Science and Technology Facilities Council

OPS-Particle: A domain specific language for Lagrangian systems and its application to particulate systems

C. Tsinginos & S. Rolfo

24/4/2026

Belfast, UK

CoSeC

ParaSolS
Particulate Solids Simulations

UK TURBULENCE CONSORTIUM

Introduction

HPC landscape

- **Diversity in HPC resources (CPUs, GPUs or a combination of those)**
- **Programming diversity (i.e., open standards, proprietary model etc)**

Domain specific languages can help us to surmount this barrier

Science and Technology Facilities Council

Challenges

- **New platforms may require different tuning efforts**
- **Cannot recode applications for each new type of architecture**

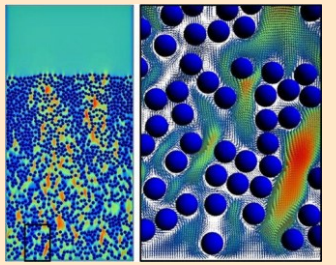
Our goal

Develop a domain-specific language (DSL) framework for modelling particulate systems

Possible applications


Turbulent laden flows


Modeling of Powders


DNS modelling of fluid-particle systems

What is a domain specific language?

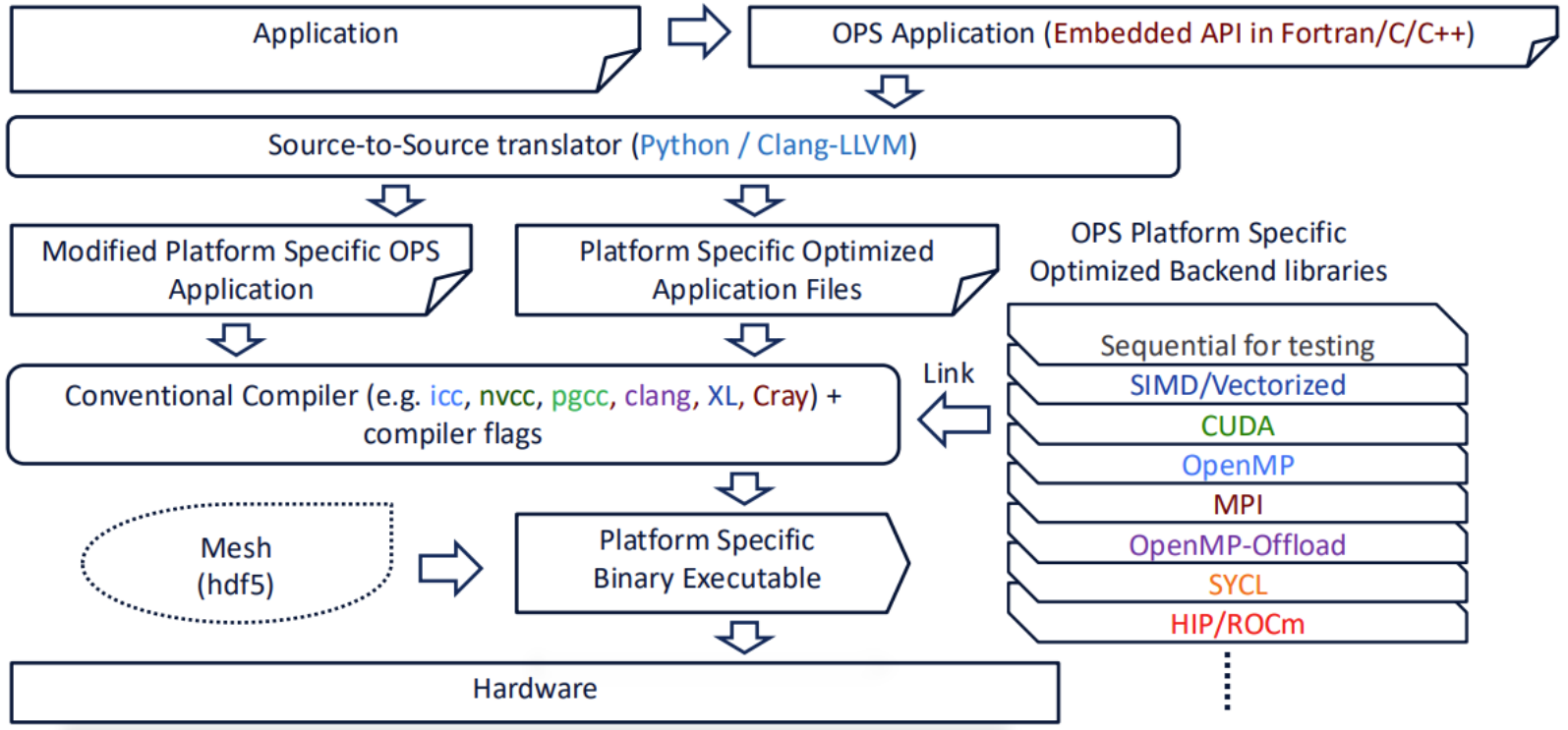
- **A computer language specialized to specific application domains**
- **New concepts and domain operations (functions) and a compiler must be introduced**

Advantages

- **Restrict writing code that is difficult for the compiler to reason and optimize**
- **Implementation of the API left to a lower level**
  - Target implementation to hardware - automatically generate implementation from specification for the context
  - Generate code in best parallelization model
  - Code usable by different applications

Examples of domain specific languages

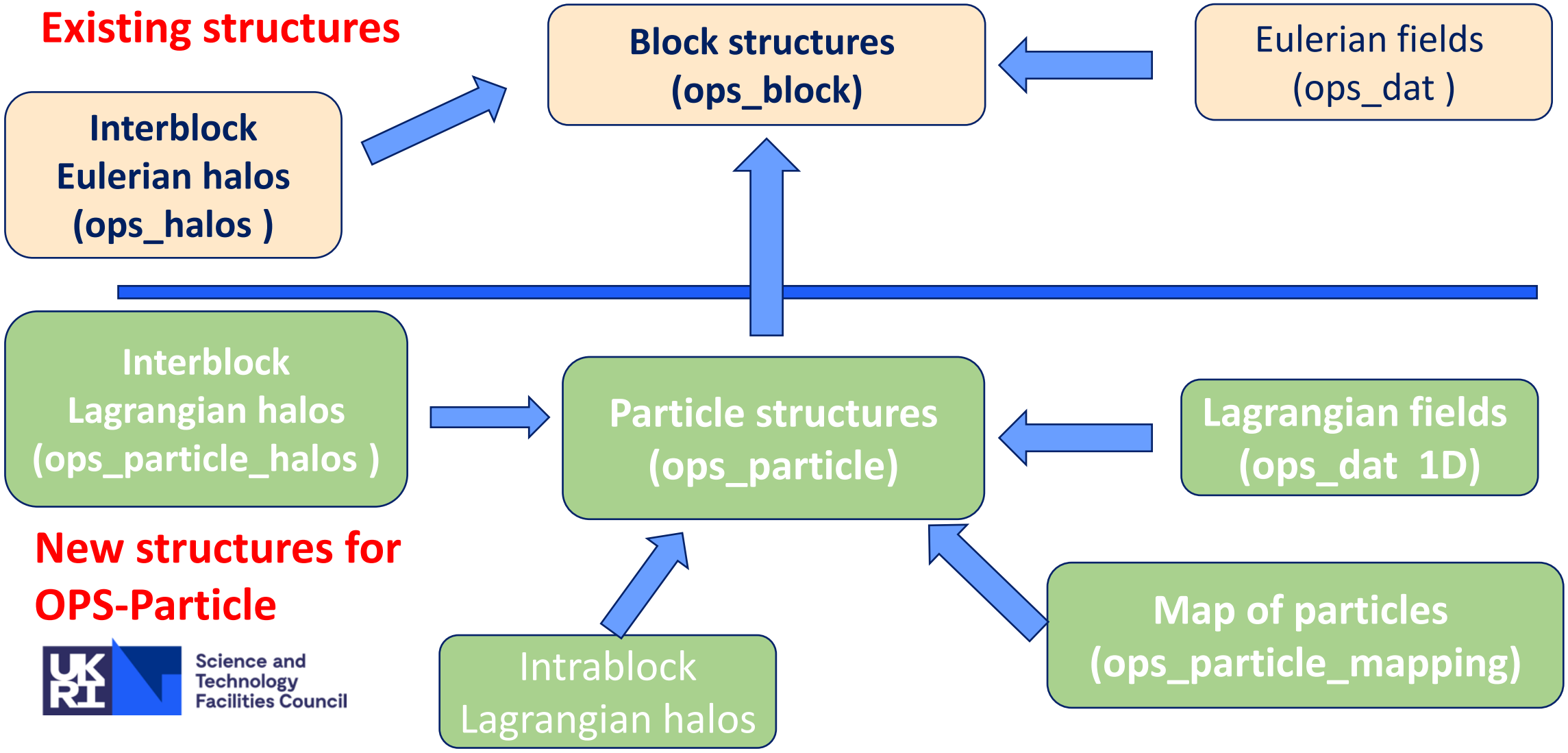
OPS in a nutshell



- A DSL for **multi-block grid structures** applications that achieves portability across different architectures
- Python translator to obtain optimized codes for different architectures

Science and Technology Facilities Council

OPS-Particle syntax: Domain entities



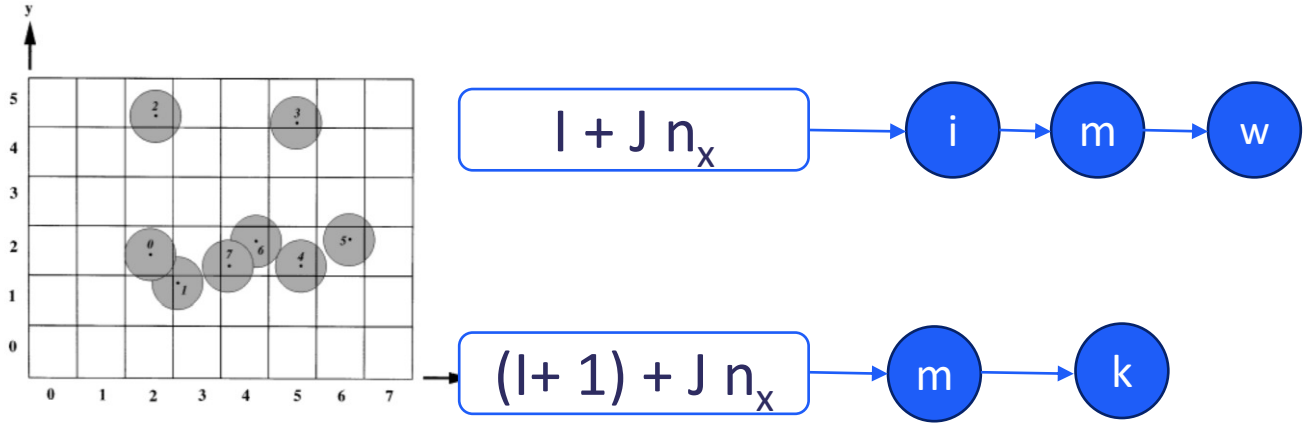
Science and Technology Facilities Council

OPS-Particle syntax: Domain operations

Operations to extend OPS into Lagrangian systems

- Mapping operations
- Interblock & intrablock communications
- Iteration operations
  - Iteration operators for particle, particle/grid and grid/particle operations need to be defined
- Initialization & deletion operations

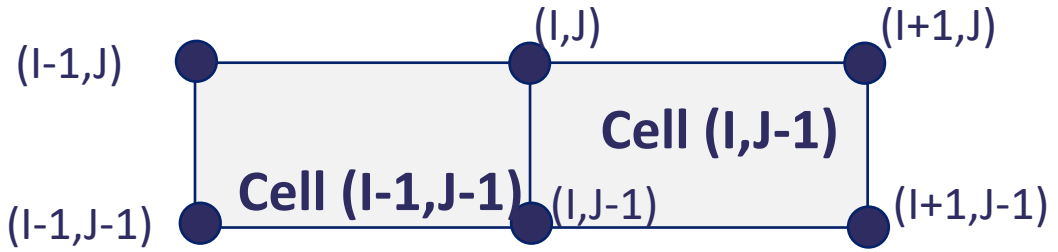
Particle maps & Halo communications in OPS-Particle



- OPS-Particle provides **intra-block** and **inter-block** communications for particle data
- **Halo types**
  - **Exchange**: Exchange of local particles
  - **Forward**: Copy data from actual to ghost (virtual) particles
  - **Reverse**: Send data from ghost to actual particles (e.g. forces)
- **Halos change with time** (Dynamic event)

- OPS-Particle provide projection operations to uniform grids for adressing grid-particle interactions (staggered) approach
- **Supported mapping strategies**
  - The decoupled (classical) approach
  - The coupled (hybrid) approach

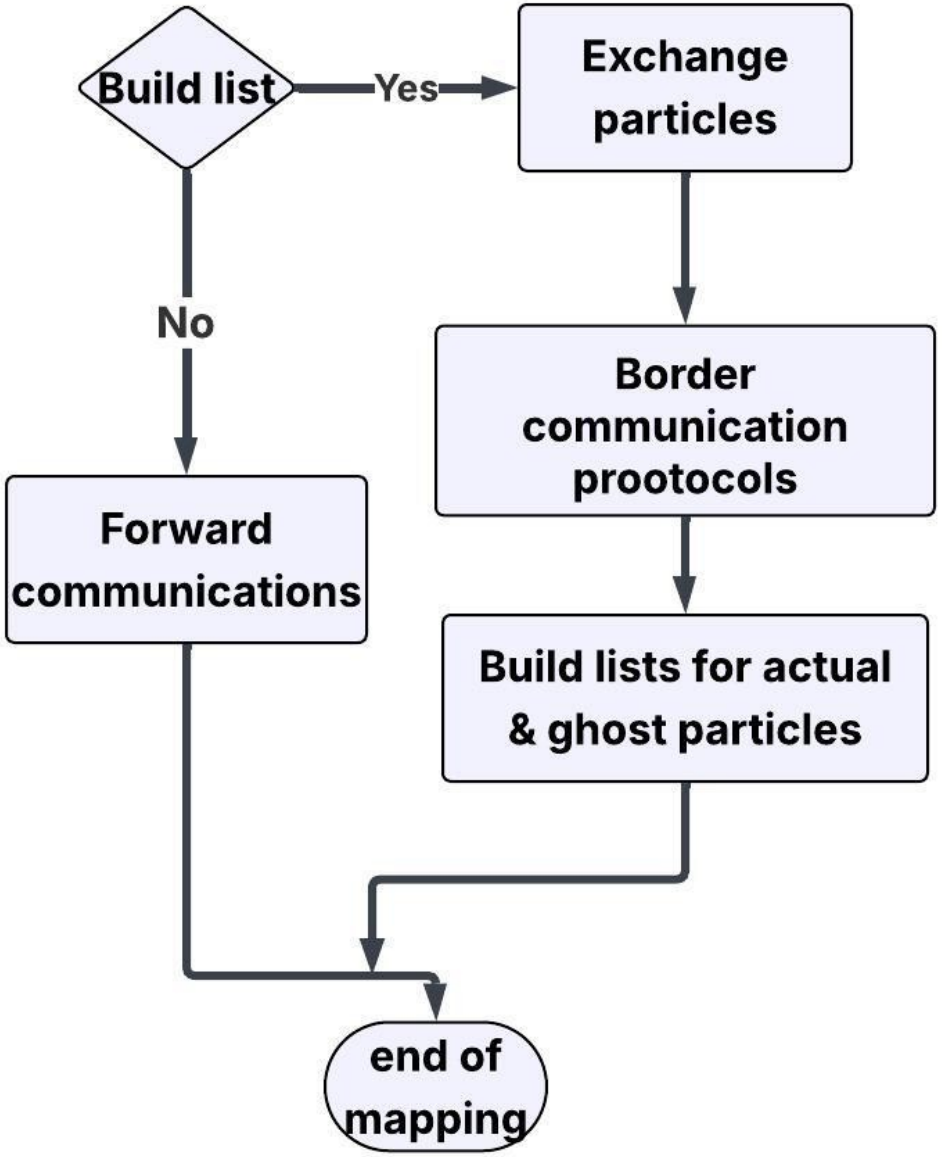
Staggered grid



Science and Technology Facilities Council

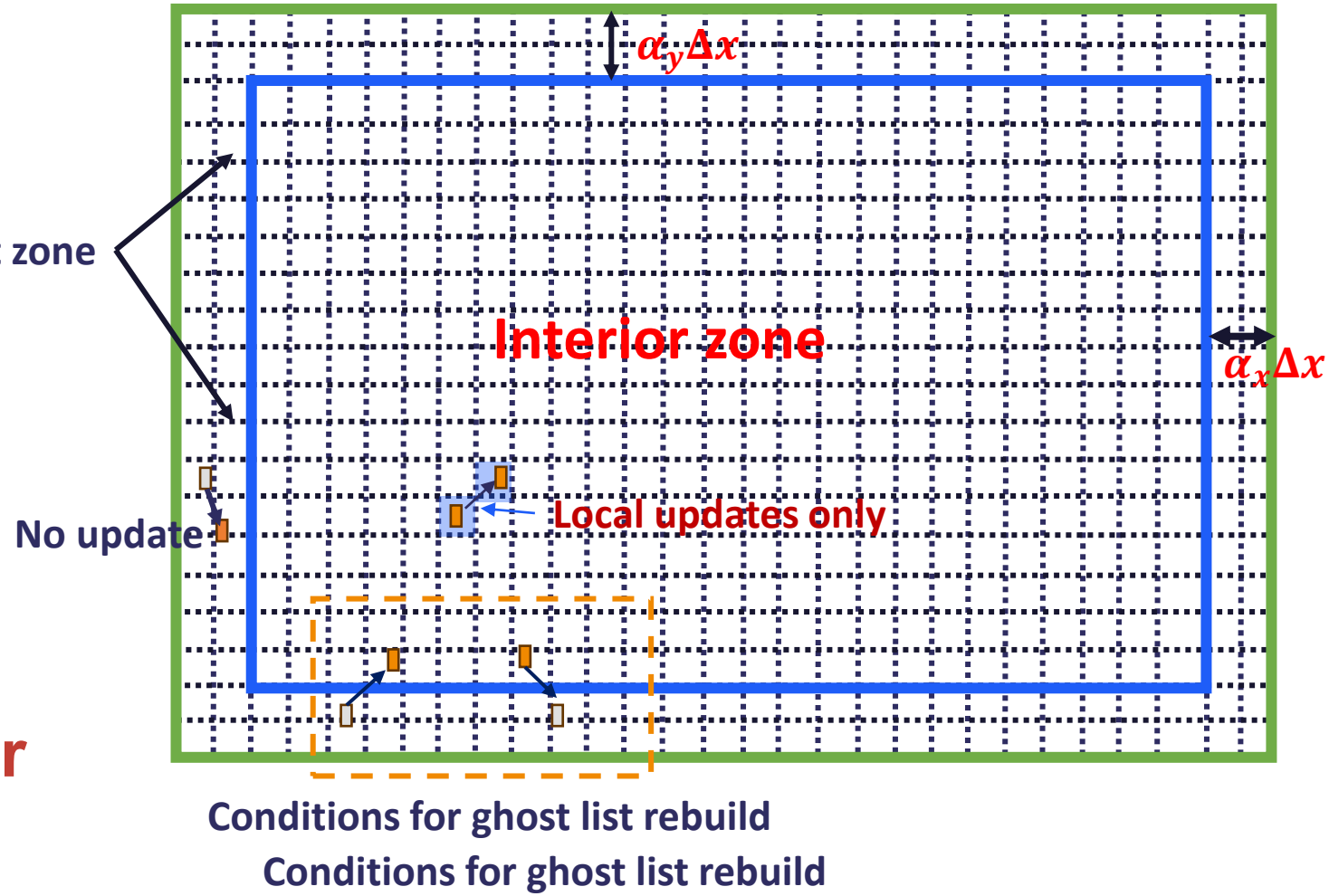
Classical approach

- Update of maps & particle halos are two independent processes
- Both built if particles in step n+1 populate different cells
- Construction of particle halos rely on box regions



Hybrid approach

- Coupling of operations of particle-to-grid projection (mapping processes) and halo reconstruction
- Reformulation of mapping rules
- Split block cells into interior & boundary regions



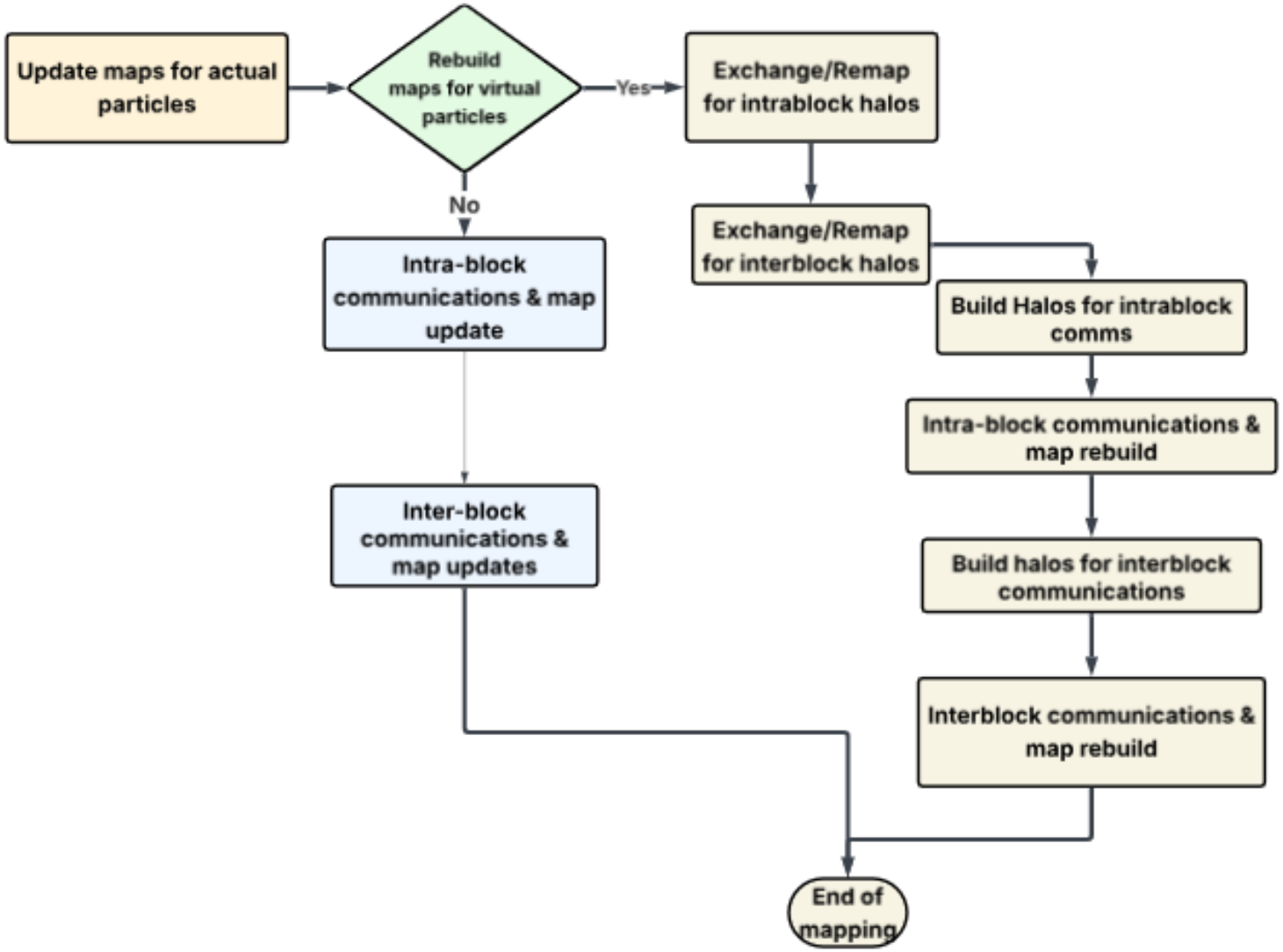
Science and Technology Facilities Cour

Engagement rules

- Partial update of maps for actual particles
- Reconstruction of virtual lists if particle enters/exits ghost zone (compression mode)
- No virtual list reconstruction if particles moves within ghost zone

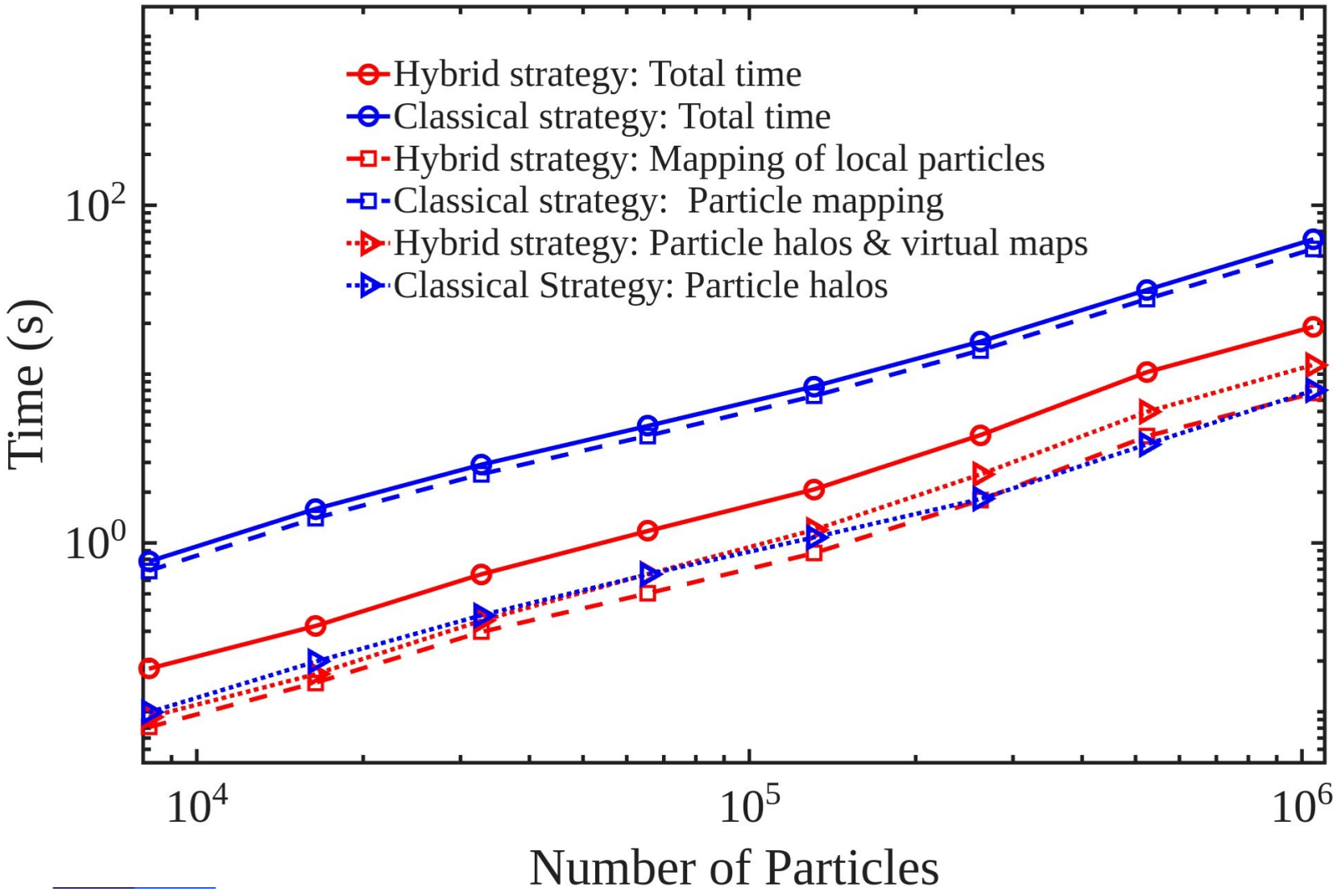
Assembly of hybrid approach for maps & halo communications

Flow chart for hybrid operations



Science and Technology Facilities Council

Performance considerations



- Hybrid scheme substantial faster
- Better performance due to
  i. Partial update of particle maps
  ii. Generation of intrablock/inter block halos based on the projection of particles to grids

Iteration operations: Particle loops

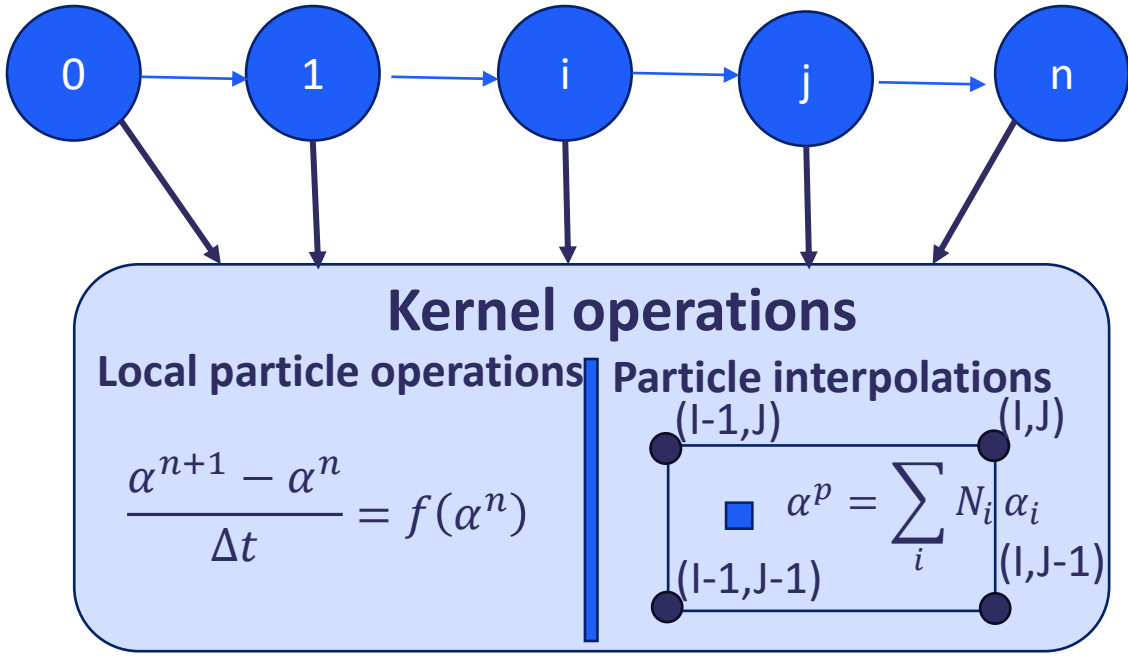
```
template <typename... ParamType, typename... OPSARG>
void
ops_particle_par_loop(void (*kernel)(ParamType...),
char const *name, ops_particle particle, int dim,
ops_particle_iterate_type iterate_type, double *range,
ops_particle_mapping map, OPSARG... arguments)
```

Function to perform basic particle operations for a given ops_particle structure

- **Update particle structures (local operation only)**
- **Rudimentary particle-grid interactions**

Features

- **`kernel`** : User operations on the particle level
- **`iterate_type`**: Flag identifying the particle set. Supported types for particle sets: Local, local with virtual and particles in given regions
- **`map`**: Map for performing rudimentary grid-particle interactions



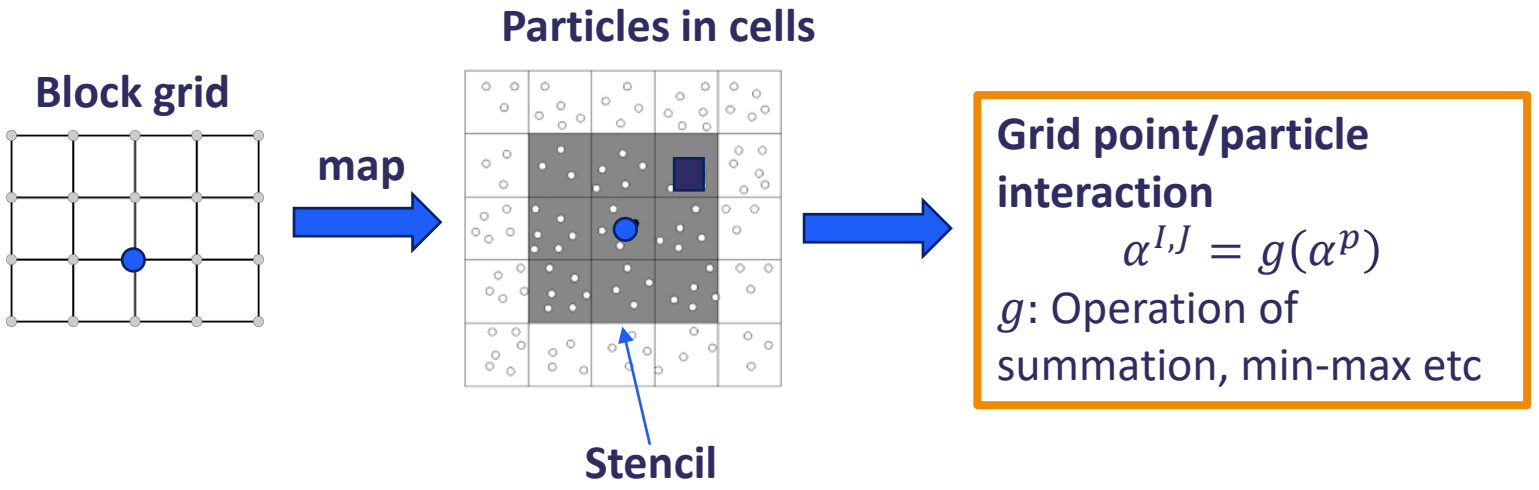
Science and Technology Facilities Council

Iteration operations: Grid-particle loops

```
template<typename... ParamType, typename... OPSARG>
void ops_par_loop(void (*kernel)(ParamType...),
     char const *name,ops_particle particle,
     ops_particle_mapping map, ops_stencil stencil,
     int dim, int *range, OPSARG... arguments)
```

- Perform advanced grid-particle operations
- Support maps with $\Delta x_M = \alpha \Delta x_G$.
- Filtering operator

- **`kernel`**: User defined grid point-particle operations
- **`map`**: Map for projecting particle to grids
- **`stencil`**: Accessing particle cells from grid points



Science and Technology Facilities Council

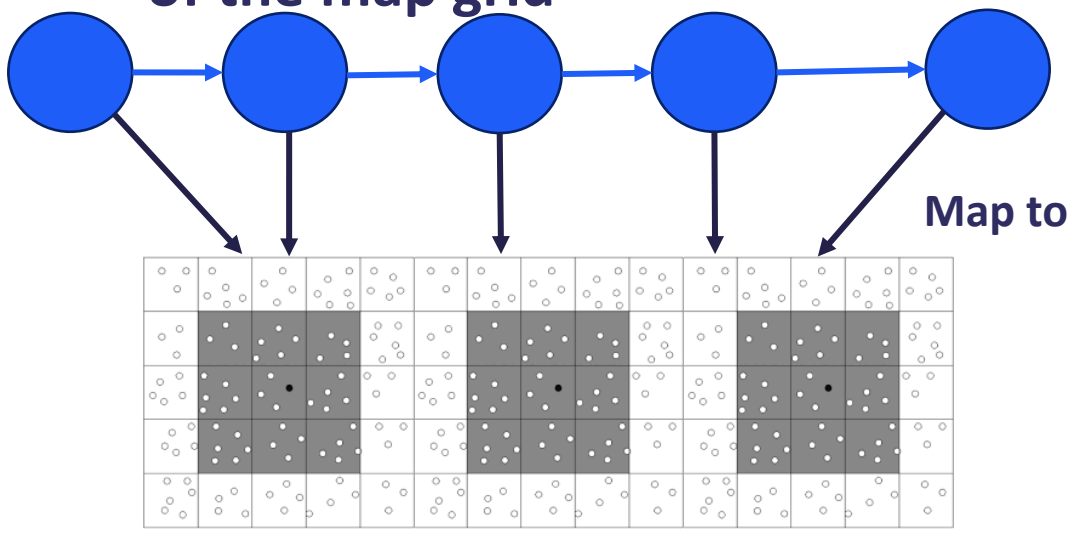
Iteration operations: Particle-Grid interactions

```
template <typename... ParamType,
typename... OPSARG>
void ops_par_particle_grid_loop(
     void (*kernel)(ParamType...),
     char const *name,
     ops_particle particle,
     ops_particle_mapping map, int dim,
     ops_particle_iterate_type iter_type,
     double *range, ops_stencil stencil,
     OPSARG... arguments)
```

- **kernel** : User-defined particle-grid node interactions
- **map** : map for accessing grid structures
- **iterate_type** : Flag to identify particle sets
- **stencil** : Stencil to access grid nodes

Perform particle-grid interactions

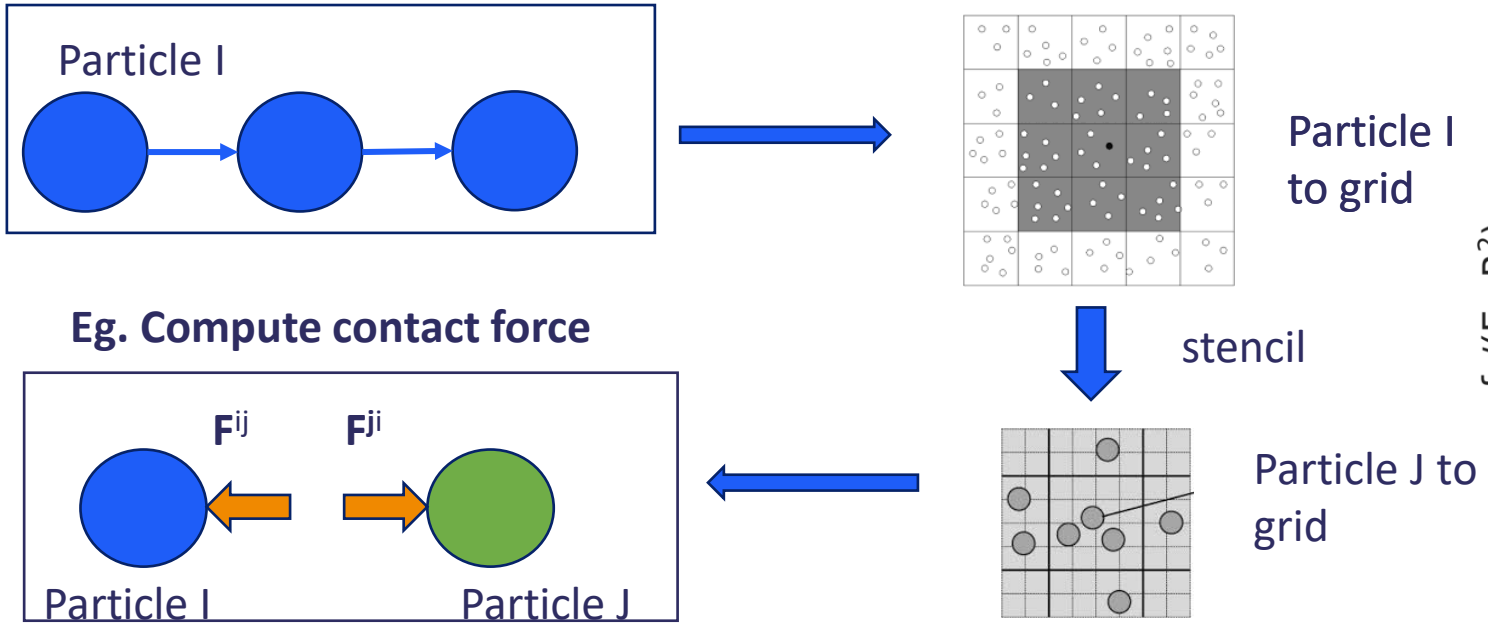
- **Support interactions with grids with sizes different than the size of the map grid**



User defines operations at particle-grid point level

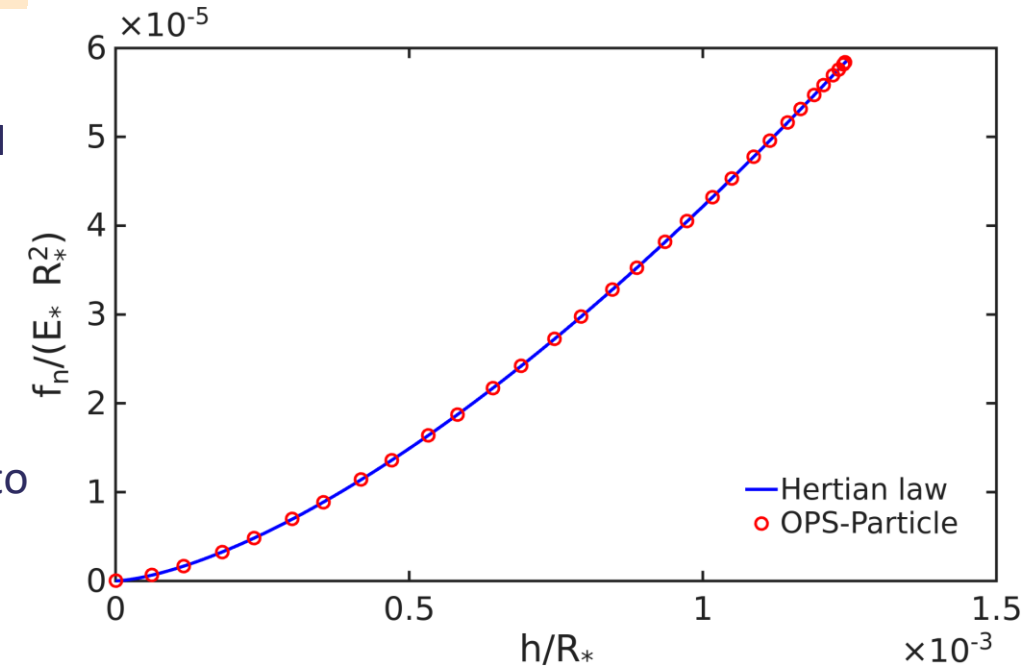
Iteration operations: Particle-Particle interactions

```
void ops_particle_inter_loop(void *kernel,
char const *name, ops_particle particleI,
ops_particle_mapping mapI, ops_particle
particleJ, ops_particle_mapping mapJ, ops_stencil
stencil, ops_particle_iterate_type iter_type,
int dim, double *range, OPSARG... arguments)
```



To handle interactions between different ops_particle structures

- **Can be used in modelling the mechanistic interactions between particles**
- **Update marker positions due to movement of individual particle**



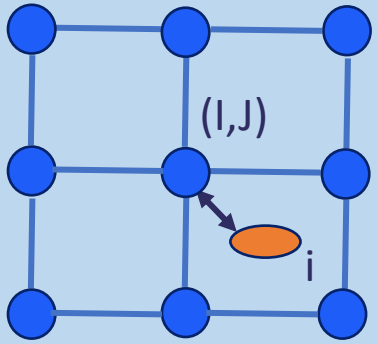
Kernel functions

Kernel functions define operations at the smallest scale

- Grid node
- Individual particle (ops_particle_par_loop)
- Interaction between a grid node with individual particle (ops_par_loop)
- Interaction between an individual particle and a grid node (ops_par_particle_grid_loop)
- The interaction between two particles of different type (ops_particle_inter_loop）

Kernel functions

Example 1: Interaction of particle with a grid to compute fluid velocity at particle I



Kernel define the contribution of node (I,J) to the center of mass of particle i

```
void KerComputeVelAtXp(ACCP<double>& ufp, const ACCP<double>& xp,
const ACC<double>& xf, const ACC<double>& u_f,
const double *dx) {

double xi = fabs((xp(0) - xf(0, 0, 0)) / (*dx));
double nu = fabs((xp(1) - xf(1, 0, 0)) / (*dx));

ufp(0) += (1. - xi) * (1. - nu) * u_f(0, 0, 0);
ufp(1) += (1. - xi) * (1. - nu) * u_f(1, 0, 0);

}
```

ACC: Access grid data

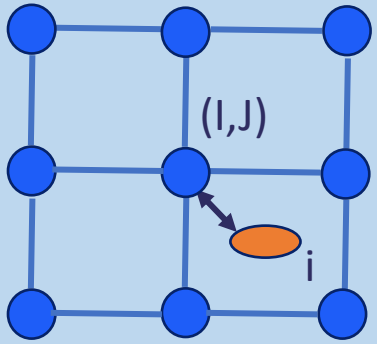
ACCP: Accessing Lagrangian data

- **Interaction between an individual particle and a grid node (ops_par_particle_grid_loop)**
- **The interaction between two particles of different type (ops_particle_inter_loop）**

Science and Technology Facilities Council

Kernel functions

Example 1: Interaction of particle with a grid to compute fluid velocity at particle I



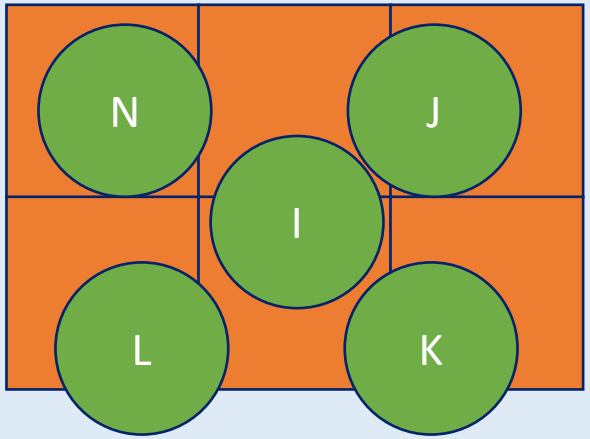
Kernel define the contribution of node (I,J) to the center of mass of particle i

```
void KerComputeVelAtXp(ACCP<double>& ufp, const ACCP<double>& xp,
const ACC<double>& xf, const ACC<double>& u_f,
const double *dx) {

double xi = fabs((xp(0) - xf(0, 0, 0)) / (*dx));
double nu = fabs((xp(1) - xf(1, 0, 0)) / (*dx));

ufp(0) += (1. - xi) * (1. - nu) * u_f(0, 0, 0);
ufp(1) += (1. - xi) * (1. - nu) * u_f(1, 0, 0);

}
```

ACC: Access grid data

ACCP: Accessing Lagrangian data

Example 2: Force calculation in particulate simulations-Frictionless particles



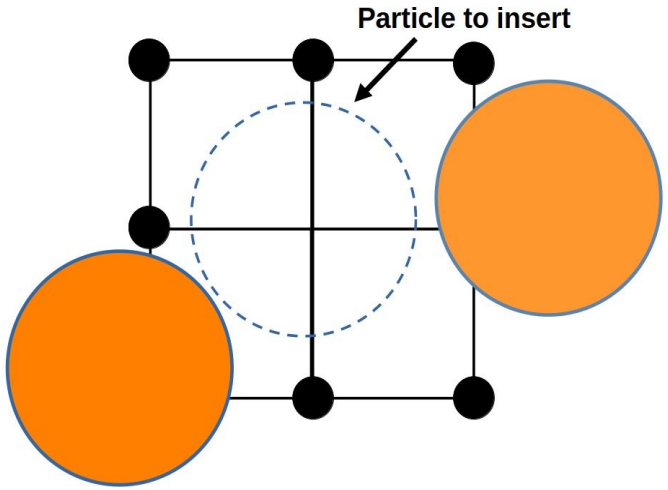
Kernel for force calculation for particles I and J

ACCPJ: To access data of particle J

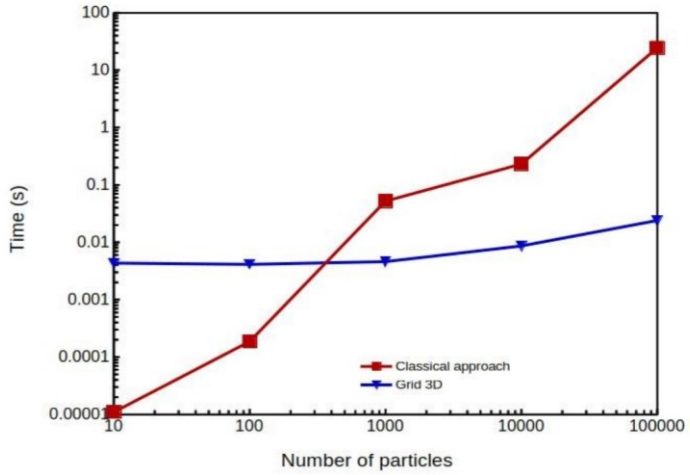
```
void KerLinearLaw(ACCP<double>& xp, ACCPJ<double>& xpj,
                  ACCP<double>& Rad, ACCPJ<double>& radj,
                  ACCP<double>& FpI, ACCPJ<double>& FpJ,
                  const double *Kn) {


  double n[2] = {xpj(0) - xp(0), xpj(1) - xp(1)};
  double d = n[0] * n[0] + n[1] * n[2];
  double rad_sq = (Rad(0) + radj(0)) * (Rad(0) + radj(0));

  if (d < rad_sq) {
  for (int i = 0; i < 2; i++) n[i] /= sqrt(d);
    double h = radj(0) + Rad(0) - sqrt(d);
    double fn = (*Kn) * h;

    FpI(0) -= fn * n[0];  //Update force to particle-I
    FpI(1) -= fn * n[1];
    FpJ(0) += fn * n[0];  //Update force to particle-J
    FpJ(1) += fn * n[1];
  }
}
```

Operations for particle initialization & deletion

Random particle insertion random data initialization

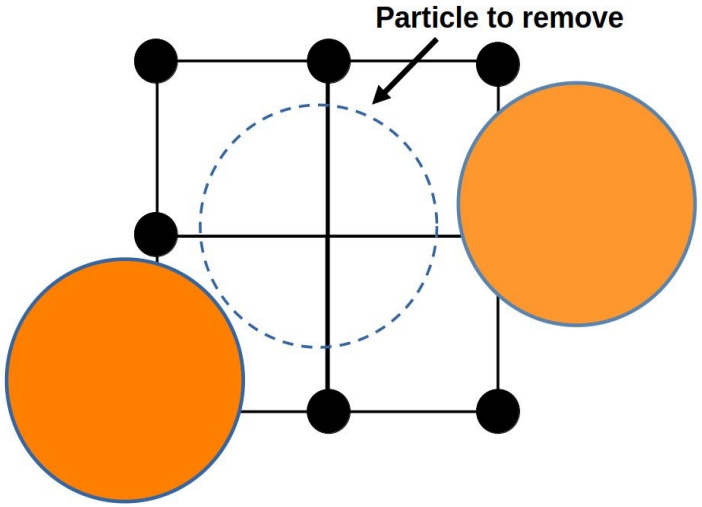



```
template<template<typename X> class
Distribution,
template<typename X1> class Distribution1,
typename T1>
void
ops_insert_random_particles(ops_particle
particle, double *region, int Nins,
int nattempts, int seed,
OPSDistribution<Distribution, double>
*rad_distr, ops_dat envelope,
OPS_dat_distr<Distribution1, T1> &distr1)
```



- Overlap checks exploit mapping structures
- Overlap checks 10x faster than traditional algorithms
- Parallel version is currently under testing

Particle removal from the system



- **Iteration functions for removing particles from the domain via user-defined criteria**

Mini-applications

Testing cases: Particles in fluids

Case I: Lagrangian tracking

Case II: Particles with D<<Δx

Case III: DNS modelling of fluid-particle systems

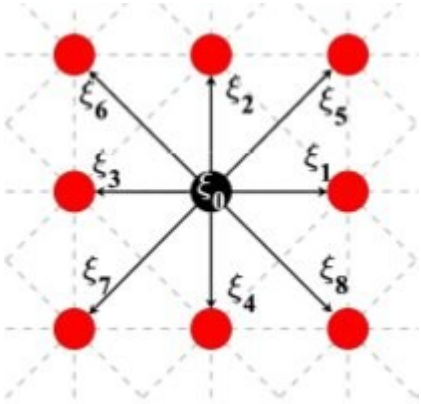
Fluid flow modeled with the Lattice Boltzmann method (LBM)

Why LBM?

- Easy to use, explicit method , excellent parallel performance

Science and Technology Facilities Council

Lattice Boltzmann in a nutshell



$$f_i(\boldsymbol{x} + \boldsymbol{\xi}_i\Delta t, t + \Delta t) = f_i(\boldsymbol{x}, t) - \frac{\Delta t}{\tau}\left(f_i - f_i^{eq}(\rho, u)\right) + \mathcal{F}_i\Delta t$$

Macroscopic variables

$$\rho = \sum_i f_i \; ; \; \rho u_a = \sum_i \xi_{i_a} f_i$$

$$\sigma_{a\beta} = \sum_i (\xi_{i_a} - u_a)(\xi_{i_\beta} - u_\beta) f_i$$

- A “finite” difference discretization of the Boltzmann-BGK equation (**Uniform grid**)
- Navier-Stokes retrieved via Chapman-Enksog expansion

Mini applications: Particle tracker in a 2D cavity flow

Simulation facts

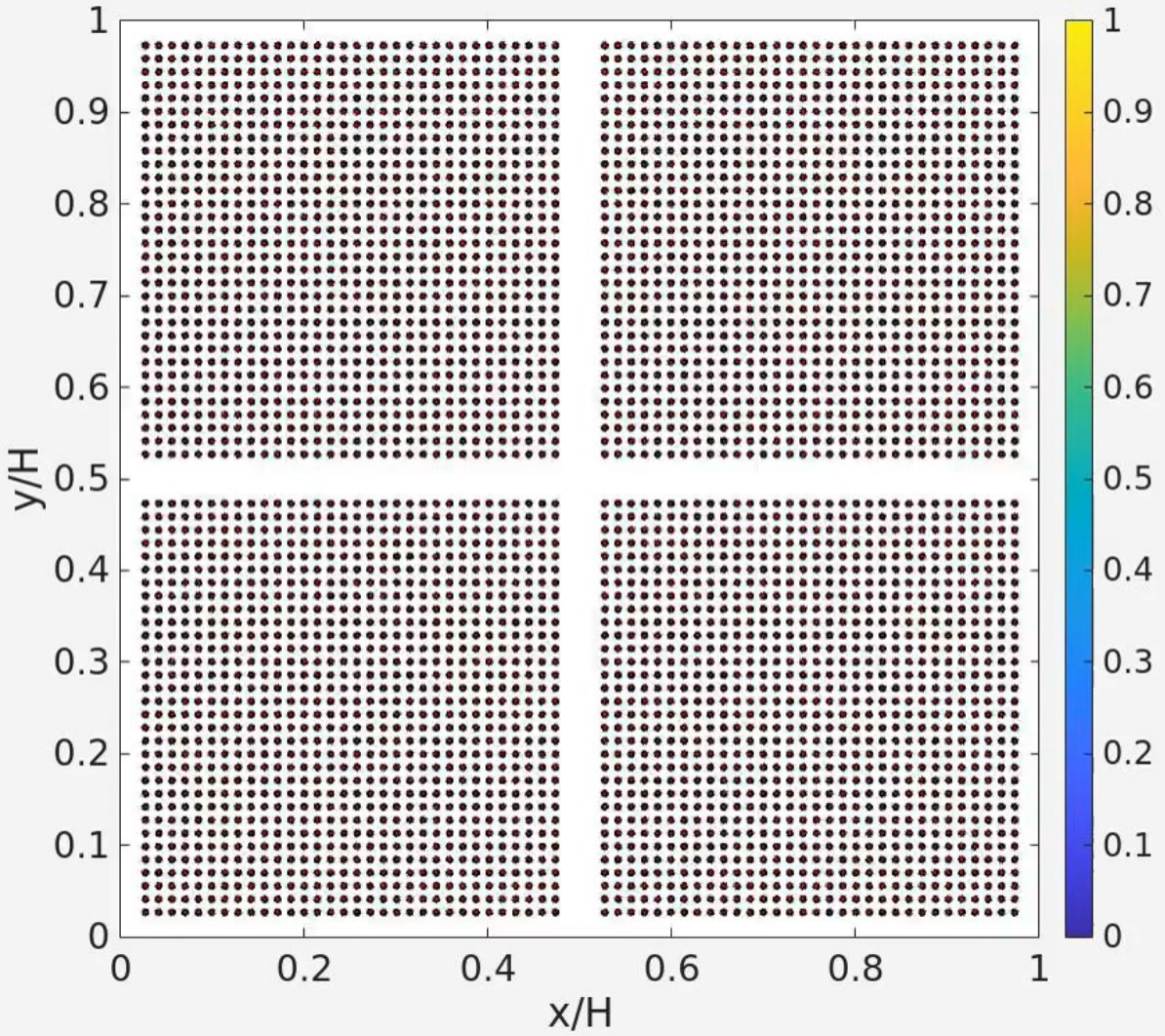
- Uniform grid of 101 x 101 points
- Relaxation time (τ) set to 0.0001
- EQDR scheme to enforce no-slip boundary condition
- U_w set to $0.01c_s$ (Re = 100)
- 4096 massless particles added in the simulation
- Run in 4 processes

Science and Technology Facilities Council

Mini applications: Particle tracker in a 2D cavity flow

Simulation facts

- Uniform grid of 101 x 101 points
- Relaxation time (τ) set to 0.0001
- EQDR scheme to enforce no-slip boundary condition
- U_w set to $0.01c_s$ (Re = 100)
- 4096 massless particles added in the simulation
- Run in 4 processes



Science and Technology Facilities Council

Mini applications: Particles immersed in a 2D cavity (D << Δx)

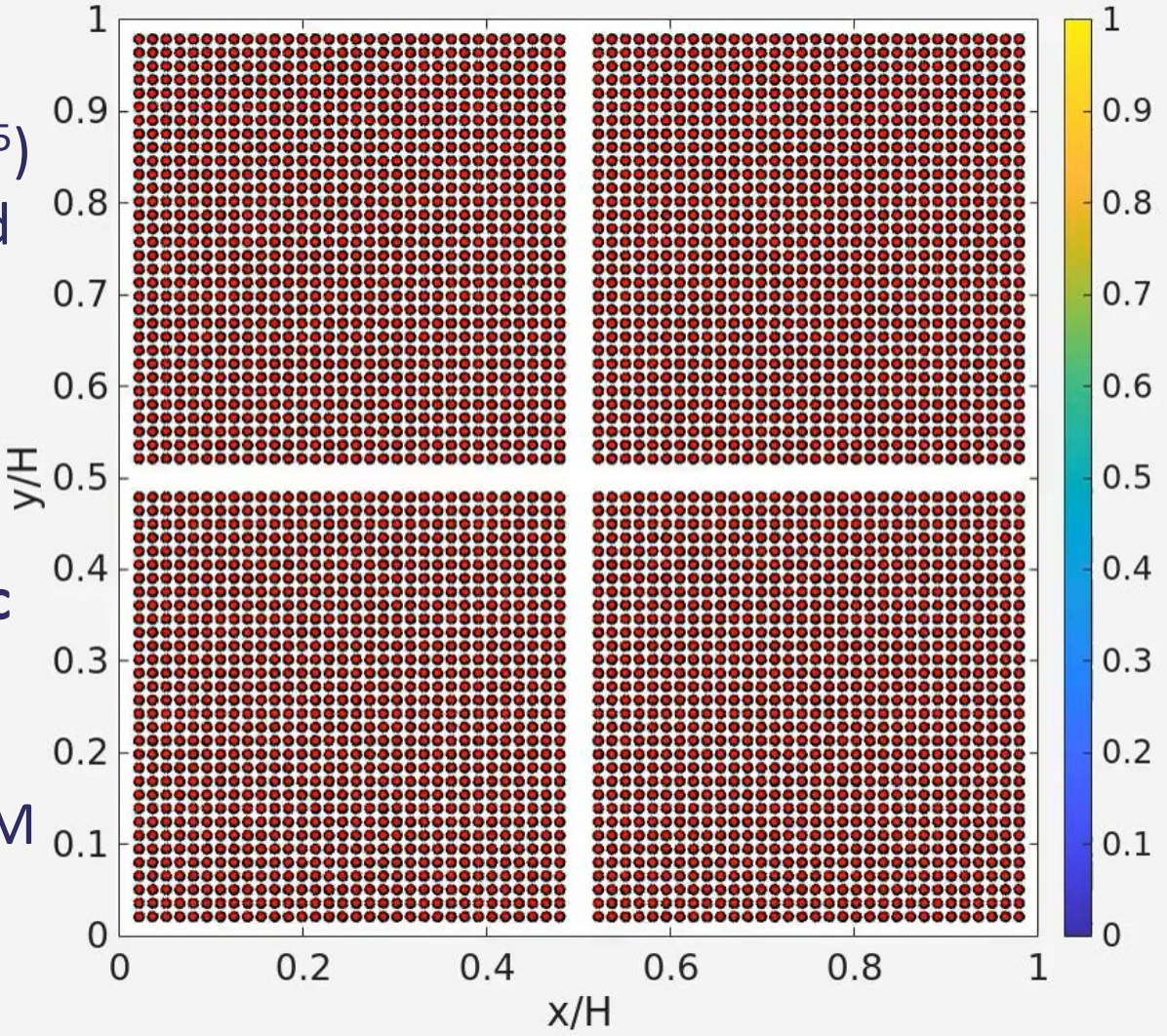
- Particles have mass now ($\rho_s = 2.6\rho$, $D = 2\ 10^{-5}$)
- Interpolation schemes used to compute fluid velocity ($\mathbf{u}^f$)at $\mathbf{x}^p$
- Drag force computed for each particle based on ($\mathbf{u}^p$-$\mathbf{u}^f$)
- Verlet algorithm to integrate particle motion
- Box filter to compute **average hydrodynamic force** at grid points
- **Martys et al**.[1] forcing scheme to compute forcing term in the evolution equation of LBM

Science and Technology Facilities Council

[1] Martys, N. S., Shan, X. & Chen, H. 1998 Evaluation of the external force term in the discrete746 Boltzmann equation. Phys. Rev. E 58 (5), 6855.747

Mini applications: Particles immersed in a 2D cavity (D << Δx)

- Particles have mass now ($\rho_s = 2.6\rho$, $D = 2\ 10^{-5}$)
- Interpolation schemes used to compute fluid velocity ($\mathbf{u}^f$)at $\mathbf{x}^p$
- Drag force computed for each particle based on ($\mathbf{u}^p$-$\mathbf{u}^f$)
- Verlet algorithm to integrate particle motion
- Box filter to compute **average hydrodynamic force** at grid points
- **Martys et al**.[1] forcing scheme to compute forcing term in the evolution equation of LBM



Science and Technology Facilities Council

[1] Martys, N. S., Shan, X. & Chen, H. 1998 Evaluation of the external force term in the discrete746 Boltzmann equation. Phys. Rev. E 58 (5), 6855.747

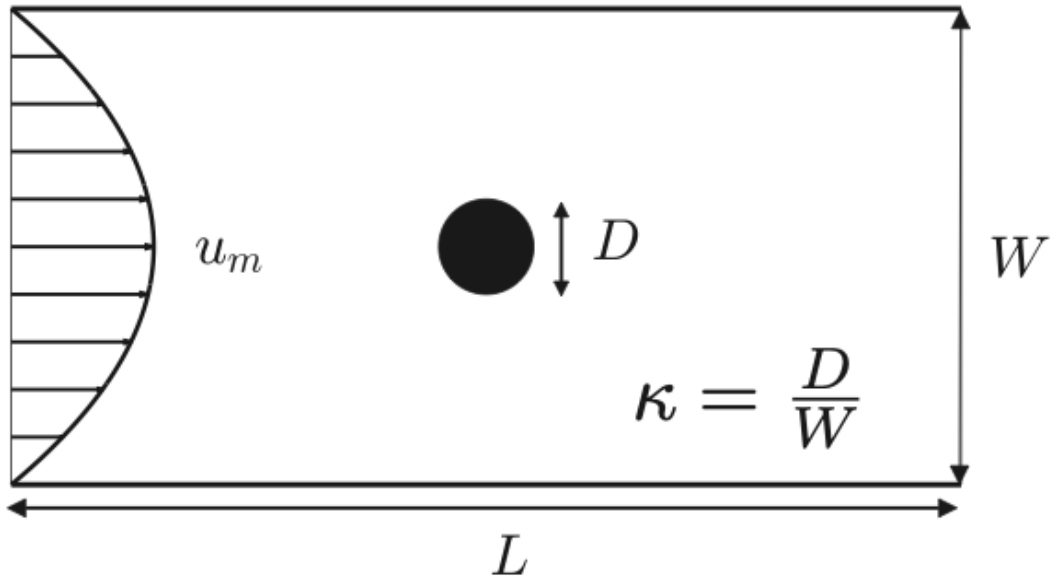
Mini-applications: DNS modeling of fluid-particle systems

- Fluid-particle interactions are modeled with the partially saturated method[1]

$$f_i(x + \xi_i \Delta t, t + \Delta t) = f_i(x, t) + (1 - B_s)\Omega_{\mathrm{i}}^{\mathrm{f}} + \mathrm{B_s}\Omega_i^s$$

$$\Omega_i^f = -\frac{1}{\tau}\left(f_i - f_i^{eq}\right) \;\&\; \Omega_i^s = \left(f_i^{eq}(\rho, u_s) - f_i\right)$$

$$B_s = \frac{\varepsilon_s\, \hat{\tau}}{\hat{\tau} - 1 - \varepsilon_s}$$



- **Testing case: Flow passed a cylinder in a narrow channel**
- **Comparison with analytical solution $\lambda_w = \frac{F_d}{\mu u_o}$**

- Particles are not mapped in the LBM grid
- Particle grid has $\Delta \mathrm{x_m} = \alpha \Delta x_f$ with $||\alpha|| > 1$

Facilities Council

[1]Noble, D. R. & Torczynski, J. R. 1998 A lattice-Boltzmann method for partially saturated computational cells. Int. J. Mod. Phys. C 9 (08), 1189–1201

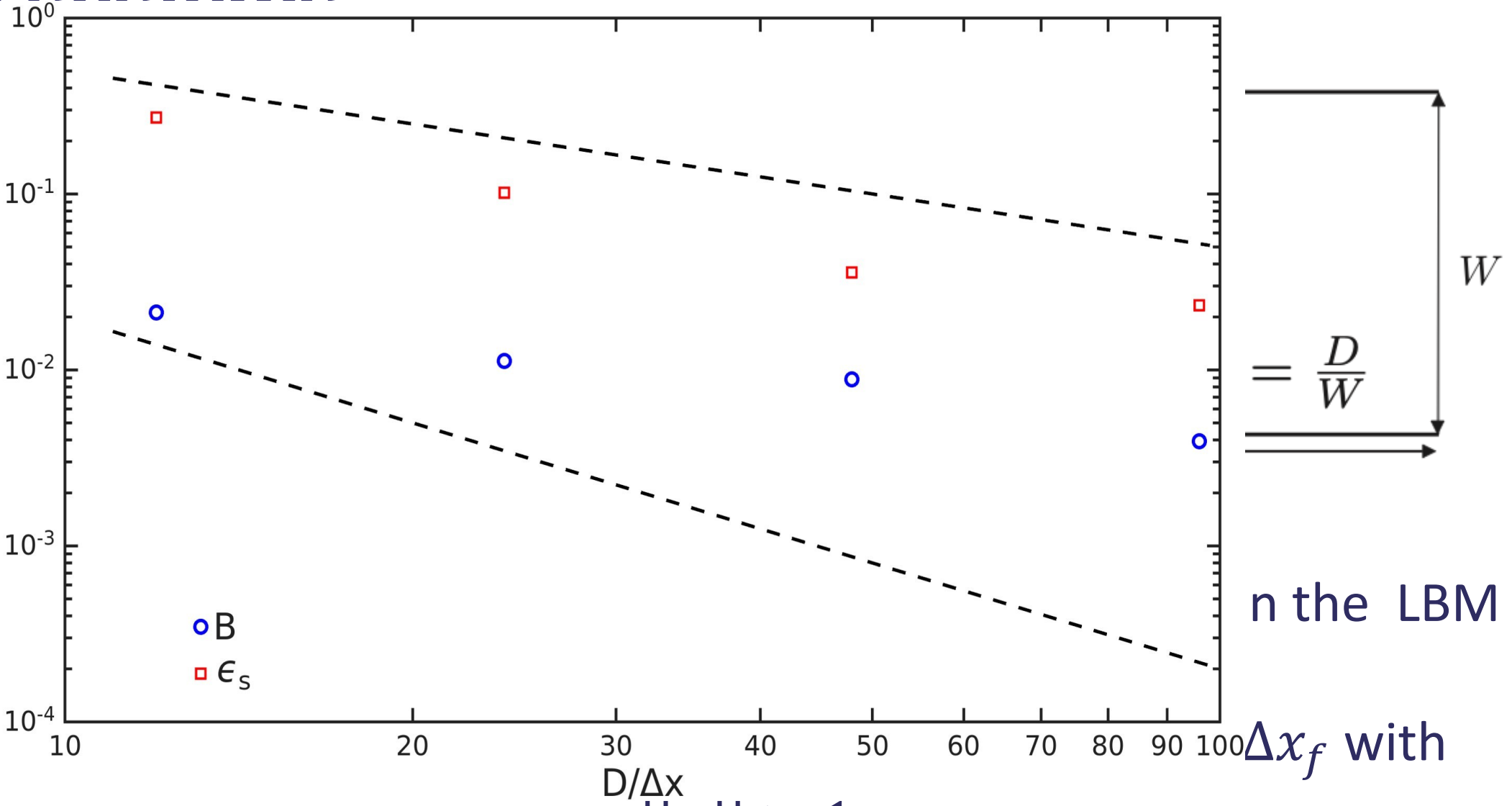
Mini-applications: DNS modeling of fluid-particle systems

- Fluid-part
  partially s
  $f_i(x + \xi_i \Delta t,$

$$\Omega_i^f = -\frac{1}{\tau}\left(f_{||E||_\lambda}\right.$$

- **Testing ca**
  narrow ch
- **Comparis**

$$\frac{F_d}{\mu u_o}$$



$= \frac{D}{W}$

n the LBM

Δx_f with

$||a|| > 1$

[1]Noble, D. R. & Torczynski, J. R. 1998 A lattice-Boltzmann method for partially saturated computational cells. Int. J. Mod. Phys. C 9 (08), 1189–1201

Conclusions & future work

- The main features of a domain specific language for Lagrangian systems are set.
- The capabilities of the OPS-Particle for modeling Lagrangian systems are demonstrated for Particle in Cell methods

Future work

- Python translator & porting to GPUs
- Code optimization
- Working on Mercury-OPS

Science and Technology Facilities Council

Science and Technology Facilities Council

Thank you

Facebook: Science and Technology Facilities Council

Twitter:@STFC_matters

YouTube: Science and Technology Facilities Council