Exceptional service in the national interest

Sandia National Laboratories

MINRES/LS: Oct 5 - Oct 7, 2022

DPG for Vlasov: Two Formulations and Selected Results

Nathan V. Roberts, Stephen D. Bond, and Eric C. Cyr
nvrober@sandia.gov
Sandia National Laboratories
SAND 2022-13451 C

U.S. DEPARTMENT OF ENERGY

National Nuclear Security Administration

Outline

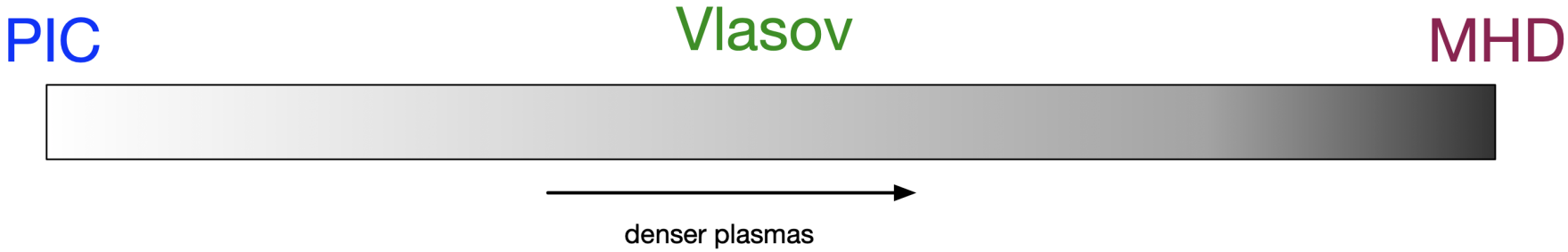
Plasma Physics

Sandia National Laboratories

Plasma physics simulations are vital for fundamental science and for clean energy (fusion power), among other things.
Applications include:

- Tokamak (and other fusion reactor) design
- Pulsed-power experimental facilities design ("Z-Next")

Plasma Physics Regimes



- Particle-in-Cell (PIC) approximations work well in rarefied regimes, where there is not too much material to simulate.
- Magnetohydrodynamics (MHD) works well, by contrast, where there is enough material that a fluid approximation of the plasma is a good model.
- In real-world applications, however, there is almost always a transition region between these regimes – for this, direct discretization of Vlasov is appropriate.

$$\partial_t f + \mathbf{v} \cdot \nabla f + \mathbf{a} \cdot \nabla_{\mathbf{v}} f = C(f)$$

Camellia 2.0 Development Effort

Sandia National Laboratories

Goals:

- Support for GPU and OpenMP execution (via Kokkos): in progress.

Camellia 2.0 Development Effort

Sandia National Laboratories

Goals:

- Support for GPU and OpenMP execution (via Kokkos): in progress.
- Support for new Intrepid2 bases, including hierarchical and serendipity bases: complete.

Camellia 2.0 Development Effort

Sandia National Laboratories

Goals:

- Support for GPU and OpenMP execution (via Kokkos): in progress.
- Support for new Intrepid2 bases, including hierarchical and serendipity bases: complete.
- Support for sum-factorized assembly (and more general "smart" assembly): in progress.

Camellia 2.0 Development Effort

Sandia National Laboratories

Goals:

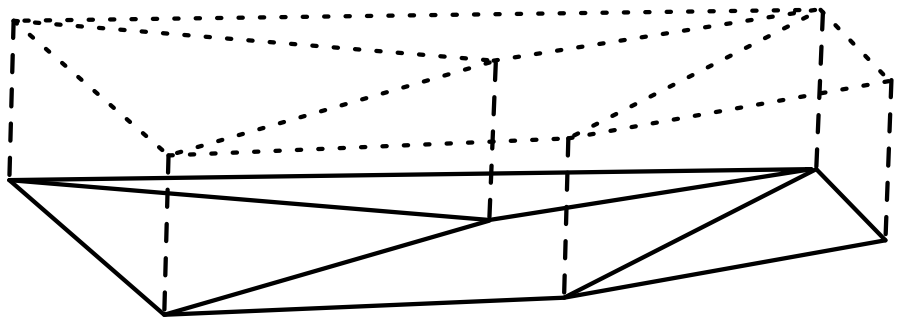
- Support for GPU and OpenMP execution (via Kokkos): in progress.
- Support for new Intrepid2 bases, including hierarchical and serendipity bases: complete.
- Support for sum-factorized assembly (and more general "smart" assembly): in progress.
- Support for matrix-free execution: aspirational.

Camellia 2.0 Development Effort

Sandia National Laboratories

Goals:

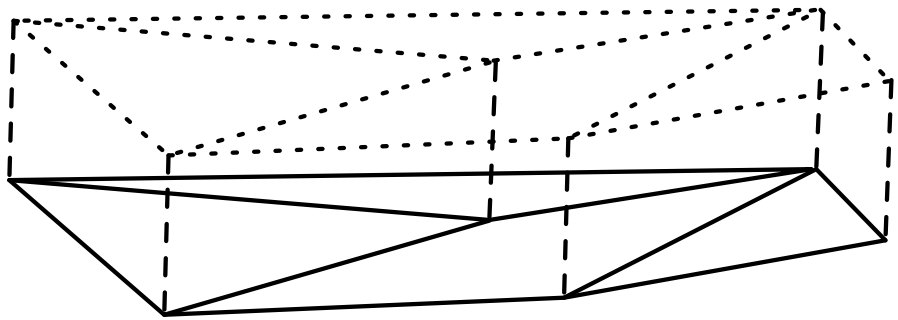
- Support for GPU and OpenMP execution (via Kokkos): in progress.
- Support for new Intrepid2 bases, including hierarchical and serendipity bases: complete.
- Support for sum-factorized assembly (and more general "smart" assembly): in progress.
- Support for matrix-free execution: aspirational.
- Support for orthogonal extrusions in up to 7D: complete for meshing; tested in up to 5D for assembly and solve.



Camellia 2.0 Development Effort

Goals:

- Support for GPU and OpenMP execution (via Kokkos): in progress.
- Support for new Intrepid2 bases, including hierarchical and serendipity bases: complete.
- Support for sum-factorized assembly (and more general "smart" assembly): in progress.
- Support for matrix-free execution: aspirational.
- Support for orthogonal extrusions in up to 7D: complete for meshing; tested in up to 5D for assembly and solve.



- Support for representing functions that only vary in specified dimension(s).

Camellia: Support for Structured Data

- Camellia aims to be quite general, with support for arbitrary PDEs on unstructured grids.
- Working to add mechanisms to preserve structure for improved performance.
- A work in progress: foundation laid for e.g. using Intrepid2's sum factorization, but not yet implemented.
- Two examples: `Function` and `ExtrudedMeshTopology` classes.

Function Class and Structured Data

The `Function` class represents an arbitrary function, which may be mesh-dependent; subclasses include:

- `ConstantScalarFunction` - a constant scalar value.
- `SimpleSolutionFunction` - mesh-based solution for a specified variable.
- `Sin_ax` - sine of ax, where a is a constant.

`values()` method: accepts an object representing the computational/geometric context (e.g., which cells and points to compute values for), and outputs a multi-dimensional array with shape (C,P) (for scalar-valued functions).

Two key additions for structure preservation:

Function Class and Structured Data

The `Function` class represents an arbitrary function, which may be mesh-dependent; subclasses include:

- `ConstantScalarFunction` - a constant scalar value.
- `SimpleSolutionFunction` - mesh-based solution for a specified variable.
- `Sin_ax` - sine of αx, where α is a constant.

`values()` method: accepts an object representing the computational/geometric context (e.g., which cells and points to compute values for), and outputs a multi-dimensional array with shape (C,P) (for scalar-valued functions).

Two key additions for structure preservation:

- a version of `values()` that outputs to an `Intrepid2::Data` object (alongside methods that allow the subclass to specify the structure of the data)

Function Class and Structured Data

The `Function` class represents an arbitrary function, which may be mesh-dependent; subclasses include:

- `ConstantScalarFunction` - a constant scalar value.
- `SimpleSolutionFunction` - mesh-based solution for a specified variable.
- `Sin_ax` - sine of ax, where a is a constant.

`values()` method: accepts an object representing the computational/geometric context (e.g., which cells and points to compute values for), and outputs a multi-dimensional array with shape (C,P) (for scalar-valued functions).

Two key additions for structure preservation:

- a version of `values()` that outputs to an `Intrepid2::Data` object (alongside methods that allow the subclass to specify the structure of the data)
- a bit-packed member variable `_variesInDimension` that allows subclasses to specify in which spatial dimensions the `Function` varies

ExtrudedMeshTopology

Sandia National Laboratories

Camellia's `MeshTopology` maintains the geometry of the mesh, including neighbor and parent-child relationships. (Contrast with `Mesh`, which additionally includes degrees of freedom for each cell.)

ExtrudedMeshTopology

Sandia National Laboratories

Camellia's `MeshTopology` maintains the geometry of the mesh, including neighbor and parent-child relationships. (Contrast with `Mesh`, which additionally includes degrees of freedom for each cell.)

`ExtrudedMeshTopology` is a subclass of `MeshTopology` that supports orthogonal extrusion of a lower-dimensional `MeshTopology` in arbitrary dimensions.

ExtrudedMeshTopology

Sandia National Laboratories

Camellia's `MeshTopology` maintains the geometry of the mesh, including neighbor and parent-child relationships. (Contrast with `Mesh`, which additionally includes degrees of freedom for each cell.)

`ExtrudedMeshTopology` is a subclass of `MeshTopology` that supports orthogonal extrusion of a lower-dimensional `MeshTopology` in arbitrary dimensions.

- constructor takes a root-level/unrefined `MeshTopology` and a set of coordinates in each extruded dimension.

ExtrudedMeshTopology

Sandia National Laboratories

Camellia's `MeshTopology` maintains the geometry of the mesh, including neighbor and parent-child relationships. (Contrast with `Mesh`, which additionally includes degrees of freedom for each cell.)

`ExtrudedMeshTopology` is a subclass of `MeshTopology` that supports orthogonal extrusion of a lower-dimensional `MeshTopology` in arbitrary dimensions.

- constructor takes a root-level/unrefined `MeshTopology` and a set of coordinates in each extruded dimension.
- maintains a 1D `MeshTopology` object for each extrusion dimension, with the rule that this is at least as fine as any corresponding phase-space cell in that dimension.

ExtrudedMeshTopology

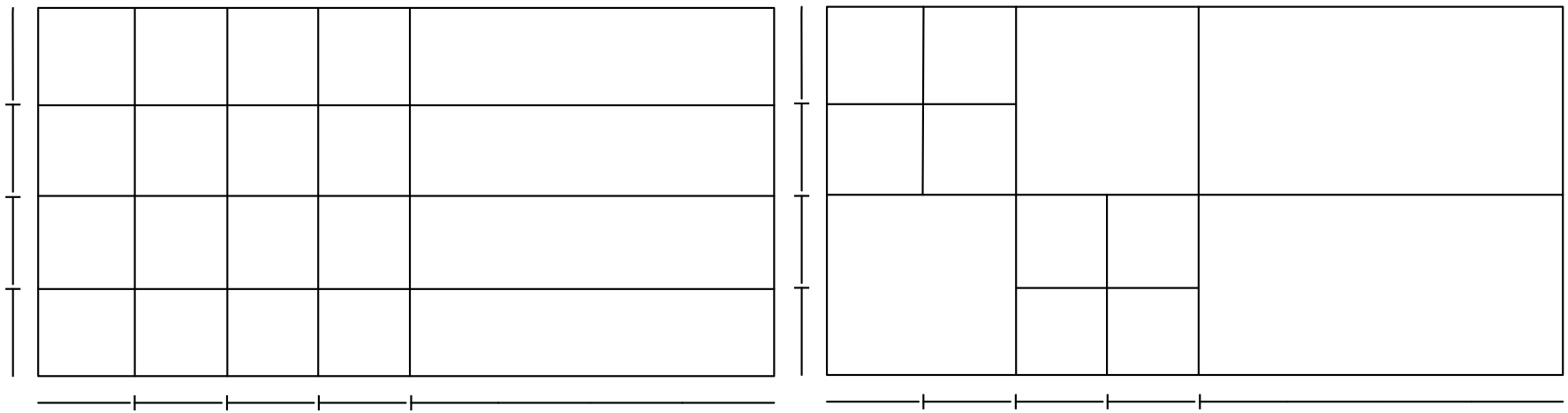
Sandia National Laboratories

Camellia's `MeshTopology` maintains the geometry of the mesh, including neighbor and parent-child relationships. (Contrast with `Mesh`, which additionally includes degrees of freedom for each cell.)

`ExtrudedMeshTopology` is a subclass of `MeshTopology` that supports orthogonal extrusion of a lower-dimensional `MeshTopology` in arbitrary dimensions.

- constructor takes a root-level/unrefined `MeshTopology` and a set of coordinates in each extruded dimension.
- maintains a 1D `MeshTopology` object for each extrusion dimension, with the rule that this is at least as fine as any corresponding phase-space cell in that dimension.
- overrides `addCell()` method (a bottleneck for refinements), and maintains maps from phase-space cells to cells in each extrusion dimension (and back).

Phase-Space Mesh Structure



Camellia supports maintains a base mesh (x axis) and set of orthogonal extrusion meshes (y axis). The phase-space mesh is a submesh of the full tensor-product mesh.

Intrepid2 Sum Factorization

Intrepid2 has support for sum-factorized assembly across the whole exact sequence, with good performance across Serial CPU, OpenMP, and CUDA platforms (CUDA shown).

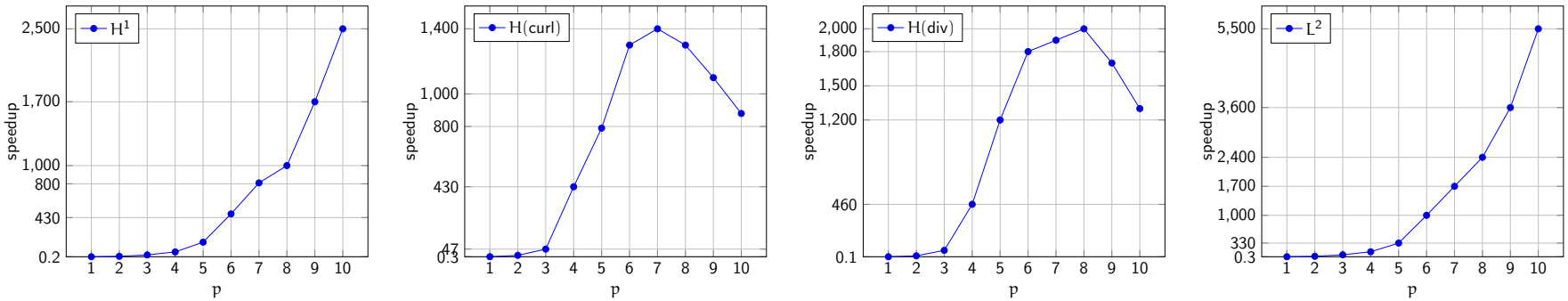


Figure: CUDA speedups compared to standard assembly for H^1, H(curl), H(div), and L^2 norms on hexahedra.

We plan to take advantage of Intrepid2's support for sum-factorized assembly in Camellia soon.

Vlasov-Poisson: 3D3V and 1D1V Equations

The 3D3V (3 space dimensions + 3 velocity dimensions) Vlasov-Poisson equations take the form:

$$\frac{\partial f}{\partial t} + \mathbf{v} \cdot \frac{\partial f}{\partial \mathbf{x}} + \frac{q}{m}\mathbf{E} \cdot \frac{\partial f}{\partial \mathbf{v}} = 0 \tag{1}$$

$$\nabla \cdot \mathbf{E} = \frac{q}{\epsilon_0} \int f d^3v \tag{2}$$

$$\mathbf{E} + \nabla \phi = 0 \tag{3}$$

Here, we have introduced a potential ϕ such that $\mathbf{E} = -\nabla\phi$ (convenient for BCs). We can simplify further by restricting to 1D1V:

$$\frac{\partial f}{\partial t} + v_x \frac{\partial f}{\partial x} + \frac{q}{m} E \cdot \frac{\partial f}{\partial v_x} = 0 \tag{4}$$

$$\frac{\partial E}{\partial x} = \frac{q}{\epsilon_0} \int f dv_x \tag{5}$$

$$E + \frac{\partial \phi}{\partial x} = 0 \tag{6}$$

Time-Marching Formulation: Vlasov

Backward Euler discretization of Vlasov:

$$\frac{1}{\delta t}\left(f^{k+1}, w\right) + \langle \hat{t}_n^{k+1}, w\rangle - \left(\begin{pmatrix} v_x f^{k+1} \\ \frac{q}{m} E_x^{k+1} f^{k+1} \end{pmatrix}, \nabla_{xv} w\right) = \frac{1}{\delta t}\left(f^k, w\right);$$

here formally $\hat{t}_n^{k+1} = \mathrm{tr}\left(\begin{pmatrix} v_x f^{k+1} \\ \frac{q}{m} E_x^{k+1} f^{k+1} \end{pmatrix} \cdot \mathbf{n}\right)$. We use the graph norm on the test space. $\hat{t}_n^{k+1} \in L^2$ is *non-conforming*; we enrich its polynomial order by 1 to match the order of conforming space.

Time-Marching Formulation: Poisson

Poisson Formulation:

$$\langle \hat{\phi}, \tau\, n_x \rangle - (\phi, \partial_x \tau) + (E_x, \tau) = 0$$

$$\langle \hat{E}_x, q\, n_x \rangle - (E_x, \partial_x q) = \left(\frac{\rho}{\epsilon_0}, q \right),$$

where ρ is computed from the plasma distribution f, and ϵ_0 is a constant. We use the graph norm on the test space.

Space-Time Formulation: Vlasov

We may write the 1D1V Vlasov equation as:

$$\nabla_{xtv} \cdot \begin{bmatrix} v_x f \\ f \\ \frac{q}{m} E_x f \end{bmatrix} = 0.$$

Multiplying by test $w \in H^1$ and integrating by parts:

$$\langle \hat{t}_n, w \rangle - \left(\begin{bmatrix} v_x f \\ f \\ \frac{q}{m} E_x f \end{bmatrix}, \nabla_{xtv} w \right) = 0,$$

where formally

$$\hat{t}_n = \mathrm{tr} \left(\begin{bmatrix} v_x f \\ f \\ \frac{q}{m} E_x f \end{bmatrix} \cdot \begin{bmatrix} n_x \\ n_t \\ n_v \end{bmatrix} \right).$$

We use the graph norm on the test space. Here again, $\hat{t}_n \in L^2$ is *non-conforming*; we enrich its polynomial order by 1 to match the order of conforming space.

Space-Time Formulation: Poisson

Our space-time Poisson Formulation:

$$\langle \hat{V}_E, \tau\, n_x \rangle - (V_E, \partial_x \tau) + (E_x, \tau) = 0$$

$$\langle \hat{E}_x, q\, n_x \rangle - (E_x, \partial_x q) = \left(\frac{\rho}{\epsilon_0}, q \right).$$

Note that the traces $\hat{V}_E, \hat{E}_x$ are only defined at the spatial interfaces (those for which $n_x \neq 0$). Note also that ρ is two-dimensional: it varies in time as well as space. The usual situation is that BCs are imposed on $\hat{V}_E$ at the left and right boundaries; for the cold diode, we impose $\hat{V}_E = 0$ at each.

We use the graph norm on the test space.

Solution Strategy: Fixed Point Iteration

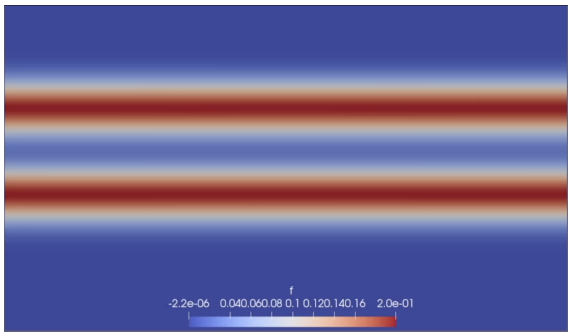
Sandia National Laboratories

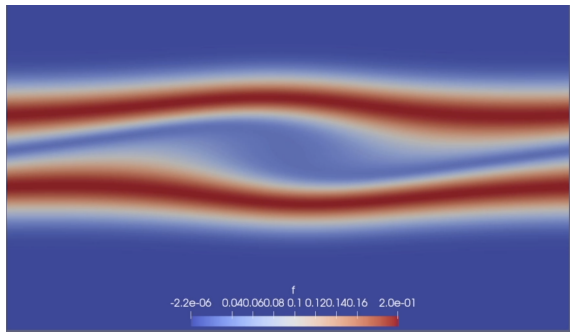
We use a fixed-point iteration with a set maximum number of iterations:

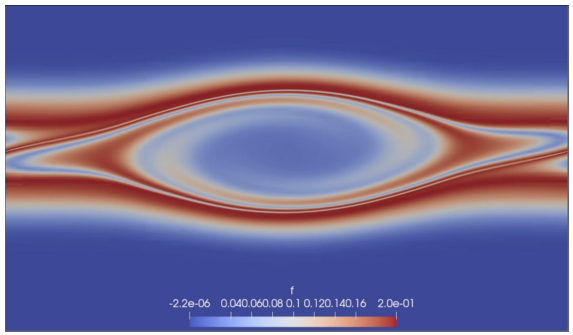
- up to 15 fixed-point iterations per solve, with early exit if the relative norm of the update falls below a tolerance (10^{-6}).
- Linear solves performed with Geometric-Multigrid-preconditioned conjugate gradient solver, tolerance between 10^{-7} and 10^{-9}.

The two-stream instability problem: classic verification problem in plasma physics:

- Two Maxwellian streams have velocities exactly opposite each other.
- A small, sinusoidal perturbation in the initial distributions generates an instability.
- There is a progression form linear, unstable regime, to nonlinear, stable regime.
- The growth rate of the electric field is known analytically.

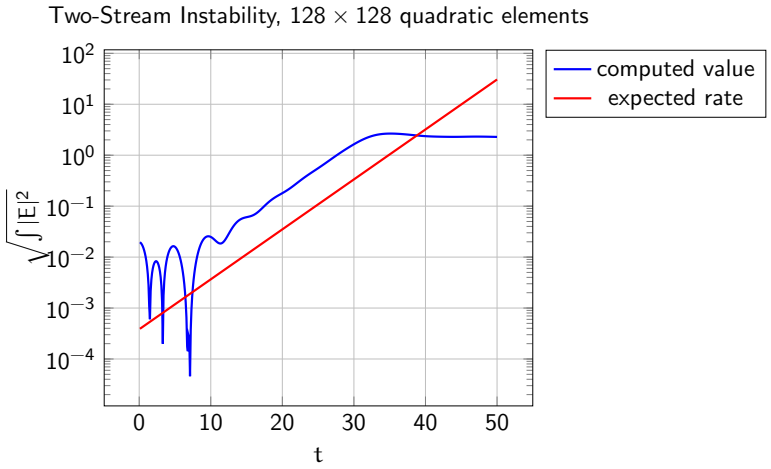




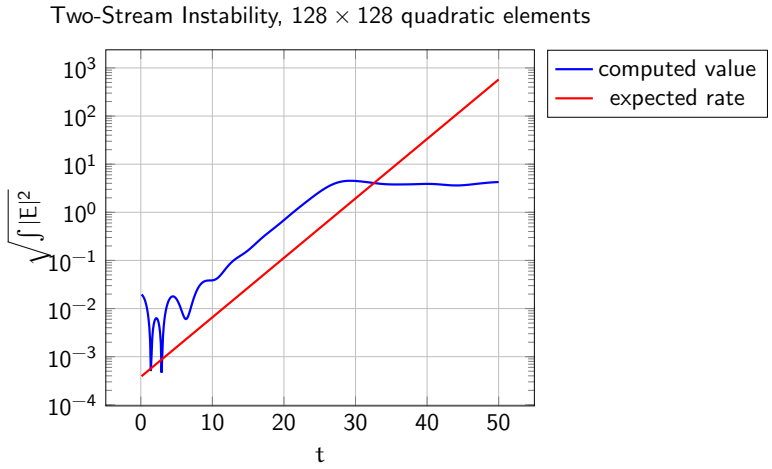


Two-Stream: Recovery of Expected Growth Rate

We run the two-stream problem with fixed $128 \times 128, p = 3$ mesh, and $\Delta t = 0.1$, for $u_0 = 2.4, u_0 = 3.0$.

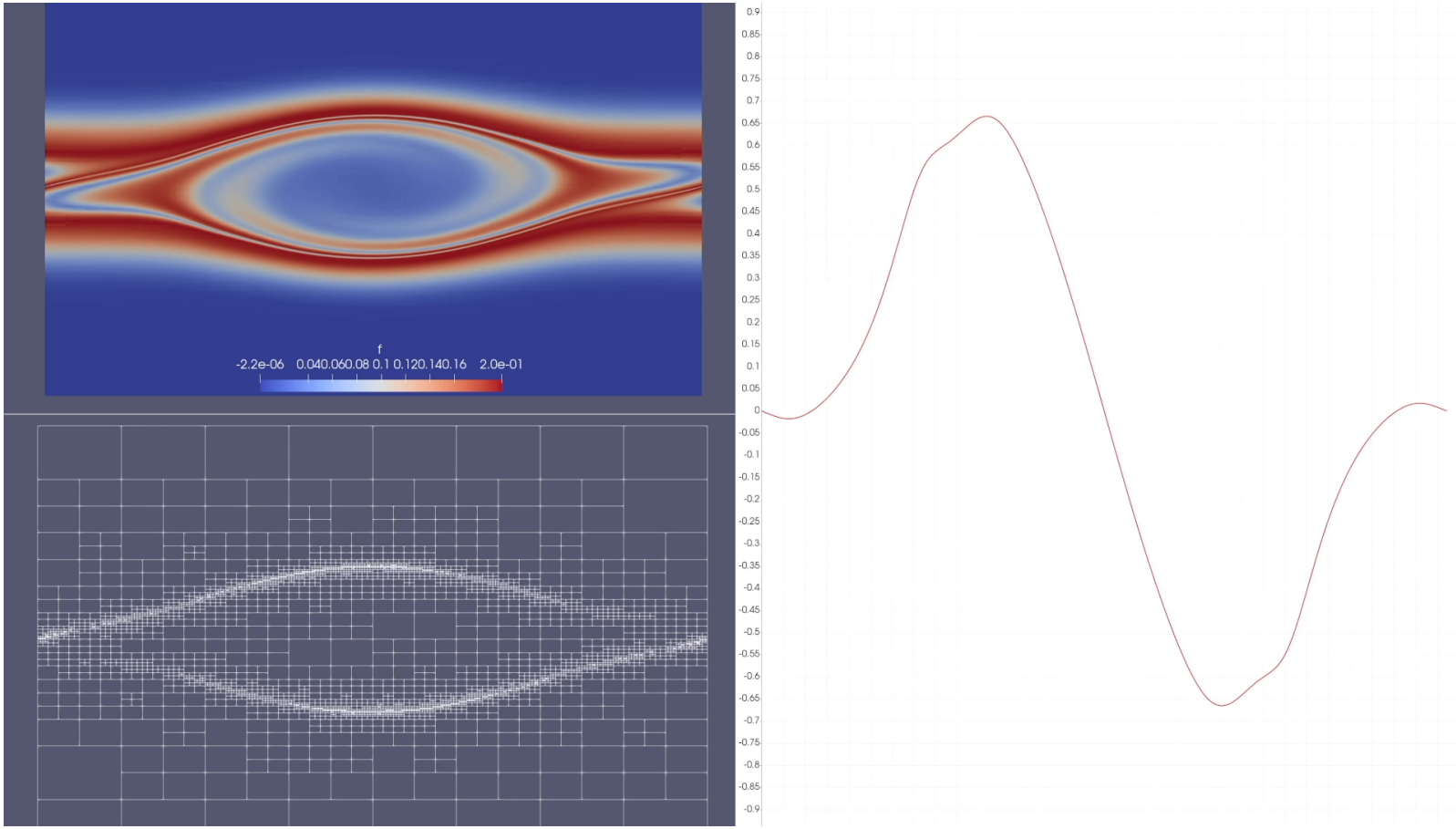


(a) Two-stream instability with $u_0 = 2.4$.



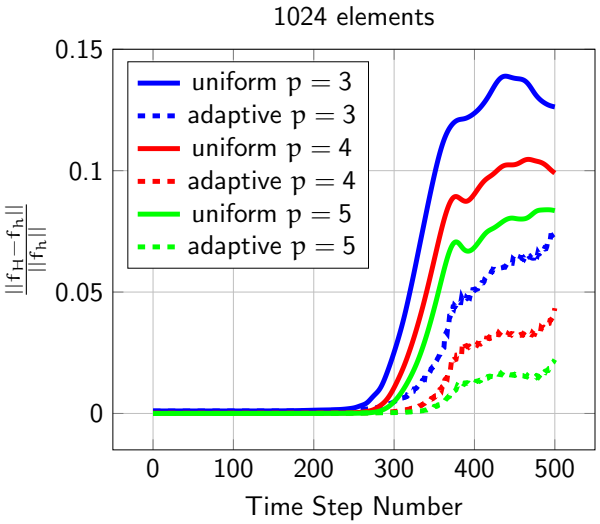
(b) Two-stream instability with $u_0 = 3.0$.

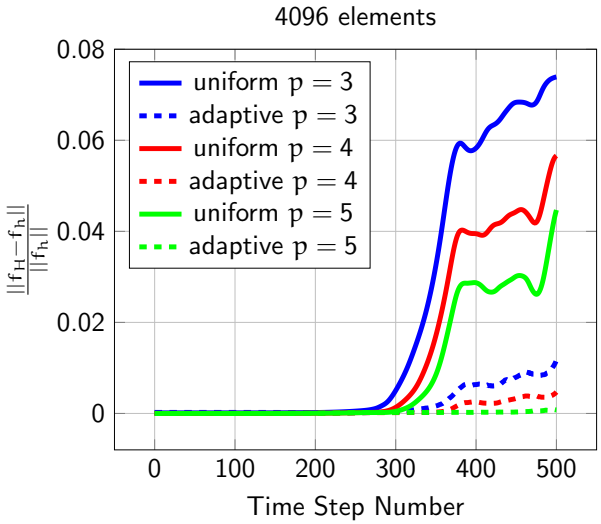
Two-Stream, Adaptive Solution



Final-time adaptive solution of the two-stream instability problem with 4,096-element budget and quadratic discretization of f. Top left: phase-space distribution f. Lower left: phase-space mesh. Right: configuration-space electric field E.

Two-Stream Adaptivity Study

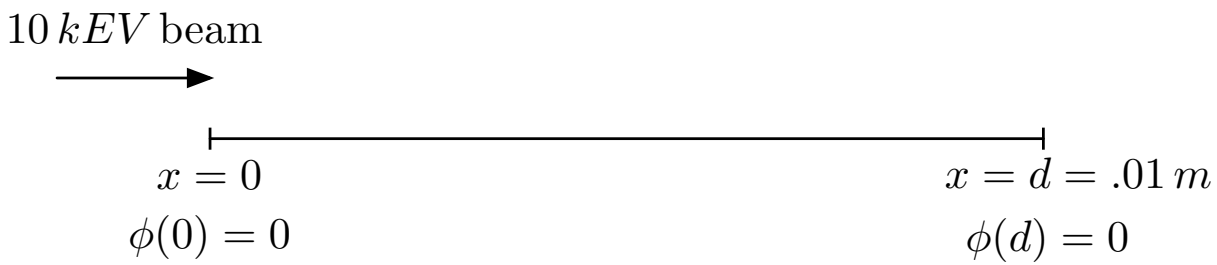




Final-time error plots for two-stream instability problem, $u_0 = 3.0$ case with uniform (32×32- and 64×64-element) and adaptive meshes with 1,024 and 4,096 elements with $u_0 = 3.0$, for several polynomial orders. Error is measured relative to an overkill mesh with $256 \times 256 = 65,536$, $p = 5$ elements. The overkill mesh has approximately 2.4 million degrees of freedom. All solves were run with a fixed timestep of $\Delta t = 0.1$.

The Cold Diode Problem

In the cold diode problem, a beam of electrons is emitted across a 1D anode-cathode gap, with an applied voltage across the gap.



- We have an exact solution due to Jaffé.
- EMPIRE-PIC has very accurate results for this problem.

EMPIRE Cold-Diode Results

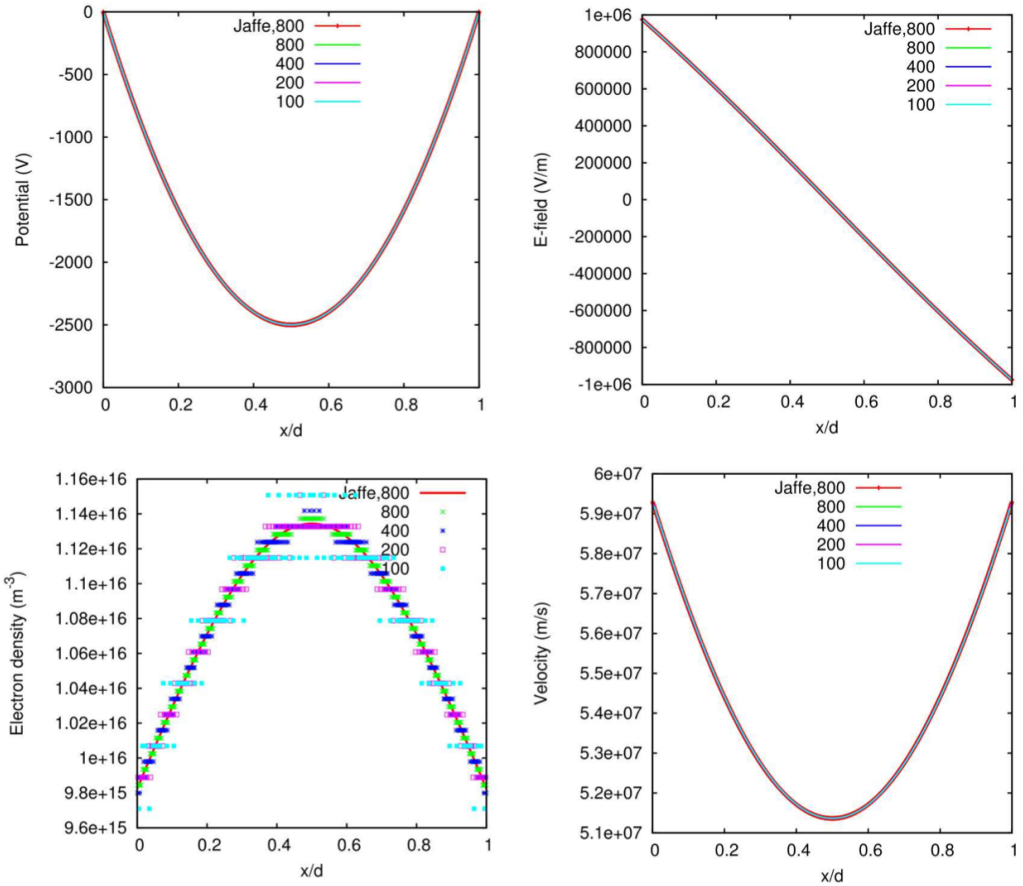


Figure 3-2. Comparison of computed solutions on four mesh levels with analytic solution (top left, potential; top right E-field; bottom left, electron number density; and bottom right, velocity).

The Cold Diode Problem and Vlasov

Some notes on our approach:

- We nondimensionalize for computations, such that $v^*_{\text{beam}} = 1$ and $t^*_{\text{final}} = 1$.
- We rescale on output for comparison to exact solution.
- Inflow BC: approximated with a Maxwellian with thermal velocity $\sigma = 0.025\, v_{\text{beam}}$.
- $\sigma > 0 \implies$ solving a slightly different problem; can expect some error due to that difference.
- Important to **resolve** the BC; we perform initial refinements to resolve to a given tolerance.
- For space-time adaptive study, we also introduce a linear temporal "ramp", phasing in the injection BC between $t = 0$ and $t = 0.25$.
- For final-time results in space-time, we average values between $t = 0.999$ and $t = 1.0$.

Time-Marching Results

For a first study, we use the following setup:

- meshes: multiples of 2×20 elements
- non-dimensional ν range $(0.5, 1.5)$
- $\sigma = 0.025$
- quadratic field variables
- test space enrichment $\Delta p = 4$.

Time-Marching Results: Uniform Refinement Study

Table: Quadratic f, Time-Marching, Relative L^2 errors

Mesh Size	Num Time Steps	E err.	ϕ err.	n_e err.	v_x err.
4x40	20	3.951E-04	3.715E-04	1.206E-03	5.041E-04
8x80	25	3.620E-04	3.638E-04	3.361E-04	3.133E-04
16x160	50	3.616E-04	3.634E-04	3.350E-04	3.126E-04
32x320	100	3.322E-04	3.333E-04	3.117E-04	3.069E-04

Solution Plots (Coarsest)

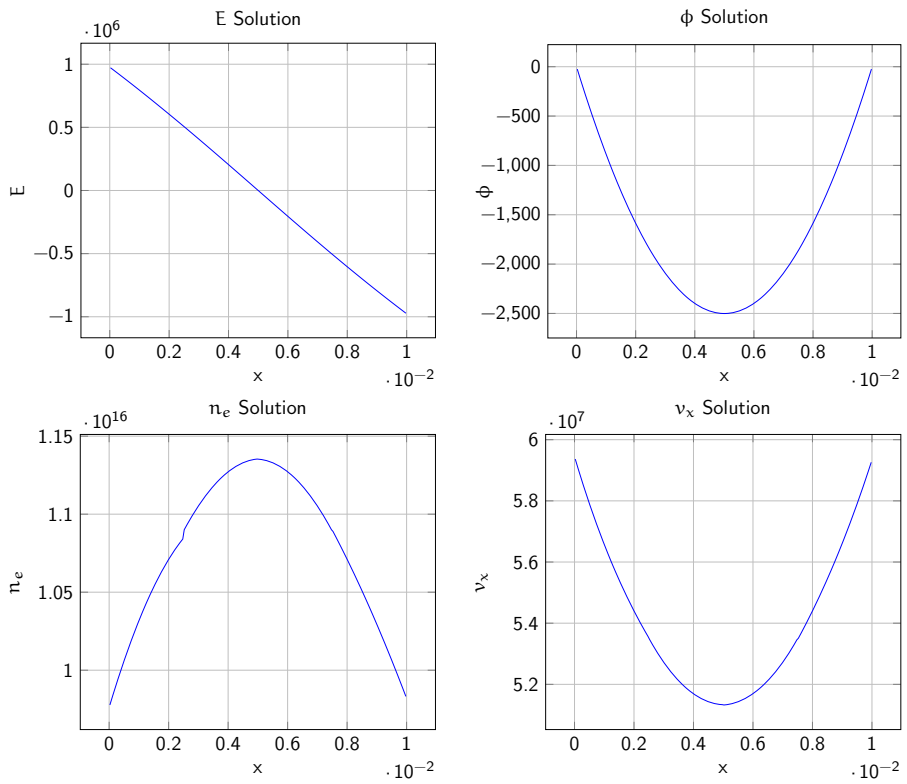


Figure: Final-time solution with 4×40 mesh, after 20 time steps. f is discretized with quadratic polynomials; the Vlasov mesh has 160 elements and 2,896 degrees of freedom; the Poisson mesh has 8 elements and 66 degrees of freedom.

Solution Plots (Finest)

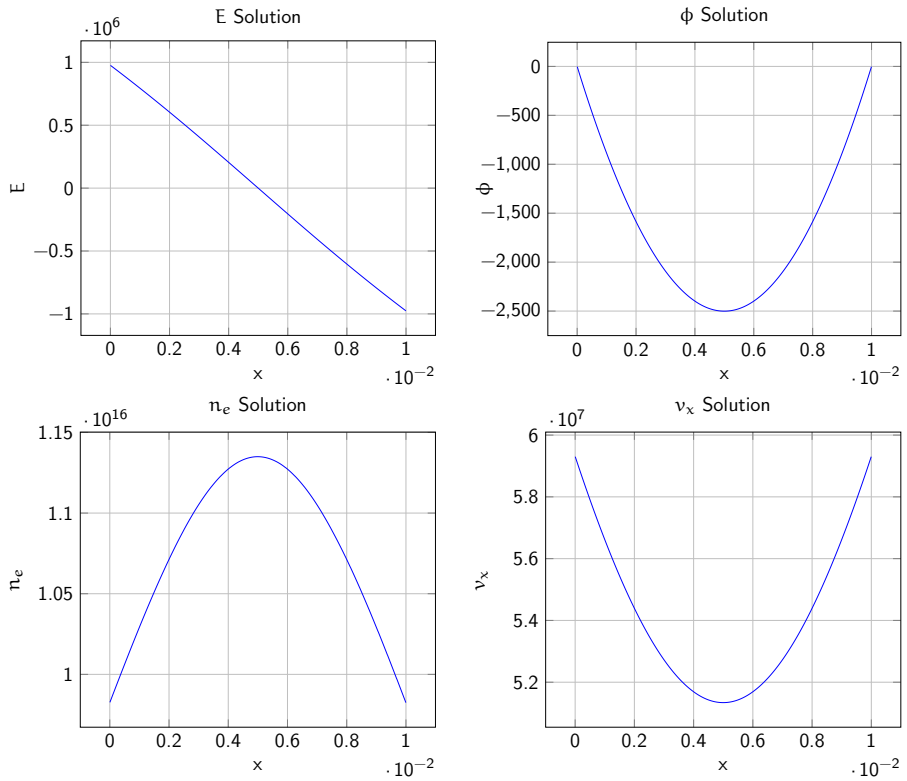
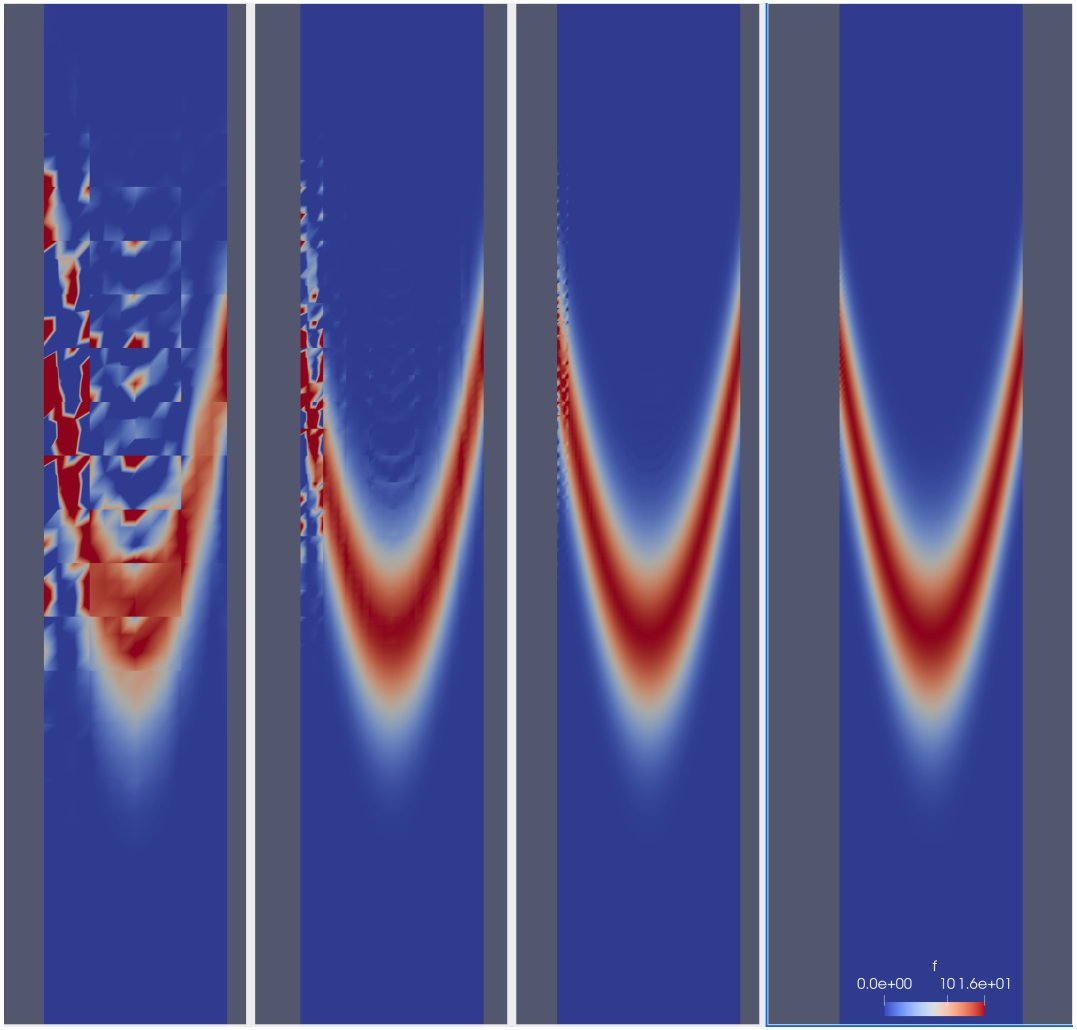


Figure: Final-time solution with 32×320 mesh, after 100 time steps. f is discretized with quadratic polynomials; the Vlasov mesh has 10240 elements and 175,488 degrees of freedom; the Poisson mesh has 64 elements and 514 degrees of freedom.

Vlasov Solution Plots



From left to right: coarsest to finest solution. Even the finest solution has *some* visible error at the inflow boundary (which may not be visible on the screen). (Note: scale clipped, dramatically for coarsest.)

σ Study

To assess the effect of σ on the error, we take 100 time steps on a fine (32×640, quadratic) mesh with $\sigma = 0.10, 0.5, 0.025$, successively.

Table: Quadratic f, Time-Marching, Relative L^2 errors

σ	E err.	ϕ err.	n_e err.	ν_x err.
0.10	6.478E-03	6.509E-03	6.007E-03	5.579E-03
0.05	1.476E-03	1.483E-03	1.367E-03	1.275E-03
0.025	3.322E-04	3.333E-04	3.117E-04	3.069E-04

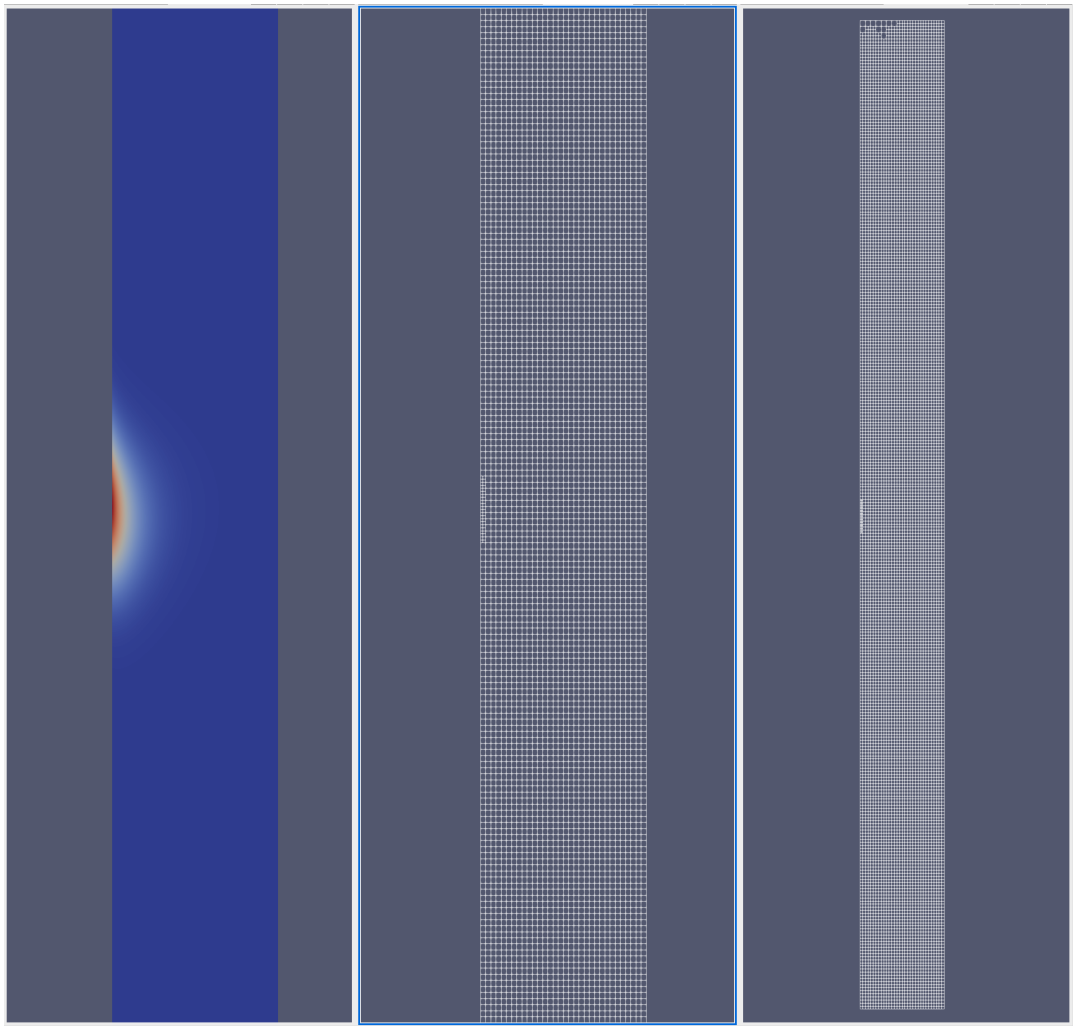
Adaptive Solve

Sandia National Laboratories

To give just one adaptive solve example:

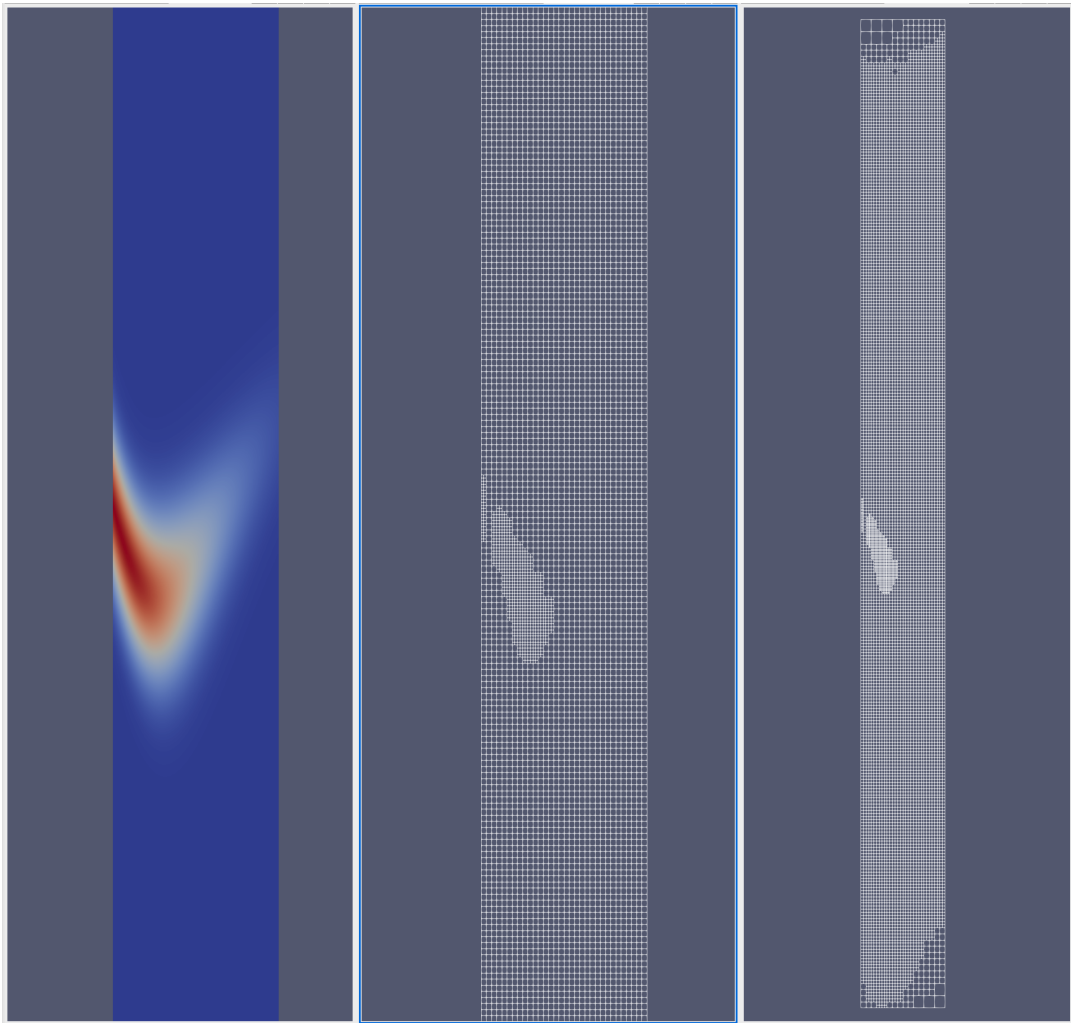
- start with a fine mesh identical to the finest fixed-size quadratic solution, 32×320 elements
- each time step, refine according to energy error, and unrefine an equal number of elements
- test space enrichment $\Delta \mathrm{p} = 5$.

Adaptive Solve



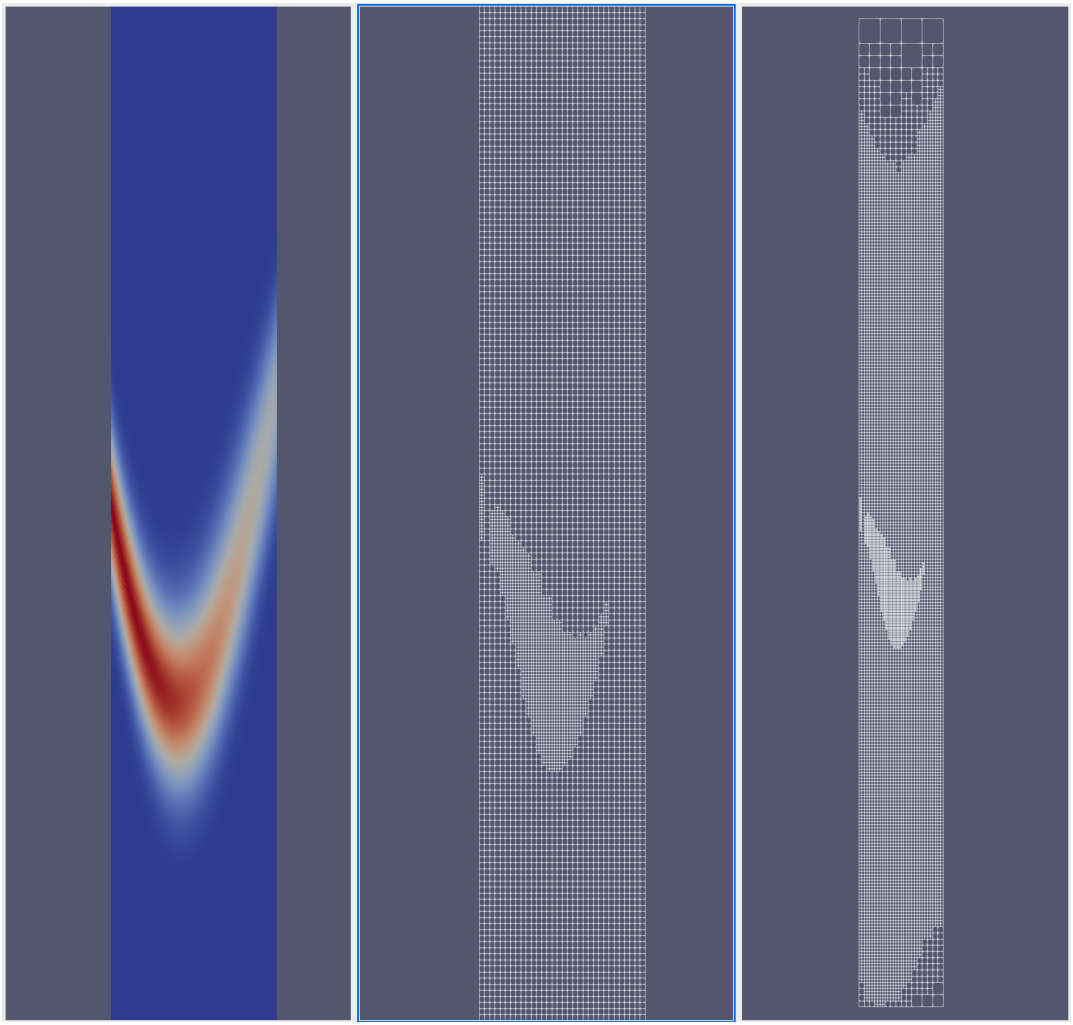
Time step 1.

Adaptive Solve



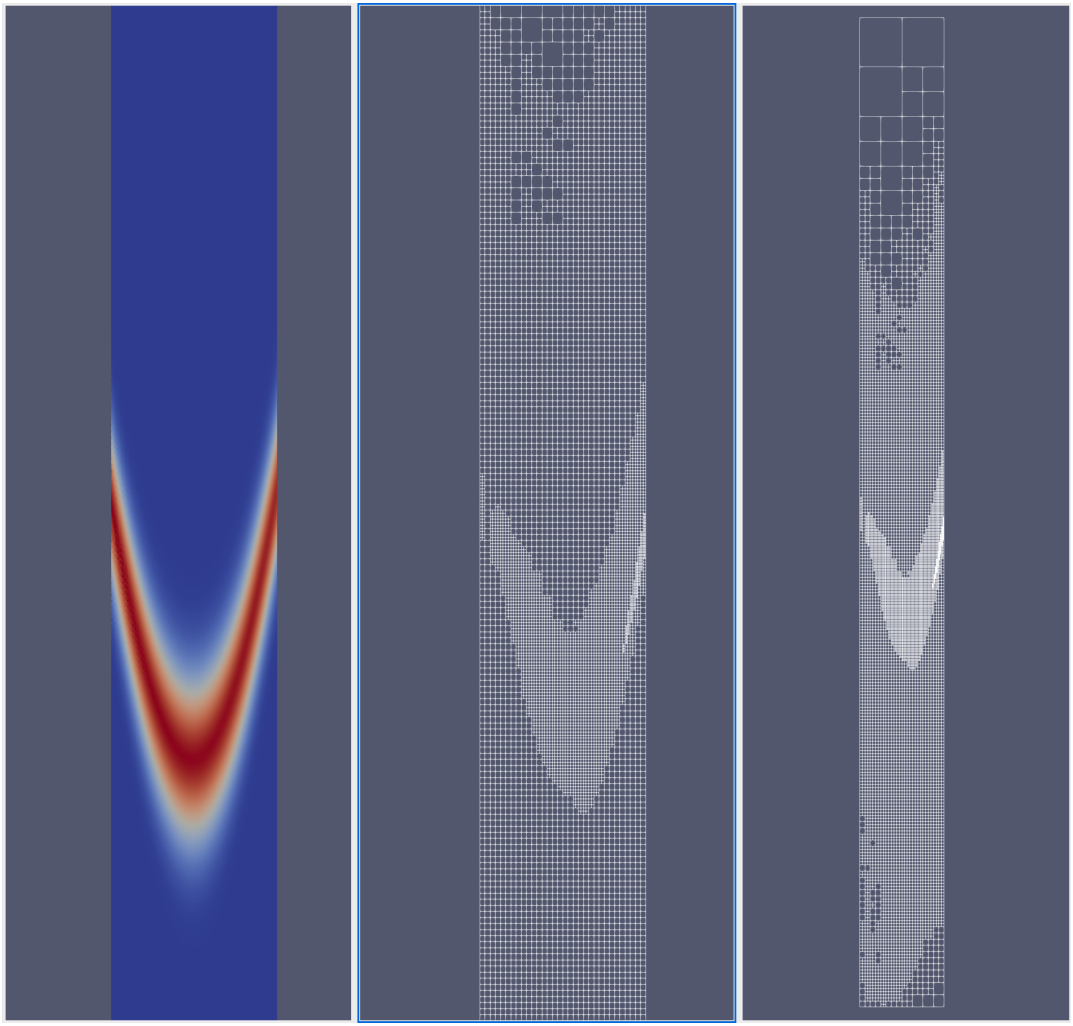
Time step 5.

Adaptive Solve



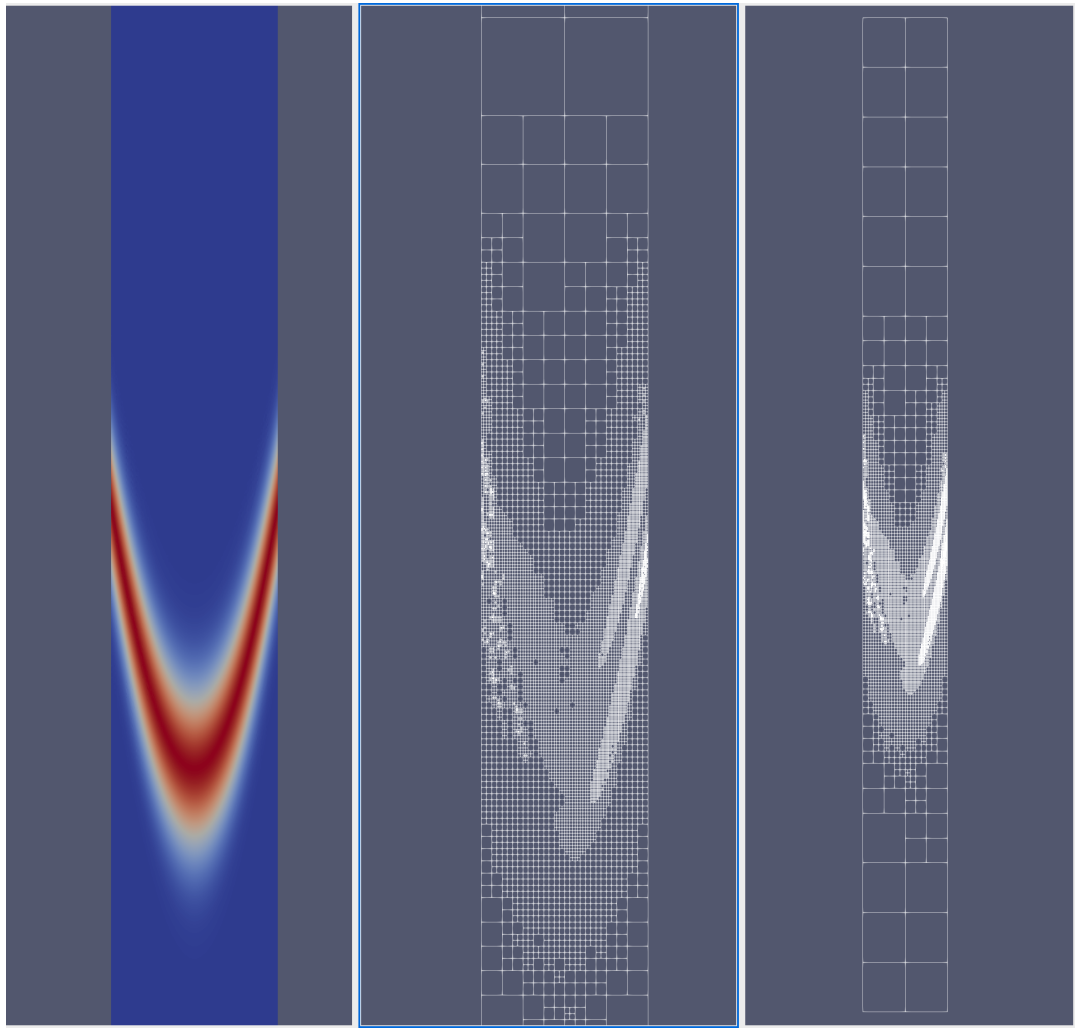
Time step 10.

Adaptive Solve



Time step 20.

Adaptive Solve



Time step 100. In contrast to the fixed-mesh solution, here there is **no** visible error accumulation at inflow (or elsewhere).

Space-Time Results: Uniform Refinement Studies

Sandia National Laboratories

Table: Relative L^2 errors

f order	Mesh Size	E err.	ϕ err.	n_e err.	ν_x err.
0	$4 \times 40 \times 40$	2.458E-01	2.228E-01	2.276E-02	2.386E-02
0	$8 \times 80 \times 80$	1.228E-01	1.133E-01	1.130E-02	1.198E-02
0	$16 \times 160 \times 160$	6.137E-02	5.690E-02	5.630E-03	5.998E-03
1	$4 \times 20 \times 40$	2.481E-03	2.505E-02	2.446E-03	2.200E-03
1	$8 \times 40 \times 80$	7.065E-04	6.266E-03	6.660E-04	6.212E-04
1	$16 \times 80 \times 160$	3.924E-04	1.605E-03	3.641E-04	3.399E-04
2	$4 \times 10 \times 40$	5.021E-04	4.206E-04	2.586E-03	6.109E-04
2	$8 \times 20 \times 80$	3.660E-04	3.673E-04	4.753E-04	3.365E-04
2	$16 \times 40 \times 160$	3.618E-04	3.635E-04	4.016E-04	3.138E-04
3	$4 \times 5 \times 40$	6.151E-03	2.189E-03	2.614E-02	3.178E-03
3	$8 \times 10 \times 80$	3.624E-04	3.632E-04	4.126E-04	3.133E-04
3	$16 \times 20 \times 160$	3.619E-04	3.637E-04	3.353E-04	3.126E-04

Uniform refinement study for space-time, for $p = 1$ to 4. As with our finest time-marching solves, we see error of roughly 3×10^{-4} in each variable, due to the nonzero value for σ. Note that the second dimension is time; we use coarser discretizations in time for higher polynomial orders so that we have roughly the same number of temporal nodes as in the time-marching scheme.

Space-time Results

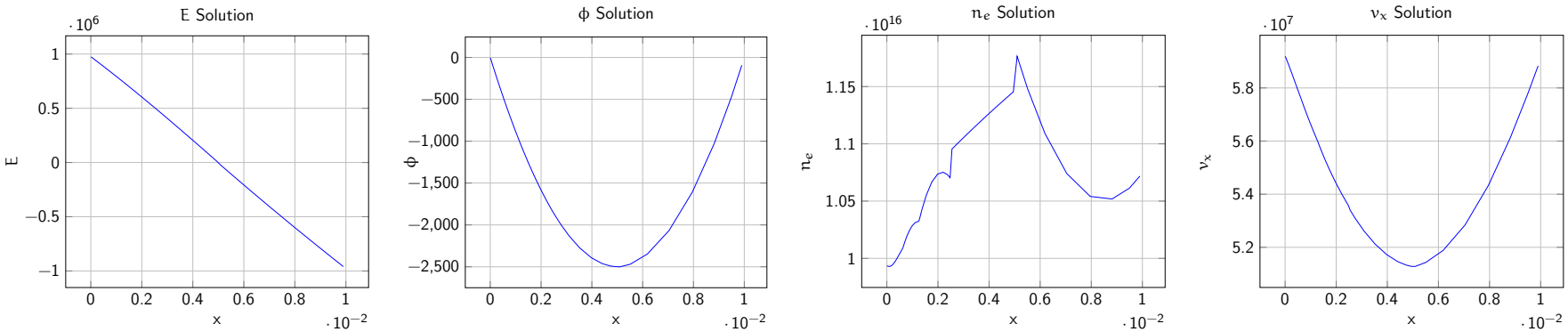


Figure: Final-time solution for the cold diode problem, using space-time DPG with initial $2 \times 4 \times 10$ mesh, with initial refinements at inflow to resolve the BC to tolerance of 10^{-5}, followed by 0 energy-error driven mesh refinements. f is discretized with quadratic polynomials; the space-time Vlasov mesh has 2,544 elements and 187,184 degrees of freedom; the space-time Poisson mesh has 179 elements and 4686 degrees of freedom. The final-time spatial output has 6 elements.

Space-time Results

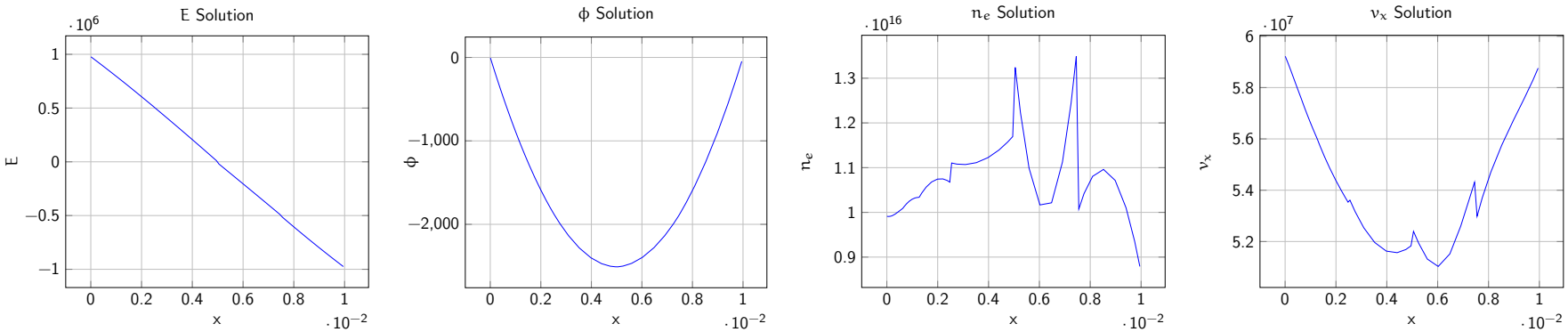


Figure: Final-time solution for the cold diode problem, using space-time DPG with initial $2 \times 4 \times 10$ mesh, with initial refinements at inflow to resolve the BC to tolerance of 10^{-5}, followed by 1 energy-error driven mesh refinements. f is discretized with quadratic polynomials; the space-time Vlasov mesh has 2,866 elements and 210,614 degrees of freedom; the space-time Poisson mesh has 224 elements and 5,888 degrees of freedom. The final-time spatial output has 7 elements.

Space-time Results

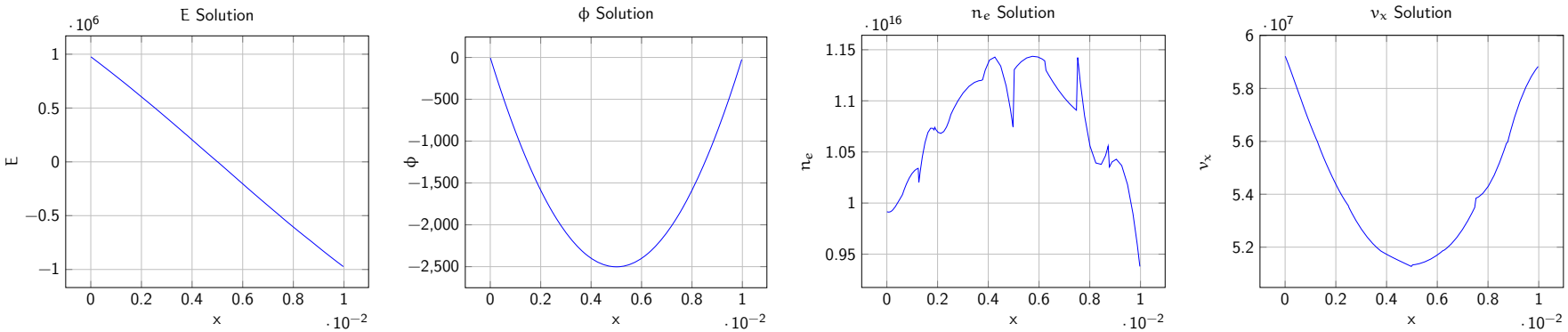


Figure: Final-time solution for the cold diode problem, using space-time DPG with initial $2 \times 4 \times 10$ mesh, with initial refinements at inflow to resolve the BC to tolerance of 10^{-5}, followed by 2 energy-error driven mesh refinements. f is discretized with quadratic polynomials; the space-time Vlasov mesh has 3,482 elements and 254,606 degrees of freedom; the space-time Poisson mesh has 317 elements and 8,346 degrees of freedom. The final-time spatial output has 11 elements.

Space-time Results

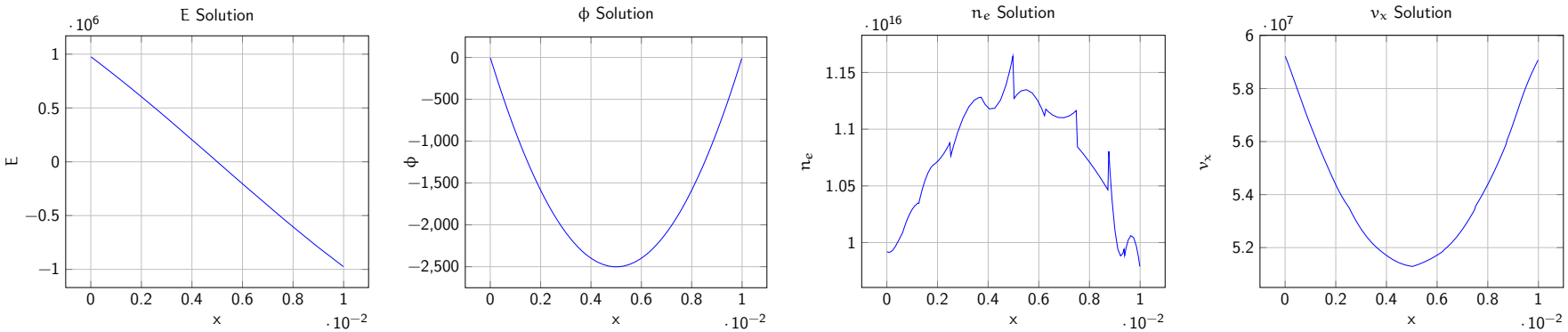


Figure: Final-time solution for the cold diode problem, using space-time DPG with initial $2 \times 4 \times 10$ mesh, with initial refinements at inflow to resolve the BC to tolerance of 10^{-5}, followed by 3 energy-error driven mesh refinements. f is discretized with quadratic polynomials; the space-time Vlasov mesh has 3937 elements and 286,811 degrees of freedom; the space-time Poisson mesh has 383 elements and 10,078 degrees of freedom. The final-time spatial output has 12 elements.

Space-time Results

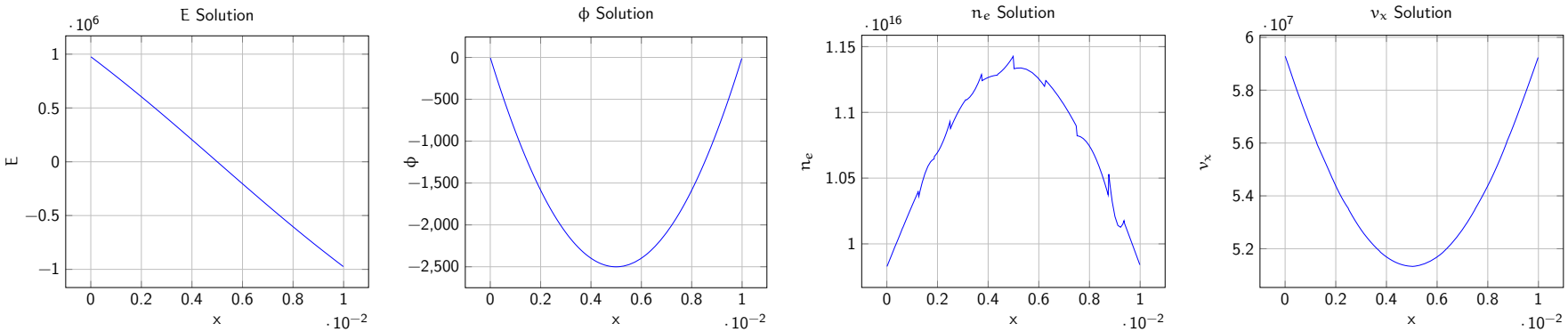


Figure: Final-time solution for the cold diode problem, using space-time DPG with initial $2 \times 4 \times 10$ mesh, with initial refinements at inflow to resolve the BC to tolerance of 10^{-5}, followed by 4 energy-error driven mesh refinements. f is discretized with quadratic polynomials; the space-time Vlasov mesh has 7,801 elements and 562,883 degrees of freedom; the space-time Poisson mesh has 893 elements and 23,250 degrees of freedom. The final-time spatial output has 16 elements.

Space-time Results

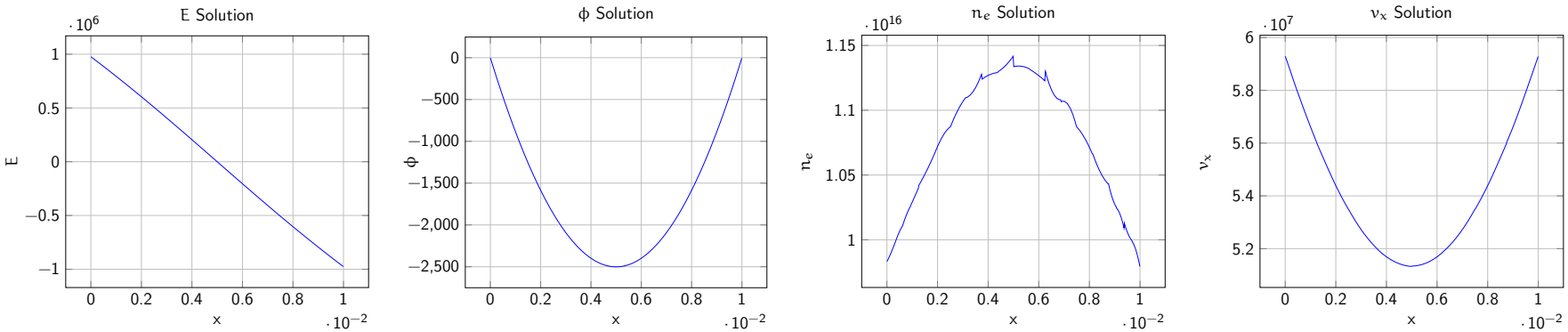


Figure: Final-time solution for the cold diode problem, using space-time DPG with initial $2 \times 4 \times 10$ mesh, with initial refinements at inflow to resolve the BC to tolerance of 10^{-5}, followed by 5 energy-error driven mesh refinements. f is discretized with quadratic polynomials; the space-time Vlasov mesh has 12,323 elements and 883,841 degrees of freedom; the space-time Poisson mesh has 1,283 elements and 33,342 degrees of freedom. The final-time spatial output has 22 elements.

Space-time Results

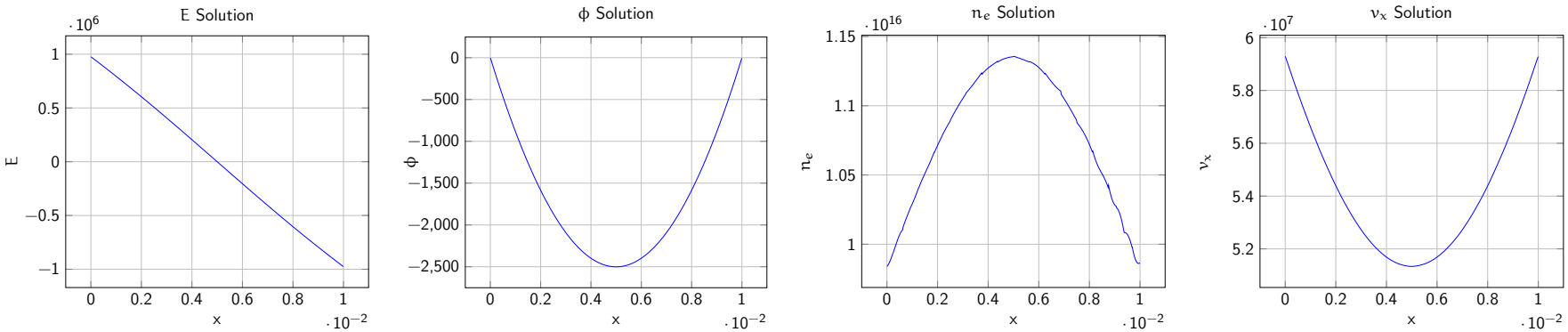


Figure: Final-time solution for the cold diode problem, using space-time DPG with initial $2 \times 4 \times 10$ mesh, with initial refinements at inflow to resolve the BC to tolerance of 10^{-5}, followed by 6 energy-error driven mesh refinements. f is discretized with quadratic polynomials; the space-time Vlasov mesh has 37,159 elements and 2,638,589 degrees of freedom; the space-time Poisson mesh has 2,858 elements and 74,156 degrees of freedom. The final-time spatial output has 31 elements.

Space-time Results

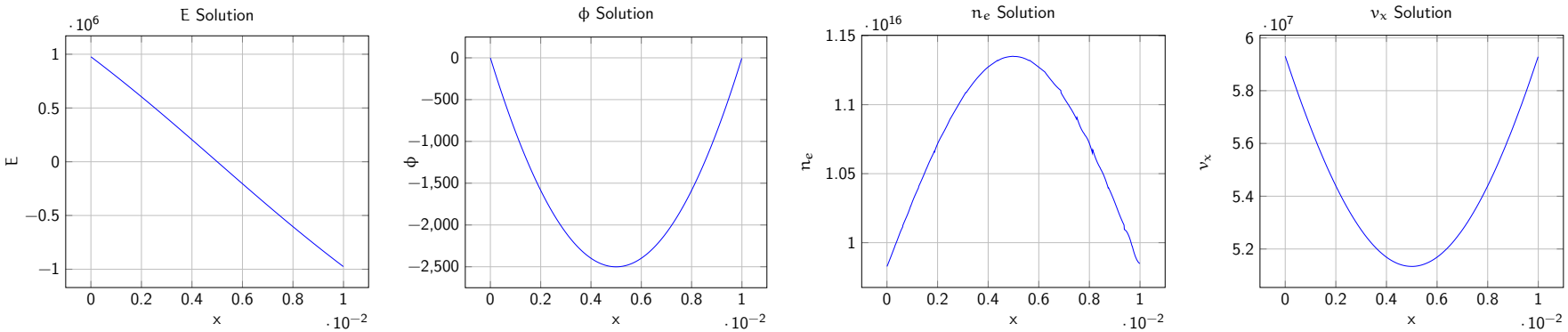


Figure: Final-time solution for the cold diode problem, using space-time DPG with initial $2 \times 4 \times 10$ mesh, with initial refinements at inflow to resolve the BC to tolerance of 10^{-5}, followed by 7 energy-error driven mesh refinements. f is discretized with quadratic polynomials; the space-time Vlasov mesh has 55,534 elements and 3,924,458 degrees of freedom; the space-time Poisson mesh has 4,157 elements and 107,490 degrees of freedom. The final-time spatial output has 36 elements.

Space-time Results

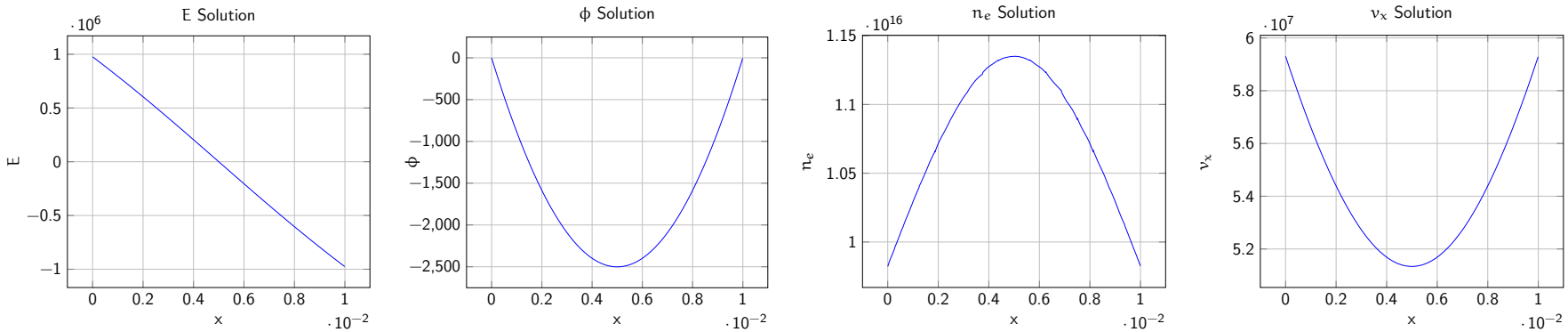


Figure: Final-time solution for the cold diode problem, using space-time DPG with initial $2 \times 4 \times 10$ mesh, with initial refinements at inflow to resolve the BC to tolerance of 10^{-5}, followed by 8 energy-error driven mesh refinements. f is discretized with quadratic polynomials; the space-time Vlasov mesh has 77,220 elements and 5,447,804 degrees of freedom; the space-time Poisson mesh has 5,198 elements and 134,004 degrees of freedom. The final-time spatial output has 38 elements.

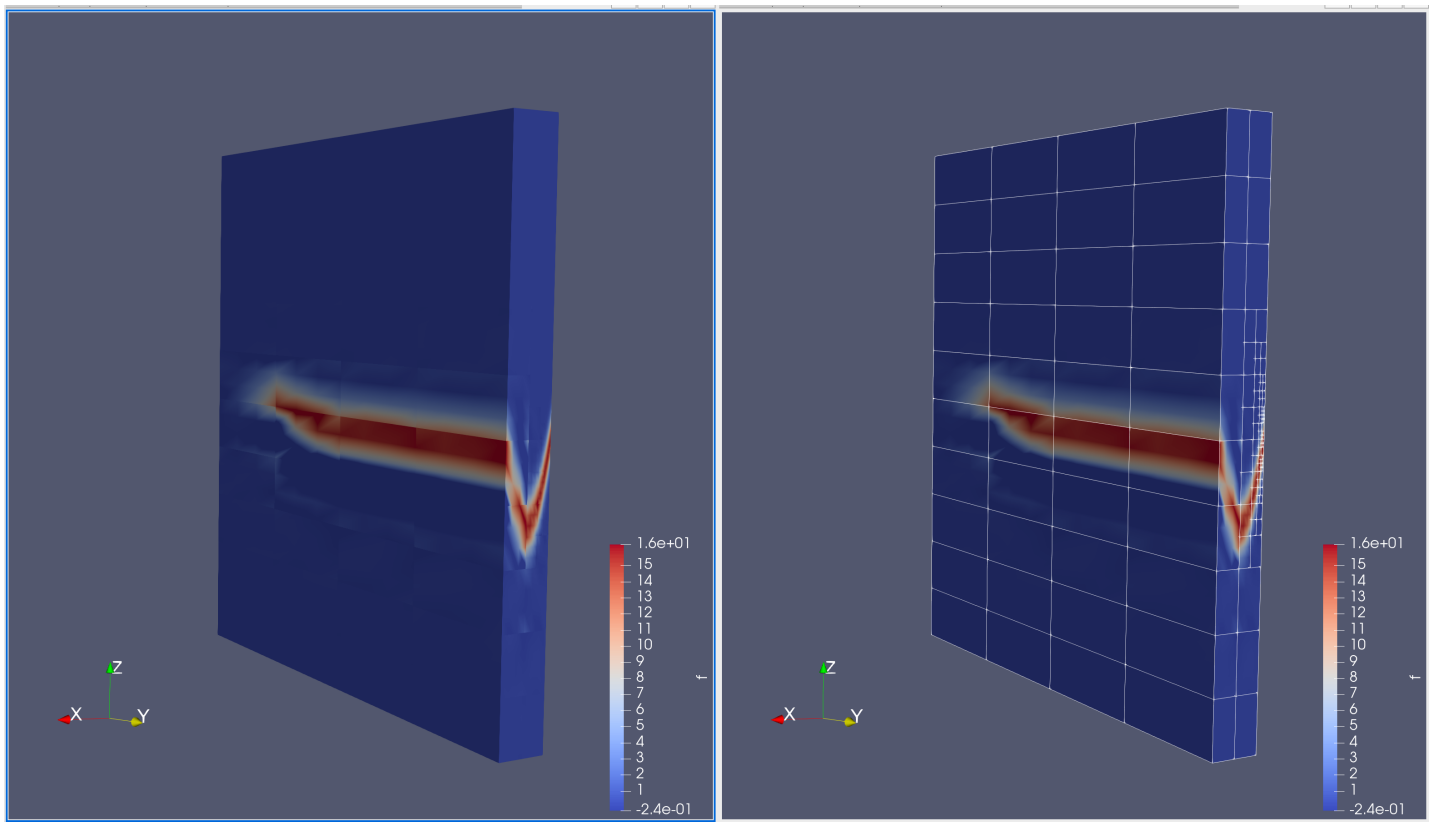
Adaptive Space-Time Results

Sandia National Laboratories

For this AMR run, we perform a set of initial refinements, driven by the error in the boundary condition, until that error is less than a specified tolerance in the relative L^2 norm on the boundary. In this run, we use the following setup:

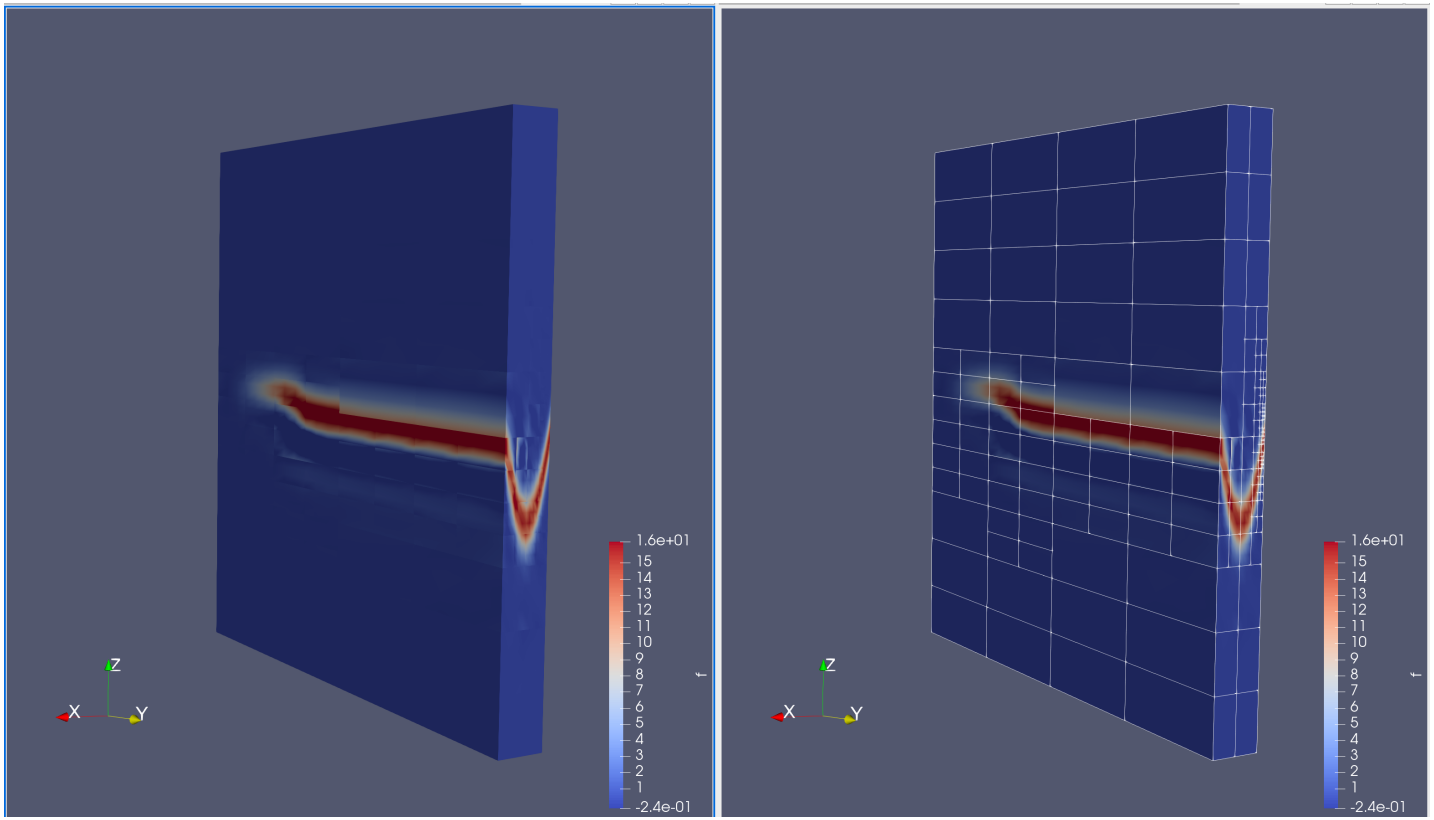
- coarse mesh: $2 \times 4 \times 10$ elements
- $\sigma = 0.025$
- BC tol: 10^{-5}
- quadratic field variables ($p = 3$)
- test space enrichment $\Delta p = 4$
- greedy refinement parameter $\theta = 0.2$

Adaptive Space-Time Results: Cold Diode



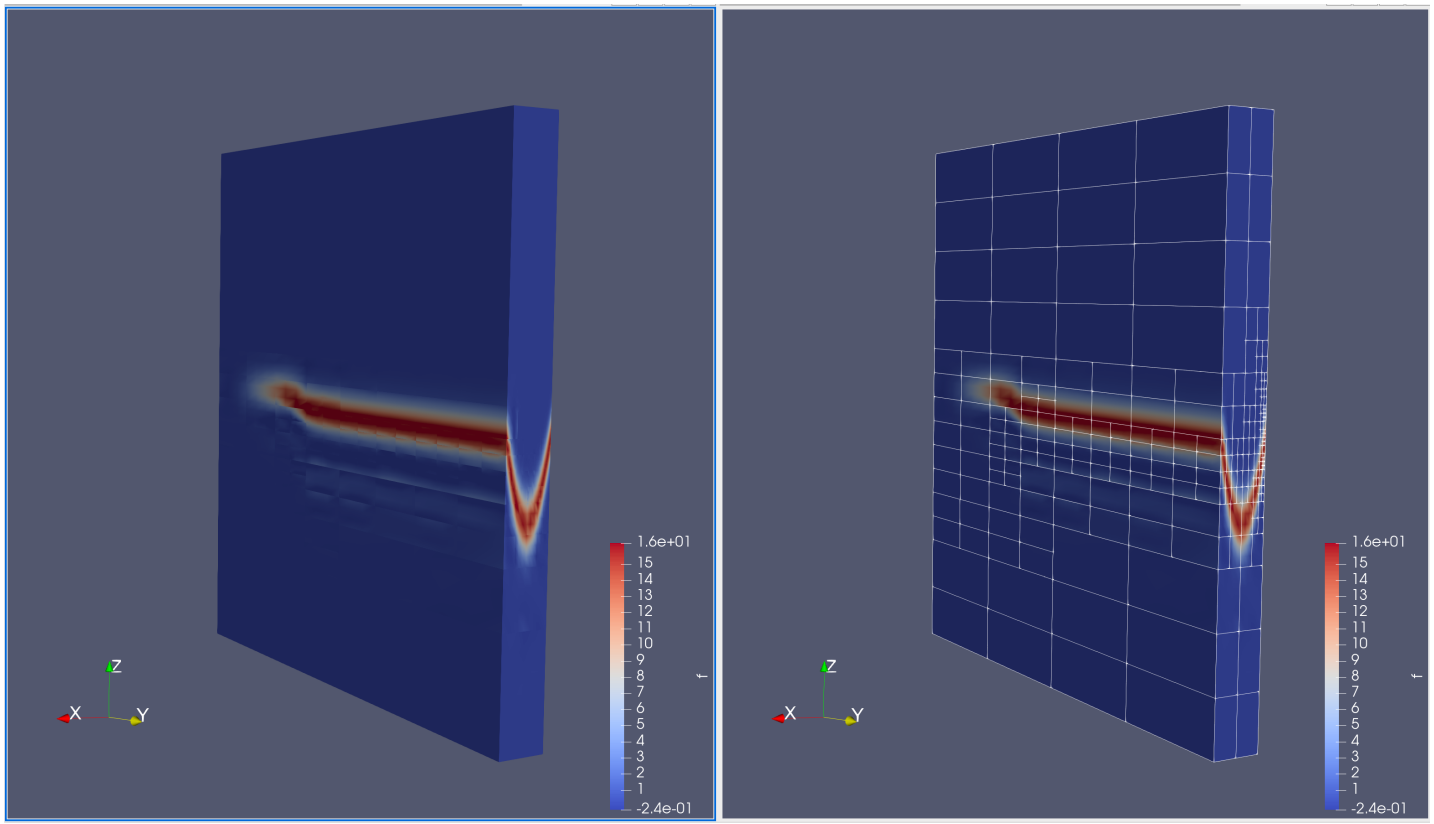
Vlasov solution for the cold diode problem, after 0 energy-error refinements. Time dimension is coming out of the screen; the left side is the spatial outflow.

Adaptive Space-Time Results: Cold Diode



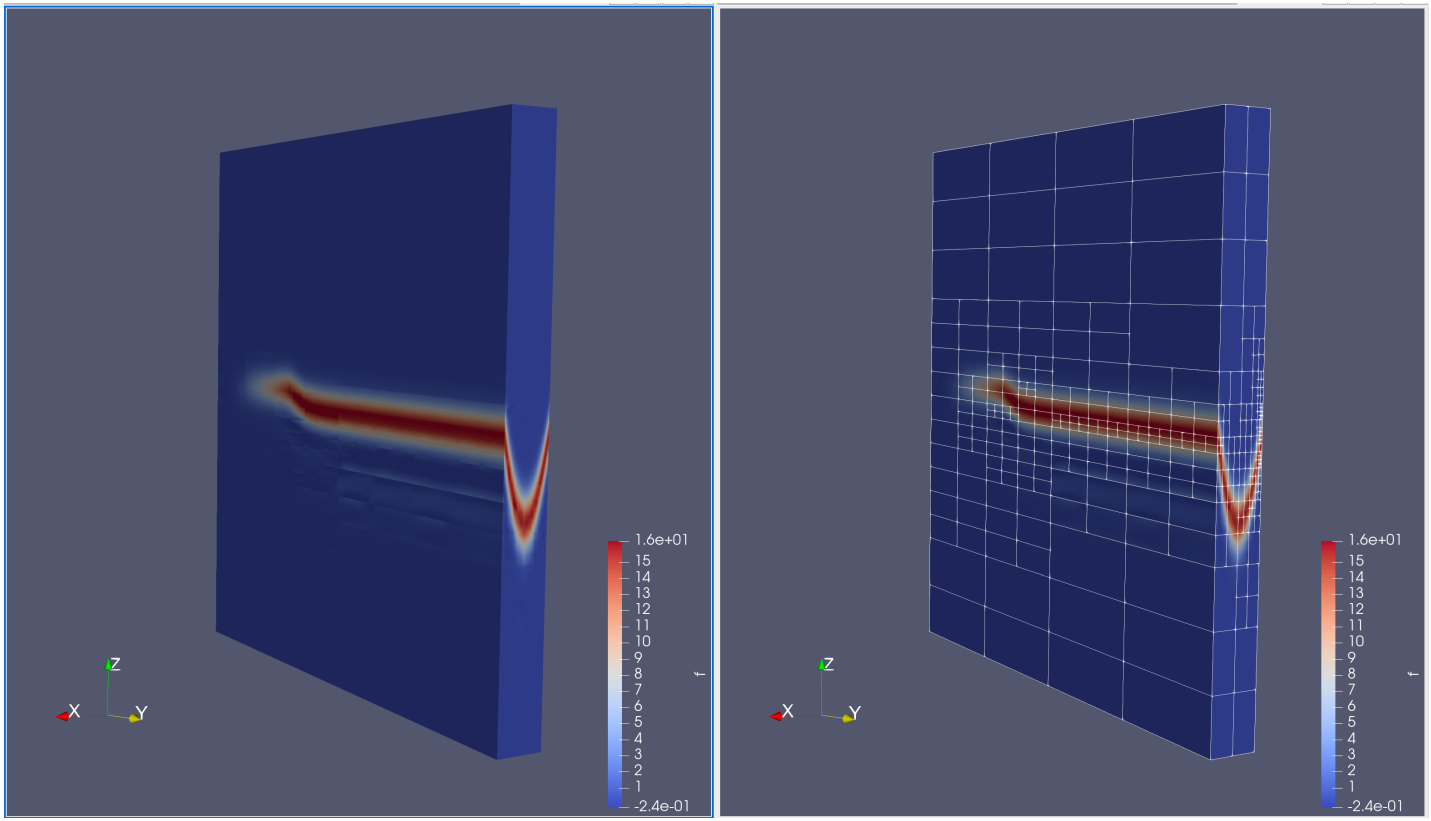
Vlasov solution for the the cold diode problem, after 1 energy-error refinement. Time dimension is coming out of the screen; the left side is the spatial outflow.

Adaptive Space-Time Results: Cold Diode



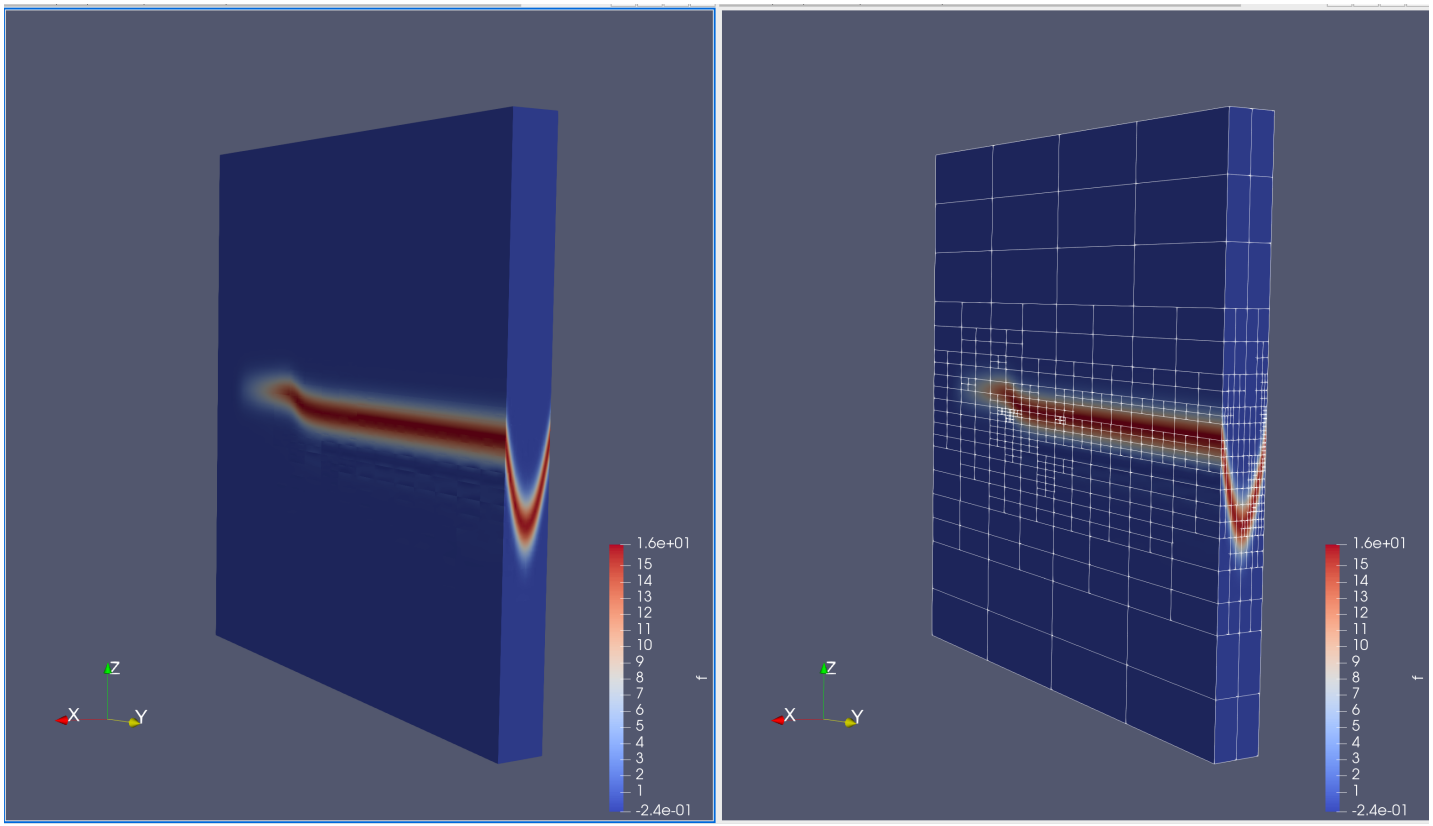
Vlasov solution for the cold diode problem, after 2 energy-error refinements. Time dimension is coming out of the screen; the left side is the spatial outflow.

Adaptive Space-Time Results: Cold Diode



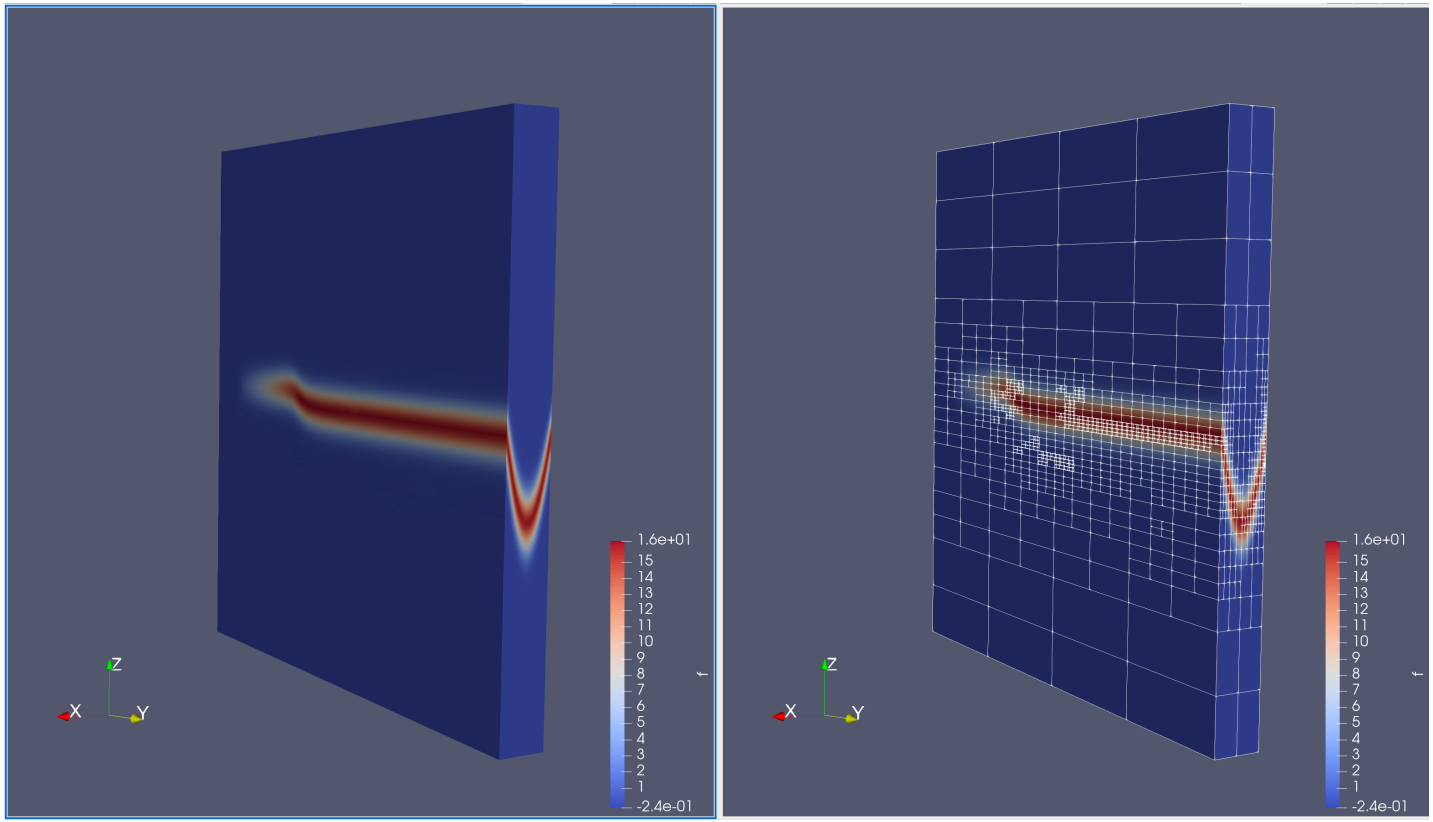
Vlasov solution for the cold diode problem, after 3 energy-error refinements. Time dimension is coming out of the screen; the left side is the spatial outflow.

Adaptive Space-Time Results: Cold Diode



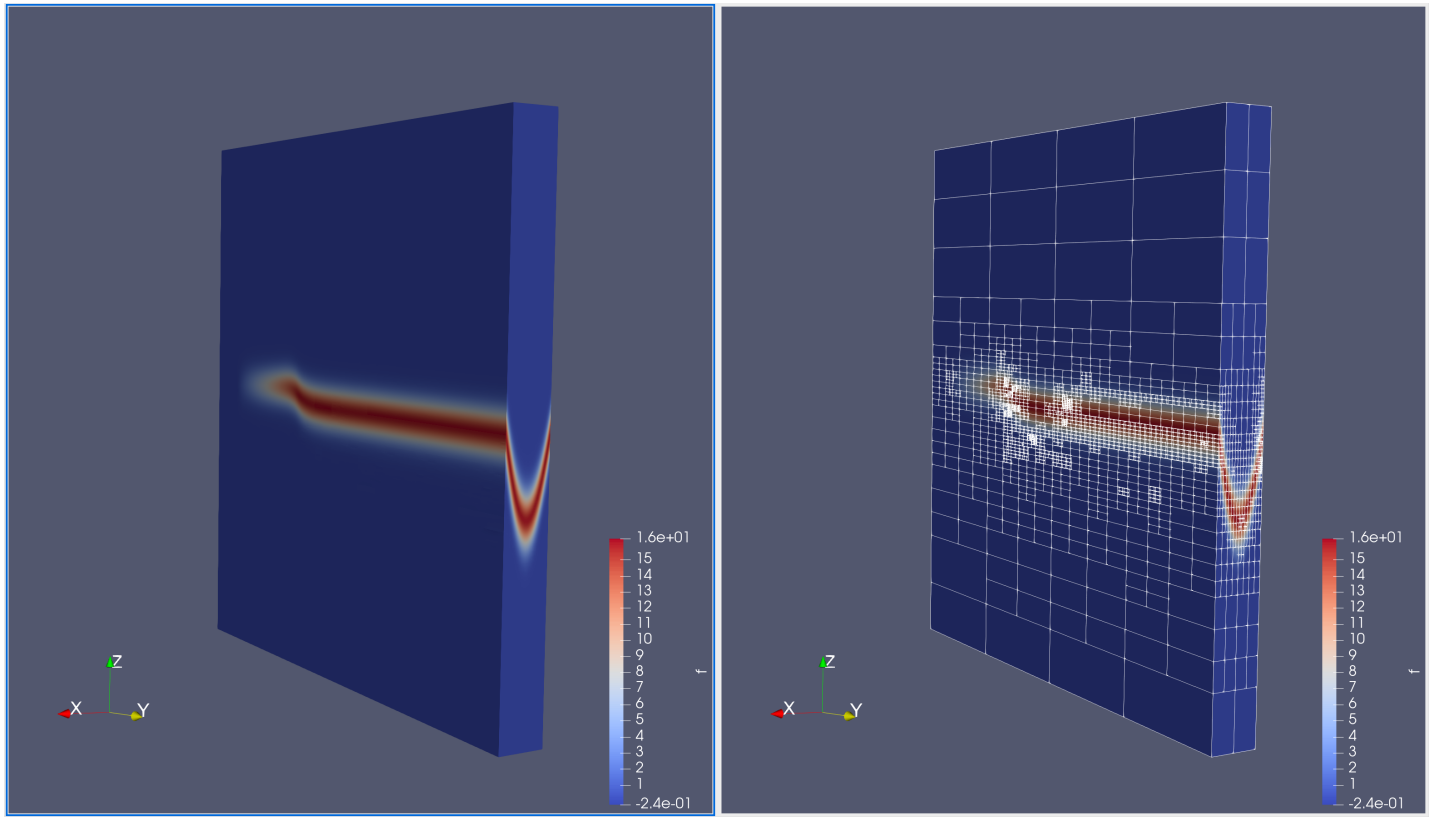
Vlasov solution for the cold diode problem, after 4 energy-error refinements. Time dimension is coming out of the screen; the left side is the spatial outflow.

Adaptive Space-Time Results: Cold Diode



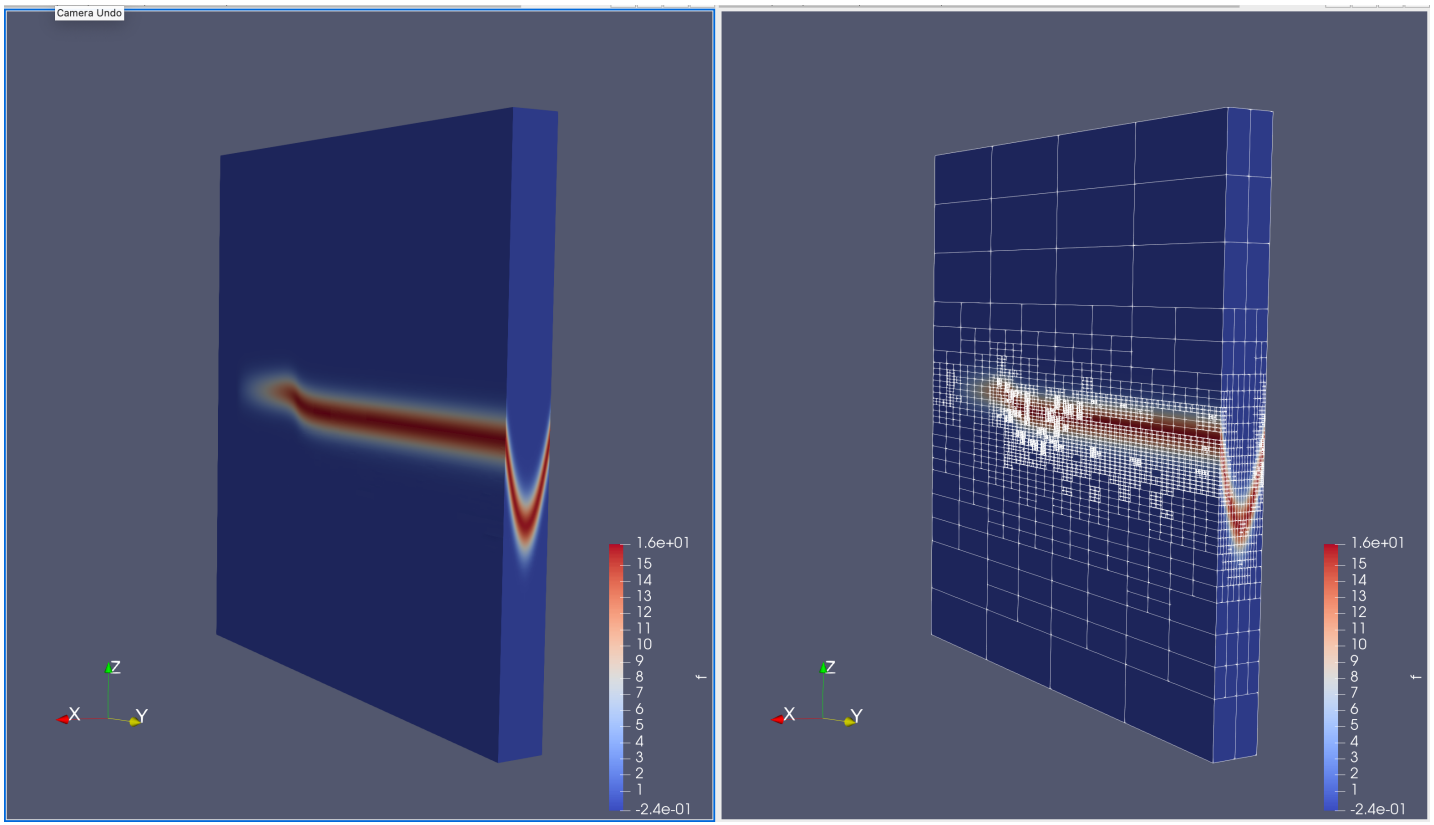
Vlasov solution for the cold diode problem, after 5 energy-error refinements. Time dimension is coming out of the screen; the left side is the spatial outflow.

Adaptive Space-Time Results: Cold Diode



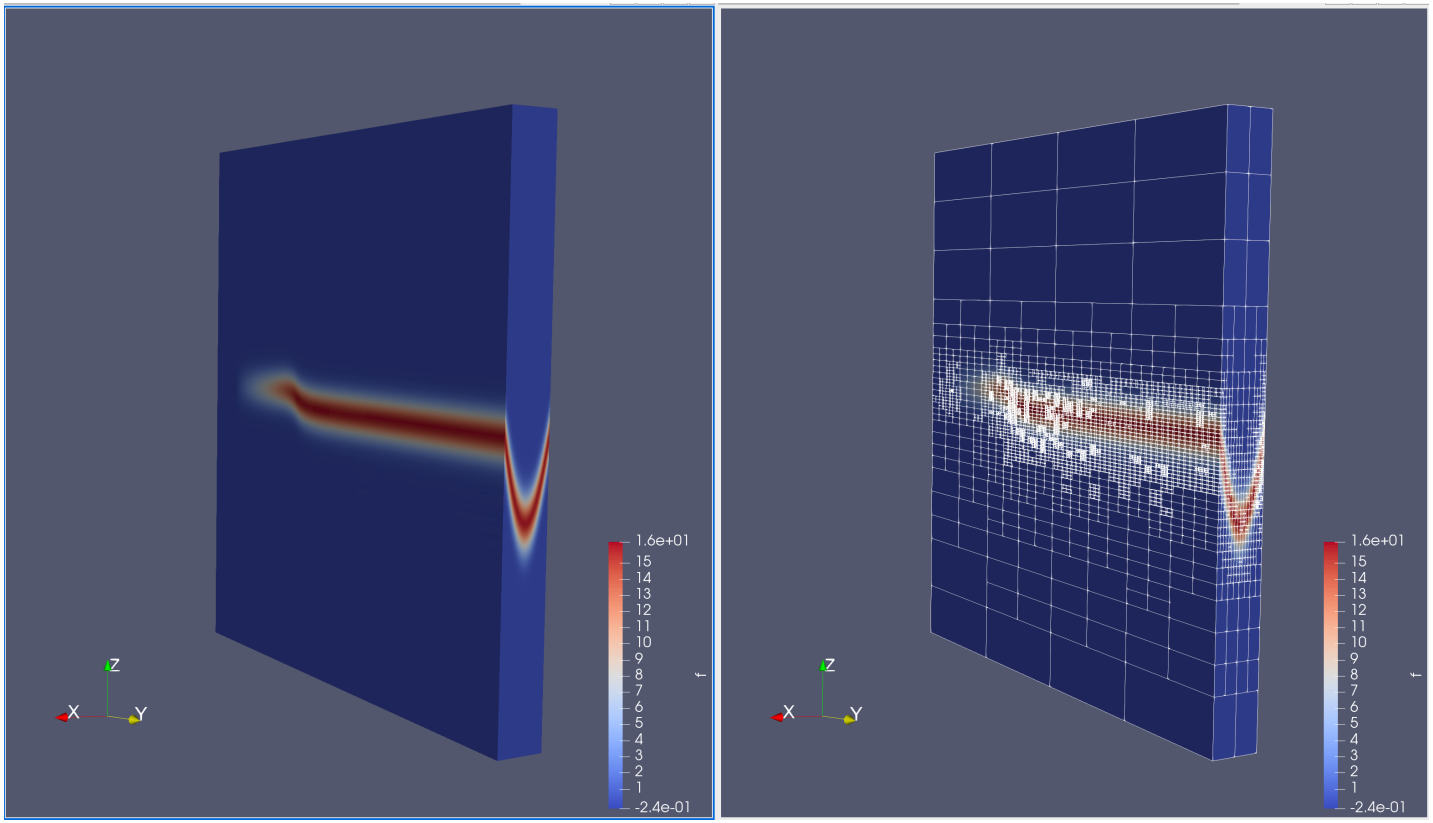
Vlasov solution for the cold diode problem, after 6 energy-error refinements. Time dimension is coming out of the screen; the left side is the spatial outflow.

Adaptive Space-Time Results: Cold Diode



Vlasov solution for the cold diode problem, after 7 energy-error refinements. Time dimension is coming out of the screen; the left side is the spatial outflow.

Adaptive Space-Time Results: Cold Diode



Vlasov solution for the cold diode problem, after 8 energy-error refinements. Time dimension is coming out of the screen; the left side is the spatial outflow.

Conclusion

We've demonstrated feasibility of DPG for Vlasov, in both time-marching and space-time formulations.

Future work:

- Study of circuit-coupled Vlasov-Poisson in the "B-dot" problem (involves experimental data).
- Faster assembly: sum factorization and "smart" assembly.
- Matrix-free implementation (Sandia is developing matrix-free preconditioners).
- Other time-marching formulations (Crank-Nicolson is low-hanging fruit).
- Study of higher dimensional Vlasov-Poisson, Vlasov-Maxwell.

I am very interested in collaborations. If you are interested in helping, please get in touch. Especially of interest:

- Anything we can do to accelerate high-dimensional DPG.
- Property preservation with DPG (e.g. non-negative field values).
- Machine learning for accelerating DPG solves and/or determining good anisotropic refinements.

Thanks for your attention!