Portable solvers for batches of small systems applied to the Landau collision operator

1st Mark F. Adams  
Lawrence Berkeley National Laboratory  
mfadams@lbl.gov

3rd Kim Liegeois  
Sandia National Laboratories  
knliege@sandia.gov

Abstract—Many small independent sparse linear system solves occur in many applications, such as the Landau collision operator in plasma physics and astrophysics simulations, chemistry in combustion applications, and subdomain solves in domain decomposition solvers. One can simply stack these systems into a global linear system and use existing general-purpose sparse solvers. However, this so-called “ensemble reduction” approach does not exploit the independent structure of these systems and the theoretical optimality of a Krylov solver is lost.

The many independent processing elements (PEs) found in contemporary (GPU) accelerators are well suited to solving each of these systems independently. This “batch” approach maintains the Krylov subspace optimality, significantly reduces the number of kernel launches, and elides (unnecessary) global communication. This study develops portable solvers that run an entire linear system solve on a PE in a single kernel launch within the PETSc (Portable Extensible Toolkit for Scientific Computing) numerical library, with both build-in versions written in Kokkos and as a front-end to versions in Kokkos Kernels.

Index Terms—Batch solvers, Landau collision operator, Kokkos, GPU

I. INTRODUCTION

A solve phase with many independent sparse linear systems arises in several applications. Single level domain decomposition solvers or multigrid smoothers [1]–[3], and multiphysics models with a tensor product-like structure of a PDE on a spatial grid with, for example, a chemistry PDE in combustion [4], or with a velocity space Landau collision operator in plasma physics and astrophysics [5]–[11], all generate many independent system solves. Sensitivity analyses run independent simulations with solves for implicit time integrators [12]. In addition to these PDE based solves, batched solves appear in a pseudo-inverse at each grid point used in conservative mapping between particle and finite element bases representations in particle-in-cell methods [13], [14]. The related problem of batched singular value decompositions has also been developed [15].

Modern accelerator hardware is well suited to these many small systems because GPUs are composed of many small independent processing elements (PEs) that are equipped with fast synchronization primitives, by definition. While one can simply combine these linear systems into a single large linear systems and solve with existing sparse solvers, this ensemble reduction approach is not optimal in several respects. For Krylov solvers, the convergence of the Krylov method applied on the ensemble system is deteriorated compared to the convergence of the worst system of the batch. This suggests the use of batch solvers that place the entire solution process for each system on a PE in a single kernel. Batch Krylov solvers allow for the correct algorithm to be used for each system, with the proper scaling of the Krylov vectors, and independent convergence checking for each system. In addition to supporting the correct algorithm, batch solvers drastically reduce the number of kernel launches in the solver from a dozen or more per iteration to a single kernel launch. Batch solvers also avoid unnecessary global communication in dot products and norms and use only the fast local synchronization primitives on devices. The Ginkgo project is developing similar methods [16].

Performance portability is currently a significant challenge in high-performance computing with accelerator devices (GPUs). One viable approach is to abandon a single source model and implement a version of your solver for each device of interest. PETSc uses this approach for portable linear algebra. Alternatively, one can use a single source model with a portable language like Kokkos, Raja or SYCL [17]–[19]. This work discusses batched Jacobi preconditioned Krylov solvers in PETSc written in Kokkos [17], and similar Kokkos Kernels solvers incorporated in PETSc as third party solver [12], [20].

This report begins with an introduction to the Landau collision operator in §II, and builds on previous work [6], with the following new material:

• a portable, batch TFQMR1 solver in §III
• new 2V and 3V performance data for the build-in PETSc batch TFQMR with the Landau collision operator in §IV
• and performance results from the Kokkos Kernels batched GMRES2 solver §V
• comparison of batch Krylov solvers with the existing ensemble approach in PETSc.

and §VI concludes the report.

II. LANDAU COLLISIONS FOR MAGNETIZED PLASMAS

The Vlasov-Maxwell-Landau system of equations is the fundamental model of magnetized plasmas [21], [22]. It evolves a distribution function for each species (one electron

1transpose-free quasi-minimum residuals, a Krylov method for asymmetric matrices

2generalized minimum residuals, Krylov method for asymmetric matrices
and potentially many ions species) in phase space with up to three configuration space dimensions plus three velocity space dimensions (6D). The Landau operator conserves density, momentum and energy and admits unstructured finite element discretizations that strictly conserve these quantities [22], [23].

The evolution of the phase space distribution of a density function \( f (\vec{x}, \vec{v}, t) \) of a plasma in an electromagnetic field is effectively modeled with a Vlasov-Maxwell-Landau system of the form

\[
\frac{df}{dt} = \frac{\partial f}{\partial t} + \vec{\partial} \cdot \nabla_x f + \vec{\partial} \cdot \nabla_v f
\]

\[
= \frac{df}{dt} + \vec{v} \cdot \nabla_x f + \frac{e}{m} \left( \vec{E} + \vec{v} \times \vec{B} \right) \cdot \nabla_v f = C
\]

with charge \( e \), mass \( m \), electric field \( \vec{E} \), magnetic field \( \vec{B} \), spatial coordinate \( \vec{x} \), velocity coordinate \( \vec{v} \) and a collision term \( C \). This equation is composed of the symplectic Vlasov-Maxwell system \( \frac{df}{dt} = 0 \) and a metric, or diffusive, collision operator \( C \), within a metriplectic formalism [24].

Landau collisions between species \( \alpha \) and \( \beta \), are given by

\[
C_{\alpha\beta} = \nu_{\alpha\beta} \frac{m_\alpha v}{m_\alpha} \int d\Omega U(\vec{u}, \vec{v}) \left( \frac{m_\alpha}{m_\beta} \vec{f}_\beta \nabla f_\alpha - \frac{m_\beta}{m_\alpha} \vec{f}_\alpha \nabla f_\beta \right)
\]

with a collision frequency \( \nu_{\alpha\beta} = e^2 e_0^2 \ln \Lambda_{\alpha\beta} / 8\pi m_0^2 e_0 \), the Coulomb logarithm \( \Lambda_{\alpha\beta} \) (\( \approx 10 \) herein), an arbitrary reference mass \( m_0 \), the vacuum permittivity \( e_0 \) and the effective charges \( e \) of each species. \( U(\vec{u}, \vec{v}) \) is the Landau tensor. Overbar terms are evaluated on the grid for the domain \( \Omega \) of species \( \beta \) and \( \vec{v} \equiv \vec{v} \) for clarity. And in the evolution of \( f_\alpha \), \( C_{\alpha\beta} = \sum_\beta C_{\alpha\beta} \).

See — for the further details, the weak form and a kernel algorithm [25].

The salient feature of (1) is the inner integral over the domain \( \Omega \) for each species \( \beta \), which results in an \( O(N^2) \) work complexity algorithm, where \( N \) is the number of integration points for a finite element formulation [25]. The two terms in (1), a divergence and Laplacian, have rank one vector and rank two tensor, respectively, “material” coefficients. These coefficients are computed in the inner integral.

### III. Batched linear solvers

Batch linear solvers, as with linear solvers in general, can be usefully categorized as direct and iterative. Direct methods for the most part use some type of sparse factorization and iterative methods combine the vectors in a Krylov subspace to generate an approximate solution that usually possesses some optimality condition such as minimizing the residual [26]. Direct solvers are attractive because they are robust and their poor asymptotic complexity is not an issue with the small systems in batch solves. However, direct solvers are inhibited by data dependencies in both matrix factorizations and solves. Experience with a single kernel launch CUDA batch band LU solver for the 2V Landau examples in §?? has been disappointing (§IIIG and the data archive in [27]). However, this effort used a band solver. The use of sparse solvers with more parallelism, such as nested dissection, should perform significantly better. Additionally, direct solvers could be critical if iterative solvers fail. The kernels of iterative methods, with simple Jacobi preconditioning, have minimal data dependencies and Krylov methods converge well for the Landau collision operator problems considered here and for combustion problems in Pele [16].

An approach to solving these systems that uses existing portable solvers in, for example, PETSc or Trilinos Kokkos Kernels [28], is to create an ensemble matrix, where each linear system is “stacked” to create a large block diagonal matrix, and use a Jacobi preconditioned Krylov solver [9]. However, the ensemble approach has several limitations. Traditional implementations of solvers abstract linear algebra operations for, among other things, portability. On a device, this results in a kernel launch for each vector operation, matrix-vector product, panel updates, preconditioner, etc., amounting to hundreds or thousands of such kernel launches. Amortizing these kernel launch costs requires a large degree of parallelism from batching, which may not be available from the application. Importantly, the ensemble approach is not consistent with Krylov iterative solvers because the scaling of each vector in the Krylov subspace is derived from the (artificial) global operator. Direct solvers do not suffer from inconsistency, but similarly require special techniques for batching [29].

The cost of a Landau collision time advance is dominated by the construction of the Jacobian matrix and the linear solve for the implicit time integrator. The matrix construction is described with CPU linear solvers in [6] and §IV discusses new optimizations to the batching of multiple spatial problems in each kernel. This report introduces new PETSc built-in GPU linear solvers that place a single solve on each PE using the Kokkos portable language in §V and compares it to a batch solver in Kokkos Kernels that supports multiple solves in each PE in §V.

#### A. Kokkos Hierarchical parallelism and batch solvers

The Kokkos portable language provides three levels of shared memory parallelism: the team level, the thread level, and the vector level. A Kokkos thread team is a collection of threads that can synchronize among themselves and that share a common memory level within the memory hierarchy. The maximal team size, i.e. the number of threads within a team, is architecture dependent. The vector level needs to be vectorizable. These three levels allow to write a hierarchical parallel algorithm within a CPU/GPU in a portable way.

Batch solvers in Kokkos Kernels [20] are implemented at the team level. A team can solve a small number \( m \) of systems at once. Choosing \( m \neq 1 \) provides more opportunity for parallelism within a team. In particular, providing that the layout of the underlying data is well chosen, choosing \( m = 8 \) allows to perfectly vectorize the code on Intel CPU architectures with double precision and AVX-512 instruction set. In order to solve a batch system, the user has to write a team parallel for loop that loops over subsets of batched indices, each team then solves the systems corresponding to
those indices (up to \( m \) systems) at once, this strategy is illustrated in Fig. 1.

![Illustration of a batched system where the colors represent different teams.](image)

**Fig. 1. Illustration of a batched system where the colors represent different teams.**

**IV. BUILT-IN PETSc BATCH SOLVER PERFORMANCE**

The important figure of merit to understand performance of the Landau collision operator time advance is the throughput of Newton iterations per second. This metric factors out the specifics of the time integrator and non-linear solver tolerance, etc., which is application dependent. Throughput is defined as the total number of, for example, Newton iterations times the batch size and number of GPUs, divided by the simulation time. A “warm-up” time step is used to setup the solver and is not timed because setup costs are amortized in a production setting.

The model problem is a deuterium plasma with two single species grids and eight species of tungsten that share one grid. The initial electron temperature is twice that of the ions and the model is run toward equilibrium for 10 time steps. One level of AMR refinement from a \( 4 \times 2 \) and a \( 4 \times 4 \times 4 \) grid is used, in \( 2 \) and \( 3V \) respectively, resulting in 14 (Q3 elements) and 120 (Q2 elements) in \( 2V \) and \( 3V \), respectively. We have observed that these grids are sufficient to measure a plasma resistivity within about 1% of the fully converged resistivity [6].

The test harness (ex2.c Landau example in PETSc, Appendix A) replicates the model problem to create a batch of problems to mimic an application’s use of this solver. Each of these problems requires a linear solver solve per species, resulting in a composition of this batch of problems with a batch of species solves per problem. To mimic variability in a real application, the density and hence collision frequency of each successive problem in a batch are varied within a range of about 10%.

Precise build parameters and instructions for reproducing the data herein are publicly available (see Appendix A).

Two linear solvers are considered: a batched TFQMR solver in PETSc, written in Kokkos, and an ensemble solver that uses Kokkos Kernels linear algebra primitives within the PETSc framework. Jacobi preconditioning is use throughout the report.

**A. NVIDIA A100 tests**

One node with four NVIDIA A100 Tensor Core GPUs based on the NVIDIA Ampere GPU architecture and 256GB of memory (Perlmutter) is used to investigate performance.

TFQMR solver is the fastest option with 60,000 and 550 Newton iterations per second in \( 2V \) and \( 3V \), respectively. Note that the GPU is pretty well saturated with a batch size of 256 in \( 2V \) and 32 in \( 3V \), and this corresponds to about 57,000 and 104,000 integration points per GPU, and thus \( 2V \) and \( 3V \) saturate at roughly the same number of integration points total as is expected.

**Figure 2** shows the \( 2V \) Linear solver throughput as a function of batch size and solver. This data shows that the batched

| Batch Size | TFQMR | Ensemble |
|------------|-------|----------|
| 256        | 3.48e+04 | 1.01e+04 |
| 64         | 6.47e+04 | 1.94e+04 |
| 16         | 1.24e+05 | 3.65e+04 |
| 8          | 1.78e+05 | 4.98e+04 |
| 4          | 2.26e+05 | 6.5e+04   |
| 2          | 2.54e+05 | 8.38e+04  |

![Graph showing Newton iterations / sec for each solver: 2V (left), 3V (right).](image)

**Figure 2.** Newton iterations / sec as function of batch size for each solver: \( 2V \) (left), \( 3V \) (right)

**Tables I and II show the total component times in \( 2V \) and \( 3V \), respectively, including mass matrix creation (“Mass”), Landau Jacobian (“Jacobian”), linear solver (“Solve”), and the total time and the total number of linear solver iterations.**

| Component | Jacobian | Mass | Solve | Total | Krylov its |
|-----------|----------|------|-------|-------|------------|
| Batch TFQMR | 1.57     | 0.22 | 0.58  | 2.44  | 3.69 |
| Ensemble TFQMR | 1.57   | 0.22 | 1.76  | 3.69  | 4.015 |

In \( 2V \) the solves, are subdominant and in \( 3V \) the time is completely dominated by the Jacobian creation, which is
TABLE II

| Component    | Jacobian | Mass | Solve | Total | Krylov its |
|--------------|----------|------|-------|-------|------------|
| Batch TFQMR  | 29.69    | 3.00 | 2.33  | 35.08 | 2,785      |
| Ensemble TFQMR | 29.67 | 3.00 | 1.51  | 34.31 | 2,326      |

is the fastest option with 24,000 and 397 Newton iterations per second in 2V and 3V, respectively. Note that the GPU is pretty well saturated with a batch size of 128 in 2V and 16 in 3V, and this corresponds to about 29,00 and 52,000 integration points per grid, and thus 2V and 3V saturate at about the same number of integration points total as is expected.

Tables III and IV show the total component times in 2V and 3V, respectively, including mass matrix creation (“Mass”), Landau Jacobian (“Jacobian”), linear solver (“Solve”), the total time and the total number of linear solver iterations. Note TFQMR has two matrix-vector products per iteration and so its work complexity is twice that of GMRES with respect to the number of iterations. Again, batch TFQMR is the fastest.

B. AMD MI250X tests

One node with four AMD MI250X, each with 2 Graphics Compute Dies (GCDs) for a total of 8 GCDs per node (Crusher) is used to investigate performance. Figure 3 shows the 2V Linear solver throughput as a function of batch size and solver. This data shows that the batched TFQMR solver is the fastest option with 24,000 and 397 Newton iterations per second in 2V and 3V, respectively. Note that the GPU is pretty well saturated with a batch size of 128 in 2V and 16 in 3V, and this corresponds to about 29,00 and 52,000 integration points per grid, and thus 2V and 3V saturate at about the same number of integration points total as is expected.

Tables III and IV show the total component times in 2V and 3V, respectively, including mass matrix creation (“Mass”), Landau Jacobian (“Jacobian”), linear solver (“Solve”), the total time and the total number of linear solver iterations. Note TFQMR has two matrix-vector products per iteration and so its work complexity is twice that of GMRES with respect to the number of iterations. Again, batch TFQMR is the fastest.

TABLE III

| Component    | Jacobian | Mass | Solve | Total | Krylov its |
|--------------|----------|------|-------|-------|------------|
| Batch TFQMR  | 17.04    | 1.04 | 1.76  | 19.88 | 3,673      |
| Ensemble TFQMR | 17.07 | 1.04 | 36.24 | 54.47 | 4,004      |

TABLE IV

| Component    | Jacobian | Mass | Solve | Total | Krylov its |
|--------------|----------|------|-------|-------|------------|
| Batch TFQMR  | 168.16   | 18.07| 11.28 | 196.81| 2,796      |
| Ensemble TFQMR | 168.62| 18.06| 39.51 | 210.84| 2,326      |

V. KOKKOS KERNELS BATCH SOLVER PERFORMANCE

VI. CONCLUSION

This report demonstrates that with the effective utilization of GPUs the gold standard for collisions in plasma simulations, the Landau collision operator, can be used effectively with an axisymmetric (2V) approximation and that full 3V may be feasible in the future. 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 to reduce cost, high-order accurate finite element discretizations with adaptive mesh refinement, and runs fully and effectively on GPUs with a portable Kokkos implementation in PETSc and in Kokkos Kernels. A new PETSc built-in batch solver has been introduced and the batch solvers in Kokkos Kernels have been discussed. Experiments have been conducted on an NVIDIA A100 node and AMD MI250X node. Artifacts and reproducibility instructions are publicly available (see Appendix A).

Future work includes the development of the solver in full 3V. This includes the development of a single 3V finite element with quadrature that is optimized to represent a Gaussian, the equilibrium Maxwellian distribution of a plasma. Beentjes shows that, for instance, 320 integration points can represent a Gaussian to an accuracy that we have observed is sufficient in our verification tests [30]. For near-Maxwellian plasmas this would bring the cost of a full 3V Landau time advance to less than a factor of 5x more expensive than the current 2V solve.
ACKNOWLEDGMENTS

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research and Office of Fusion Energy Sciences, Scientific Discovery through Advanced Computing (SciDAC) program.

REFERENCES

[1] S. Vanka, “Block-implicit multigrid solution of Navier-Stokes equations in primitive variables,” J. Comp. Phys., vol. 65, pp. 138–158, 1986.
[2] B. Smith, P. Bjorstad, and W. Gropp, Domain Decomposition. Cambridge University Press, 1996.
[3] P. E. Farrell, M. G. Knepley, L. Mitchell, and F. Wechsung, “Pefach: Software for the topological construction of multigrid relaxation methods,” ACM Trans. Math. Softw., vol. 47, no. 3, jun 2021. [Online]. Available: https://doi.org/10.1145/3445791
[4] H. Sitaraman, S. Yellapantula, M. T. Henry de Franh, B. Perry, J. Rood, R. Grout, and M. Day, “Adaptive mesh based combustion simulations of direct fuel injection effects in a super sonic cavity flame-holder,” Combustion and Flame, vol. 232, p. 111531, 2021. [Online]. Available: https://doi.org/10.1016/j.combustflame.2021.111531
[5] 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, vol. 315, pp. 644–660, Jun. 2016. [Online]. Available: https://doi.org/10.1016/j.jcp.2016.03.064
[6] M. F. Adams, D. P. Brennan, M. G. Knepley, and P. Wang, “Exascale Landau collision operator in the CUDA programming model applied to thermal quench plasmas,” in IEEE International Parallel and Distributed Processing Symposium, 2022.
[7] M. Lemou and P.-H. Chavanis, “Escape of stars from gravitational clusters in the Chandrasekhar model,” Physica A: Statistical Mechanics and its Applications, vol. 389, no. 5, pp. 1021–1040, 2010. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0378437109009194
[8] J. Heyvaerts, J.-B. Fouvy, P.-H. Chavanis, and C. Pichon, “Dressed diffusion and friction coefficients in inhomogeneous multicomponent self-gravitating systems,” Monthly Notices of the Royal Astronomical Society, vol. 469, no. 4, pp. 4193–4220, 05 2017. [Online]. Available: https://doi.org/10.1093/mnras/stt1092
[9] P.-H. Chavanis, “Landau equation for self-gravitating classical and quantum particles: Application to dark matter,” 2020. [Online]. Available: https://arxiv.org/abs/2012.12858
[10] J.-B. Fouvy, C. Pichon, P.-H. Chavanis, and L. Monk, “Resonant thickening of self-gravitating discs: imposed or self-induced orbital diffusion in the tightly wound limit,” Monthly Notices of the Royal Astronomical Society, vol. 471, no. 3, pp. 2642–2673, 06 2017. [Online]. Available: https://doi.org/10.1093/mnras/stx1625
[11] J.-B. Fouvy, P.-H. Chavanis, and C. Pichon, “Functional integral approach to the kinetic theory of inhomogeneous systems,” Physica A: Statistical Mechanics and its Applications, vol. 459, pp. 117–128, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0378437116301376
[12] K. Liegeois, R. Boman, E. T. Phipps, T. Wiesner, and M. Arntz, “Gmres with embedded ensemble propagation for the efficient solution of parametric linear systems in uncertainty quantification of multiphysics models,” Computer Methods in Applied Mechanics and Engineering, vol. 369, p. 113188, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S004578252030377X
[13] A. Mollm, M. F. Adams, M. G. Knepley, R. Hager, and C. S. Chang, “Implementation of higher-order velocity mapping between marker particles and grid in the particle-in-cell code XGC,” Journal of Plasma Physics, vol. 87, no. 2, 2021.
[14] J. V. Pusztay, M. G. Knepley, and M. F. Adams, “Conservative projection between fem and particle bases,” SIAM Journal on Scientific Computing, 2022, accepted.
[15] W. H. Boukaram, G. Turkyyah, H. Ltaief, and D. E. Keyes, “Batched qr and svd algorithms on gpus with applications in hierarchical matrix compression,” Parallel Computing, vol. 74, pp. 19–33, 2018, parallel Matrix Algorithms and Applications (PMAA’16). [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0018921917304611

APPENDIX A

ARTIFACT DESCRIPTION AND REPRODUCIBILITY

PETSc output files with performance data and provenance information, build instructions for each platform, reproducibility instructions, and verification data can be found with git clone gitlab.com/markadams4/batch_paper_data.