loops over particles and DOFs
        3. Pairwise particle loops (for particle-particle interactions)

- Implementation
    1. SYCL – low level target language
    2. Python code generation framework
    3. DSL embedded in Python
    4. PPMD/pyOP2 inspired

```
advection = ParticleLoop(
    target_device, # where to execute
    Kernel(
        "advection_kernel",
        """
        P[ix, 1] += V[ix, 1] * $dt
        P[ix, 2] += V[ix, 2] * $dt
        P[ix, 3] += V[ix, 3] * $dt
        global[1] += 1
        """
    ),
    Dict( # map from kernel symbols to data
structures
         "P" => (particle_group["P"], WRITE),
         "V" => (particle_group["V"], READ),
         "global" => (global_data, INC),
    )
)
execute(advection)
```


ExCALIBUR