UK Atomic Energy Authority

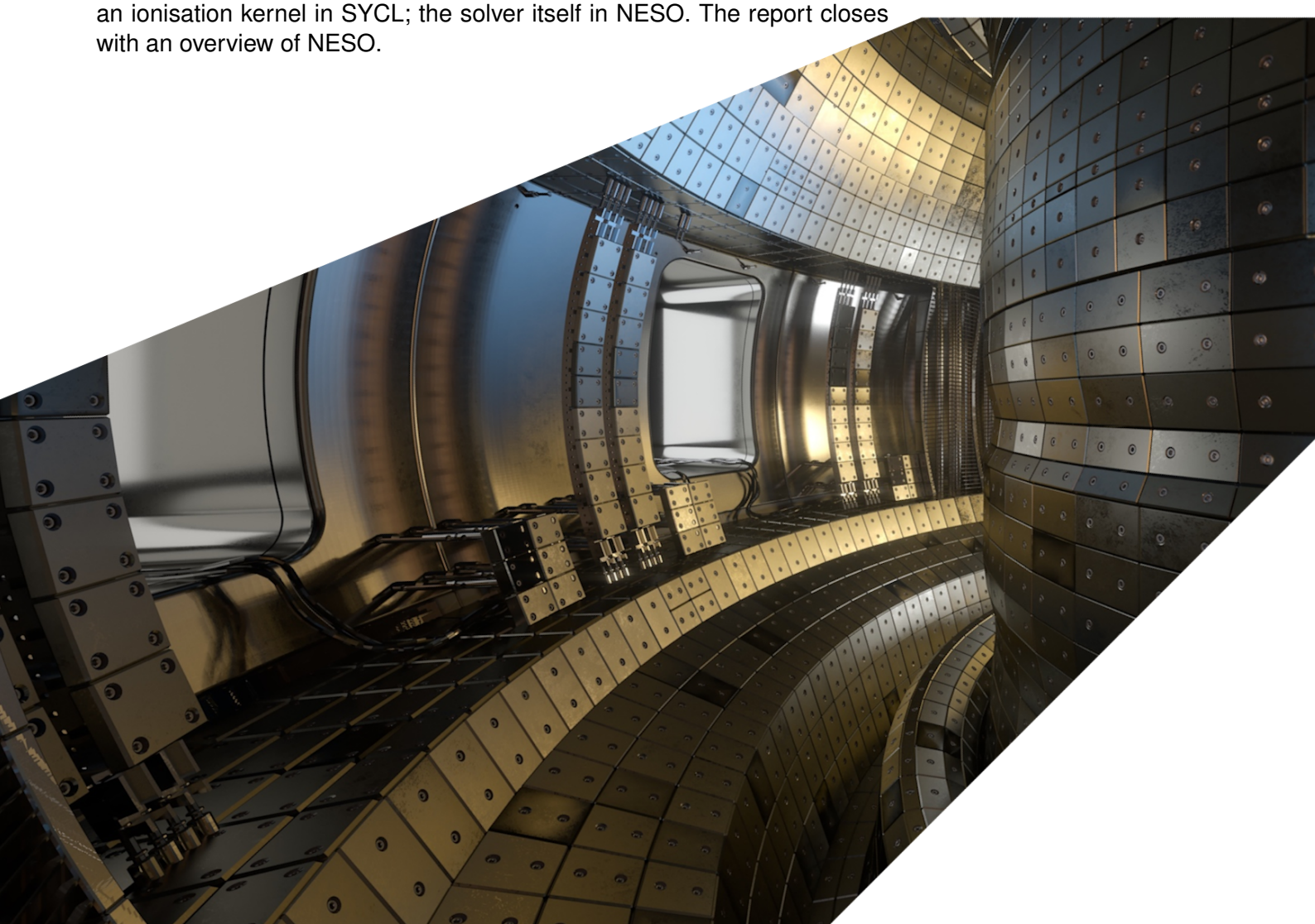
ExCALIBUR

3-D integrated particle and continuum model

M4c.3 Version 1.00

Abstract

This report describes work for ExCALIBUR project NEPTUNE at Milestone M4c.3. An integrated particle and continuum model has been developed in $3d3v$, where $3d3v$ implies that there is variation in three space dimensions (3-D) and three velocity space coordinates (3-V). Nektar++ provides the 3-D finite elements to represent the plasma as a fluid, whereas 3-D 3-V phase space is populated with neutral particles by the SYCL-enabled NESO-Particles library. The calculation is orchestrated by the NESO software, which has been developed under project NEPTUNE. We have implemented a set of fluid equations that exhibit turbulent behaviour, as shown in this report, namely the Hasegawa-Wakatani equations as a subset of the system of equations solved by HERMES-3 to model the LAPD device. Particles interact with the finite-element fluid via an ionisation source term, and variation of the fluid density changes the particle ionisation. This report describes the following developments: upgrades to NESO-Particles; an ionisation kernel in SYCL; the solver itself in NESO. The report closes with an overview of NESO.



UKAEA REFERENCE AND APPROVAL SHEET

	Client Reference:	
	UKAEA Reference:	CD/EXCALIBUR-FMS/0079
	Issue:	1.00
	Date:	Sept 30, 2023
Project Name: ExCALIBUR Fusion Modelling System		

	Name and Department	Signature	Date
Prepared By:	James Cook Will Saunders Owen Parry BD	N/A N/A N/A	Sept 30, 2023 Sept 30, 2023 Sept 30, 2023
Reviewed By:	Ed Threlfall Deputy Principal Investigator		Sept 30, 2023

1 Introduction

Interactions between turbulence and neutral particles is, as yet, an under-studied area of plasma physics despite being crucial to understanding the nuances of plasma exhaust in tokamaks. Plasma leaks out from the confined region of the tokamak past the last closed flux surface where it travels down, and to some extent radially outwards, towards the divertor region. This plasma may become turbulent, from built up gradients within the SOL, from its "initial conditions" coming out of the confined region, or otherwise. Turbulence in the SOL can cause peaks in heatloads at the divertor, in attached regions (where the plasma is in contact with the divertor), which can be deleterious to the divertor causing erosion and embrittlement, for example. It is possible for the ablated high charge state material to migrate back into the core plasma, thereby diluting the fuel and causing cooling via radiation. In detached regimes, where there is an ionisation front between the SOL plasma and the neutrals in front of the divertor, turbulence may perturb this equilibrium potentially causing a breakdown of the detachment, which should be avoided during operation. For these reasons it is important to understand the role of neutral interactions with turbulence in the SOL, and this is what motivates the work done for this report and the wider project. To this end, we have implemented the Hasegawa-Wakatani equations in 3-D with particles, as a natural extension from previous 2-D work, as a child class of a more complete set of SOL turbulence equations as implemented in the HERMES-3 software [1] and deployed on a system modelling the LAPD machine [2].

The structure of the rest of the report is as follows. In the following section we highlight the developments made to the particle functionality, namely the creation of halos and the evaluation of basis functions in various element types in support of 3-D simulations. Section 3 details the implementation of the ionisation kernel and how the neutral particles interact with the finite element fluid. In section 4 we describe the Hasegawa-Wakatani implementation in 3-D and show some initial results involving particles. Next, we describe the layout and some architectural decisions behind NESO where we discuss some ways in which we aim to make development more productive as well as highlighting some areas where improvements can be made in this direction.

2 Task Work

2.1 Particles

The extension from 2-D to 3-D adds four new mesh element types that we must consider in order to provide a functional 3-D implementation. These 3-D elements are the Tetrahedron, Pyramid, Prism and Hexahedron. We assume for this report that these are linear elements and hence do not feature curved edges or faces.

The implementation work is formed of two main components. The first component is the implementation that enables particles to exist on and be tracked over 3-D meshes. The second component couples 3-D particles with 3-D finite element functions via projection and evaluation operations.

2.1.1 3-D Halo Creation

In the mesh decomposition approach to parallelisation each MPI rank holds a portion of the overall mesh. The cells in this sub-domain are considered to be "owned" by that MPI rank. However it is computationally advantageous for an MPI rank to have copies of mesh elements that are owned by neighbouring ranks. We refer to these copies as "halo cells", or more concisely "halo(s)", however these copies are also called "ghost cells" in other literature.

In NESO-Particles (NP; [3]) halo regions are combined with a global grid structure to enable global particle transfer. This global communication algorithm relies on the existence of the halo regions to function correctly and efficiently. Figure 1 illustrates the sub-domain owned by an MPI rank augmented with halo cells from neighbouring ranks.

When a particle leaves the sub-domain owned by an MPI rank one of two scenarios occurs. In the first scenario the new position is contained within a halo cell held by the rank. If the position is in a halo cell then the MPI rank which owns the halo cell should be sent the particle. If the new particle position falls outside the halo region then the particle should be sent through the global communication algorithm via the global data structure. A description of this global move algorithm is described in [4].

Halo regions are constructed during the setup of the simulation. This setup stage is a two part process: 1) identify which mesh cells are required on each MPI rank to form the halo region, and 2) pack, exchange and unpack these geometry objects. As in the 2-D case, the coarse Cartesian mesh, which forms the global data structure, provides the information required to efficiently identify the communication pattern for these geometry objects to conduct step 1).

For step 2) we identify that, in Nektar++, 3-D geometry objects are constructed, in the C++ sense of the word constructed, from the 2-D geometry objects which form the faces. As NESO already contained the implementation required to exchange 2-D geometry objects, the natural implementation to exchange 3-D objects is to exchange the 2-D faces using the existing 2-D mechanisms along with the additional information required to construct the 3-D objects.

In Nektar++ all geometry objects are given a globally unique (within each dimension) integer label. As all 2-D geometry objects have a globally unique integer label, the description of a 3-D object

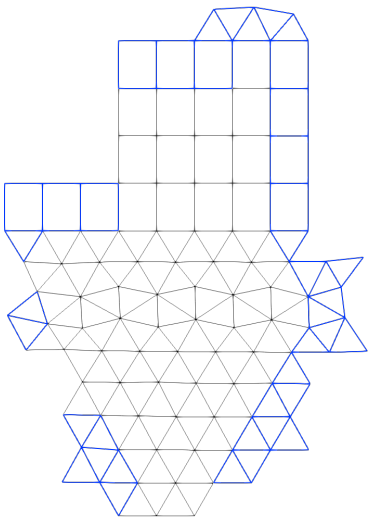


Figure 1: 2-D sub-domain on a single rank with owned elements illustrated by thin black lines and halo elements illustrated by thicker blue lines.

can be simply constructed by a single integer that describes the 3-D object type followed by the integer labels of the constituent faces. As the receiving MPI rank already holds the 2-D geometry objects, which we communicated via the 2-D communication functions, a set of 3-D geometry objects can be communicated via an array of integers. In Figure 2 we present an example of a sub-domain, as owned by an MPI rank, along with the built halo region.

2.1.2 Halo Extension

When a particle leaves a sub-domain it holds a position which is either inside or outside the halo region. If this particle position is inside the halo region then the particle is transferred directly to the new owning rank with an MPI communication pattern which is both local and determined at simulation setup. This is in contrast to the global communication procedure which is global in nature and hence less efficient. Hence the larger the halo region the greater the probability that a leaving particle is communicated to the new owning rank via a local communication approach instead of a global communication approach.

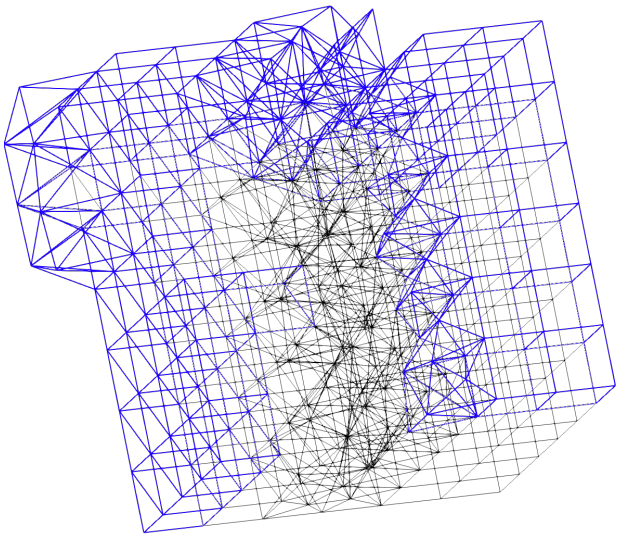


Figure 2: 3-D sub-domain on a single rank with owned elements illustrated by thin black lines and halo elements illustrated by thicker blue lines.

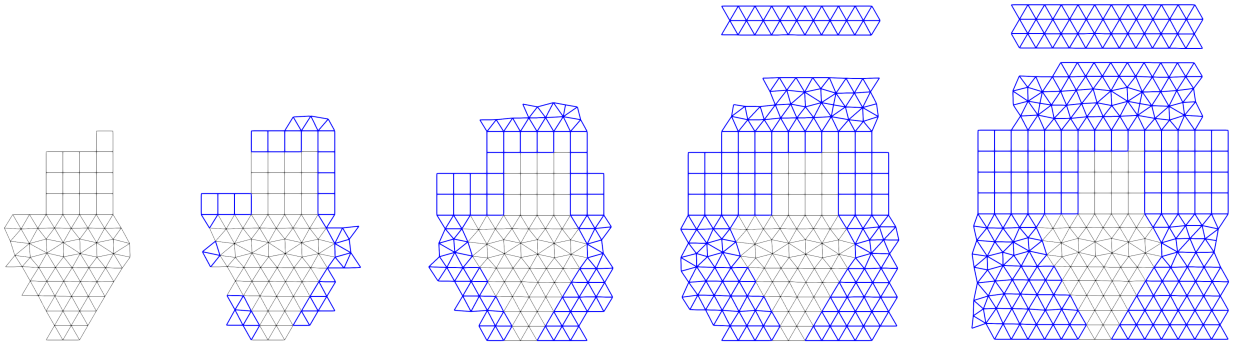


Figure 3: 2-D sub-domain on a single rank with owned elements illustrated by thin black lines and halo elements illustrated by thicker blue lines. Moving left to right increases the width of the halo region from no halos (pre-setup) to an extremely large halo which wraps around the periodic boundary.

For the plasma use case we may model highly directional plasma flow where it is advantageous to grow the size of the halo in particular directions. For the moment NESO can grow the halo region in all directions. In Figure 3 we illustrate halo extension on a periodic square domain for increasing halo width. Although this example is 2-D the implementation will extend the halo regions on 3-D domains by applying the same algorithm.

2.1.3 3-D Cell Mapping

In Nektar++ there are three coordinate systems which we consider to determine if a particle resides within a particular mesh element. These three spaces are illustrated in Figure 4 for an example triangle. As an input the user specifies a mesh containing many elements; these elements are specified in a coordinate system which we will refer to as "physical" coordinates. In many FEM implementations, for computational efficiency reasons, it is beneficial to perform operations on a "reference" element for each geometry object type and to map from the reference space (ξ) to the physical space.

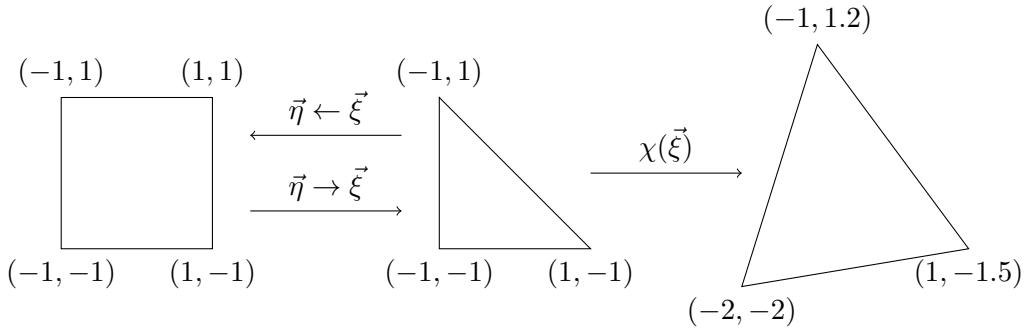


Figure 4: Illustration of Nektar++ coordinate systems in 2-D. Left: "collapsed" coordinate system ($\vec{\eta}$). Middle: "standard" triangle reference element ($\vec{\xi}$). Right: physical space coordinate system as found in the input mesh ($\chi(\vec{\xi})$).

We denote the map from reference space $\vec{\xi}$ to physical space $\vec{x}$ as $\chi(\vec{\xi})$. Typically χ is a non-linear map that is relatively straightforward to evaluate. However the inverse map χ^{-1} is non-trivial and is typically evaluated via a Newton iteration. In addition to the "standard" reference elements Nektar++ contains a "collapsed" coordinate space which is defined as $[-1,1] \times [-1,1]$ in 2-D and $[-1,1] \times [-1,1] \times [-1,1]$ in 3-D for all element types. The maps between the reference coordinate space $\vec{\xi}$ and collapsed coordinate space $\vec{\eta}$ are given in closed form for each direction and element and are relatively easy to evaluate.

At each time step of a simulation the positions of each particle in the system are updated and the element that contains each particle must be computed. More formally, for a given particle position $\vec{r}_i$ we must find an element e and reference position $\vec{\xi}_i$ such that $\chi_e(\vec{\xi}_i) = \vec{r}_i$. The element e contains the particle if $\vec{\xi}_i$ is contained within the reference element. As determining whether $\vec{\xi}_i$ is inside the reference element is itself a non-trivial computation, and would be element type-dependent, we instead use the collapsed coordinates $\vec{\eta}_i$ and test if $\vec{\eta}_i$ is inside $[-1,1]^d$. An overview of the process to determine the element containing a given particle position is given in Algorithm 1.

for *particle* i *with position* $\vec{r}_i$ **do**
  for *candidate element* e *for position* $\vec{r}_i$ **do**
    Compute $\vec{\xi}_i$ such that $\chi_e(\vec{\xi}_i) = \vec{r}_i$ via Newton iteration
    Set $\vec{\eta}_i =$ to_collapsed$(\vec{\xi}_i)$
    if $\vec{\eta}_i \in [-1, 1]^d$ **then**
      $\vec{r}_i$ is contained inside element e.
      Store $\vec{\eta}_i$ for basis function evaluation.
      break.
    end
  end
end

Algorithm 1: Algorithm to determine the element e containing a particle i with position $\vec{r}_i$. We assume that a coarse map exists to map positions $\vec{r}$ to candidate elements e based on bounding boxes.

For linear sided elements each of the four 3-D element types have a map from reference space $\vec{\xi}$ to physical space $\chi(\vec{\xi})$. These maps are functions constructed using the vertices of the element in physical space and are essentially a one-to-one mapping between a vertex in reference space and a vertex in physical space. These χ mappings are explicitly defined as

$$\chi_{\text{triangle}}(\vec{\xi}) = \frac{1}{2}\left[\vec{v}_1 - \vec{v}_0, \vec{v}_2 - \vec{v}_0\right]\left[\vec{\xi} + \begin{bmatrix}1\\1\end{bmatrix}\right] + \vec{v}_0 \tag{1}$$

$$\begin{aligned}\chi_{\text{quadrilateral}}(\vec{\xi}) = \frac{1}{4}\Bigg(&\vec{v}_0(1-\xi_0)(1-\xi_1) + \vec{v}_1(1+\xi_0)(1-\xi_1)\\ &+ \vec{v}_3(1-\xi_0)(1+\xi_1) + \vec{v}_2(1+\xi_0)(1+\xi_1)\Bigg)\end{aligned} \tag{2}$$

$$\chi_{\text{tetrahedron}}(\vec{\xi}) = \frac{1}{2}\left[\vec{v}_1 - \vec{v}_0, \vec{v}_2 - \vec{v}_0, \vec{v}_3 - \vec{v}_0\right](\vec{\xi} + [1,1,1]^T) + \vec{v}_0. \tag{3}$$

The pyramid mappings are more involved and are written in terms of the collapsed coordinates $\vec{\eta}$,

$$\chi_{\text{pyramid}}(\vec{\xi}) = c_0\vec{v}_0 + c_1\vec{v}_1 + c_2\vec{v}_2 + c_3\vec{v}_3 + c_4\vec{v}_4 + c_5\vec{v}_5 \tag{4}$$

$$\text{where } d_2 = 1 - \xi_2 \tag{5}$$

$$\eta_0 = 2((1+\xi_0)/d_2) - 1 \tag{6}$$

$$\eta_1 = 2((1+\xi_1)/d_2) - 1 \tag{7}$$

$$\eta_2 = \xi_2 \tag{8}$$

$$c_0 = (1/8)(1-\eta_0)(1-\eta_1)(1-\eta_2) \tag{9}$$

$$c_1 = (1/8)(1+\eta_0)(1-\eta_1)(1-\eta_2) \tag{10}$$

$$c_2 = (1/8)(1+\eta_0)(1+\eta_1)(1-\eta_2) \tag{11}$$

$$c_3 = (1/8)(1-\eta_0)(1+\eta_1)(1-\eta_2) \tag{12}$$

$$c_4 = (1+\eta_2)/2. \tag{13}$$

In a similar fashion, for the prism,

$$\chi_{\text{prism}}(\vec{\xi}) = c_0\vec{v}_0 + c_1\vec{v}_1 + c_2\vec{v}_2 + c_3\vec{v}_3 + c_4\vec{v}_4 + c_5\vec{v}_5 \tag{14}$$
$$\text{where } \eta_0 = 2((1+\xi_0)/(1-\xi_2)) - 1 \tag{15}$$
$$c_0 = (1/8)(1-\eta_0)(1-\xi_1)(1-\xi_2) \tag{16}$$
$$c_1 = (1/8)(1+\eta_0)(1-\xi_1)(1-\xi_2) \tag{17}$$
$$c_2 = (1/8)(1+\eta_0)(1+\xi_1)(1-\xi_2) \tag{18}$$
$$c_3 = (1/8)(1-\eta_0)(1+\xi_1)(1-\xi_2) \tag{19}$$
$$c_4 = (1/4)(1-\xi_1)(1+\xi_2) \tag{20}$$
$$c_5 = (1/4)(1+\xi_1)(1+\xi_2). \tag{21}$$

Finally the hexahedron mapping is given by

$$\begin{aligned}\chi_{\text{hexahedron}}(\vec{\xi}) = \frac{1}{8}\big(&\vec{v}_0(1-\xi_0)(1-\xi_1)(1-\xi_2)\\ &+ \vec{v}_1(1+\xi_0)(1-\xi_1)(1-\xi_2)\\ &+ \vec{v}_2(1+\xi_0)(1+\xi_1)(1-\xi_2)\\ &+ \vec{v}_3(1-\xi_0)(1+\xi_1)(1-\xi_2)\\ &+ \vec{v}_4(1-\xi_0)(1-\xi_1)(1+\xi_2)\\ &+ \vec{v}_5(1+\xi_0)(1-\xi_1)(1+\xi_2)\\ &+ \vec{v}_6(1+\xi_0)(1+\xi_1)(1+\xi_2)\\ &+ \vec{v}_7(1-\xi_0)(1+\xi_1)(1+\xi_2)\big)\end{aligned} \tag{22}$$

where for all mappings we use the notation $\vec{\xi} = [\xi_0, \xi_1]^T$ in 2-D and $\vec{\xi} = [\xi_0, \xi_1, \xi_2]^T$ in 3-D.

To perform the Newton iterations we applied a static polymorphism technique known as Curiously Recurring Template Pattern (CRTP) to separate the particular definition of χ from the Newton iteration loop. This separation allows NESO to implement generic Newton method iterations which are templated for a particular element type at compile time. The abstract Newton method interface is defined in an abstract base class `MappingNewtonIterationBase` and specialisations are implemented for each of the element types.

The specialisations of the Newton iteration base class define various methods for each of the element types. Most importantly the specialisation defines how a Newton iteration occurs, ie. compute new iteration $\vec{\xi}^{n+1}$ from current iteration $\vec{\xi}^{n}$, and defines the residual computation $|\chi(\vec{\xi}) - \vec{x}|$ where $\vec{x}$ is the target physical coordinate. These methods are non-trivial to implement by hand but are readily generated by symbolic computation packages such as *Sympy*[5]. An example of the generated implementation for a residual computation is provided in Listing 1 for a simple example map. From a software engineering point of view this code generation approach allows these involved expressions to be computed and tested in a robust manner.

Listing 1: Generated implementation to compute the residual for a $\chi_{\text{quadrilateral}}$ map.

```
/**
 * Compute and return F evaluation where
 *
```

```
 * F(xi) = X(xi) - X_phys
 *
 * where X_phys are the global coordinates. X is defined as
 *
 *
 * X(xi) = 0.25 * v0 * (1 - xi_0) * (1 - xi_1) +
 *         0.25 * v1 * (1 + xi_0) * (1 - xi_1) +
 *         0.25 * v3 * (1 - xi_0) * (1 + xi_1) +
 *         0.25 * v2 * (1 + xi_0) * (1 + xi_1)
 *
 *
 * This is a generated function. To modify this function please edit
 * the script that generates this function. See top of file.
 *
 * @param[in] xi0 Current xi_n point, x component.
 * @param[in] xi1 Current xi_n point, y component.
 * @param[in] v00 Vertex 0, x component.
 * @param[in] v01 Vertex 0, y component.
 * @param[in] v10 Vertex 1, x component.
 * @param[in] v11 Vertex 1, y component.
 * @param[in] v20 Vertex 2, x component.
 * @param[in] v21 Vertex 2, y component.
 * @param[in] v30 Vertex 3, x component.
 * @param[in] v31 Vertex 3, y component.
 * @param[in] phys0 Target point in global space, x component.
 * @param[in] phys1 Target point in global space, y component.
 * @param[in, out] f0 Current f evaluation at xi, x component.
 * @param[in, out] f1 Current f evaluation at xi, y component.
 */
inline void newton_f_linear_2d(
  const REAL xi0, const REAL xi1, const REAL v00,
  const REAL v01, const REAL v10, const REAL v11,
  const REAL v20, const REAL v21, const REAL v30,
  const REAL v31, const REAL phys0,
  const REAL phys1, REAL *f0, REAL *f1
) {
  const REAL x0 = xi0 - 1;
  const REAL x1 = xi1 - 1;
  const REAL x2 = xi0 + 1;
  const REAL x3 = 0.25 * x1 * x2;
  const REAL x4 = xi1 + 1;
  const REAL x5 = 0.25 * x0 * x4;
  const REAL f0_tmp = -phys0 + 0.25 * v00 * x0 * x1 - v10 * x3 +
                      0.25 * v20 * x2 * x4 - v30 * x5;
  const REAL f1_tmp = -phys1 + 0.25 * v01 * x0 * x1 - v11 * x3 +
                      0.25 * v21 * x2 * x4 - v31 * x5;
```

```
  *f0 = f0_tmp;
  *f1 = f1_tmp;
}
```

2.1.4 3-D Basis Function Evaluation

In report [6] we describe an L2 Galerkin approach to convert a particle representation of a field into a FEM based representation. This projection approach relies on evaluating the FEM basis functions at the location of each particle in the system. As we perform the projection, and evaluation, operations at each time step and for each particle a set of basis functions must be evaluated. The overall cost of these operations is computationally very significant. By using recursion relations we implement the evaluation of the required basis functions for N particles with $\mathcal{O}(Np^d)$ for polynomial order p and dimension d.

The four sets of so-called "modified" basis functions are defined as

$$\psi_p^A(z) = \begin{cases} (1-z)/2 & \text{if } p = 0 \\ (1+z)/2 & \text{if } p = 1 \\ (1/4)(1-z)(1+z)P_{p-2}^{1,1}(z) & \text{otherwise} \end{cases} \tag{23}$$

$$\psi_{pq}^B(z) = \begin{cases} \psi_q^A(z) & \text{if } p = 0 \\ ((1-z)/2)^p & \text{if } p \neq 0, q = 0 \\ ((1-z)/2)^p((1+z)/2)P_{q-1}^{2p-1,1}(z) & \text{otherwise} \end{cases} \tag{24}$$

$$\psi_{pqr}^C(z) = \psi_{(p+q)r}^B(z) \tag{25}$$

$$\psi_{pqr}^{PyrC}(z) = \begin{cases} \psi_{qr}^B(z) & \text{if } p = 0 \\ \psi_{1r}^B(z) & \text{if } p = 1, q = 0 \\ \psi_{qr}^B(z) & \text{if } p = 1, q \neq 0 \\ \psi_{pr}^B(z) & \text{if } p > 1, q < 2 \\ ((1-z)/2)^{p+q-2} & \text{if } p > 1, q \geq 2, r = 0 \\ ((1-z)/2)^{p+q-2}((1+z)/2)P_{r-1}^{2p+2q-3,1}(z) & \text{otherwise} \end{cases} \tag{26}$$

where $P_p^{\alpha,\beta}$ denotes a p^{th} order Jacobi polynomial. Note that although these equations specify the underlying basis functions additional modifications are made for particular element types.

For each d-dimensional element type in Nektar++ there exists a d-tuple of basis functions for that element. This set of basis functions can be considered as an order p basis for each dimension. The basis functions for the element types are as follows:

Quadrilateral ψ^A, ψ^A

Triangle ψ^A, ψ^B

Tetrahedron ψ^A, ψ^B, ψ^C

Pyramid ψ^A, ψ^A, ψ^{PyrC}

Prism ψ^A, ψ^A, ψ^B

Hexahedron ψ^A, ψ^A, ψ^A

For each of these element types there exists a double, in the 2-D case, or triple in the 3-D case, loop over the p,q and r (in 3-D) indices that index the basis. It is in these triple loops that corrections to the basis functions are made when evaluating the entire set of basis functions. As these loops are non-trivial we provide a reference implementation in NESO.

Evaluating these basis functions is a major computational cost in NESO based simulations. Furthermore the tensor product structure of the loops provides high arithmetic intensity for larger polynomial orders. As for the Newton iteration implementation we apply the CRTP method to separate the parallel loop over particles and DOFs from the particular specialisation required for each element type.

A significant advantage of the projection approach in comparison to traditional deposition approaches is the natural extension to complex geometry. In Figure 5 we plot a source particle distribution and the corresponding L2 projection onto a Continuous Galerkin function space constructed with cubic polynomials and homogeneous Dirichlet boundary conditions. The domain for this projection example is half a torus discretised into a Tetrahedron mesh.

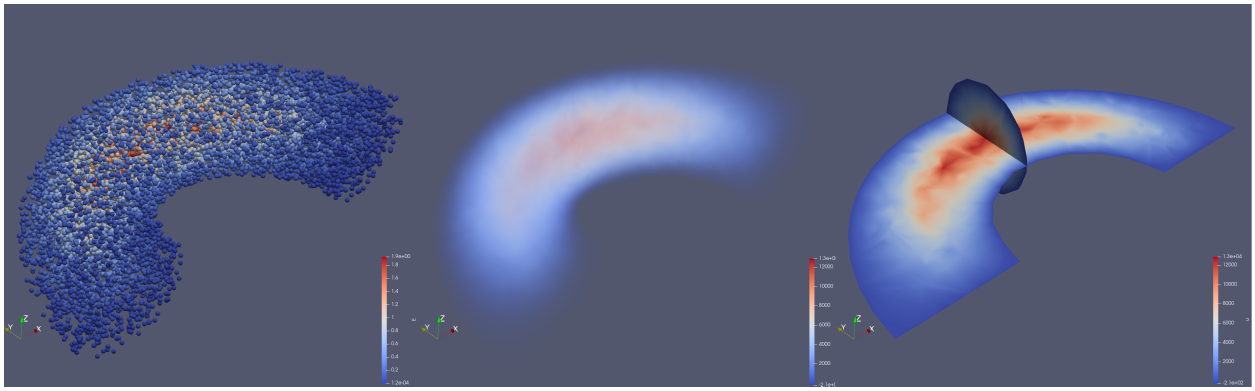


Figure 5: Left: Subset of particles (10^4 out of 10^6 total) to visualise the quantity-of-interest which is represented in the finite element function space. Middle: Volume representation of FEM representation. Right: Slices through FEM representation.

This concludes the present discussion of the particles implementation.

2.2 Charge Exchange

Reactions (ionization, charge exchange, etc.) are used to couple the particles and plasma fluid implementations. Charge exchange reactions have been added to NESO since the date of the last report.

Charge exchange is the process in which an electron is exchanged between two colliding atoms. To include this process into NESO involves considering the following reaction

$$H + H^+ \rightarrow H^+ + H. \tag{27}$$

As the number of H and H^+ is the same in the final state as it was in the initial state, then the ion density (n_{ions}) and neutral density ($n_{neutrals}$) are unchanged in this particular reaction. The momentum of the H atom in the final state does not however generally equal the momentum of the H atom in the initial state in general though, as its value takes on the momentum of the initial H^+ ion, and vice versa for the final state momentum of the H^+ ion. This means that charge exchange is a mechanism by which momentum can be exchanged between the charged and neutral species. In this work, the neutral species is represented by particles and the charged species are represented by a fluid. As such, charge exchange is a mechanism by which momentum can be exchanged between particles and fluid components of the plasma, without affecting density. As the Hasegawa-Wakatani equations describe the evolution of the plasma density and vorticity, but not the momentum, it was decided that charge exchange would not be included in the solver for these equations. However, a charge exchange kernel will be needed for future use within NESO, so work continued on this topic. The first requirement for such a kernel was to find an estimate of the rate at which charge exchange is likely to occur on a per-particle basis (R_{CE}), given a neutral's kinetic energy. A `csv` file was created that contains this rate as a function of the H kinetic energy by digitising figure 37 of ref. [7]. A `csv` reader class was built to enable this. Once this reference data has been read in it can then be used in conjunction with a 1-D linear interpolator to estimate the R_{CE} value for a neutral based on the average kinetic energy of a particle within a macro-particle, which represents w physical particles. Δw which represents the number of particles which undergo charge exchange in time-step Δt can be calculated using the following formula

$$\Delta w = R_{CE}\, w\, n_{ions}\, \Delta t, \tag{28}$$

For a neutral particle with velocity v, the total change in velocity Δv as a function of Δw is given by

$$\Delta v = \frac{\Delta w}{m_{neutral} w}(m_{neutral} v - m_{ion} v_{fluid}), \tag{29}$$

and the total change in the fluid momentum

$$\Delta p_{fluid} = \Delta w(m_{ion} v_{fluid} - m_{neutral} v), \tag{30}$$

where $m_{neutral}$ is the mass of a single neutral H atom, m_{ion} is the mass of a single H^+ ion, and v_{fluid} is the velocity of the fluid. Having completed the 1-D linear interpolator and `csv` reader in prior work, we are currently in the process of creating a charge exchange kernel. Once the charge exchange kernel had been created, several tests will be put in place. At a basic level, conservation of momentum will be tested for by accounting for the total amount of momentum initialised and injected into the system in all species. Next, the rates of change of momentum will be calculated from the code and tested against analytical solutions.

2.3 NEKTAR++

In this subsection we present work on implementing plasma turbulence equations in a 3-D domain using Nektar++. In Section 2.3.1 we describe the implementation of the Hasegawa-Wakatani equations and present encouraging results that suggest the Nektar framework is capable of capturing 3-D plasma turbulence. Section 2.3.2 outlines modifications required to integrate the Nektar solver with a system of neutrals, modelled using NESO-Particles, and demonstrates the behaviour of the coupled software. Lastly, Section 2.3.3 reports on progress towards implementing a more complex system of equations and outlines a number of items to be addressed in upcoming work.

2.3.1 A '2-D-in-3-D' Hasegawa-Wakatani solver

A solver for the 2-D Hasegawa-Wakatani equations has previously been presented as a NEPTUNE proxyapp[8] called Nektar-driftwave. Rather than extend the existing code to work in a 3-D domain, we reimplement the equation system in an inheritance hierarchy, with the intent of coupling a particle system to the base class later. This approach allows all systems in the hierarchy to use the same particle interface, avoiding code duplication and making it trivial to switch to a more complex set of plasma turbulence equations at a later date.

Note that the present plasma equations represent a proof-of-concept rather than a system of particular physical interest. The equations to be solved are unchanged from those implemented in Nektar-driftwave [8], but are replicated below for convenience:

$$\frac{\partial n}{\partial t} + [\Phi, n] = \alpha(\Phi - n) - \kappa\frac{\partial \Phi}{\partial y}, \tag{31}$$

$$\frac{\partial \omega}{\partial t} + [\Phi, \omega] = \alpha(\Phi - n), \tag{32}$$

where n is number density, ω is vorticity and Φ is the electrostatic potential.

$[a, b]$ is the Poisson bracket operator, defined as

$$[a, b] = \frac{\partial a}{\partial x}\frac{\partial b}{\partial y} - \frac{\partial a}{\partial y}\frac{\partial b}{\partial x}. \tag{33}$$

Φ is obtained at each time-step by solving the Poisson equation

$$\nabla_\perp^2 \Phi = \omega \tag{34}$$

where the $_\perp$ subscript denotes the restriction of the Laplacian to the two-dimensional subspace transverse to the magnetic field lines (which are assumed to be parallel to the z-axis).

In these equations, n is assumed to be a perturbation on a fixed background (n_0), the profile of which is exponential in the x coordinate with a scale length κ, that is

$$n_0 = \exp(-\kappa x), \tag{35}$$

and α is an 'adiabiacity operator', which describes parallel electron behaviour and is a constant in the 2-D version of these equations.

In order to implement the $\nabla_\perp$ operator that appears in Equation (34), we use a feature of the Nektar++ API that has previously been applied to modelling anisotropic diffusion. For Poisson problems, we can supply elements of a matrix $\mathcal{C}$, such that Nektar solves

$$\nabla \cdot (\mathcal{C} \nabla \Phi) = \text{ rhs} \tag{36}$$

where the matrix coefficients are labelled according to

$$\mathcal{C} = \begin{bmatrix} d_{00} & d_{01} & d_{02} \\ d_{01} & d_{11} & d_{12} \\ d_{02} & d_{12} & d_{22} \end{bmatrix}. \tag{37}$$

We set $d_{00} = d_{11} = 1$ and all other coefficients other to zero, such that the operator becomes $\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$. Note that this approach means the solution is not unique - an arbitrary z-dependence can be added to Φ without changing the right hand side. Hence, it must be understood how the solution changes from one iteration to the next, since the HW equations rely on the value of Φ directly.

The two advected fields (n and ω) are discretised using a Discontinuous Galerkin (DG) formulation, while Φ is represented as a Continuous Galerkin (CG) field, which is recomputed from ω at each time step. Fluxes between DG elements are calculated using a Riemann solver (implementation due to Toro [9]). We use basis functions constructed from a modified form of 7^{th} order Legendre polynomials, described by Karniadakis and Sherwin [10]. The fluid fields are evolved using an explicit, 4^{th} order, Runge-Kutta time stepping scheme.

The domain was chosen to be a cuboid with dimensions (arbitrary length units) 5x5x10. A mesh with 8x8x16 hexahedral elements was created in GMSH and converted to Nektar++ format using NekMesh.

Component	Label	Description	Value
Nektar			
	TimeStep	Fluid time-step	1.25e-3
	TFinal	Simulation duration	40.0
HW			
	α	HW equation coefficient	0.1
	κ	HW equation coefficient	3.5
	B_{xy}	Magnetic field strength	1.0

Table 1: Parameter values used in the 3-D HW solver. Values listed in the final column are dimensionless unless otherwise specified.

The simulation is configured via a standard Nektar++ XML *session file*. Some of the most important parameters are listed in Table 1.

n and ω are initialised to a Gaussian profile in x and y, modulated by a sinusoidal function in the z direction. Boundary conditions are chosen to be periodic in all three dimensions.

Note that this configuration closely resembles the simulation setup described in section 2.2.1 of report M6c.4 [11] but uses an initial density perturbation with six times greater amplitude, a higher value of κ, a shorter time-step and a longer total duration (these values were chosen empirically to obtain non-trivial behaviour in the plasma ie. to give the appearance of a turbulent state).

Figure 6 shows the evolution of the plasma density at four output times between $t = 0$ (the initial conditions) and $t = 40$ (the last time-step of the simulation).

Several possibilities exist for improving the performance of our HW solver. Typically, solvers for this problem add a hyper-diffusion term to suppress numerical instabilities. As far as the authors are aware, hyper-diffusion has not been employed in Nektar applications before, so no guidance is immediately available as to how the corresponding flux terms (required by DG solvers) should be implemented. Further work will be needed to determine the form of those functions. While it is possible to use regular diffusion instead, for explicit methods this places a further constraint on the time-step which is inversely proportional to the diffusion coefficient, though we note that other Nektar applications have bypassed this problem by implementing diffusion implicitly. Finally, further investigation is needed to assess, and possibly replace, our current implementation of the $\nabla_\perp$ operator (see discussion following Equation (36)), in collaboration with Nektar++ developers.

2.3.2 Coupled 3-D solver

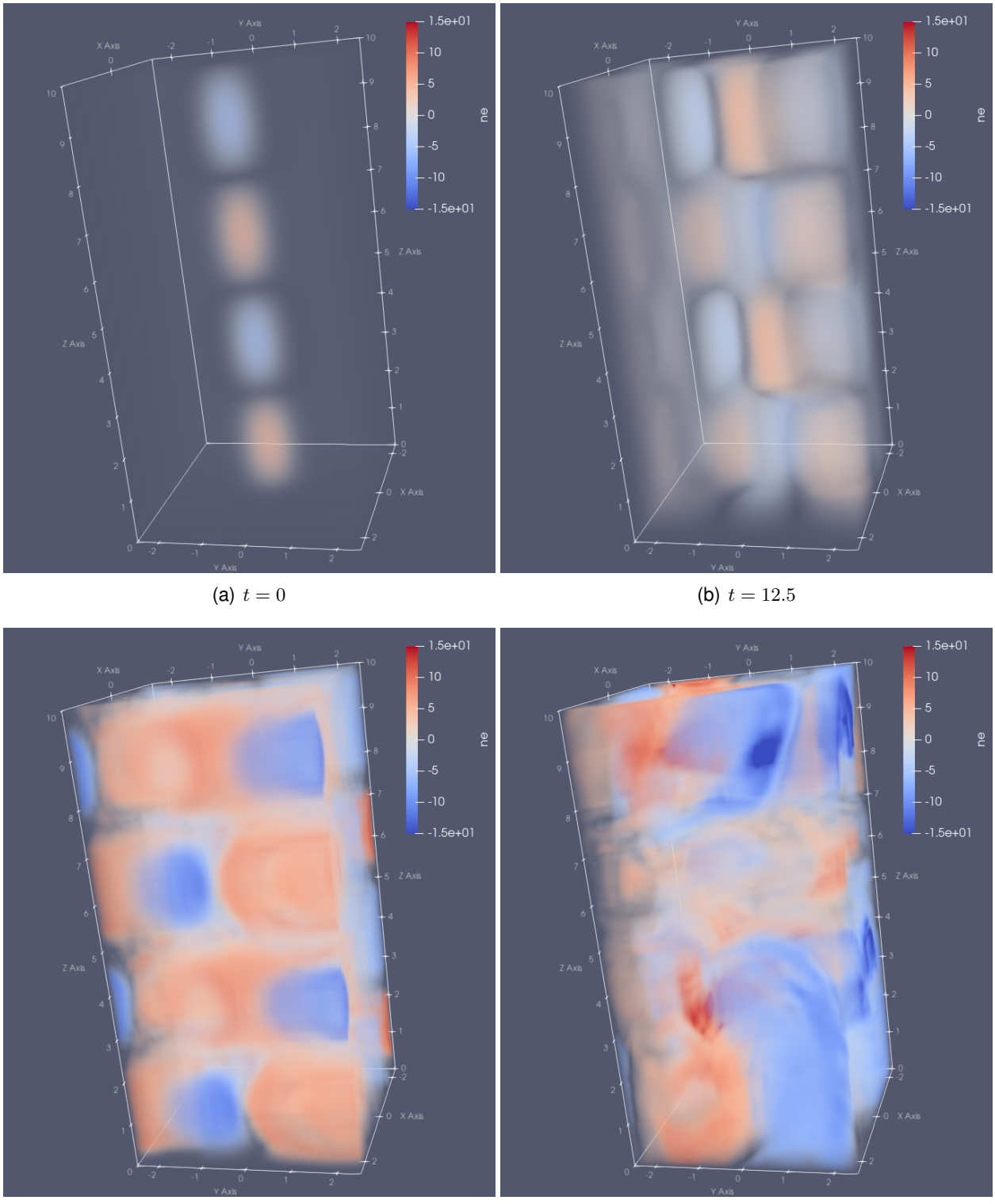
The fluid solver described in Section 2.3.1 was augmented in a number of ways in order to couple it to a system of neutrals, which are modelled using the NESO-Particles framework.

1. A new field was added to the Nektar++ configuration file to store the particle density sources. Note that this field is excluded (along with the potential, Φ) from the advection operations that affect n and ω.
2. Objects were added to the equation system class to handle *evaluation* of the Nektar++ electron density field at the particle positions and *projection* of the particle weights onto the new source field.
3. An additional member function was implemented to add the source field to the right-hand side of the density time evolution equation.

The domain dimensions used for the coupled simulation were the same as those used in the fluid-only case. The number of elements for the coupled simulation was 8x8x16 and the order of the Nektar++ basis functions was six.

The initial conditions were set to be zero throughout the domain for both n and ω, such that all non-trivial evolution of the fluid fields is triggered by particle sources. The boundary conditions remain periodic in all three dimensions for the fluid solver and the same conditions are adopted for the neutral particle system. All other parameters used by the fluid solver were set to the values listed in Table 1.

The parameters used to configure the neutral particle system are listed in Table 2. The code generates *num_particles_total* computational particles and assigns initial weights (number of neutral



(a) $t = 0$ (b) $t = 12.5$

(c) $t = 25$ (d) $t = 40$

Figure 6: Evolution of the electron density in the fluid-only 2-D-in-3-D Hasegawa-Wakatani solver.

Label	Description	Value
drift_velocity	Bulk velocity given to new particles	2.0
number_density	Overall number density of neutrals in m^{-3}	10^{16}
num_particles_total	Number of computational particles	10^6
source_width	Width of the 3-D Gaussian used to draw random initial particle positions	0.2
thermal_velocity	Width of the 3-D Gaussian used to draw random initial particle velocities	1.0
n_{bg}	Assumed background density in m^{-3}	10^{18}
T_e	Electron temperature in eV	10

Table 2: Parameter values used to configure the particle system in the coupled 3-D solver. Values listed in the final column are dimensionless, unless otherwise specified.

particles) such that the overall number density in the domain matches the requested value. Initial particle positions are chosen by drawing values at random from a 3-D Gaussian distribution centred at the origin. Initial velocities are assigned by adding random thermal velocities, drawn from a second Gaussian, to a bulk drift velocity.

Recall that the electron density in Equation (31) is actually the magnitude of a perturbation on a fixed background. To compute an ionisation rate, therefore, the perturbed value evaluated at each particle is multiplied by an SI conversion factor and added to a constant, n_{bg}, in order to arrive at a number density in m^{-3}. In combination with the assumed electron temperature, T_e, this value is used to calculate the number of particles ionised per unit time, which is projected back onto the source field and then added to the right-hand side of Equation (31) by the fluid solver.

To demonstrate the behaviour of the coupled fluid-particle solver, we present two sets of images, illustrating first the effects of ionisation at the beginning of the simulation and then the subsequent evolution of the fluid fields.

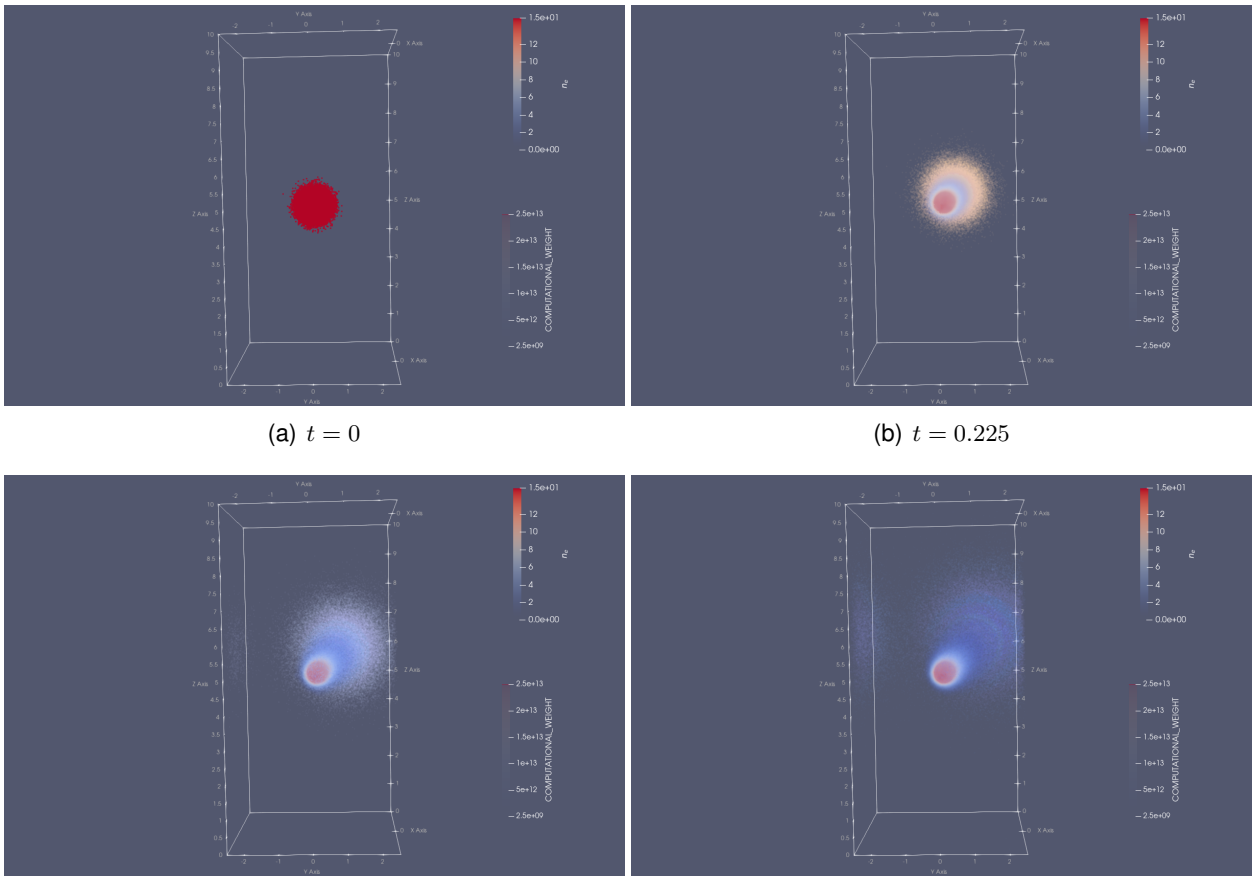
Figure 7 shows the initial evolution of the fluid, with particle positions overlaid. Time increases from left to right and top to bottom. Over this interval most of the neutrals ionise, as indicated by the changing colours of the points in the four panels, leaving a “hot spot” of plasma which then evolves according to the 2-D HW equations.

Figure 8 shows the evolution of the electron density and vorticity in the remainder of the simulation. Note the larger range of output times relative to Figure 7, reflecting a relatively slow evolution compared to the particle dynamics.

2.3.3 The LAPD problem

In this subsection we report on progress using Nektar++ to solve a more complex system of equations that captures additional physics used to study plasma turbulence.

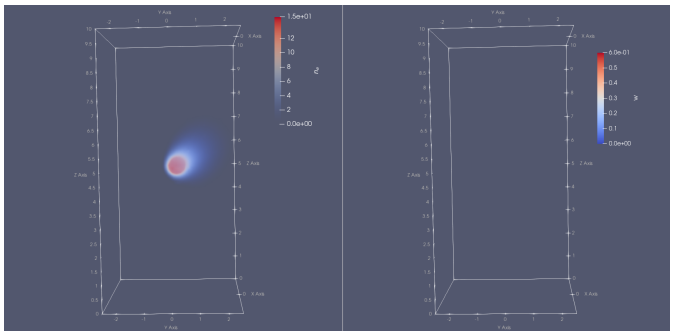
Our aim is to simulate the LArge Plasma Device (LAPD; [2, 12]) - a linear, pulsed-discharge device designed for plasma physics research. This problem has already been tackled using the HERMES-3 finite difference solver [13] (based on the BOUT++ [14] framework), offering



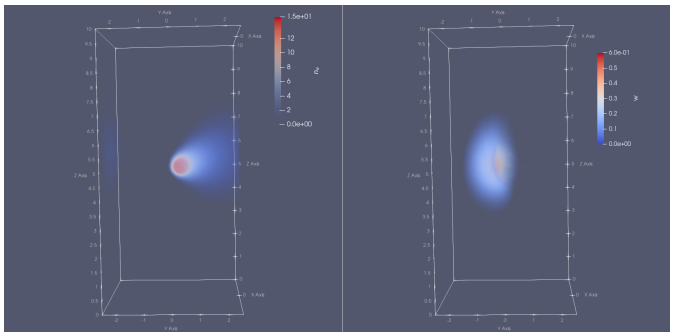
(a) $t = 0$ (b) $t = 0.225$

(c) $t = 0.45$ (d) $t = 0.675$

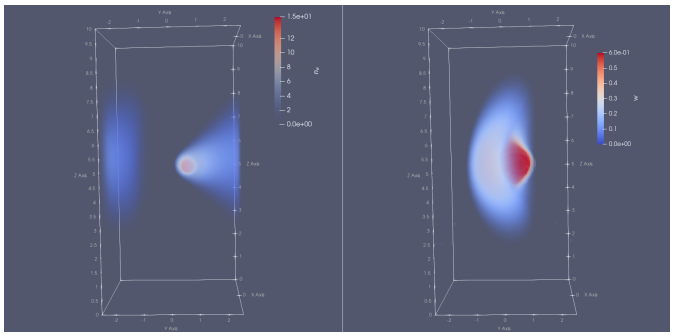
Figure 7: Evolution of the particle distribution (points) and electron density (coloured contours) in the coupled 2-D-in-3-D Hasegawa-Wakatani / particles solver. Particles are coloured according to their computational weight, corresponding to the number of physical neutrals that they represent. Captions under each panel indicate the simulation time.



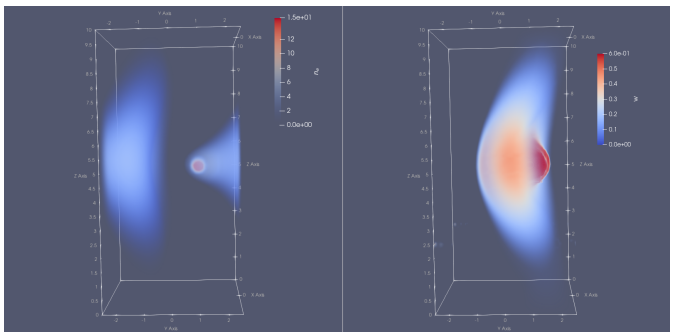
(a) $t = 1.5$



(b) $t = 3$



(c) $t = 4.5$



(d) $t = 6$

Figure 8: Evolution of the electron density (left) and vorticity (right) in the coupled 2-D-in-3-D Hasegawa-Wakatani / particles solver. Captions under each panel indicate the simulation time.

a straightforward path to validating our results.

The LAPD is reasonably well approximated as a 17 m cylinder, with a circular cross section of diameter $\sim$ 1m and a plasma source at each end. Typically, electron number densities reach 10^{18} m^{-3} and electron temperatures are ≤ 10 eV. For an image of a mesh designed for the LAPD problem, see Fig. 4 of [11].

The LAPD equation system has been implemented as a parent class of the system described in Section 2.3.1, meaning that all of the same apparatus for coupling neutral particles can be reused without modifications to the code. Switching between the two systems simply involves setting a different equation system label as a parameter in the XML configuration file.

The system of equations to be modelled is

$$\frac{\partial n_e}{\partial t} = -\nabla \cdot \left[n_e(\mathbf{v}_{E\times B} + \mathbf{b}v_{\|e})\right] \tag{38}$$

$$\frac{\partial (m_e n_e v_{\|e})}{\partial t} = -\nabla \cdot \left[m_e n_e v_{\|e}(\mathbf{v}_{E\times B} + \mathbf{b}v_{\|e})\right] - \partial_{\|} p_e - e n_e E_{\|} + m_e n_e \nu_{ei}(v_{\|d+} - v_{\|e}) \tag{39}$$

$$\frac{\partial (m_{d+} n_{d+} v_{\|d+})}{\partial t} = -\nabla \cdot \left[m_{d+} n_{d+} v_{\|d+}(\mathbf{v}_{E\times B} + \mathbf{b}v_{\|d+})\right] - \partial_{||} p_{d+} + e n_{d+} E_{\|} - m_e n_e \nu_{ei}(v_{\|d+} - v_{\|e}) \tag{40}$$

$$\frac{\partial \omega}{\partial t} = -\nabla \cdot (\omega \mathbf{v}_{E\times B}) + \nabla \cdot (n_{d+} v_{\|d+} - n_e v_{\|e}) \tag{41}$$

where Equation (38) describes the evolution of the electron density, Equations(39) and (40) the parallel momentum of electrons and (Deuterium) ions respectively and Equation (41) the vorticity. Quasi-neutrality is assumed, hence no additional equation is required for the ion number density. Electrons and ions are taken to be isothermal with temperatures T_e and T_{d+} respectively, meaning that the pressure of species x can be trivially obtained using $p_x = n_e T_x$. The electron-ion collision rate, ν_{ei}, is calculated from the Coulomb logarithm, Λ, according to

$$\nu_{ei} = \frac{|q_e||q_i| n_i \log\Lambda \, (1 + m_e/m_i)}{3\pi^{3/2}\epsilon_0^2 m_e^2 \left(v_e^2 + v_i^2\right)^{3/2}} \tag{42}$$

with $v_x^2 = 2kT_x/m_x$.

q_x, m_x, n_x and v_x are, respectively, the charge, mass, number density and velocity of species x and ϵ_0 is the permittivity of free space.

As with the simpler HW system described in Section 2.3.1, the drift velocity is calculated from the potential using $\mathbf{v}_{E\times B} = \frac{\mathbf{b}\times\nabla\Phi}{B^2}$ and the potential is obtained by solving the Poisson-like equation:

$$\nabla \cdot (\frac{\bar{m_i}\bar{n}}{B^2}\nabla_{\perp}\Phi) = \omega \tag{43}$$

where $\bar{m_i}\bar{n}$ is a characteristic density.

As a first approximation, we will assume a magnetic field aligned with the z-axis, but plan to relax that assumption when initial results have been validated. Further development of the solver may also include generalisation of the above equations to multiple ion species.

In order to compare our results to HERMES-3 directly, we choose a density source and initial conditions to match their implementation. The (steady) density source term is set to be Gaussian in the radial direction and constant in the axial direction, via a Nektar++ session file function. This function is added to the right-hand side of Equation (38) at each time step.

The initial conditions for n_e are set to

$$0.1\ e^{-x^2} + 10^{-5}\ (\mathrm{mixmode}(z) + \mathrm{mixmode}(4 * z - x)) \tag{44}$$

where 'mixmode' is a function defined as

$$\mathrm{mixmode}(x) = \sum_{i=1}^{14} \frac{1}{(1 + |i - 4|)^2} \cos[ix + \Phi(i, \mathrm{seed})] \tag{45}$$

where i is the mode number and Φ is a random phase between $-\pi$ and $+\pi$. As explained in the "Variable initialisation" section of the BOUT++ documentation [14]: "The factor in front of each term is chosen so that the 4th harmonic ($i = 4$) has the highest amplitude. This is useful mainly for initialising turbulence simulations, where a mixture of mode numbers is desired."

Substantial progress towards implementing the LAPD equation system in NESO has already been made, but there remain a number of future work items before results can be compared to HERMES-3 and to experimental data. Firstly, boundary condition implementations need to be developed that are appropriate to the physical conditions expected at the edges of the LAPD domain. At each end of the domain along the magnetic axis, for instance, a *sheath* is expected, characterised by outflow at, or close to, the sound speed. A successful implementation of sheath boundary conditions should enforce appropriate outflow rates whilst avoiding the creation of steep velocity/momentum gradients that might degrade numerical stability. Typically the potential is constrained with a Dirichlet condition $\Phi = 0$, but that approach can cause the formation of artificial boundary layers. HERMES-3 prevents this by employing a relaxation technique [15] which it may make sense to emulate in our Nektar++ solver. Secondly we note that, in the current version of the solver, Equation (43) is solved in an identical fashion to Equation (34). While the LAPD equations do not depend on the value of Φ directly, as the HW equations do, the parallel electric field $E_{\parallel} = \partial\Phi/\partial z$ does feature in Equations(39) and (40). Hence, further work is required to examine and improve our implementation of the $\nabla_{\perp}$ operator, as highlighted in the Section 2.3.1. Finally, Nektar++ does not support non-Cartesian coordinate systems, meaning that cylindrical polar coordinates, which are a more natural choice for the LAPD problem, cannot be used. While this does not add any significant difficulty to the implementation, it will add an extra layer of processing when drawing comparisons to HERMES-3 results.

2.4 Synthesis

NESO [16] is an open source C++ framework for solving equations involving a coupling of continuum fields to computational Lagrangian markers, or particles. The specific use-case in mind for project NEPTUNE is the exhaust region of a tokamak power plant where the dynamics may be captured, in some regimes, by the interactions between a continuum fluid plasma and point particle neutral species. These two requirements are met by the use of Nektar++, for solving fluid-like PDEs, and NESO-Particles [3] for representing particles. NESO brings together both

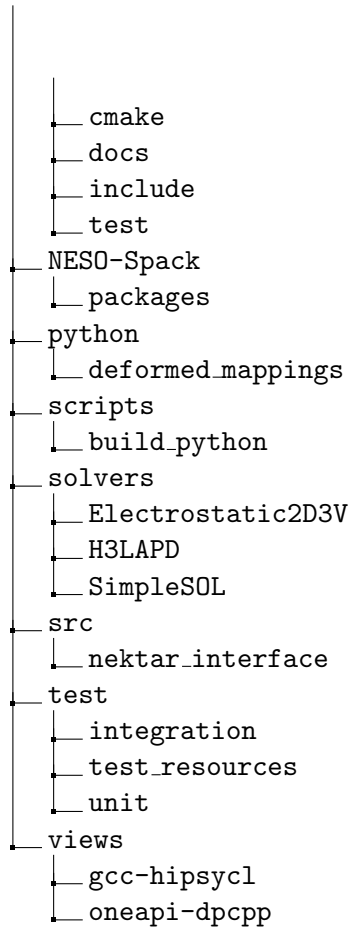
of these in a single framework and provides the functionality required for the two to interoperate: NESO-Particles particles deposit sources to Nektar++ fields and the Nektar++ fields are evaluated to provide information at each particle's location.

The following sections describe some of the design aspects of NESO behind the work described in this document.

2.4.1 Framework layout

This section explains the layout of NESO during development. The directory tree showing only 2 layers is as follows:

```
builds
├── gcc-hipsycl
└── oneapi-dpcpp
cmake
docker
docs
├── doxygen
└── sphinx
examples
├── Electrostatic2D3V
├── H3LAPD
├── MaxwellWave2D3V
├── poisson
└── SimpleSOL
include
├── nektar_interface
├── particle_utility
└── solvers
nektar
├── builds
├── cmake
├── docker
├── docs
├── library
├── pkg
├── solvers
├── templates
├── tests
├── ThirdParty
└── utilities
neso-particles
├── build
└── builds
```



The `builds` directory contains two sets of binaries and library files: one set is built with oneAPI and the other with hipSYCL. By always building against two implementations of SYCL we have a higher degree of certainty that our code will build against any implementation, at least with a minimum of changes. The `cmake` directory needs little introduction; the `docker` directory contains docker files. `docs` is where the `.rst` and `.md` documentation for the code is kept. Code that is shared between different solvers within NESO has been written in header files within the `include` directory. The `nektar`, `neso-particles` and `neso-spack` directories contain the corresponding git-submodules for these tools.

The layout of NESO is driven to some extent by the desire to build separate executables for different equation systems, aka solvers, such that users may incrementally explore the numerical and physical parameter spaces of their problems. This has led to the separation of different solver code into directories within `solvers`. Penultimately, we have the `src` and `test` files for NESO code, and finally the `views` directory that spack creates.

2.4.2 Building solvers

Recursive cloning of the NESO repository avails the user of the `NESO-Spack` git submodule. Users may easily (at least significantly more easily than any alternative method) build NESO and all of its dependencies for both the OpenSYCL (previously known as hipSYCL) and oneAPI SYCL builds; that is to say, NESO-Particles can be compiled with both implementations of the SYCL standard.

2.4.3 Hasegawa-Wakatani solver and example

The code for the HERMES-3 LAPD system of equations is located in the `solvers/H3LAPD` directory relative to the top directory of NESO.

```
solvers/H3LAPD
├── CMakeLists.txt
├── Diagnostics
│   └── mass_conservation.hpp
├── EquationSystems
│   ├── H3LAPDSystem.cpp
│   ├── H3LAPDSystem.hpp
│   ├── HWSystem.cpp
│   └── HWSystem.hpp
├── H3LAPD.cpp
├── H3LAPD.hpp
├── main.cpp
└── ParticleSystems
    └── neutral_particles.hpp
```

The hierarchy of classes is designed to re-use code where available and to taper down functionality to create two sets of integration tests, see Fig. 9. `HWSystem` inherits from `H3LAPDSystem`, which itself inherits directly from the Nektar++ class `AdvectionSystem`. In this way, terms may be tested in the boiled down Hasegawa-Wakatani system before deployment in the more complicated HERMES-3 LAPD system.

2.4.4 Diagnostics and data processing

Primarily of value are diagnostic dumps of fields and particles, where post-processing may be performed in eg. Python or ParaView. NESO-Particles uses the HDF5 particles standard, `H5part`, to output particle data, which may then be read in by any number of tools. Nektar++ outputs `.chk` files, which can be converted to the widespread `.vtu` format with the use of the Nektar++ tool `FieldConvert`.

The team most commonly use ParaView for post-processing of results. There will be a time when detailed calculations will be required of the simulation results. At that time it will be necessary to spawn a new repository dedicated to the task, based on versioned data output schemes.

2.4.5 Adding SYCL to Nektar++

Work has begun, via collaboration between the Cambridge Open Zettascale lab and UKAEA, to SYCL-ise bottlenecks found in Nektar++. The Intel Advisor software, which emulates ports of CPU code to SYCL and characterises its performance, has been deployed on the SOL2D with particles NESO solver and has highlighted a number of areas for speed up via GPUs. Although this work

has just begun, there is hope that performance gains may be found for universal components of Nektar++, such as the Riemann solver of Toro.

2.5 Opportunities to increase productivity

Working with a cutting edge software stack puts developers up against the limit of implementations and as such can cause delays in progress. This report covers coupling of particles to the Hasegawa-Wakatani system and progress towards coupling to the full HERMES-3 LAPD system. In the process, we have encountered some opportunities for increasing our productivity when working with Nektar++.

It is well known to the community that the Hasegawa-Wakatani equations rely on a hyper-diffusion term for numerical stability. The derivation of the physics does not include this term, so it is important to include it sparingly with only just enough damping to keep the simulation stable. Unfortunately, fluxes for this term, required by the DG method, were not formulated during the work for this report and it remains a matter for future work. It was posited that the strongly discontinuous values of the 4^{th} derivative of high-order basis functions would decrease stability unless the fluxes were fully accounted for, hence it was decided, in the interest of time, to rely on the dissipative qualities of upwinded flux. Productivity on the project would increase given time to fully understand the way fluxes are imposed and create a quality-of-life wrapper or interface, if necessary.

A productivity improvement would be the addition of a DG-enabled implicit diffusion term. We experimented with the explicit diffusion term that was taken from the nektar-diffusion proxyapp, itself a product of a prior ExCALIBUR -NEPTUNE grant from UKAEA. Unfortunately, it was found that large diffusion coefficients caused the code to crash, likely due to a breach of the CFL condition of the explicit algorithm.

It is necessary to revisit the implementation of the perpendicular Laplacian having found failure cases whilst performing the work for this deliverable. Future work involves collaborating with the Nektar++ developers to augment the test suite and finesse the implementation to make improvements.

Automatic timestep modifiers exist in the form of simple Proportionalintegralderivative (PID) controllers, and the inclusion of one into Nektar++ would be beneficial. Another improvement would be to allow error-handling to re-run a calculation with a reduced timestep.

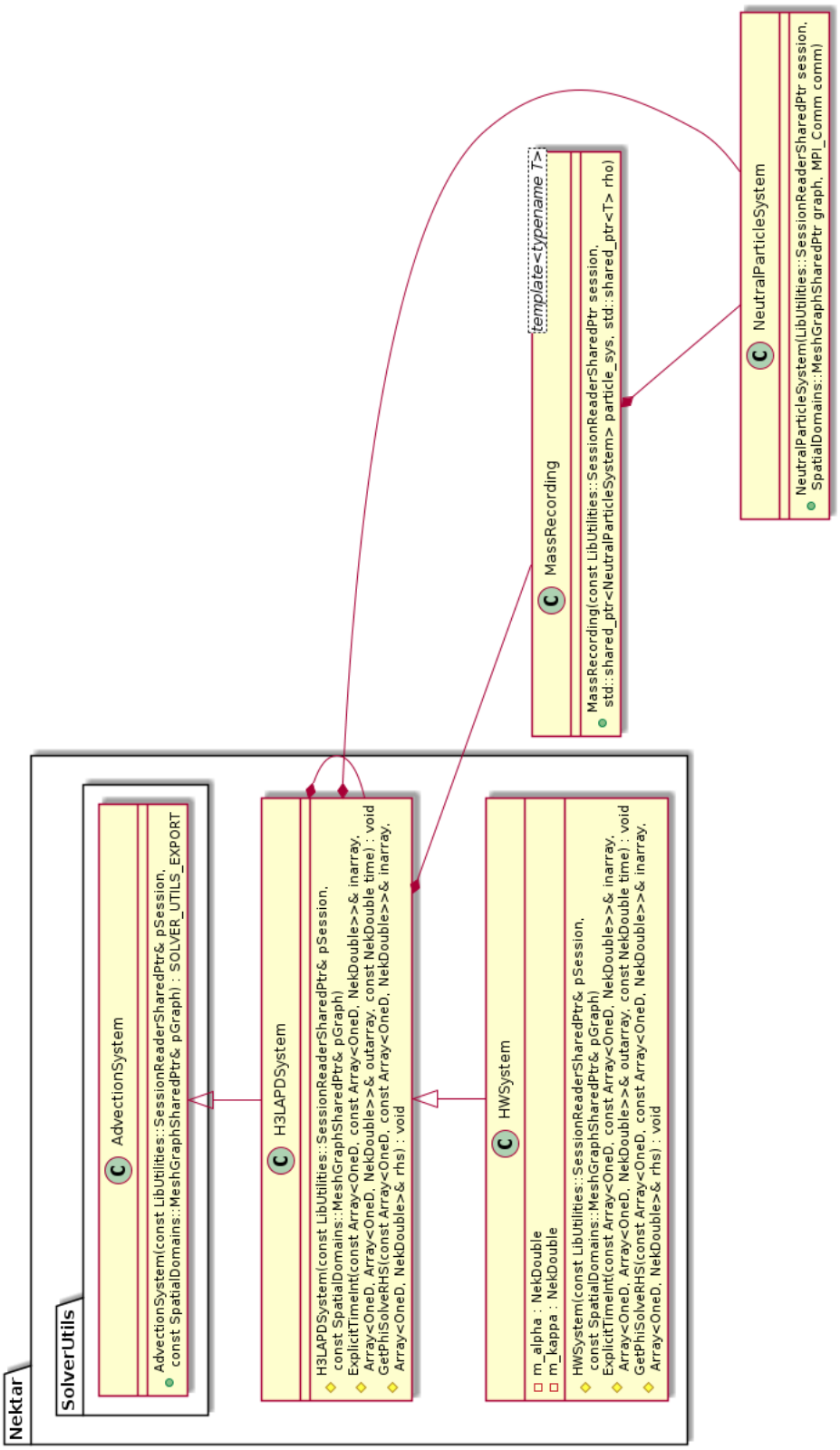


Figure 9: A PlantUML diagram showing the inheritance and composition of classes within the H3LAPD (HERMES-3-LAPD) solver.

3 Summary

In this report we describe the development of a system of fluid equations that exhibits turbulence in the presence of neutral particles, by full coupling of GPU-enabled particles and high order finite element fluid fields. A significant amount of development has gone into the particle framework to enable the communication of particles over a 3-D mesh. Further work has augmented the NESO framework to create classes for the HERMES-3 LAPD system of equations and the Hasegawa-Wakatani system. The latter was run with and without particle sources giving encouraging initial results showing turbulence for the former case and the production and transport of fluid plasma density in the latter. As such, this report describes the successful development of a fluid turbulence model coupled to kinetic particles, in three spatial dimensions and three velocity dimensions, in a finite element framework coupled to a GPU-enabled particles library.

Acknowledgement

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

References

[1] Hermes-3: multifluid drift-reduced model. `https://github.com/bendudson/hermes-3`. Accessed: September 2023.

[2] LArge Plasma Device (LAPD). `https://plasma.physics.ucla.edu/large-plasma-device.html`. Accessed: September 2023.

[3] W. R. Saunders. NESO-Particles. Zenodo. `https://doi.org/10.5281/zenodo.8386763`, 2022.

[4] W. Saunders, J. Cook, and W. Arter. High-dimensional Models Complementary Actions 2. Technical Report CD/EXCALIBUR-FMS/0062-M4.3, UKAEA, 3 2022. `https://github.com/ExCALIBUR-NEPTUNE/Documents/blob/main/reports/ukaea_reports/CD-EXCALIBUR-FMS0062-M4.3.pdf`.

[5] Aaron Meurer, Christopher P. Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B. Kirpichev, Matthew Rocklin, AMiT Kumar, Sergiu Ivanov, Jason K. Moore, Sartaj Singh, Thilina Rathnayake, Sean Vig, Brian E. Granger, Richard P. Muller, Francesco Bonazzi, Harsh Gupta, Shivam Vats, Fredrik Johansson, Fabian Pedregosa, Matthew J. Curry, Andy R. Terrel, Štěpán Roučka, Ashutosh Saboo, Isuru Fernando, Sumith Kulal, Robert Cimrman, and Anthony Scopatz. Sympy: symbolic computing in python. *PeerJ Computer Science*, 3:e103, January 2017.

[6] J. Cook, W. Saunders, and W. Arter. One-Dimensional and Two-Dimensional particle models. Technical Report CD/EXCALIBUR-FMS/0070-M4c.1, UKAEA, 1

2023. https://github.com/ExCALIBUR-NEPTUNE/Documents/blob/main/reports/ukaea_reports/CD-EXCALIBUR-FMS0070-M4c.1.pdf.

[7] R.L. Freeman, E.M. Jones, United Kingdom Atomic Energy Authority, and H.M.S.O. *Atomic collision processes in plasma physics experiments. Analytic expressions for selected cross-sections and Maxwellian rate coefficients*. Number v. 1 in CLM-R / Culham Laboratory. H.M. Stationery Office, 1974.

[8] Nektar++ solver for Hasegawa-Wakatani equations. https://github.com/ExCALIBUR-NEPTUNE/nektar-driftwave, 2021. Accessed: September 2021.

[9] Eleuterio F. Toro. *Riemann Solvers and Numerical Methods for Fluid Dynamics*. Springer Berlin Heidelberg, 2009.

[10] G. Karniadakis and S. Sherwin. *Spectral/hp element methods for computational fluid dynamics 2nd Ed.* Oxford University Press, 2005. https://doi.org/10.1093/acprof:oso/9780198528692.001.0001.

[11] E. Threlfall and O. Parry. Complementary actions. Code integration, acceptance and operation 4. Technical Report CD/EXCALIBUR-FMS/0078-M6.4, UKAEA, 09 2023. https://github.com/ExCALIBUR-NEPTUNE/Documents/blob/main/reports/ukaea_reports/CD-EXCALIBUR-FMS0078-M6c.4.pdf.

[12] W. Gekelman, P. Pribyl, Z. Lucky, M. Drandell, D. Leneman, J. Maggs, S. Vincena, B. Van Compernolle, S. K. P. Tripathi, G. Morales, T. A. Carter, Y. Wang, and T. DeHaas. The upgraded Large Plasma Device, a machine for studying frontier basic plasma physics. *Review of Scientific Instruments*, 87(2):025105, 02 2016.

[13] Hermes plasma edge simulation model: Hermes-3, a hot ion multifluid drift-reduced model. https://github.com/bendudson/hermes-3, 2021. Accessed: June 2021.

[14] B.D. Dudson. BOUT++ website. https://boutproject.github.io/, 2020. Accessed: June 2020.

[15] Ben Dudson, Mike Kryjak, Hasan Muhammed, Peter Hill, and John Omotani. Hermes-3: Multi-component plasma simulations with bout++, 2023.

[16] UKAEA. NESO. https://github.com/ExCALIBUR-NEPTUNE/NESO, 2022.