A PERFORMANCE PORTABLE, FULLY IMPLICIT LANDAU COLLISION OPERATOR WITH BATCHED LINEAR SOLVERS

MARK F. ADAMS*, PENG WANG†, JACOB MERSON‡, KEVIN HUCK§, AND MATTHEW G. KNEPLEY¶

In fond memory of Ravindra Samtaney

Abstract. Modern accelerators use hierarchical parallel programming models that enable massive multithreading within a processing element (PE), with multiple PEs per device driven by traditional processes. *Batching* is a technique for exposing PE-level parallelism in algorithms that have traditionally run on MPI processes or multiple threads within a single process. Opportunities for batching arise in, for example, kinetic discretizations of magnetized plasmas where collisions are advanced in velocity space at each spatial point independently.

This paper builds on previous work on a high-performance, fully nonlinear, Landau collision operator by batching the linear solver, as well as batching the spatial point problems and adding new support for multiple grids for multiscale, multi-species problems. An anisotropic relaxation verification test that agrees well with previous published results and analytical models is presented. The performance results from NVIDIA A100 and AMD MI250X nodes are presented with hardware utilization analysis for each architecture. The entire implicit Landau operator time advance is implemented in Kokkos for performance portability, running entirely on the device and is available in the PETSc numerical library.

Key words. Batch solvers, Landau collision operator, GPU finite elements, kinetic methods

MSC codes. 76X05, 68N01, 65F10, 65F50, 65Y20, 65Z05

1. Introduction. The programming model used for modern accelerator hardware, introduced in the *CUDA* programming language, is now supported in several languages and libraries, such as Kokkos [9]. This model supports massive multithreading within a processing element (PE) and encourages vector processing with C/C++ syntax, in addition to traditional MPI processes for coarse-grain parallelism. The Kokkos programming model supports accelerator languages, such as *SYCL*, *HIP* and *CUDA*, as well as *OpenMP*.

High-dimensional applications with tensor product structures, such as combustion with chemistry [3], kinetic methods with collision operators [10, 12], ensemble problems in uncertainty quantification [17, 18], and others [5, 19] run many small problems independently. These problems have traditionally been run in MPI processes or with small-scale thread parallelism within a process. However, these solves may be small enough to run effectively on a single PE. Techniques known as *batching* are designed to explicitly expose this PE-level parallelism to the accelerator.

Phase space problems, from $2D$ to $6D$ are used to accurately model many problems in computational physics. Magnetized plasmas are one such application. Plasmas are governed by the symplectic Vlasov-Maxwell system [26], coupled with a metric or dissipative collision operator. This work focuses on Fokker-Planck collisions in Landau form, which is the gold standard for small-angle collision-dominated plasmas [16], using discretizations that preserve the geometric structure of this system in the *metriplectic* formalism [12, 15]. The evolution of the collision operator is computed

*Lawrence Berkeley National Laboratory, Berkeley, CA (mfadams@lbl.gov).
†NVIDIA Corporation, Santa Clara, CA.
‡Rensselaer Polytechnic Institute, Troy, NY.
§University of Oregon, Eugene, OR.
¶University of Buffalo, Buffalo, NY.

at each spatial point independently and is well-suited to batch processing, including the linear solver in a nonlinear solver used in implicit time integrators.

This paper builds on previous work on a high-performance, portable, structure-preserving, grid-based, Landau collision operator with high-order accurate finite element discretizations and block-structured adaptive mesh refinement (AMR) [12, 2, 1]. Background from previous work is presented in §2 followed by new material:

- multiple grids and batching of the collision operator (§3),
- performance with a ten species test on NVIDIA and AMD device nodes (§4),
- unstructured adaptive meshing for collision operators (§5),
- an anisotropic relaxation verification test (§6),
- and performance data on NVIDIA A100 and AMD MI250X nodes (§7),
- MI250X and A100 hardware utilization analysis (§8),

and §9 concludes the paper. All codes are available in PETSc. Data, plotting scripts and reproducibility instructions are publicly available (Appendix A).

2. Landau collisions for magnetized plasmas. This section reviews previous work on structure preserving methods for the time evolution of the Landau collision operator [12, 2, 1]. The density of each species α, $f_\alpha(t,\vec{v})$, evolves, in velocity space $\vec{v}\in\mathbb{R}^3$, from collisional effects with each species β according to $\frac{df_\alpha}{dt}=\sum_\beta C\left[f_\alpha,f_\beta\right]_{\alpha\beta}$, with

$$C\left[f_\alpha,f_\beta\right]_{\alpha\beta}=\nu_{\alpha\beta}\frac{m_0}{m_\alpha}\nabla\cdot\int_{\bar{\Omega}}d\bar{v}\;\mathbf{U}(\vec{v},\bar{v})\cdot\left(\frac{m_0}{m_\alpha}\bar{f}_\beta\nabla f_\alpha-\frac{m_0}{m_\beta}f_\alpha\bar{\nabla}\bar{f}_\beta\right), \tag{2.1}$$

a collision frequency $\nu_{\alpha\beta}=e_\alpha^2e_\beta^2\ln\Lambda_{\alpha\beta}/8\pi m_0^2\varepsilon_0^2$, the Coulomb logarithm $\ln\Lambda_{\alpha\beta}$, an arbitrary reference mass m_0, the vacuum permittivity ε_0 and the effective charges e_α of each species. $\mathbf{U}(\vec{v},\bar{v})$ is the Landau tensor [16, 12]. Overbar terms are evaluated on the grid for the domain $\bar{\Omega}$ of species β and $\bar{v}\equiv\vec{\bar{v}}$ for clarity.

This inner integration over all domains, $\bar{\Omega}$, results in an $\mathcal{O}(N^2)$ work complexity algorithm, where N is the sum of the number of species times the number of integration points on each species' grid. This coupling between species is only through the inner integral and results in the property that each species can use a separate grid, which is common in physics codes [10].

2.0.1. Nondimensionalization and Landau structure. The system (2.1) is nondimensionalized with a reference speed v_0, such as the thermal speed of electrons, to define a velocity coordinate $\vec{x}=\vec{v}/v_0$, not to be confused with the spatial coordinate. The distribution function variable is normalized with $\tilde{f}_\alpha=f_\alpha v_0^3/n_0$ with a number density n_0. The electrons are used to set v_0 and m_0 in this work. Time and the collision frequency are nondimensionalized with

$$t_0=\frac{8\pi m_0^2\varepsilon_0^2v_0^3}{e^4\ln\left(\Lambda_{ee}\right)n_0},\quad\tilde{\nu}_{\alpha\beta}=\frac{t_0n_0}{v_0^3}\nu_{\alpha\beta}, \tag{2.2}$$

with elementary charge e. Further, $d\vec{x}=v_0^{-3}d\vec{v}$, $\mathbf{U}(\vec{x},\bar{x})=v_0\mathbf{U}(\,\vec{v},\bar{v})$ and $\frac{\partial}{\partial\vec{x}}=v_0\frac{\partial}{\partial\vec{v}}$.

2.1. Discretizations. The time evolution of the collisional dynamics of the density of each species $f(\vec{x})$ with that of all species β, after nondimensionalization and assuming all species are identical for clarity, can be written as

$$\frac{df}{dt}=\sum_\beta\nabla\cdot\int_{\bar{\Omega}}d\bar{x}\;\mathbf{U}(\vec{x},\bar{x})\cdot\left(\bar{f}_\beta\nabla f-f\bar{\nabla}\bar{f}_\beta\right)=\left(\nabla\cdot\mathbf{D}\left(\vec{x}\right)\nabla-\nabla\cdot\mathbf{K}\left(\vec{x}\right)\right)f, \tag{2.3}$$

with tensor $\mathbf{D}(\vec{x}) \equiv \sum_\beta \int_{\bar{\Omega}} d\bar{x}\, \mathbf{U}(\vec{x},\bar{x}) \cdot \bar{f}_\beta$ and vector $\mathbf{K}(\vec{x}) \equiv \sum_\beta \int_{\bar{\Omega}} d\bar{x}\, \mathbf{U}(\vec{x},\bar{x}) \cdot \bar{\nabla} \bar{f}_\beta$.

This system is evolved in time with a "solver stack" of time integrators, nonlinear and linear algebraic solvers, and discretizations. The fully implicit time integrators in this paper, backward Euler or a three stage adaptive Runge-Kutta method, are sufficient for energy conservation but do not result in an entropy stable, or monotonic in entropy, method. A discrete gradient time integrator can be entropy stable [15] and is under development.

A finite element discretization of (2.1) that conserves moments up to the order of the polynomial that can be represented exactly by the finite element space was developed by Hirvijoki and Adams [12]. For example quadratic elements, $P2$ or $Q2$, conserves up to the second moment, which is energy and is of particular interest to physicists. The spatial discretization enforces weak equivalence by multiplying (2.3) with a test function ϕ_i in a set of basis vectors and integrating over the domain in a standard finite element process with the solution expressed as a weighted sum of the same set of basis vectors. After integrating by parts, the i^{th} equation is

$$\int_\Omega d\vec{x}\, \phi_i \frac{df(\vec{x})}{dt} = -\int_\Omega d\vec{x}\, \nabla\phi_i \cdot (\nabla \cdot \mathbf{D}(\vec{x}) \nabla - \nabla \cdot \mathbf{K}(\vec{x})) f(\vec{x}). \tag{2.4}$$

The exact Jacobian of the system is dense, however an approximation that ignores linearizing the inner integrals is sparse. That is, at each point $\vec{x}$ one integrates over all domains to compute $\mathbf{D}(\vec{x})$ and $\mathbf{K}(\vec{x})$ and then discretizes $\nabla \cdot \mathbf{D}(\vec{x}) \nabla$ and $\nabla \cdot \mathbf{K}(\vec{x})$.

2.2. Data and work partitioning model. In matrix form, with S species and letting $u = [f_1, f_2, ..., f_S]^T$, the system can be written as an $S \times S$ block matrix, A, defined by $A[u](\alpha,\beta) \equiv C[f_\alpha, f_\beta]_{\alpha\beta}$ from (2.1). An implicit time integrator, backward Euler for example, evolves a state u^k at time t to the state u^{k+1} at time $t + \Delta t$ according to $u^{k+1} \leftarrow u^k + \Delta t A\left[u^{k+1}\right] u^{k+1}$, which requires a nonlinear solve of $\left(I - \Delta t A\left[u^{k+1}\right]\right) u^{k+1} = u^k$ for u^{k+1}. The finite element discretization results in the nonlinear algebraic system $(M_h + \Delta t A_h[u_h])\, u_h^{k+1} = u_h^k$, with the finite element *mass* matrix M_h (see §III in [12] for details). The mass matrix is added in a separate kernel due to the PETSc time stepper interface design.

A Newton iteration first computes the residual with the current solution u, $r \leftarrow u^k - A_h[u]\, u$, then solves for the correction δ in $(M + \Delta t A_h[u])\, \delta = r$ and updates the solution $u \leftarrow u + \delta$. Figure 1 diagrams a backward Euler time step with the nonlinear

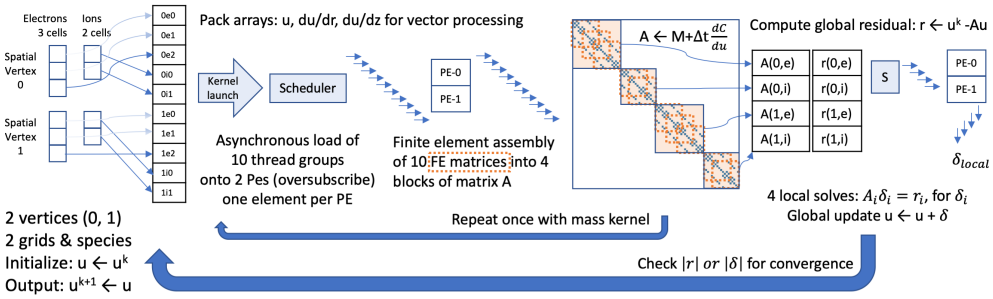


Fig. 1: Diagram of one time step on a generic accelerator

solver, batched matrix construction and linear solver with two spatial vertices, a two

species plasma, a three cell electrons (e) and two cell ions (i) mesh on a generic accelerator with asynchronous scheduler (S). All work and data remain on the device throughout the full collision time advance.

2.2.1. Jacobian and mass matrix construction. The algorithm for computing the Jacobian matrix of (2.1) can be expressed as six nested loops (Algorithm 1, [2]), and implemented with a naive kernel (Algorithm 3, [2]) and optimized by computing sub-expressions and loop hoisting (Algorithm 1, [1]). The data structures are optimized for vector processing (Algorithm 2, [2]). Our strategy for partitioning data and work is to compute each element's matrix with one Kokkos thread group, or PE, and assemble the global Jacobian matrix on the device with the COO interface in PETSc [20], as illustrated in Figure 1.

The finite element mass matrix is linear and does not change during the simulation because the basis functions on different grids are trivially orthogonal. One can compute the mass matrix once, on the CPU for convenience, copy it to the GPU and add it to the shifted Jacobian matrix. We found that while this approach was a bit faster with small number of batches the memory bandwidth demands with large batch sizes resulted in poor performance and recomputing the mass term on-the-fly on the GPU was faster and used less memory.

2.2.2. Shared memory. The batch solvers offer the opportunity to place data into shared memory explicitly instead of relying on the cache system to pull data from global memory. Shared memory tends to be partitioned into a user managed partition and a system cache partition. PETSc, like the Kokkos Kernels batch solvers, fills the user shared memory space with as many Krylov work vectors as will fit, prioritizing vectors appropriately, and relying on the system to manage the rest. We observe that the cache system is effective, with only a small performance increase from explicit use of shared memory. The matrix construction also explicitly places data in shared memory, such as the accumulation of integrals for **D** and **K** in (2.3), as is described in §III.D and §III.E of [1] .

2.3. Coordinate systems. The Landau integral is inherently three dimensional, but in a strong magnetic guide field, a gyrokinetic approximation allows for the use of cylindrical coordinates, $\vec{x} = (r, z)$, to reduce the $3D$ problem to a $2D$ computation, or in velocity space $2V$, and (2.3) is modified accordingly (§III.A [1]). The z coordinate is aligned with the magnetic field and is referred to as $v_{\parallel}$. Likewise r is referred to as $v_{\perp}$. Both the $2V$ and full $3V$ models are investigated in this report. The $3V$ model is required for extension to relativistic regimes [4, 6, 23], but $2V$ is the focus of performance optimization and is currently practical [10].

2.4. The Vlasov-Maxwell-Landau system. While the collision operator in this paper can be used as-is, it is intended for use within a fully kinetic model, such as the *Vlasov-Maxwell-Landau* system, the fundamental model of magnetized plasmas [26, 16]. The density for each species is evolved in phase space, using $\vec{x}$ for the spatial coordinate and $\vec{v}$ for the velocity coordinate, with up to three *configuration* space dimensions and three velocity space dimensions (6D or $3X + 3V$) according to

$$\frac{df_\alpha}{dt} \equiv \frac{\partial f_\alpha}{\partial t} + \frac{\partial \vec{x}}{\partial t} \cdot \nabla_x f_\alpha + \frac{\partial \vec{v}}{\partial t} \cdot \nabla_v f_\alpha = \sum_\beta C\left[f_\alpha, f_\beta\right]_{\alpha\beta} .$$

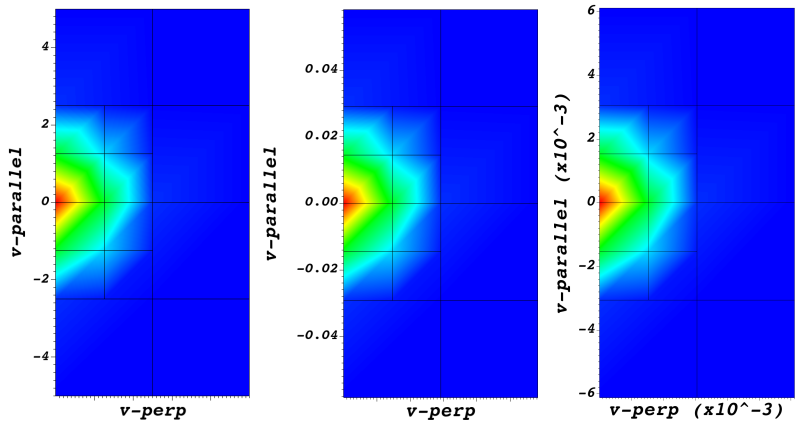
Maxwell's equations provide electro-magnetic acceleration forces in the $\partial\vec{v}/\partial t$ term.

The evolution of the system is split between a global symplectic time evolution of

the Vlasov-Maxwell system and a velocity space evolution of the collision operator. This paper focuses on the performance and verification of the collision operator. Our test problems, which are examples in PETSc, assume $\nabla_x = 0$ and $\partial\vec{v}/\partial t = a = 0$. However $\partial\vec{v}/\partial t$ terms are folded into the collision operator for a plasma resistivity verification test where a constant electric field is applied, inducing a current that is measured and the effective resistivity compared to the NRL Plasma Formulary [1].

3. Multiple grids and batching. Separate grids for each species simplify meshing because only a single scale needs to be resolved for the near-Maxwellian distributions common in plasmas, and adaptive meshing is less critical than if a single grid is used [2]. Additionally, species with similar thermal speeds, such as with many ionization states of impurities, can share a grid and amortize the cost of the Landau tensor [12], which is substantial in $2V$ (see Appendix in [12]).

Figure 2 shows three block-structured AMR grids used for the 10 species example in §4, with an electron grid, a light ion and heavy ion grid, plotted with a Maxwellian distributions in the axisymmetric coordinate system. Note the different scales on each of the grids, otherwise the images are identical.



(a) electrons (b) Deutarium (c) Tungsten

Fig. 2: Three grid example with Maxwellian distributions with scales for each species group. (Linear interpolation in Visit results in visualization artifacts)

3.1. Batching and aggregation. Kinetic applications commonly use operator split time integrators, where the symplectic Vlasov system evolves the full phase space distribution and the metric collision operator evolves the velocity space distribution separately at each spatial grid point. This algorithm has a high degree of parallelism and is well-suited to modern hardware with large numbers of simple processing elements (PEs). *Batching* refers to writing algorithms in a single, or a few, kernels that run asynchronously on PEs so as to fully exploit available parallelism in the algorithm. In the Landau operator the computation and assembly of element "stiffness" matrices and the subsequent linear solve are amenable to batching.

An alternative to batching is to *aggregate* the many small systems into a single system that can use sparse linear algebra operators in vendor libraries or higher level libraries like PETSc. Aggregation requires block diagonal matrices as illustrated in Figure 1. The aggregated methods use separate kernel launches for each primitive operation such as a matrix-vector product. The time integrators use high-level operations that are not compute intensive, are not iterative, and with adaptive time

stepping require global synchronization, and therefore aggregation is appropriate. Nonlinear solvers also use high-level operations and can be aggregated effectively, but the asynchronous processing of batching would benefit problems with nonhomogeneous vertex problems. We currently use PETSc's (aggregated) nonlinear solvers – batching is the subject of future work.

3.2. Linear solvers and batching. Iterative solvers lend themselves to both aggregated and batched solvers. The sparse matrix-vector product kernels in iterative solvers are amenable to vector processing, and with few data dependencies they are well-suited to modern accelerators. Aggregation solvers compute residuals, solve linear systems, and update solutions with traditional high-level sparse linear algebra on systems composed of aggregated small systems. Batch solvers require writing the entire solver and preconditioner in device kernel code. Batch solvers have several advantages in both performance and accuracy: the entire solve requires a single kernel launch, each solve can run asynchronously, exiting the kernel as soon as it converges to be immediately replaced with another solve by the scheduler, and (unwanted) global synchronization is avoided. Aggregate solvers merge the systems together with a single (L2 norm) convergence test, which obscures the residual of each system. Batch solvers maintain the semantics of the Krylov method for each system. Batched sparse solvers are available in the PETSc, Kokkos Kernels and Ginkgo libraries [18, 14].

In addition to code reuse, the aggregation approach has the advantage of being able to run effectively on a larger range of problem sizes because an individual linear system can run on just part of a PE or multiple PEs, transparently. Note, Kokkos Kernels and Ginkgo batch solvers support scaling down with multiple problems per thread group, but the PETSc batch solver does not. For scaling up, NVIDIA provides *cooperative groups* since CUDA-9 and the new *Hopper* architecture introduces hardware support for inter-PE communication called a *thread block cluster* that should allow for Kokkos to increase the size of its thread groups to use 16 SMs in the future. The $3V$ solves in §4.1.2 can be seen to suffer from this scaling up problem.

4. Landau time advance performance. The vast majority of the time in the full Landau time advance is spent in matrix construction and linear solves (e.g., Table 5), and the high-level timings in this section, and §7, focuses on these two phases. §8 analyzes low-level hardware utilization. This section uses a 10 species model problem to investigate throughput with respect to batch size and $2V$ vs $3V$ in §4.1, and component times with batched and aggregate linear solves in §4.2. §7 investigates component times with a two species anisotropic relaxation test with respect to order of the finite space with tensor and simplex elements. The test harness replicates the model problem to create a batch of problems in each MPI process, to mimic an application. Each NVIDIA GPU or AMD GCD on a compute node is driven by one MPI process.

A 10 species model problem. The model problem is a deuterium plasma with the addition of eight species of tungsten with different ionization states. This resembles a fusion energy science benchmark for plasma models with impurities. The electrons and deuterium each have a dedicated grid and the tungsten species share one grid, for a total of three block-structured AMR grids as shown in Figure 2. One level of AMR refinement about the origin is used with a 4×2 and a $4 \times 4 \times 4$ initial grid, in $2V$ and $3V$ respectively, resulting in 14 (Q3 elements) in $2V$ and 120 (Q2 elements) in $3V$. We have observed that these grids are sufficient to converge a plasma resistivity test to within about 1% of the fully converged state [1]. Each problem has 10 linear systems, one for each species, with 142 equations and 22.2 average non-zeros per row,

in $2V$, and 1,045 equations with an average of 54.0 non-zeros per row in $3V$. Each distribution is initialized with a Maxwellian with a given thermal temperature. An implicit backward Euler time integrator with electron temperature initialized to twice that of the ions is run toward equilibrium for ten time steps.

4.1. Throughput experiment. This section analyzes the throughput of the entire Landau time advance as well as that of the linear solvers, with respect to batch size for both the $2V$ and full $3V$, and for batched and aggregate solvers. Newton throughput is defined as the number of Newton iterations in the entire simulation times the batch size per device times the number of devices per node, divided by the *total run time*, which includes the Jacobian and mass matrix construction, the linear solve, and some linear algebra in the control logic of the time integrator and nonlinear solver, with a (CPU) setup phase that is amortized by the solve phase. Given that all tests use the same Landau solver with a slightly different linear solver, one could use many quantities of interest, such as flop rate, but Newton iterations is chosen because it is relevant to application scientists. Solver throughput is defined similarly, with the number of linear solves (with $10 - 15$ per nonlinear solve) divided by the time spent in the linear solve phase only.

Each of these problems requires a linear solve per species, resulting in ten batched linear solves for each problem in the batch. Two linear solvers are considered: a batched TFQMR solver in PETSc, written in Kokkos, and an aggregated TFQMR solver that uses Kokkos Kernels linear algebra primitives within the PETSc framework. Jacobi preconditioning is used throughout. Two architectures are examined:

- one node with four NVIDIA A100 Tensor Core GPUs with 256GB of memory, the *Perlmutter* machine at NERSC and,
- one node with four AMD MI250Xs, each with 2 Graphics Compute Dies (GCDs), with ROCm 5.1, the *Crusher* machine at ORNL.

4.1.1. Newton iteration throughput. Table 1 shows the $2V$ Newton iteration throughput, and Table 2 shows the $3V$ data, as a function of batch size with the batch linear solver and aggregated linear solver for the A100 and MI250X.

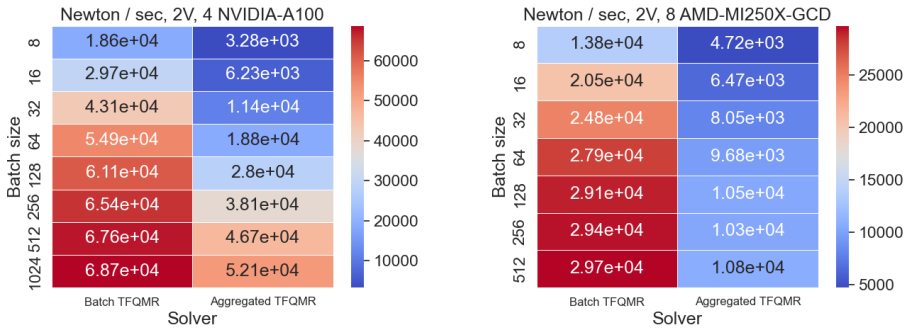


Table 1: 2V Newton throughput: NVIDIA A100 (left) and AMD MI250X (right)

The next section investigates the linear solver throughput, but this data shows that the A100 is more than $2x$ faster than the MI250X in $2V$ and about 50% faster in $3V$. Note that the GPU is pretty well saturated with a batch size of 128 in $2V$ and 16 in $3V$, and this corresponds to 57,344 and 42,240 integration points per GPU, and thus $2V$ and $3V$ saturate with a similar number of total integration points.

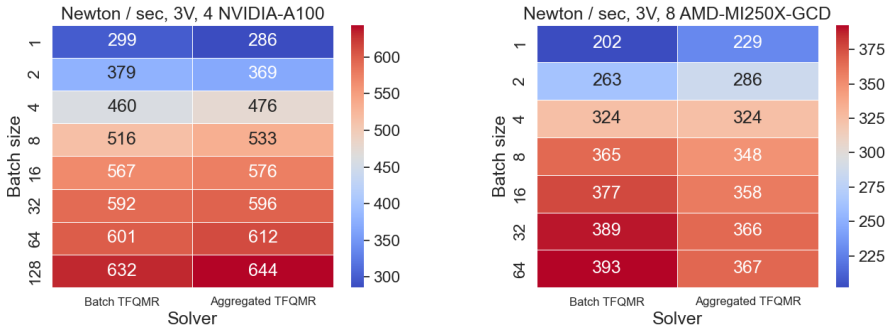


Table 2: 3V Newton throughput: NVIDIA A100 (left) and AMD MI250X (right)

4.1.2. Linear solver throughput. Table 3 shows the $2V$ linear solver throughput, and Table 4 shows the $3V$ throughput data, as a function of batch size with the batch linear solver and aggregated linear solver for the A100 (left) and MI250X (right).

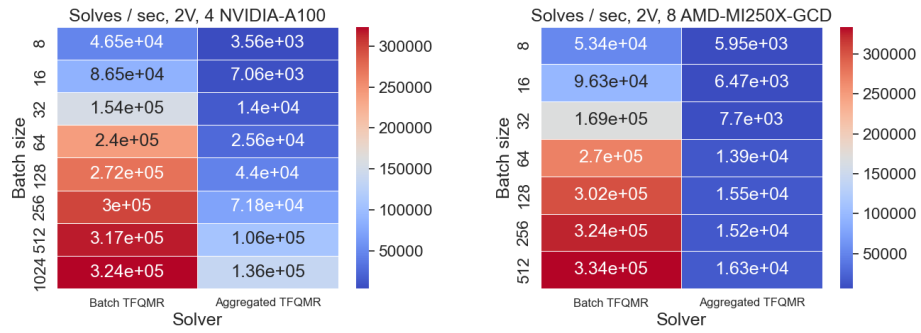


Table 3: 2V Linear solver throughput: NVIDIA A100 (left) and AMD MI250X (right)

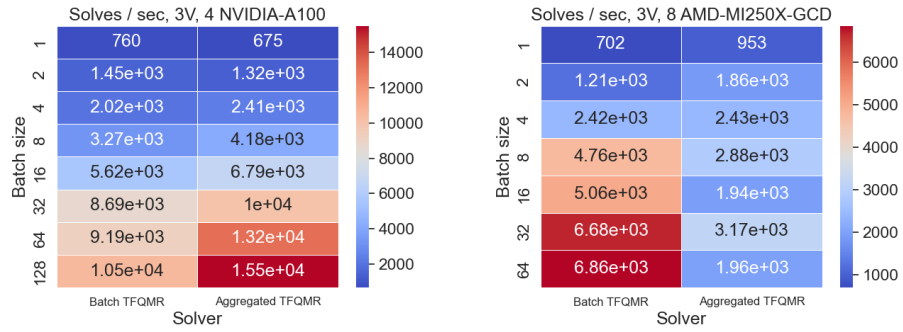


Table 4: 3V Linear solver throughput: NVIDIA A100 (left) and AMD MI250X (right)

This data shows that the batched TFQMR solver over $20x$ faster than the aggregated solver in $2V$ and the MI250X and over $2x$ faster on the A100, and that the two batch solvers have similar performance. The A100 batch solver is almost $2x$ faster in

$3V$, but the aggregated solver is faster than the batch solver on the A100. The batch solver is $3.5x$ faster than the aggregate solver on the MI250X. This under-performance of the batched solver in $3V$ illustrates the difficulty in fitting the problem to a fixed hardware resources as discussed in §3.2. The linear solver also does not saturate as clearly as the Jacobian dominated Newton iteration throughput.

4.2. Component times. Tables 5 and 6 show the component times on the A100 in $2V$ and $3V$, respectively, including mass matrix ("Mass"), Landau Jacobian ("Jacobian"), linear solver ("Solve"), the total time and the sum of the Krylov iterations for the aggregate solver and the sum of the maximum number of iterations of any system for the batch solver (see §4.2.1 for the batch solver iteration count distribution). Tables 7 and 8 show the same data on the MI250X in $2V$ and $3V$, respectively.

Table 5: 2V Component times (batch size = 256), NVIDIA-A100

Component	Jacobian	Mass	Solve	Total	Krylov iterations
Batch TFQMR	1.57	0.22	0.58	2.44	3,648
Aggregated TFQMR	1.57	0.22	1.76	3.69	4,015

Table 6: 3V Component times (batch size = 32), NVIDIA-A100

Component	Jacobian	Mass	Solve	Total	Krylov iterations
Batch TFQMR	29.69	3.00	2.33	35.08	2,785
Aggregated TFQMR	29.67	3.00	1.51	34.31	2,326

Table 7: 2V Component times (batch size = 128), AMD-MI250X-GCD

Component	Jacobian	Mass	Solve	Total	Krylov its
Batch TFQMR	4.28	0.29	0.49	5.08	3,642
Aggregated TFQMR	4.31	0.29	9.54	14.10	4,011

Table 8: 3V Component times (batch size = 64), AMD-MI250X

Component	Jacobian	Mass	Solve	Total	Krylov iterations
Batch TFQMR	168.16	18.07	11.28	196.81	2,796
Aggregated TFQMR	168.62	18.06	39.51	210.84	2,326

The balance of time between the Jacobian and linear solver is skewed toward the Jacobian construction on the AMD relative to the NVIDIA data. This is partially due to the AMD accelerator outperforming the NVIDIA node in the linear solver, but the MI250X node underperforms in the matrix construction.

4.2.1. Batch solver iteration distributions. The Krylov iteration counts for the batched solvers in Tables 5, 6, 7 and 8 obscures the distribution of iterations within the 10 linear species solve. Table 9 shows average iterations per linear solve for

aggregate (Agg.) and batch (Bat.) solvers in $2V$ and $3V$, with batch solver iteration count averages for each species 1-10, with one time step of the test problem. The $2V$ batch size is 256 and the batch size in $3V$ is 32.

Table 9: Average linear solver iteration counts: aggreate, batched and each of ten species in the batched solve

V	Agg.	Bat.	1	2	3	4	5	6	7	8	9	10
2	23.9	11.8	23.6	10.0	11.2	11.2	10.1	10.1	9.9	11.0	10.0	9.2
3	16.6	14.8	14.0	15.9	17.0	16.2	15.8	15.3	15.4	15.5	15.1	14.8

In $2V$ the electrons (first species) have a much higher iteration count, likely due to the higher speed of electrons (see Figure 2) and other implicit Landau solvers have noticed this phenomenon [14]. The $3V$ data, however, does not exhibit this behavior, which is not understood.

5. Unstructured mesh generation with simplex and tensor elements. The finite element formulation and implementation in this work supports unstructured simplex (triangles) and tensor cell (quadrilaterals and hexahedra) meshes as well as the block-structured AMR meshes used in §4. This section focuses on simplex mesh generation and presents the unstructured quadrilateral meshes newly available. Simplex meshes are read from a file and quadrilateral mesh can be similarly input, but the solver provide simple quadrilateral mesh generation (§5.2).

5.1. Simplex mesh generation. The simplex meshes are constructed on a semicircle centered at $(0, 0)$ with a radius of one. They are scaled within the solver to obtain the final dimensions. Unstructured meshes have been generated using simmetrix-simmodsuite version 18.0-220913 [24].

Table 10: Mesh quality metrics for each of the unstructured meshes.

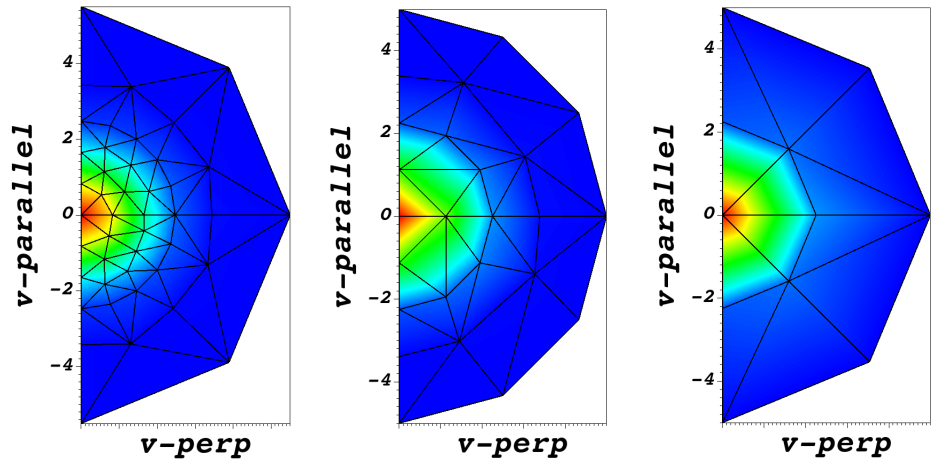
		Aspect Ratio		Skew	
# cells	P	Max	Median	Max	Median
144	2	2.618	1.512	0.552	0.161
68	3	2.926	1.762	0.650	0.246
24	4	2.801	1.304	0.254	0.097

Each model is partitioned at $z = 0$ which promotes a more symmetric mesh construction. Additionally, the model is partitioned at $r = 0.45$. We found that high aspect ratio elements near the boundary of the Maxwellian distribution caused a loss in accuracy. Mesh construction required a balance between the number of elements in the outer region ($r > 0.45$), largely controlled by gradation rate, and aspect ratio of these elements. We found that prioritizing the aspect ratio criteria over other element shape metrics during the mesh optimization and smoothing phases improved the overall mesh quality and solution accuracy. The shape quality metrics for the meshes used herein (Fig. 3) are given in table 10, where the aspect ratio is the ratio of the length of the longest triangle edge to the shortest triangle edge and the skew is defined as

$$\text{Skew} = \frac{A_{\text{opt}} - A}{A_{\text{opt}}}. \tag{5.1}$$

For 2D simplices the optimal area $A_{\text{opt}} = 3\sqrt{3}/4R^2$ where R is the circumcircle radius corresponds to the equilateral triangle. The skew ranges from 0 (equilateral) – 1 (degenerate), where quality meshes of complex geometries typically have values of Skew < 0.8.

These meshes are read into the solver from a file. The three meshes used for the anisotropic relaxation model (§6) with the initial electron distribution is shown in Figure 3.

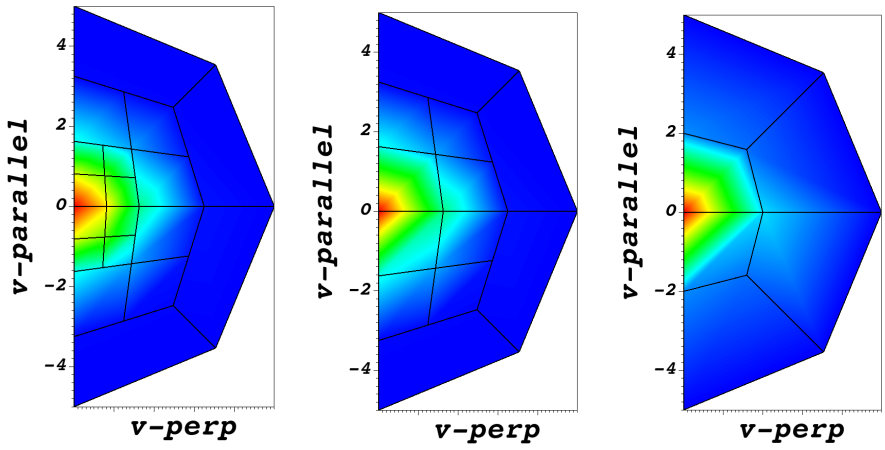


(a) $P2$ grid (b) $P3$ grid (c) $P4$ grid

Fig. 3: Semi-circular grids for the three orders of finite elements ($P2$, $P3$ and $P4$) tested with Maxwellian distribution of electrons in units of the thermal speed (Linear interpolation in Visit results in visualization artifacts)

5.2. Unstructured quadrilateral meshes. Tensor cell meshes have the advantage that our non-conforming AMR package, *p4est*, can accommodate unstructured tensor cells, quadrilaterals and hexahedra, but not simplicies. Nonconforming meshes can coarsen faster while maintaining well-shaped elements for better accuracy, which helps to reduce the number of cells required for a given level of accuracy. The disadvantage of tensor cells is that mesh generation is much less automated. Velocity space distributions are, however, relatively simple in many cases with near-Maxwellian distributions. The same adaptivity strategies developed for the block structured meshes apply to unstructured tensor cell meshes, such as refinement about the origin (eg, Figures 6f) and the parallel axis (eg, Figures 6a-6e). Figure 4 shows an example of a quadrilateral mesh used in the verification test in §7.2.

6. Anisotropic relaxation verification test. A plasma resistivity test was presented in previous work to verify the Landau collision operator and place constraints on the mesh sizes (§IV.B in [1]). Plasma resistivity does not test the collision rate, the $\nu_{\alpha\beta}$ term in (2.1), nor does it include anisotropic distribution functions. An anisotropic relaxation test, where each species of a deuterium plasma is initialized with different parallel and perpendicular temperatures, was reported by Hager et al. [10] and is the basis of this test.



(a) Q2 (b) Q3 (c) Q4

Fig. 4: Semi-circular grids for electrons with tensor elements, from the $Q2$, $Q3$, $Q4$ test in §7 (linear interpolation in Visit results in visualization artifacts).

6.1. Model problem. A deuterium plasma is initialized with an anisotropic distribution with respect to mean parallel and perpendicular speed within each species and between the two species, and evolved to equilibrium. The modified Maxwellian distribution is defined with an anisotropic parameter α ($\alpha = 1.3$ herein) as

$$f_s\left(v_\parallel, v_\perp\right) = \frac{n_s}{\alpha}\left(\frac{1}{\pi\theta}\right)^{3/2} \exp[-\frac{(v_\parallel - u_s)^2 + v_\perp^2/\alpha}{\theta}], \tag{6.1}$$

with $\theta = 2k_bT_s/m_sv_0^2$ where n_s is the number density of species s ($n_s = 10^{20}$ herein) and for the shifted case the shift $u_s = -1.5 \cdot sgn(s) \cdot m_0/m_s$ with $sgn(s) = -1$ for electrons and $sgn(s) = 1$ for ions, which results in zero net parallel momentum and a significant shift of the electrons.

The total temperature is defined as $T_s = (2 \cdot T_\perp + T_\parallel)/3$ and, in a finite element context, is computed with

$$T_s\left(v_\parallel, v_\perp\right) \equiv \frac{m_s v_0^2 \int dv f_s\left(v_\perp, v_\parallel\right) \cdot v_\perp \cdot \left(\left(v_\parallel - u_s\right)^2 + v_\perp^2\right)}{3 \int dv f_s\left(v_\perp, v_\parallel\right) \cdot v_\perp}.$$

These temperatures are evaluated in MKS units in the code and scale by $\sim 6.24e{\times}10^{18}$ to convert from Joules to electron volts (eV). Likewise the parallel and perpendicular temperatures are computed according to

$$T_{s,\perp}\left(v_\parallel, v_\perp\right) \equiv \frac{m_s v_0^2 \int dv f_s\left(v_\perp, v_\parallel\right) \cdot v_\perp \cdot v_\perp^2}{2 \int dv f_s\left(v_\perp, v_\parallel\right) \cdot v_\perp}$$

and

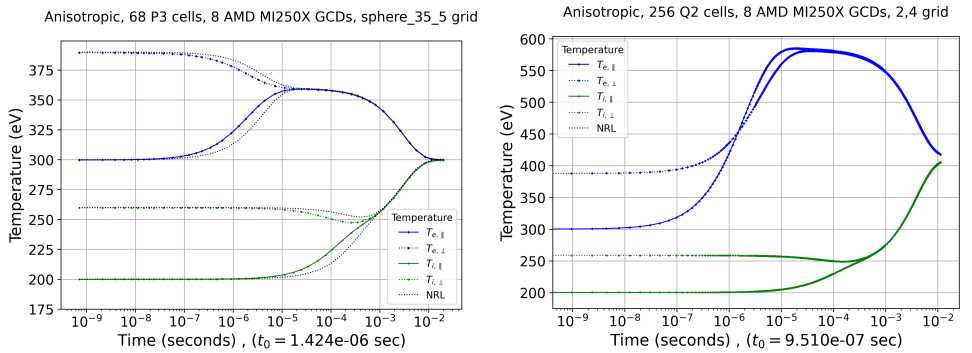
$$T_{s,\parallel}\left(v_\parallel, v_\perp\right) \equiv \frac{m_s v_0^2 \int dv f_s\left(v_\perp, v_\parallel\right) \cdot v_\perp \cdot \left(v_\parallel - u_s\right)^2}{\int dv f_s\left(v_\perp, v_\parallel\right) \cdot v_\perp}.$$

These definitions generate an electron parallel temperature $T_{e,\parallel} = 300$ eV, perpendicular temperature $T_{e,\perp} = 390$ eV, and ion parallel temperature $T_{i,\parallel} = 200$ eV,

perpendicular temperature $T_{i,\perp} = 260$ eV, as reported by Hager et al. [10], and as can be seen at $t = 0$ in the plots in Figure 5. Two cases of anisotropic initial conditions are investigated: 1) *non-shifted* $u_s = 0$ and 2) *shifted* $u_e = 1.5, u_i = 1.5(m_e/m_i)$, $m_e/m_i \approx 1/3,671$.

6.2. Comparison with analytical results. The NRL Plasma Formulary provides formulas for inter and intra-species thermalization rates (Appendix B) that generate analytical temperature histories for comparison with the computed results. The Plasma Formulary also provides formulas for Coulomb logarithms, which are used for both the computed and analytical temperature histories. All three relevant Coulomb logarithms are between 6.8 and 7.5 in this study.

Figure 5a shows the temperatures as a function of time and the electron-electron collision period t_0 in the non-shifted cases, with $P3$ elements. The discrepancy between the computed and NRL results is due to the fact that the NRL rates are based on Maxwellian distributions. While this test thermalizes to a Maxwellian distribution, the plasma is significantly non-Maxwellian during the evolution. Our results are in line with other published results [10, 27, 21].



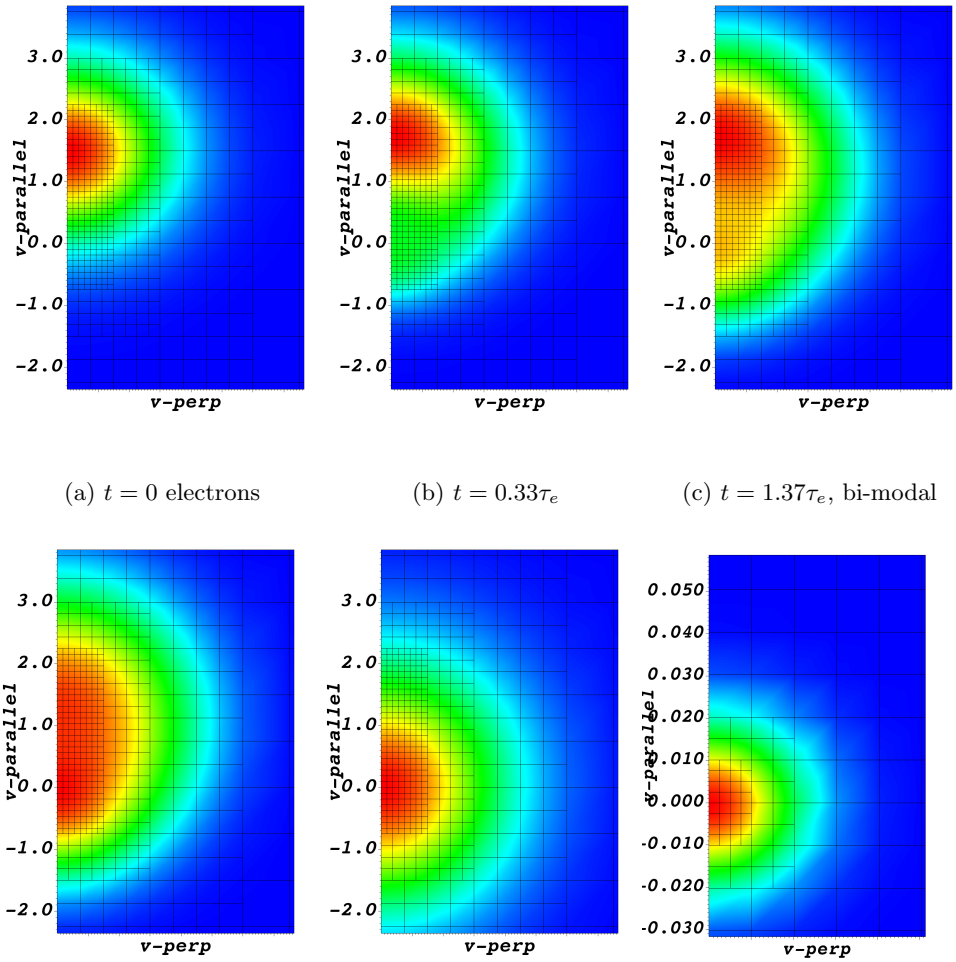
(a) $P3$, 34 cell per species semi-circular mesh (b) Shifted Bi-Maxwellian case

Fig. 5: Anisotropic relaxation test temperature vs. time of the $P3$ element case, plotted with an analytical NRL results for the non-shifted case (left), and the shifted Maxwellian case with a block-structured AMR grid and $Q2$ elements (right)

6.3. Qualitative observation of the shifted Maxwellian dynamics. Figure 5b shows a solution to the shifted Maxwellian case, illustrated in Figure 6, where the kinetic energy from the parallel flow is first transmitted to the parallel temperature of the electrons via collisions, followed by the isotropization of the electrons, and then the ion isotropization and the complete thermalization of the plasma. Note, the kinetic energy in the parallel flow or drift velocity is not used in the temperature calculation, resulting in an increase temperature as this case thermalizes.

The dynamics of the shifted Maxwellian thermalization is of interest in developing an intuitive understanding of the physics of collisions. Figure 6 shows a detail of the electron distribution function for several early times in the thermalization of the shifted problem on a highly refined mesh using block-structured mesh adaptivity. Figures 6a-6c show the initial thermalization of electrons, followed by the shift of the bulk of the electrons to eventual (near) thermalization in Figures 6d-6e. Figure 6f shows the initial ion distribution (note, electron and ion grids have different topology and scaling). The times of each figure refers to the temperature histories, normalized

with t_0, shown in Figure 5b.



(a) $t = 0$ electrons (b) $t = 0.33\tau_e$ (c) $t = 1.37\tau_e$, bi-modal

(d) $t = 2.49\tau_e$ electrons (e) $t = 23.1\tau_e$ (f) $t = 0$, ions

Fig. 6: Detail of the electron distributions, shifted Maxwellian deuterium plasma, ions near origin: (a) initial condition, (b) penumbra in shift and early Maxwellian population, (c) bi-modal distribution, (d) mid-thermalization, (e) near full thermalization, (e) initial condition of ions

7. Performance results for the anisotropic relaxation test. This section investigates the performance of the anisotropic relaxation test with the two single node configurations in §4, using the unstructured grids shown in Figure 3, with $P2 - P4$ triangle elements, and the quadrilateral meshes in Figure 4 with $Q2 - Q4$ elements. The domain radius is optimized by hand for the $P2$ grid and uses a domain size of 5.5, but otherwise a radius of 5.0 is used. The quadrilateral meshes are optimized to some extent, although $Q2$ elements are not served well with the current meshing.

A scalar error metric is desired to understand costs versus accuracy and to con-

strain the degree of mesh refinement. To this end, error is defined as the difference in any of the four measured temperatures from the correct value of 300 eV at the converged state: $error \equiv \max_{s\in[e,i],d\in[\parallel,\perp]}|T_{s,d} - 300|$, and is listed in Tables 11, 12 and 13. This is not a rigorous metric because the two species grids are nondimensionalized with the initial parallel thermal speed of the single species on the grid, which are different. While both species have the same thermal speed and (Maxwellian) distribution at equilibrium, different computational domains results in different discretization errors in the temperature integrals. We observe decent correlation of this metric and refinement, but these results should be verified with a qualitative assessment of the temperature histories compared to the NRL data (see Appendix C). We observe a decent correlation between this metric and the qualitative assessment of the temperature histories.

7.1. Solver stack. An adaptive three-stage Runge-Kutta time integrator with an initial time step of 0.001, in units of the electron-electron collision time τ_e, is used to capture the early dynamics accurately. The time step quickly increases to an imposed maximum value of 1.0 as the plasma reaches equilibrium. Each of the three stages uses a nonlinear solve with a relative residual tolerance of 10^{-14} and all three moments of interest, density, parallel momentum, and energy, are conserved to this scale in each time step. The total simulation ($T = 14,000\tau_e$ and $> 14,000$ time steps) conserves energy to about 12 digits.

7.2. Anisotropic relaxation test performance data. The AMD MI250X results use ROCm 5.2 and on the *Frontier* machine and the NVIDIA A100 results use the *Perlmutter* machine. Both CUDA and HIP allow for specifying launch bounds at compile time, defined as `<maxThreadsPerBlock, minBlocksPerMultiprocessor>` and `<maxThreadsPerBlock, minWarpsPerEU>` respectively. We found the default launch bounds of $< 1024, 1 >$ were not ideal, especially for the MI250X. We found that $< 256, 1 >$ was reasonable for the MI250X and that $< 256, 2 >$ was reasonable for the A100. A batch size of 128 on AMD with 8 MPI processes and 256 on NVIDIA with 4 MPI processes is used, resulting in 1,024 full anisotropic relaxation problems running simultaneously. These 1,024 problems are identical and mimic an application setting where multiple spatial points are processed simultaneously.

Three simplex cases are investigated, $P2, P3$ and $P4$ elements with the grids shown in Figure 3, as well as three quadrilateral cases, $Q2, Q3$ and $Q4$ elements with the grids in Figure 4. Tables 11, 12 and 13 list the number of cells, the number of integration points (IPs), the number of equations in the matrix for each problem (with two species, each linear system is one half this size), the average number of non-zeros per row, the order P or Q of the finite elements, the matrix construction and linear solver times, the total run time, and the percent error.

Table 11: Anisotropic thermalization timings (seconds) on 4 NVIDIA A100 GPUs

# cells	# IPs	# eqs	nnz/row	P	Jacobian	Solve	Total time	error (%)
144	864	318	10.8	2	4,930	424	5,424	0.67
68	816	350	15.5	3	4,785	495	5,352	0.18
24	384	226	20.8	4	1,376	544	2,009	0.84

This data shows that the A100 and MI250X performance is comparable with the solver running faster on the A100 and the matrix construction running faster on the

Table 12: Anisotropic thermalization timings (seconds) on 8 AMD MI250X GCDs

# cells	# IPs	# eqs	nnz/row	P	Jacobian	Solve	Total time	error (%)
144	864	318	10.8	2	3,948	660	4,746	0.67
68	816	350	15.5	3	3,510	826	4,421	0.18
24	384	226	20.8	4	959	943	1,982	0.84

Table 13: Anisotropic times (seconds) using tensor elements on the NVIDIA node

# cells	# IPs	# eqs	nnz/row	Q	Jacobian	Solve	Total time	error (%)
36	324	154	14.3	2	1,189	236	1,543	2.83
24	384	236	22.7	3	1,447	328	1,897	0.77
12	300	226	31.4	4	1,375	456	1,953	0.64

MI250X node. These times correlate with the accuracy with $P4$ being the fastest, by far, and $P3$ being the most accurate on the simplex grids, and the quadrilateral grids being a bit more efficient than the simplex grids with the exception of the $Q2$ case.

8. Hardware utilization. This section analyses the efficacy of the hardware utilization in the results in §4 and §7. The throughput performance study in §4 is analysed on the A100 with NVIDIA's Nsight Systems in §8.1 and the anisotropic relaxation test in §7 is analysed on the AMD MI250X with the TAU performance tools in §8.2 [22]. One of the four A100 GPUs and one of the eight MI205X GCDs are used with one MPI process on our nodes for analysis.

PETSc supports the use of these two architectures with both CUDA and HIP back-ends as well as a *Kokkos Kernels* back-end that supports both architectures. The *Kokkos Kernels* back-end, using the built-in *Kokkos* numerical kernels, is used for all performance results herein. The codes used for these tests are examples in PETSc. Instructions for reproducing these results, the raw data, and the scripts that generate the plots are publicly available (Appendix A).

The NVIDIA's Nsight Systems data in §4 shows that 97% of the total run time is spent on the GPU and 100% flops in the full collision time advance are executed on the device, after the grid setup phase. The linear solver and Jacobian construction kernels are written in *Kokkos*, the nonlinear solver and time integrator use the appropriate *Kokkos Kernels* sparse matrix a vector back-ends.

8.1. NVIDIA hardware utilization. The analysis of the hardware utilization in the A100 kernel is divided into the analysis of the Jacobian matrix and the mass matrix construction, and the batch solver in $2V$ and $3V$ on the experiments in §4. The NVIDIA Nsight Compute tool is used to gather several hardware metrics from the largest batch size in Tables 5 and 6. Table 14 presents some of the raw Nsight Compute data.

Some points can be seen in this data.

- The Jacobian kernel, with a high arithmetic intensity (AI) of 55.8 with respect to DRAM memory movement in $2V$, is not a simple loop of fused multiply add (FMA) instructions as can be seen from lines 4-6 with only 62% of the flops in FMA instructions. This limits the achievable percent of theoretical peak for this algorithm.
- The flop rate (line 7) is about $2x$ higher in $2V$ than $3V$. This is at least

Table 14: Nsight Compute data: Jacobian (Jac), Mass (M), Solver (Sol)

Data	Jac-2V	M-2V	Sol-2V	Jac-3V	M-3V	Sol-3V
DRAM (GB/s)	75.80	1230	28.18	38.33	946	538
L1 (TB/s)	1.92	3.58	1.43	1.92	2.39	1.59
L2 (GB/s)	747	4010	881	266	2810	1870
dadd/cycle	163	155	156	76.20	91.80	35.12
dfma/cycle	1155	0	168	546	0	36.11
dmul/cycle	526	329	64.50	305	198	3.14
TFlop/sec	4.23	0.68	0.89	2.06	0.41	0.16
AI-L1	2.20	0.19	0.50	1.07	0.17	0.10
Roofline-L1 %	43.60	18.27	9.18	21.27	12.19	8.11
AI-L2	5.66	0.17	0.72	7.75	0.15	0.08
Roofline-L2 %	43.60	54.20	16.70	21.27	38	25.30
AI-DRAM	55.80	0.56	23.60	53.80	0.43	0.29
R.F.-DRAM %	43.60	63.60	9.18	21.30	48.90	27.80

partially due to the Landau kernel **U** in (2.1) being more complex with a higher AI in $2V$, but this requires futher investigation.

- There are few flops and no FMAs in the mass matrix as this is essentially all assembly.
- The solver AI-DRAM is very high in $2V$ (23.6) and low in $3V$ (0.29). The theoretical AI of the solver (no cache) is about $\frac{1}{6}$. This data indicates that the solves are fitting in cache well in $2V$ but not at all in $3V$.

Tables 15 and 16 tabulate conclusions and notes from the Nsight Compute data.

Table 15: Nsight Compute Bottlenecks

Jacobian-2V	Mass-2V	Solve-2V	
FP64 pipe (57%)	L2 (70%), DRAM (64%)	L1 and instruction latency bound: L1 (43%) instruction issue (39%)	FP64 pipe (31%), L1 (24%)

Table 16: Nsight Compute Notes

Jacobian-2V	Mass-2V	Jacobian-3V
Roofline lower than FP64 pipe utilization b/c DFMA instruction is 62% of all FP64 instructions	low roofline peak b/c 1) low pipe utilization due to being L1 latency bound. 2) instruction dominated by branch and integers. FP64 instructions $\approx$ 10% of total instructions	Low pipe utilization due to L1 latency bound

8.2. AMD hardware utilization. The analysis of the hardware utilization of the AMD MI250X uses the anisotropic relaxation test in §7, which has two grids with one species each ($S = 1, 1$), as opposed to ten species and three grids ($S = 1, 1, 8$) in NVIDIA model problems. These two tests have different complexity profiles in the Jacobian kernel (Algorithm 1, [1]), with the the substantial work complexity and low memory complexity, in $2V$, of the Landau tensor ([12], Appendix A) being amortized

in multi-species per grid problem. The inner loop accesses data proportional to S, for the field and gradient field data, imposing more register pressure in the eight species per grid case, and the arithmetic intensity is nominally higher in the eight species case with reuse of this field data. The individual grids are slightly larger in the anisotropic relaxation test used for the AMD studies, resulting in lower batch size requirements to saturate the hardware. Both studies increase the batch size to observe the saturation of the hardware.

Table 17: Rocprof / TAU data: Jacobian (Jac), Mass (M), Solver (Sol).

Data	Jac-2V	M-2V	Sol-2V
Workgroups	256	256	256
LDS usage	512	16896	512
Scratch usage	248	176	0
Vector registers used	120	56	48
Scalar registers used	120	120	112
Effective LDS BW GB/s	744	764	1638
Effective vL1D BW GB/s	3814	4640	2761
Effective L2 BW GB/s	216	722	393
Effective HBM BW GB/s	107	622	188
AI-vL1D	< 0.005	< 0.005	< 0.005
AI-L2	0.03	< 0.005	< 0.005
AI-HBM	0.07	< 0.005	< 0.005
TFLOP/s	3.85	0.50	0.40
L2 Cache Hit %	77.07	57.22	74.32
vL1D Cache Hit %	94.33	84.83	85.77
Vector Inst. %	74.63	52.96	55.56
Scalar Inst. %	14.82	22.61	18.49
Branch Inst. %	2.05	4.34	7.28
Vector Memory Inst. %	4.33	10.83	4.15
LDS Inst. %	1.22	2.20	5.15
Avg. Dependency Wait Cycles %	55.22	72.8	81.86
Avg. Issue Wait Cycles %	10.60	14.66	1.77
Avg. Active Issue Cycles %	37.18	12.54	16.37
Avg. Active Wavefronts	58.07	55.78	55.82

The AMD nodes have eight GCDs, each with 110 Compute Units that are each similar to the 108 Streaming Multiprocessors on NVIDIA A100. As with the A100, the analysis of the hardware utilization in the MI250X kernel is divided into the analysis of the Jacobian matrix and the mass matrix construction. The TAU Performance System and Rocprof tools are used to gather several hardware metrics from the $P4$ test in Table 12 and a batch size of 64. Table 17 presents some of the raw Rocprof data, post-processed with a Python script written with guidance from AMD engineers for interpreting the metrics.

Because the set of available AMD metrics differ from that provided by NVIDIA, different points can be seen in this data, but it is still a useful comparison.

- Both as an absolute value and relative to the A100 results, the Jacobian kernel has low arithmetic intensity (AI-HBM, computed as total FLOP / effective bandwidth) of 0.07 with respect to DRAM memory movement in $2V$. The kernel is apparently suffering from the cost of the relatively smaller L1 cache of the MI250X, relatively low L2 hit rate, and register spilling to scratch despite the application of launch bounds. This limits the achievable

performance for this algorithm.

- Despite the lower algorithmic intensity, the flop rate (TFLOP/s) for all three kernels is still comparable to the performance on the A100, although slightly lower.
- The solver AI-DRAM is low in $2V$ (< 0.005) relative to the A100 data. This indicates that the solves are not fitting in cache in $2V$. The number of nonzeros in each of the linear systems is similar in this $P4$ unstructured test and the $Q3$ semi-structured grids of the NVIDIA test data.
- All three kernels are suffering from relatively low L2 cache hit rate, while the Mass and Solver kernels are suffering from a relatively low L1 cache rate.

9. Conclusion. This report concludes a series of papers on a grid-based structure preserving Landau collision operator with advanced numerical methods and a performance portable implementation in the PETSc numerical library [12, 2, 1]. This Landau solver supports multiple independent grids to efficiently resolve the domain of each species group, with multiple species per grid for species with like velocity profiles, and high-order accurate finite element discretizations with static, fully unstructured and block-structured, adaptive mesh refinement. A new batch solver has been introduced and experiments with a well optimized code on an NVIDIA A100 and an AMD MI250X node is presented. A new anisotropic relaxation test is presented that shows good agreement with analytical models and other published results.

The entire implicit time advance, after an initial setup phase, is written in the Kokkos programming language, and good hardware utilization is demonstrate, especially given the relative complexity of the kernel. We observe 57% FP64 pipe utilization (i.e., theoretical peak flop rate) on the NVIDIA A100 in the main computational kernel, the $2V$ Jacobian matrix construction, and comparable overall performance on the AMD MI250X. Many expressions in the Landau kernel have three or four multiply operands resulting in only 62% of the flops being in fused multiple add instructions on the A100. Thus, the 57% theoretical peak is about 70% of the theoretical peak for this algorithm on this hardware.

Both the Landau time advance and batched linear solvers are publicly available in the PETSc (Portable Extensible Toolkit for Scientific Computing) numerical library, and the two performance test codes are examples in PETSc (Appendix A). This grid-based method complements new full $3V$ particle-based Landau operators currently under development [27, 11, 21], and this grid-based method can be used with PIC codes with new conservative translation operators between particle and grid representations of distribution functions [21]. Future work involves understanding the entropy generation of this translation method, which is a type of *coarse-graining* used to add numerical entropy [7, 8, 25].

Acknowledgments. This work was supported in part by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Scientific Discovery through Advanced Computing (SciDAC) program through the FASTMath Institute under Contract No. DE-AC02-05CH11231 at Lawrence Berkeley National Laboratory.

Appendix A. Artifact description and reproducibility.

PETSc output files with all data, provenance information, and reproducibility instructions for all tables and plots can be obtained from `git@gitlab.com:markadams4/batch_paper_data.git`. This includes the python scrips that generates the plots and run scripts, makefiles and PETSc resource files used to generate the data. The

`landau_throughput_nsight` directory has throughput and NVIDIA hardware utilization data, and `landau_anisotropic_shift` has verification test data and AMD hardware utilization data. The exact PETSc versions (SHA1) are in the data files, with the provenance data, and any PETSc version from v3.19 should suffice to reproduce this data. The two driver codes are in `src/ts/utils/dmplexlandau/tutorials`, ex1.c (anisotropic temperature relaxation §7) and ex2.c (throughput studies in §4).

Appendix B. NRL Plasma Formulary isotropization rates.

The NRL Plasma Formulary provides relaxation evolution equations for inter and intra-species thermalization [13], according to

$$\frac{dT_\perp}{dt} = -\frac{1}{2}\frac{dT_\parallel}{dt} = -v_T^\alpha (T_\perp - T_\parallel),$$

where if $A \equiv T_\perp/T_\parallel - 1 > 0$,

$$v_T^\alpha = \frac{2\sqrt{\pi}e^2 n_\alpha \ln\Lambda_{\alpha\alpha}}{m_\alpha^{1/2}(kT_\parallel)^{3/2}} A^{-2}\left[-3 + (A+3)\,\frac{tan^{-1}\left(A^{1/2}\right)}{A^{1/2}}\right].$$

The two species thermal equilibrium evolution equation is

$$\frac{dT_\alpha}{dt} = \bar{v}\,(T_\alpha - T_\beta)\,,$$

and

$$\bar{v} = 1.8\times 10^{-19}\frac{\sqrt{m_e m_i}n_0 \ln\Lambda_{ei}}{(m_e T_i + m_i T_e)^{3/2}} sec^{-1}.$$

Appendix C. Additional anisotropic relaxation test data.

Figure 7 shows anisotropic relaxation test data from the non-shifted Maxwellian case with $P2$ and $P4$ elements, where inaccuracies are visible. Figure 8 shows aniso-

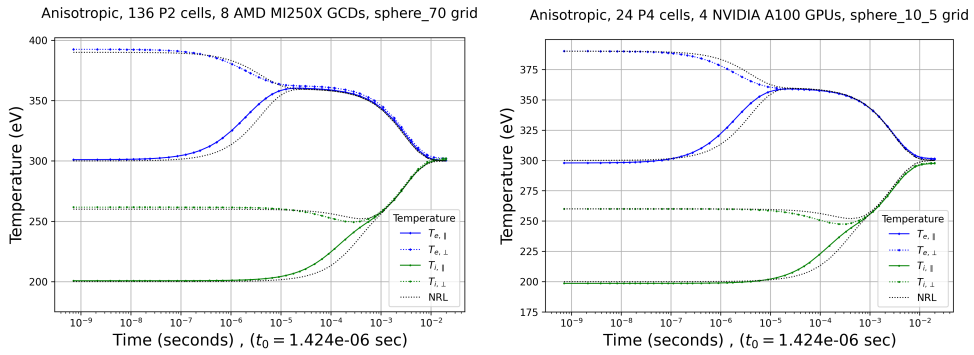


Fig. 7: Anisotropic relaxation test data from the $P2$ element case (left) and the $P4$ elements (right), plotted with analytical NRL model data

tropic relaxation test data on the semicircle quadrilateral mesh where the poor quality of the $Q2$ mesh is clearly visible.

REFERENCES

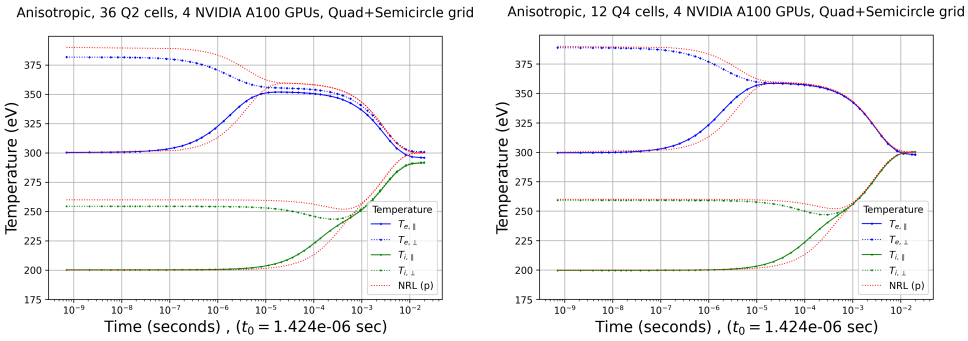


Fig. 8: Temperature vs. time with unstructured quadrilateral meshes and $Q2$ element case (left) and the $Q4$ elements (right), plotted with analytical NRL model data

[1] M. F. Adams, D. P. Brennan, M. G. Knepley, and P. Wang, *Landau collision operator in the CUDA programming model applied to thermal quench plasmas*, in 2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS), 2022, pp. 115–123, https://doi.org/10.1109/IPDPS53621.2022.00020.

[2] M. F. Adams, E. Hirvijoki, M. G. Knepley, J. Brown, T. Isaac, and R. T. Mills, *Landau collision integral solver with adaptive mesh refinement on emerging architectures*, SIAM Journal on Scientific Computing, 39 (2017), pp. C452–C465, https://doi.org/10.1137/17M1118828, http://epubs.siam.org/doi/abs/10.1137/17M1118828, https://arxiv.org/abs/1702.08880.

[3] I. Aggarwal, A. Kashi, P. Nayak, C. J. Balos, C. S. Woodward, and H. Anzt, *Batched sparse iterative solvers for computational chemistry simulations on GPUs*, in 2021 12th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (ScalA), 2021, pp. 35–43, https://doi.org/10.1109/ScalA54577.2021.00010.

[4] S. T. Beliaev and G. I. Budker, *The Relativistic Kinetic Equation*, Soviet Physics Doklady, 1 (1956), pp. 218–222.

[5] W. H. Boukaram, G. Turkiyyah, H. Ltaief, and D. E. Keyes, *Batched QR and SVD algorithms on GPUs with applications in hierarchical matrix compression*, Parallel Computing, 74 (2018), pp. 19–33, https://doi.org/https://doi.org/10.1016/j.parco.2017.09.001, https://www.sciencedirect.com/science/article/pii/S0167819117301461. Parallel Matrix Algorithms and Applications (PMAA'16).

[6] B. J. Braams and C. F. F. Karney, *Differential form of the collision integral for a relativistic plasma*, Physical Review Letters, 59 (1987), pp. 1817–1820, https://doi.org/10.1103/PhysRevLett.59.1817, https://arxiv.org/abs/physics/0501067.

[7] S. Brunner, E. Valeo, and J. A. Krommes, *Collisional delta-f scheme with evolving background for transport time scale simulations*, Physics of Plasmas, 6 (1999), pp. 4504–4521.

[8] Y. Chen and S. E. Parker, *Coarse-graining phase space in δf particle-in-cell simulations*, Physics of Plasmas, 14 (2007), p. 082301, https://doi.org/10.1063/1.2751603, https://doi.org/10.1063/1.2751603, https://arxiv.org/abs/https://doi.org/10.1063/1.2751603.

[9] H. C. Edwards, C. R. Trott, and D. Sunderland, *Kokkos: Enabling manycore performance portability through polymorphic memory access patterns*, Journal of Parallel and Distributed Computing, 74 (2014), pp. 3202 – 3216, https://doi.org/https://doi.org/10.1016/j.jpdc.2014.07.003, http://www.sciencedirect.com/science/article/pii/S0743731514001257. Domain-Specific Languages and High-Level Frameworks for High-Performance Computing.

[10] R. Hager, E. Yoon, S.-H. Ku, E. F. D'Azevedo, P. H. Worley, and C.-S. Chang, *A fully non-linear multi-species Fokker–Planck–Landau collision operator for simulation of fusion plasma*, Journal of Computational Physics, 315 (2016), pp. 644–660, https://doi.org/10.1016/j.jcp.2016.03.064, http://dx.doi.org/10.1016/j.jcp.2016.03.064.

[11] E. Hirvijoki, *Structure-preserving marker-particle discretizations of Coulomb collisions for particle-in-cell codes*, Plasma Physics and Controlled Fusion, 63 (2021), p. 044003, https://doi.org/10.1088/1361-6587/abe884, https://doi.org/10.1088/1361-6587/abe884.

[12] E. Hirvijoki and M. F. Adams, *Conservative discretization of the Landau collision integral*, Physics of Plasmas, 24 (2017), p. 032121, https://doi.org/10.1063/1.4979122, http://dx.doi.org/10.1063/1.4979122, https://arxiv.org/abs/http://dx.doi.org/10.1063/1.4979122.

[13] J. D. Huba, *NRL PLASMA FORMULARY Supported by The Office of Naval Research*, Naval Research Laboratory, Washington, DC, 2013, http://wwwppd.nrl.navy.mil/nrlformulary.

[14] A. Kashi, P. Nayak, D. Kulkarni, A. Scheinberg, P. Lin, and H. Anzt, *Batched sparse iterative solvers on gpu for the collision operator for fusion plasma simulations*, in 2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS), 2022, pp. 157–167, https://doi.org/10.1109/IPDPS53621.2022.00024.

[15] M. Kraus and E. Hirvijoki, *Metriplectic integrators for the Landau collision operator*, Physics of Plasmas, 24 (2017), https://doi.org/10.1063/1.4998610.

[16] L. D. Landau, *Kinetic equation for the Coulomb effect*, Phys. Z. Sowjetunion, 10 (1936), p. 154.

[17] K. Liegeois, R. Boman, E. T. Phipps, T. A. Wiesner, and M. Arnst, *GMRES with embedded ensemble propagation for the efficient solution of parametric linear systems in uncertainty quantification of computational models*, Computer Methods in Applied Mechanics and Engineering, 369 (2020), p. 113188, https://doi.org/https://doi.org/10.1016/j.cma.2020.113188, https://www.sciencedirect.com/science/article/pii/S004578252030373X.

[18] K. Liegeois, S. Rajamanickam, and L. Berger-Vergiat, *Performance portable batched sparse linear solvers*, IEEE Transactions on Parallel and Distributed Systems, 34 (2023), pp. 1524–1535, https://doi.org/10.1109/TPDS.2023.3249110.

[19] J. Merson and M. S. Shephard, *Using hierarchical parallelism to accelerate the solution of many small partial differential equations*, 2023, https://arxiv.org/abs/2305.07030.

[20] R. T. Mills, M. Adams, S. Balay, J. Brown, J. Faibussowitsch, T. Isaac, M. Knepley, T. Munson, H. Suh, S. Zampini, H. Zhang, and J. Zhang, *Petsc/tao developments for early exascale systems*, 2024, https://arxiv.org/abs/2406.08646.

[21] J. V. Pusztay, M. G. Knepley, and M. F. Adams, *Conservative projection between finite element and particle bases*, SIAM Journal on Scientific Computing, 44 (2022), pp. C310–C319, https://doi.org/10.1137/21M1454079, https://doi.org/10.1137/21M1454079, https://arxiv.org/abs/https://doi.org/10.1137/21M1454079.

[22] S. S. Shende and A. D. Malony, *The tau parallel performance system*, Int. J. High Perform. Comput. Appl., 20 (2006), p. 287–311, https://doi.org/10.1177/1094342006064482, https://doi.org/10.1177/1094342006064482.

[23] T. Shiroto and Y. Sentoku, *Structure-preserving strategy for conservative simulation of the relativistic nonlinear landau-fokker-planck equation*, Phys. Rev. E, 99 (2019), p. 053309, https://doi.org/10.1103/PhysRevE.99.053309, https://link.aps.org/doi/10.1103/PhysRevE.99.053309.

[24] *Simulation modeling suite*, 2023, http://www.simmetrix.com/index.php/simulation-modeling-suite (accessed 2023-05-05).

[25] T. Vernay, S. Brunner, L. Villard, B. F. McMillan, S. Jolliet, T. M. Tran, A. Bottino, and J. P. Graves, *Neoclassical equilibria as starting point for global gyrokinetic microturbulence simulations*, Physics of Plasmas, 17 (2010), p. 122301, https://doi.org/10.1063/1.3519513, https://doi.org/10.1063/1.3519513, https://arxiv.org/abs/https://doi.org/10.1063/1.3519513.

[26] A. A. Vlasov, *The vibrational properties of an electron gas*, Soviet Physics Uspekhi, 10 (1968), pp. 721–733, https://doi.org/10.1070/pu1968v010n06abeh003709, http://dx.doi.org/10.1070/PU1968v010n06ABEH003709.

[27] F. Zonta, J. V. Pusztay, and E. Hirvijoki, *Multispecies structure-preserving particle discretization of the Landau collision operator*, Physics of Plasmas, 29 (2022), p. 123906, https://doi.org/10.1063/5.0105182, https://doi.org/10.1063/5.0105182, https://arxiv.org/abs/https://doi.org/10.1063/5.0105182.