SIAM J. SCI. COMPUT.
Vol. 39, No. 6, pp. C452–C465

LANDAU COLLISION INTEGRAL SOLVER WITH ADAPTIVE MESH REFINEMENT ON EMERGING ARCHITECTURES*

MARK F. ADAMS†, EERO HIRVIJOKI‡, MATTHEW G. KNEPLEY§, JED BROWN¶, TOBIN ISAAC‖, AND RICHARD MILLS#

Abstract. The Landau collision integral is an accurate model for the small-angle dominated Coulomb collisions in fusion plasmas. We investigate a high order accurate, fully conservative, finite element discretization of the nonlinear multispecies Landau integral with adaptive mesh refinement using the PETSc library (www.mcs.anl.gov/petsc). We develop algorithms and techniques to efficiently utilize emerging architectures with an approach that minimizes memory usage and movement and is suitable for vector processing. The Landau collision integral is vectorized with Intel AVX-512 intrinsics and the solver sustains as much as 22% of the theoretical peak flop rate of the Second Generation Intel Xeon Phi ("Knights Landing") processor.

Key words. Landau collision integral, fusion plasma physics

AMS subject classification. 82D10

DOI. 10.1137/17M1118828

1. Introduction.

The simulation of magnetized plasmas is of commercial and scientific interest and is integral to the fusion energy research program of the U.S. Department of Energy (DOE) [1, 2, 3]. Although fluid models are widely employed to model fusion plasmas, the weak collisionality and highly non-Maxwellian velocity distributions in such plasmas motivate the use of kinetic models, such as the so-called *Vlasov–Maxwell–Landau* system. The evolution of the phase-space density or distribution function f of each species (electrons and multiple species of ions in general) is modeled with

$$\frac{df}{dt} \equiv \frac{\partial f}{\partial t} + \frac{\partial \boldsymbol{x}}{\partial t} \cdot \nabla_x f + \frac{\partial \boldsymbol{v}}{\partial t} \cdot \nabla_v f = \frac{\partial f}{\partial t} + \boldsymbol{v} \cdot \nabla_x f + \frac{e}{m} (\boldsymbol{E} + \boldsymbol{v} \times \boldsymbol{B}) \cdot \nabla_v f = C,$$

where e is charge, m mass, $\boldsymbol{E}$ electric field, $\boldsymbol{B}$ magnetic field, $\boldsymbol{x}$ spatial coordinate, $\boldsymbol{v}$ velocity coordinate, and t time. The Vlasov operator d/dt describes the streaming of particles influenced by electromagnetic forces, the Maxwell's equations provide the electromagnetic fields, and the Landau collision integral [4], C, produces entropy and embodies the transition from many-body dynamics to single particle statistics. As

*Submitted to the journal's Software and High-Performance Computing section February 28, 2017; accepted for publication (in revised form) October 12, 2017; published electronically December 6, 2017.

http://www.siam.org/journals/sisc/39-6/M111882.html

Funding: This work was partially funded from the Intel Parallel Computing Center program and from the DOE contract DE-AC02-09-CH11466. This research used resources of the National Energy Research Scientific Computing Center, a DOE Office of Science User Facility supported by the Office of Science of the U.S. Department of Energy under contract DE-AC02-05CH11231.

†Scalable Solvers Group, Lawrence Berkeley Laboratory, Berkeley, CA 94720 (mfadams@lbl.gov).

‡Princeton Plasma Physics Laboratory, Princeton, NJ 08540 (ehirvijo@pppl.gov).

§Computational and Applied Mathematics, Rice University, Houston, TX 77005 (knepley@gmail.com).

¶Department of Computer Science, University of Colorado, Boulder, Boulder, CO 80309 (jed@jedbrown.org).

‖School of Computational Science and Engineering, Georgia Institute of Technology, Atlanta, GA 30332 (tisaac@cc.gatech.edu).

#Intel Corporation.

such, the Vlasov–Maxwell–Landau system of equations is the gold standard for high-fidelity fusion plasma simulations.

The Vlasov–Maxwell–Landau system also conserves energy and momentum, and guaranteeing these properties in numerical simulations is of paramount importance to avoid plasma self heating and false momentum transfer during long-time simulations. Hirvijoki and Adams recently developed a finite element discretization of the Landau integral, which is able to preserve the conservation properties of the Landau collision integral with sufficient order accurate finite element spaces [5]. We now continue this work with the development of a multispecies Landau solver with adaptive mesh refinement (AMR), which is designed for emerging architectures and implemented on the Second Generation Intel Xeon Phi, ("Knights Landing", or KNL) processor.

Due to the nonlinearity of the Landau collision integral, it has an intensive work complexity of $\mathcal{O}(N^2)$ with N global integration or quadrature points. Given this high-order work complexity, reducing the total number of quadrature points decreases computational cost substantially. We use high-order accurate finite elements and AMR to maximize the information content of each quadrature point and thus minimize the solver cost. We adapt nonconforming tensor product meshes using the *p4est* library [6, 7, 8], as a third party library in the PETSc library [9, 10].

We develop algorithms and techniques for optimizing the Landau solver on emerging architectures, with emphasis on KNL, and verify the order of accuracy on a model problem. We vectorize the kernel using Intel AVX-512 intrinsics and achieve a flop rate as high as 22% of the theoretical peak floating point rate of KNL. This is an important step towards facilitating near-future gold-standard full Vlasov–Maxwell–Landau simulations where multiple Landau solvers must be run in parallel, each solver addressing one of the spatial-configuration mesh nodes on which the fields $\boldsymbol{E}$ and $\boldsymbol{B}$ are evolved.

2. Conservative finite element discretization of the Landau integral. We consider the multispecies version of the conservative finite element discretization of the Landau collision integral presented by Hirvijoki and Adams [5]. Under small-angle dominated Coulomb collision, the distribution function $f_\alpha(\boldsymbol{v}, t)$ of species α evolves according to

$$\frac{\partial f_\alpha}{\partial t} = \sum_\beta \nu_{\alpha\beta} \frac{m_o}{m_\alpha} \nabla_v \cdot \int_{\bar{\Omega}} d\bar{\boldsymbol{v}}\, \mathbf{U}(\boldsymbol{v}, \bar{\boldsymbol{v}}) \cdot \left(\frac{m_o}{m_\alpha} \bar{f}_\beta \nabla_v f_\alpha - \frac{m_o}{m_\beta} f_\alpha \bar{\nabla}_{\bar{v}} \bar{f}_\beta \right). \tag{1}$$

Here $\nu_{\alpha\beta} = e_\alpha^2 e_\beta^2 \ln \Lambda_{\alpha\beta} / (8\pi m_o^2 \varepsilon_0^2)$, $\ln \Lambda$ is the Coulomb logarithm, m_o is an arbitrary reference mass, ε_0 is the vacuum permittivity, m is mass, e is electric charge, and $\boldsymbol{v}$ is the velocity. Overbar terms are evaluated on the $\bar{\boldsymbol{v}}$ grid that covers the domain $\bar{\Omega}$ of species β. The Landau tensor $\mathbf{U}(\boldsymbol{v}, \bar{\boldsymbol{v}})$ is a scaled projection matrix defined as

$$\mathbf{U}(\boldsymbol{v}, \bar{\boldsymbol{v}}) = \frac{1}{|\boldsymbol{v} - \bar{\boldsymbol{v}}|^3} \left(|\boldsymbol{v} - \bar{\boldsymbol{v}}|^2 \mathbf{I} - (\boldsymbol{v} - \bar{\boldsymbol{v}})(\boldsymbol{v} - \bar{\boldsymbol{v}}) \right) \tag{2}$$

and has an eigenvector $\boldsymbol{v} - \bar{\boldsymbol{v}}$ corresponding to a zero eigenvalue.

Given a test function $\psi(\boldsymbol{v})$, the weak form of the Landau operator (1) for species α is given by

$$\left(\psi, \frac{\partial f_\alpha}{\partial t} \right)_\Omega = \sum_\beta (\psi, f_\alpha)_{\boldsymbol{K},\alpha\beta} + (\psi, f_\alpha)_{\mathbf{D},\alpha\beta} \tag{3}$$

where $(\cdot,\cdot)_\Omega$ is the standard L^2 inner product in Ω and the weighted inner products present the advective and diffusive parts of the Landau collision integral

$$(\psi,\phi)_{\boldsymbol{K},\alpha\beta} = \int_\Omega d\boldsymbol{v}\nabla_v\psi\cdot\hat{\nu}_{\alpha\beta}\frac{m_o}{m_\alpha}\frac{m_o}{m_\beta}\boldsymbol{K}(f_\beta,\boldsymbol{v})\,\phi, \tag{4}$$

$$(\psi,\phi)_{\mathbf{D},\alpha\beta} = -\int_\Omega d\boldsymbol{v}\nabla_v\psi\cdot\hat{\nu}_{\alpha\beta}\frac{m_o}{m_\alpha}\frac{m_o}{m_\alpha}\mathbf{D}(f_\beta,\boldsymbol{v})\cdot\nabla_v\phi. \tag{5}$$

The collision frequency is normalized with $\hat{\nu}_{\alpha\beta} = \nu_{\alpha\beta}/\nu_o$ so that time t is dimensionless, and f_β is the distribution function of species β. The vector $\boldsymbol{K}$ and the tensor $\mathbf{D}$ are defined as

$$\boldsymbol{K}(f,\boldsymbol{v}) = \int_{\bar{\Omega}} d\bar{\boldsymbol{v}}\ \mathbf{U}(\boldsymbol{v},\bar{\boldsymbol{v}})\cdot\bar{\nabla}_{\bar{v}}f(\bar{\boldsymbol{v}}), \tag{6}$$

$$\mathbf{D}(f,\boldsymbol{v}) = \int_{\bar{\Omega}} d\bar{\boldsymbol{v}}\ \mathbf{U}(\boldsymbol{v},\bar{\boldsymbol{v}})f(\bar{\boldsymbol{v}}). \tag{7}$$

Assuming a finite-dimensional vector space V_h that is spanned by the set of functions $\{\psi_i\}_i$, the finite-dimensional approximation of the weak form (3) can be written in a matrix form

$$\mathbf{M}\dot{\boldsymbol{f}}_\alpha = \mathbf{C}_\alpha[\boldsymbol{f}]\boldsymbol{f}_\alpha, \tag{8}$$

where $\boldsymbol{f}_\alpha$ is the vector containing the projection coefficients of f_α onto V_h and the vector $\boldsymbol{f}$ is the collection of all species $\boldsymbol{f}_\alpha$, The mass and collision matrices are defined

$$\mathbf{M}_{ij} = (\psi_i,\psi_j)_\Omega, \quad \mathbf{C}_{\alpha,ij}[\boldsymbol{f}] = \sum_{\beta=1}^{S}(\psi_i,\psi_j)_{\boldsymbol{K},\alpha\beta} + (\psi_i,\psi_j)_{\mathbf{D},\alpha\beta}\,. \tag{9}$$

The integrals in (6), (7), with the Landau tensors in the kernel, have $\mathcal{O}(N)$ work for each species β and each equation in (3). With $\mathcal{O}(N)$ equation this leads to an $\mathcal{O}(N^2)$ algorithm for computing a Jacobian or residual when solving (8) for each species.

We would like to note that while the direct discretization of the Landau integral has a complexity of $\mathcal{O}(N^2)$, the collision operator can be formulated as a coupled set of pure partial differential equations with a complexity of $\mathcal{O}(N)$ [11, 12]. The pure PDE formulations, however, have to introduce artificial numerical fudge-factors if strict conservation properties are desired while in the direct Landau approach the properties are guaranteed by construction [5].

3. Algorithm design for emerging architectures. This section discusses the algorithms and techniques used to effectively utilize emerging architectures for a Landau integral solver. While the Landau operator has $\mathcal{O}(N^2)$ work complexity, this work is amenable to vector processing. We focus on KNL, but the algorithm is designed to minimize data movement and simplify access patterns, which is beneficial for any emerging architecture.

The discrete Landau Jacobian matrix construction, or residual calculation, can be written as six nested loops. Algorithm 1 shows high level pseudocode for construction the Landau Jacobian matrix, with $|G|$ cells in the set G, Nq quadrature points in each element, distribution functions f, S species, and weights $w_{q_j} = |J(q_j)|\cdot q_j.weight\cdot q_j.r$, where $q_j.r$ is the axisymmetric term of the element Jacobian, $q_j.weight$ is the quadrature weight of q_j, and $J(q_j)$ is the element Jacobian at point q_j.

Algorithm 1. Simple algorithm to compute Landau Jacobian $\mathbf{C}$ with state f.

for all cells $i \in G$ **do**
  for all quadrature points $q_i \in i$ **do**
    for $\alpha = 1 : S$ **do**
      for all cells $j \in G$ **do**
        for all quadrature points $q_j \in j$ **do**
          for $\beta = 1 : S$ **do**
            $\mathbf{U} \leftarrow \mathbf{LandauTensor}\left(q_i.r, q_i.z, q_j.r, q_j.z\right)$
            $\mathbf{K} \leftarrow \hat{\nu}_{\alpha\beta} \frac{m_o}{m_\alpha} \frac{m_o}{m_\beta} \mathbf{U} \cdot \nabla f_\beta\left(q_j\right) w_{q_j}$
            $\mathbf{D} \leftarrow -\hat{\nu}_{\alpha\beta} \frac{m_o}{m_\alpha} \frac{m_o}{m_\alpha} \mathbf{U} f_\beta\left(q_j\right) w_{q_j}$
            $\mathbf{C} \leftarrow FiniteElementAssemble\left(\mathbf{C}, w_{q_i}, \mathbf{K}, \mathbf{D}\right)$
          end for
        end for
      end for
    end for
  end for
end for

The Landau tensor $\mathbf{U}$ in (6), (7) is computed, or read from memory, in the inner loop. A vector $\boldsymbol{K} = \mathbf{U} \cdot \nabla f_{q_j} w_{q_j}$ and a tensor $\mathbf{D} = \mathbf{U} f_{q_j} w_{q_j}$ are accumulated in the inner loop. With S species, the accumulation of $\boldsymbol{K}$ and $\mathbf{D}$ requires $6S$ words. These accumulated values are transformed in a standard finite element process from the reference to the real element geometry and assembled with finite element shape functions into the element matrix. The six loops of Algorithm 1 can be processed in any order, and blocked, giving different data access patterns, which is critical in optimizing performance.

The first two issues that we address in the design of the Landau solver are (1) whether to precompute the Landau tensors or compute them as needed and (2) whether to use a single mesh with multiple degrees of freedom per vertex or use a separate mesh for each species.

3.1. To precompute the Landau tensor or not to precompute. The Landau tensor is only a function of mesh geometry and can be computed and stored for each mesh configuration. The cost of computing the Landau tensor is amortized by the number of nonlinear solver iterations and the number of time steps that the mesh is used, and can be ignored if it is precomputed and stored. In the axisymmetric case there are two Landau tensors and they require approximately 165 floating point operations (flops) as measured by both the Intel Software Development Emulator (SDE) and code analysis, including four logarithms and square roots (see Appendix [5]). Storing these two tensors requires eight words of storage, or 64 bytes with double precision words. There are $\mathcal{O}(N^2)$ unique mesh (i, j) pairs for which the tensors are computed or stored. The decision to precompute or compute as needed depends on several factors.

A simple analysis on KNL, for instance, suggests that both approaches are viable. Assuming the equivalent of 200 ordinary flops per axisymmetric Landau tensor pair calculation and 64 bytes of data, the flop to byte ratio is about three. The 68 core Intel Xeon Phi 7250 version of the KNL processor has a theoretical peak floating point capacity of about 2.6×10^{12} flops/second and around 400×10^9 bytes/second on-package memory bandwidth capacity, as measured by STREAMS, or a flop to byte ratio of

about six. This simple analysis, assuming peak STREAMs, ignoring caches and assuming peak flop rates, suggests that the precomputing approach would be two times slower. We achieve about 20% of theoretical peak flop rate and, thus, a precomputed implementation would need to achieve about 40% of STREAMS bandwidth to match the run time of each kernel evaluation, which is feasible. Additionally, the stored approach could process blocks of (i, j) pairs to utilize the cache and thus increase the available memory bandwidth. This simple analysis suggests that either approach is viable on KNL, but the trends in hardware are increasing the gap between flop and memory movement capacity, in the form of more vector lanes and more hardware resources per lane, which benefits the compute as needed approach.

An optimized stored tensor implementation would allow for experimental comparison of these two approaches, which would be interesting and provide a potentially useful option, but this is the subject of future work. See Hager et al., for discussion of a stored tensor approach [13].

3.2. Single and multiple meshes. We use a single mesh, adapted for all S species, with S degrees of freedom per vertex. One can, however, use multiple meshes or a mesh for each species. Observe that the integrals in (6), (7) are decoupled from the outer integral in (4), (5). In theory, one can use a separate grid, or different quadrature, or even a different discretization for each species. An advantage of using a single mesh is that the two loops over species in (9) can be processed after the Landau tensors are computed, and hence these tensors can be reused S^2 times. However, if all of the species have "orthogonal" optimal meshes, that is each quadrature point only has significant information for one species, which is a good assumption for ions and electrons because of their disparate velocities, then a single mesh requires about as many vertices as the sum of each of the putative multiple meshes. Thus, with this simple model, the size of the single mesh is S times larger than each single mesh and is S^2 more expensive because the algorithm has $\mathcal{O}(N^2)$ complexity. This S^2 increase in cost cancels the saving from hoisting the tensor computation out of the inner species loops. With this model of orthogonal optimal meshes and kernel dominated computation or communication, and with N_α quadrature points for each species α, the complexity of a Landau solve is $\mathcal{O}\big(\sum_{\alpha=1}^{S} N_\alpha\big)^2$ for both the single and multiple mesh approach.

The "orthogonal" mesh assumption is less valid with respect to multiple ion species, because ions have similar velocities and hence the optimal meshes for each species are similar. The Jacobian matrix for the single mesh approach has about S times more nonzeros and the same fill pattern and so the solver cost is linear in S with both approaches. The single mesh method has larger accumulation register demands and larger element matrices, which places more pressure on the memory system. The result of the increased register pressure can be seen in the decreased flop rates in Table 2 with the increase in the number of species. Single and multiple mesh approaches are both viable; we have chosen a single mesh but the method is valid for multiple meshes and we may pursue this option as future work. Additionally, others have used multiple meshes successfully for Landau integrals [11, 13].

3.3. Our algorithm. For demonstration purposes, we focus on implementing the axially symmetric version using cylindrical velocity coordinates $\boldsymbol{x} = (r, \theta, z)$. Under axial symmetry the distribution function is independent of the angular velocity coordinate ($\partial_\theta f = 0$) and the evaluation of the vector $\boldsymbol{K}$ and the tensor $\mathbf{D}$ requires two different Landau tensors $\mathbf{U_K}$ and $\mathbf{U_D}$, respectively (see Appendix in [5]). We choose to compute the required Landau tensors as needed and use a single mesh with

a degree-of-freedom for each species on each vertex. We fuse the two inner loops over cells and quadrature points, inline the function call of the Landau tensor function. Algorithm 2 shows the initialization of the vectors r, z, w, f, and the two gradient vectors $df\,[1]$ and $df\,[2]$, with $|G|$ cells in the set G, S species, and weights w_{q_i} at each quadrature point i. Each quadrature point q_i is located at a 2D coordinate $(q_i.r, q_i.z)$.

Algorithm 2. Initialization of vectors r, z, w, f, and df with state f.

```
for all cells i ∈ G do
  for all quadrature points q_i ∈ i do
    r.append(q_i.r)
    z.append(q_i.z)
    w.append(w_{q_i})
    for α = 1 : S do
      f[α].append(f_α(q_i))
      df[1][α].append(∇f_α(q_i)[1])
      df[2][α].append(∇f_α(q_i)[2])
    end for
  end for
end for
```

Algorithm 3 shows the algorithm for the construction of the Landau collision integral Jacobian. This algorithm is designed to minimized data movement by computing the Landau tensors as needed and exploits a single mesh by hoisting the tensor kernel outside of the two inner loops over species.

4. Numerical methods and implementation. We implement the Landau solver with the PETSc numerical library [9, 10]. PETSc provides finite element (FE) and finite volume discretization support, mesh management, interfaces to several third party mesh generators, fast multigrid solvers, interfaces to third party direct solvers, and AMR capabilities, among other numerical methods. We adapt nonconforming tensor product meshes using the third party *p4est* library [6, 7, 8], and unstructured conforming simplex meshes with PETSc's native AMR capabilities [14]. Our experiments use biquadratic (Q2) elements with *p4est* adaptivity, with the PETSc's Plex mesh management framework.

The velocity is normalized according to $\boldsymbol{v} = \boldsymbol{x}L$ where $\boldsymbol{x}$ is dimensionless and L is chosen freely and can present, e.g., a multiple of thermal velocity. For us, the computational domain is chosen to be $\Omega = \{(r, z) \mid 0 \le r \le L, -L \le z \le L\}$. We use Neumann boundary conditions and shifted Maxwellian initial distribution functions, for each species, of the form

$$f_\alpha(\boldsymbol{x}, t = 0) = \frac{1}{2}\left(\pi\sigma_\alpha^2\right)^{-3/2} \exp\left(-\frac{r^2 + (z - s_\alpha)^2}{\sigma_\alpha^2}\right),$$

where $\sigma_\alpha^2 = 2T_\alpha/(m_\alpha L^2)$, $s_i = 0$, $s_e = -1$, T_α is temperature.

We solve the boundary value problem

$$\frac{\partial f_\alpha}{\partial t}(\mathbf{v}, t) - \mathbf{C}_\alpha[f]f = 0$$

in axisymmetric coordinates, with standard FE methods and time integrators. A Newton nonlinear solver with the SuperLU direct linear solver is used at each time stage or step [15]. These experiments use a Crank–Nicolson time integrator.

Algorithm 3. Algorithm to compute $\mathbf{C}$ with r, z, w, f, and df from Algorithm 2.

for all cells $i \in G$ **do**
  $\mathbf{ElemMat} \leftarrow 0$
  for all quadrature points $q_i \in i$ **do**
    $\mathbf{K} \leftarrow 0$
    $\mathbf{D} \leftarrow 0$
    $w_i \leftarrow q_i.weight \cdot |J(q_i)| \cdot q_i.r$
    $N \leftarrow Nq \cdot |G|$
    for $n = 1 : N$ **do** // Vectorized loop
      $[\mathbf{U_K}, \mathbf{U_D}] \leftarrow \mathbf{LandauTensor}(q_i.r, q_i.z, r[n], z[n])$
      for $\alpha = 1 : S$ **do**
        for $\beta = 1 : S$ **do**
          $\mathbf{K}[\alpha] \leftarrow \mathbf{K}[\alpha] + \hat{\nu}_{\alpha\beta} \frac{m_o}{m_\alpha} \frac{m_o}{m_\beta} \mathbf{U_K} \cdot df[:][\beta][n] w[n]$
          $\mathbf{D}[\alpha] \leftarrow \mathbf{D}[\alpha] - \hat{\nu}_{\alpha\beta} \frac{m_o}{m_\alpha} \frac{m_o}{m_\alpha} \mathbf{U_D} f[\beta][n] w[n]$
        end for
      end for
    end for
    for $\alpha = 1 : S$ **do**
      $\mathbf{G2}[\alpha] \leftarrow \mathbf{J}(q_i)^{-1} \mathbf{K}[\alpha] w_i$ // transform point integral to global space
      $\mathbf{G3}[\alpha] \leftarrow \mathbf{J}(q_i)^{-1} \mathbf{D}[\alpha] \mathbf{J}(q_i)^{-1} w_i$
    end for
    // Project point value to vertices of cell i
    $\mathbf{ElemMat} \leftarrow Transform\&Assemble(\mathbf{ElemMat}, \mathbf{G2}, \mathbf{G3}, \mathbf{B}(q_i))$
  end for
  // Sum element matrix into global Jacobian
  $\mathbf{C} \leftarrow GlobalAssemble(\mathbf{C}, i, \mathbf{ElemMat})$
end for

A global kinetic model would include a 3D spatial component and the 3V version of this solver would be used at each cell in either a particle method [13], or a grid based kinetic method [16]. Our numerical experiments use up to 272 Message Passing Interface (MPI) processes on one KNL socket, redundantly solving the problem, to include memory contention that one would see in a 5D method. The timing experiments run one time step with one Newton iteration, which results in the Landau operator being called twice (one more than the number of Newton iterations), and with one linear solve (one per Newton iteration). The time for this step is reported, which does not include the AMR mesh construction (see section 4.6 for a discussion of AMR). We observe a variability in times with 272 processes: we have run each test several times in several sessions, in both batch and interactive modes, and we report the fastest observed time. This reported time is the maximum time for all processes; we see about a 10% ratio between the maximum and the minimum time of any process with large process counts.

4.1. Overview of test problem. To illustrate the capabilities and behavior of the solver, we run the code to near equilibrium, initializing electrons with a shifted Maxwellian distribution hitting a stationary single proton ion population with a Maxwellian distribution. Figure 1 (left) shows the initial electron distribution with the ion grid at the origin, a partially thermalized electron distribution (center, left), and Maxwellian ion distribution near equilibrium (center, right). The ion distribution

has been shifted from the origin by collision with the electrons. The ions are resolved with AMR at the origin and have a near Maxwellian distribution. Note, the visualization in Figure 1 uses linear interpolation from the three corners of two triangles created for each quadrilateral, whereas the numerics use biquadratic interpolation with nine vertices per quadrilateral, which results in distored visualization.

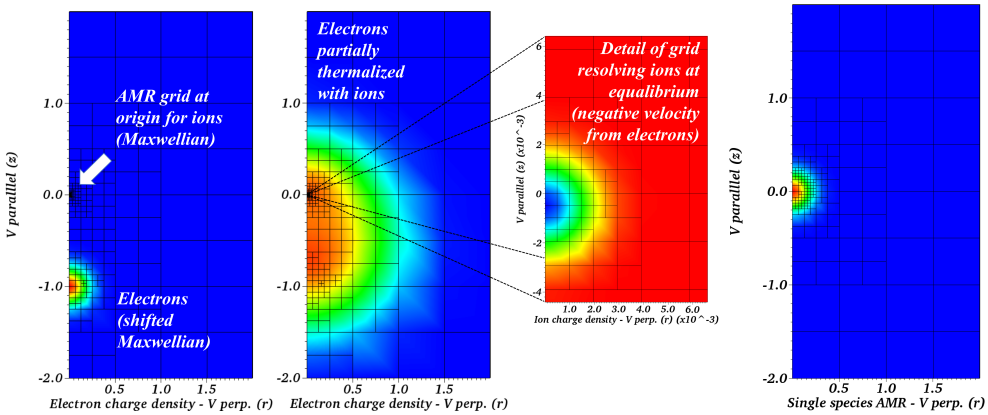


FIG. 1. *Charge density with initial Maxwellian distribution functions relaxing towards equilibrium, initial electron distribution (left), partially thermalized electrons (center, left), detail of the ion distribution (center, right), electron only adapted mesh (right).*

4.2. Optimization and performance. Most of the work in the Landau solver is in the inner integral of (4,5) (lines 8–16 in Algorithm 3). This kernel is vectorized with Intel AVX-512 intrinsics. The Landau tensors calculation includes two logarithms, a square root, a power, seven divides, about 85 multiplies and 165 total flops. The power is converted to an inverse square root, an intermediate divide is reused, resulting in five divides, two logarithms, a square root, and an inverse square root. The KNL sockets used for this study are equipped with 34 tiles, each with a 1 megabyte shared L2 cache and two cores; each core has two vector processor units (VPUs) with 8 double-precision SIMD lanes and can issue one fused multiply add (FMA) per cycle per VPU. Each core has four hardware threads, for a total of 272 threads per socket. The nominal KNL clock rate is 1.4 GHz, but is clocked down to 1.2 GHz in sustained AVX-512 code segments. This results in a theoretical flop peak rate of 2.6×10^{12} flops/second. The peak flop rate that can be achieved with this solver is reduced because the kernel is not entirely composed of FMAs and the four logarithms and square roots require considerably more than one cycle each.

We note that our early implementations of this kernel did not make use of AVX-512 intrinsics; instead, we relied on the Intel compiler to transform loops into vectorized code. Guided by the compiler optimization reports, we made several code modifications and appropriate use of pragmas, after which the compiler was able to generate vectorized executables that achieved over 90% of the performance of the AVX-512 intrinsics-based version of the code. We ultimately decided to use the AVX-512 intrinsics approach because of the slightly better performance offered, and because we found that relying on the compiler vectorizer requires code changes and maintenance that may not be obvious.

4.3. Performance overview. The performance data in this section uses a simplified version of the test problem: a grid with 176 cells and 1,584 quadrature points,

a mass ratio of $\frac{m_i}{m_e} = 1$ and $T_e = T_i = 0.2$ keV, and no Maxwellian shifts ($s_i = s_e = 0$) as shown in Figure 1 (right). All experiments herein set $\ln \Lambda_{\alpha\beta} = 10$ and $L = 2$.

The major code segments have been instrumented with PETSc timers. Table 1 shows the maximum time from any process for major components of the Landau operator, the total Landau operator, and the linear solver. This data shows that the Landau kernel, though vectorized, is still responsible for most of the run time.

TABLE 1
Major component times (maximum of any process) from one time step with two species, double precision, and 272 processes.

Component (times called)	Time (maximum)	% of total
Landau initial vector data setup, Algorithm 2	0.019	2
Landau kernel with AVX512 intrinsics	0.533	66
Landau FE transforms & assemble	0.030	4
Landau FE global matrix assembly	0.072	9
Landau operator total (2)	0.682	85
Linear solver (1)	0.12	15
Total time step time (1)	0.803	100

4.4. Performance and complexity analysis. There are two types of work in the kernel: (1) computing the two Landau tensors and (2) the accumulation of the $\boldsymbol{K}$ vector and $\mathbf{D}$ tensor. The accumulation requires $20S^2$ flops (lines 12–13 in Algorithm 3). Instrumenting this inner loop would be invasive, but we can infer the percentage of time and work in these two parts with a complexity model and global measurements. Assume both the time and work cost of the entire solver are of the form $C = aS^0 + bS^1 + cS^2$. The solve times and flop counts with $S = 1, 2, 3$, shown in Table 2, generate right-hand sides for a system of three equations and three unknowns a, b, and c, which are the time spent, or work, in each of the three types of components. The Landau tensor cost is formally independent of the number of species, and the work in the accumulation has S^2 work complexity. Most of the rest of the costs, given a mesh, order of elements, etc., has $\mathcal{O}(S^1)$ complexity. Table 3 shows the percentage of time and work in each component, inferred from the one process data in Table 2. This analysis with the 272 process data results in invalid percentages, presumably due to performance variations between cores. This analysis shows that, in the case of one process and two species, about 97% of the work, and about 82% of the time, is in the kernel. The measurements of the kernel time in Table 1 is 66% with 272 processes. This discrepancy is probably due to performance noise and memory contention in the 272 process timings, as well as inaccuracy of this model. The nonkernel time percentage (18) increases by a factor of about eight from the flop percentage (2.2) for the two species case and is similar for one and three species cases. This factor of eight is expected because the KNL vector unit has eight vector lanes.

TABLE 2
Time (seconds) with 1 and 272 processes on one KNL socket; flop counts from Intel SDE, and flop rates.

# Species	1 proc.	272 proc.	Gflops 1 proc.	Gflops/sec. (% of peak)
1	0.21	0.47	1.01	572 (22)
2	0.28	0.79	1.34	455 (18)
3	0.38	1.38	1.88	370 (14)

TABLE 3
Percentage of flops (F) and time (T) in species independent work, work linear in S, and work quadratic in S, with double precision and one process.

Work type (# of species)	F (1)	T (1)	F (2)	T (2)	F (3)	T (3)
S^0 (Landau tensors)	88	81	66	61	47	45
S^1 (nonkernel work)	1.5	12	2.2	18	2.4	20
S^2 (accumulation)	10.5	7	31	21	50	36

4.5. Memory performance. We perform a kind of *weak* speedup study, where the same serial problem is replicated in each process as we scale up to the full 272 hardware threads on one KNL socket. Unlike a typical weak speedup study we do not suffer from interprocess communication, but we do see increased run time from memory contention as we increase the number of processes per tile. Given that the number of flops per process is constant, this increase in run time translates to a decrease in flop rate. Table 4 shows timing data with increasing number of processes on a single KNL socket, with single and double precision. KNL's architecture allows for twice as many vector lanes with single (32 bit) versus double (64 bit) precision and can run, in theory, twice as fast in single precision. This data shows that we are achieving about 80% of the perfect factor of two speedup with single precision.

TABLE 4
Weak speedup time (seconds) with single and double precision.

Processes per socket	272	136	68	34	1
Processes per core (*with idle cores)	4	2	1	1*	1*
Processes per tile (*with idle tiles)	8	4	2	1	1*
Single	0.51	0.29	0.18	0.17	0.17
Double	0.80	0.46	0.30	0.28	0.28

Recall from section 4.2 that the KNL processors we use have 34 tiles, 68 cores (2 per tile), 272 hardware threads (4 per core), and 136 vector units (2 per core). One might expect that using more processes than 136 processes would not be useful because there are 136 vector units; however, the kernel has serial dependencies that result in bubbles in the pipeline, especially in the ninth and tenth order polynomial evaluations in the elliptic integrals. These holes can be filled by interleaving a second process in another hardware thread. We do see about a 15% increase in total throughput from the added parallelism of using all 272 hardware threads.

This data shows that the flop rate per process decreases by a factor of about three in going from one process per tile to eight processes per tile (time increase by a factor of three). There is no difference in per process flop rate between the one process and 34 process runs, suggesting that the 34 processes are indeed placed with one per tile and there is no contention at the level of main memory. There is little degradation going from one to two processes per tile, suggesting that the problem still fits in the L2 cache. The degradation from one to two processes per core suggests there is contention in the L1 cache. The fact that the 272 process solve time does not exactly double suggests that the two hardware threads per vector unit is allowing for some instruction interleaving.

4.5.1. Precision and accuracy. We investigate the effects of single versus double precision on the accuracy of the solver by considering the accuracy in the energy with the test problem in section 4.7, with an 8x16 cell version of the problem. Floating point error, with any given precision, will cause the norm of the nonlinear residual to

stagnate at a relative reduction of a little less than the number of digits in the precision. We found that with single precision we could use a relative residual tolerance in the nonlinear solver of $rtol = 10^{-5}$ and for double precision we use $rtol = 10^{-12}$ (for the Q9 test we had to reduce this to $rtol = 10^{-11}$). Table 5 shows the energy after one time step of the test problem, with the digits highlighted that do not change in subsequent order elements. This data indicates that we can get about twice as many digits with double precision, as one would expect, and that we can effectively use up to about Q8 elements for accuracy of the total energy with double precision.

TABLE 5

Single precision (SP) energy (left) and double precision (DP) energy (right), for Tensor element polynomial order Qx, **Boldface digits** *do not change is subsequent order test,* $^*rtol = 10^{-11}$.

Tensor Element type (order)	Energy (SP)	Energy (DP)
Q2	**5.7**588756084442e-02	**5.7**588695494153e-02
Q3	**5.760**9990239143e-02	**5.76**10025387605e-02
Q4	**5.7607**710361481e-02	**5.7607**449991296e-02
Q5	5.7607315480709e-02	**5.760743**7129653e-02
Q6		**5.7607439**155331e-02
Q7		**5.760743913**8728e-02
Q8		**5.760743913**8879e-02
Q9*		5.7607439139733e-02

4.6. Dynamic mesh adaptivity. Our dynamic mesh adaptivity is a hierarchical quadtree-based method as implemented by the *p4est* library [17]. *p4est* implements a forest-of-quadtree data structure that maps quadtrees onto arbitrary base meshes made of conforming quadrilateral, and partitions the leaves of the quadtrees in parallel, although our numerical tests are serial, according to a space filling curve. *p4est* provides data structures and algorithms for the typical refinement cycle used in this work: refining and coarsening cells (while obeying a 2:1 condition between neighboring cells) and efficiently constructing the adjacency information needed for finite element methods from the bare list of quadtree leaves [7]. The library is designed to be both efficient and highly scalable.

In our code, *p4est* is not directly referenced: rather, it is used as a backend to the mesh interface in PETSc, as recently described in [18].

4.6.1. Mesh adaptivity numerical experiment of equilibrium test problem. We consider a problem of the form and domain shown in Figure 2, a shifted Maxwellian electron population colliding with a stationary Maxwellian population of ions and evolving to equilibrium. The ion refinement is not visible in Figure 2, but it resembles that of Figure 1 (center, right). We use a realistic mass ratio of $\frac{m_i}{m_e} = 1836.5$, and $T_e = 2$ keV and $T_i = 1$ keV. Figure 2 shows the mesh adaptivity of the initial state and the electron distribution. Note, this visualization uses linear interpolation resulting in an image that is much rougher than the actual data, but this provides a qualitative view of the test. The ion distribution at the origin is not visible at this scale as in Figure 1 (right). All test problems use electrons (e) and ions (i) with equal and opposite charge, $\ln \Lambda_{\alpha\beta} = 10$, $m_o = m_e$, and $\nu_o = \nu_{00}$.

We use a time step of 10^{-9} for 900 time steps, at which point the energies, which start with the ions having 44.8 times more energy than the electrons, to equilibrate to within 13% of each other. This test was run on one Intel Xeon Processor E5-2698 v3 ("Haswell") socket (2.3 GHz nominal frequency), with 64 hardware threads and AVX2 support (that is, with 256 bit vector registers). We run 64 MPI processes all running the same (serial) problem to mimic the memory contention of a spatial code

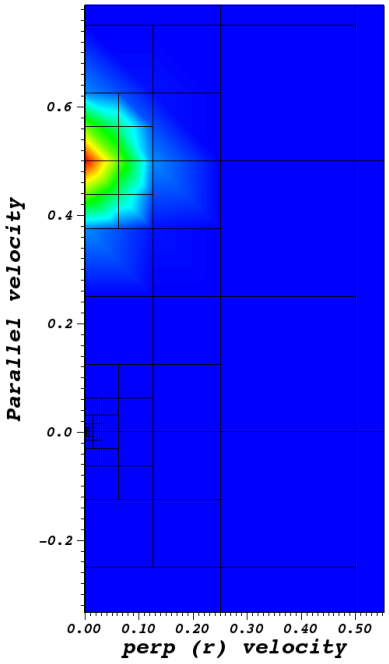


FIG. 2. *Detail of initial electron distribution of full analysis test, with mesh adaptivity.*

where the Landau solver is run at each grid point. This problem ran in 413 seconds and 2.8 of these seconds was spent in the adaptivity routines. Thus the time of the processing adaptivity, which includes computing an error mesh at each cell after each time step, is small.

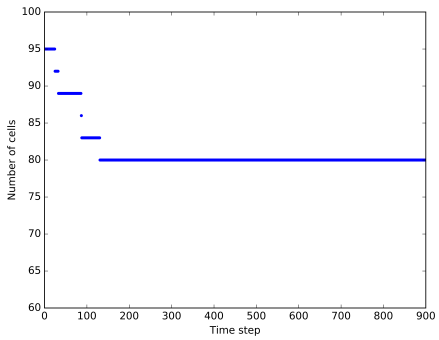
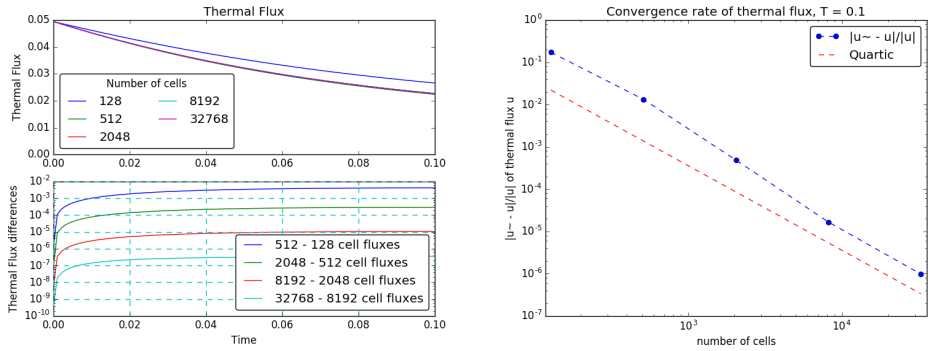


FIG. 3. *History of number of cells in equilibrium adaptivity study.*

This problems starts with 95 cells (Figure 2) and after 900 time steps has coarsened to 80 cells. Because the electrons move to the origin with a slight shift to conserve parallel momentum, the adaptivity around the initial electron cloud is completely coarsened and what is essentially the ion mesh is all that remains at equilibrium. Figure 3 shows the history of the number of cells.

4.7. Verification of order of accuracy. We verify the expected order of accuracy with a convergence study using the third moment, thermal flux. We do not have an analytical flux for this problem and use Richardson extrapolation to construct an approximate exact flux. The mass ratio is 4, $T_e = 0.2$ keV and $T_i = 0.02$ keV, and Cartesian grids are used. The flux history, with a series of refined grids starting with a 8×16 grid, is shown in Figure 4 (top, left). Figure 4 also shows the differences between fluxes on successive grids, and the error convergence.



(a) Thermal flux (top), and flux differences

(b) Error versus cell count

FIG. 4. (a) *Therm flux over time (top, left), flux differences in grid sequence (bottom, left), and quartic convergence rate (right).*

We can see from this data that we achieve fourth order convergence.

5. Closure. We have implemented a high-order accurate finite element implementation of the Landau collision integral with adaptive mesh refinement in the PETSc library using AVX-512 intrinsics for the Second Generation Intel Xeon Phi ("Knights Landing") processor. We have developed a memory-centric algorithm for emerging architectures that is amenable to vector processing. We have achieved up to 22% of the theoretical peak flop rate of KNL and analyzed the performance characteristics of the algorithm with respect to process memory contention, single and double precision, and the results of vectorization. We have verified fourth order accuracy with a biquadratic, Q2, finite element discretization. Future work includes building models for runaway electrons in tokamak plasmas with this kernel [19, 20, 21, 22], and building up complete kinetic models (6D AMR) that also preserve the geometric structure of the governing equations of fusion plasmas [23]. The repository for this work is publicly available [24].

Acknowledgments. We are grateful for discussions and insights from Intel engineer Vamsi Sripathi. This work has benefited from many discussions with Sam Williams.

REFERENCES

[1] *Exascale Requirements Review for Fusion Energy Sciences*, U.S. Department of Enereg, Washington, D.C., 2016; available online from http://exascaleage.org/wp-content/uploads/sites/67/2017/06/DOE-ExascaleReport-FES-Final.pdf.

[2] P. BONOLI AND L. CURFMAN MCINNES, *Report of the workshop on integrated simulations for magnetic fusion energy sciences*, 2015.

[3] C. Greenfield and R. Nazikian, *Report on scientific challenges and research opportunities in transient research*, 2015.

[4] L.D. Landau, *Kinetic equation for the Coulomb effect*, Phys. Z. Sowjetunion, 10 (1936), pp. 154–164.

[5] E. Hirvijoki and M.F. Adams, *Conservative discretization of the Landau collision integral*, Phys. Plasmas, 24 (2017), 032121.

[6] G. Stadler, M. Gurnis, C. Burstedde, L.C. Wilcox, L. Alisic, and O. Ghattas, *The dynamics of plate tectonics and mantle flow: From local to global scales*, Science, 329 (2010), pp. 1033–1038.

[7] T. Isaac, C. Burstedde, L.C. Wilcox, and O. Ghattas, *Recursive algorithms for distributed forests of octrees*, SIAM J. Sci. Comput., 37 (2015), pp. C497–C531, https://doi.org/10.1137/140970963.

[8] J. Rudi, A. Cristiano I. Malossi, T. Isaac, G. Stadler, M. Gurnis, P.W.J. Staar, Y. Ineichen, C. Bekas, A. Curioni, and O. Ghattas, *An extreme-scale implicit solver for complex PDEs: Highly heterogeneous flow in Earth's mantle*, in Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC '15, ACM, New York, 2015, pp. 5:1–5:12, a winner of the Gordon Bell Prize for special achievement.

[9] S. Balay, S. Abhyankar, M.F. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, V. Eijkhout, W.D. Gropp, D. Kaushik, M.G. Knepley, L. Curfman McInnes, K. Rupp, B.F. Smith, S. Zampini, H. Zhang, and H. Zhang, *PETSc Web page*, http://www.mcs.anl.gov/petsc, 2016.

[10] S. Balay, S. Abhyankar, M.F. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, V. Eijkhout, W.D. Gropp, D. Kaushik, M.G. Knepley, L. Curfman McInnes, K. Rupp, B.F. Smith, S. Zampini, H. Zhang, and H. Zhang, *PETSc Users Manual*, Technical Report ANL-95/11 - Revision 3.7, Argonne National Laboratory, 2016.

[11] W.T. Taitano, L. Chacón, A.N. Simakov, and K. Molvig, *A mass, momentum, and energy conserving, fully implicit, scalable algorithm for the multi-dimensional, multi-species Rosenbluth-Fokker-Planck equation*, J. Comput. Phys., 297 (2015), pp. 357–380.

[12] A. Pataki and L. Greengard, *Fast elliptic solvers in cylindrical coordinates and the Coulomb collision operator*, J. Comput. Phys., 230 (2011), pp. 7840–7852.

[13] R. Hager, E.S. Yoon, S. 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*, J. Comput. Phys., 315 (2016), pp. 644–660.

[14] N. Barral, M.G. Knepley, M. Lange, M.D. Piggott, and G.J. Gorman, *Anisotropic mesh adaptation in Firedrake with PETSc DMPlex*, in 25th International Meshing Roundtable, S. Owen and H. Si, eds., Washington, D.C., 2016, pp. 1–5.

[15] X.S. Li, *An overview of SuperLU: Algorithms, implementation, and user interface*, ACM Trans. Math. Software, 31 (2005), pp. 302–325.

[16] E.A. Belli and J. Candy, *Gyrokinetics with advanced collision operators*, in APS Meeting Abstracts, abstract NP8.026, 2014.

[17] C. Burstedde, L.C. Wilcox, and O. Ghattas, `p4est`*: Scalable algorithms for parallel adaptive mesh refinement on forests of octrees*, SIAM J. Sci. Comput., 33 (2011), pp. 1103–1133, https://doi.org/10.1137/100791634.

[18] T. Isaac and M.G. Knepley, *Support for non-conformal meshes in PETSc's DMPlex interface*, ACM Trans. Math. Software, submitted.

[19] C. Liu, D.P. Brennan, A. Bhattacharjee, and A.H. Boozer, *Adjoint Fokker-Planck equation and runaway electron dynamics*, Phys. Plasmas, 23 (2016), 010702.

[20] J. Decker, E. Hirvijoki, O. Embreus, Y. Peysson, A. Stahl, I. Pusztai, and T. Fulop, *Numerical characterization of bump formation in the runaway electron tail*, Plasma Physics and Controlled Fusion, 58 (2016), 025016.

[21] E. Nilsson, J. Decker, Y. Peysson, R.S. Granetz, F. Saint-Laurent, and M. Vlainic, *Kinetic modeling of runaway electron avalanches in tokamak plasmas*, Plasma Physics and Controlled Fusion, 57 (2015), 095006.

[22] A.H. Boozer, *Runaway electrons and magnetic island confinement*, Phys. Plasmas, 23 (2016), 082514.

[23] P.J. Morrison, *Structure and Structure-Preserving Algorithms for Plasma Physics*, preprint, https://arxiv.org/abs/1612.06734, 2016.

[24] M.F. Adams, *Landau*, https://bitbucket.org/madams/landaufem, 2017.