Optimization and Large Scale Computation of an Entropy-Based Moment Closure[*]

C. Kristopher Garrett[†] Cory Hauck[‡] Judith Hill[§]

January 28, 2016

Abstract

We present computational advances and results in the implementation of an entropy-based moment closure, M_N, in the context of linear kinetic equations, with an emphasis on heterogeneous and large-scale computing platforms. Entropy-based closures are known in several cases to yield more accurate results than closures based on standard spectral approximations, such as P_N, but the computational cost is generally much higher and often prohibitive. Several optimizations are introduced to improve the performance of entropy-based algorithms over previous implementations. These optimizations include the use of GPU acceleration and the exploitation of the mathematical properties of spherical harmonics, which are used as test functions in the moment formulation. To test the emerging high-performance computing paradigm of communication bound simulations, we present timing results at the largest computational scales currently available. These results show, in particular, load balancing issues in scaling the M_N algorithm that do not appear for the P_N algorithm. We also observe that in weak scaling tests, the ratio in time to solution of M_N to P_N decreases.

1 Introduction

Kinetic equations, such as the Boltzmann equation and the radiation transport equation, are integro-differential equations with up to seven independent variables: three space, three momentum, and time. Moment methods track the evolution of only a finite number of weighted momentum averages, or *moments*, of the kinetic distribution, thus reducing the dimensionality of the problem. However, this reduction requires a closure that approximates in some way the kinetic information that is lost in the averaging process. Thus, various methods will differ by the closure used in their formulation.

In the context of radiation transport, the classical moment method is the spherical harmonic expansion, colloquially termed P_N in the radiation transport community [11, 32, 40]. This method uses a simple truncation closure that results in a linear hyperbolic balance law. However, the method may suffer from numerical artifacts, most notably large oscillations that can result in negative

[*]The submitted manuscript has been authored by a contractor of the U.S. Government under Contract No. DE-AC05-00OR22725. Accordingly, the U.S. Government retains a non-exclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U.S. Government purposes.

[†]Computer Science and Mathematics Division, Oak Ridge National Laboratory, Oak Ridge, TN. E-mail: `garrettck@ornl.gov`.

[‡]Computer Science and Mathematics Division, Oak Ridge National Laboratory, Oak Ridge, TN & Department of Mathematics, University of Tennessee, Knoxville, TN. E-mail: `hauckc@ornl.gov`.

[§]Oak Ridge Leadership Computing Facility, Oak Ridge National Laboratory, Oak Ridge, TN. E-mail: `hilljc@ornl.gov`.

particle concentrations, especially in the streaming particle regime, where collisions are rare [7]. An alternative to the P_N closure is a more complicated nonlinear closure based on minimizing a physically relevant, convex function related to the entropy of the physical system [13, 20, 31]. The resulting method, colloquially termed M_N in the radiation transport community, yields a nonlinear hyperbolic balance law that, unlike the P_N method, formally captures the correct streaming limit [13], but at the same time, is accurate in scattering dominated regimes [9, 17]. The M_N method has been shown to be more accurate in several test cases [8, 14, 16, 20] and in several different applications [6, 15, 18, 33, 36, 45]. However, it requires the solution of a minimization problem at every spatial cell of the discretized domain. In general, the minimization must be solved numerically. This causes the M_N method to require many more floating point operations than the P_N method, even though the methods have the same data communication patterns. To avoid the computational overhead of the minimization problem, some approaches generate approximations of the entropy-based closure using look-up tables or interpolation schemes [34, 35, 47]. Such approximations are generally less robust than the original entropy-based closures and have, so far, been limited to low-order moment systems. Even so, in some cases, they maintain enough of the structure from the entropy-based approach to be considered as a suitable alternative.

The computational expense of the M_N method makes it prohibitive for serial or even small-scale parallel implementations. However, for large scale computations on high performance machines, it is expected that the computing time for the P_N method will eventually become dominated by communication, and in such cases, the M_N method will be more competitive in time to solution. Therefore progress in this area depends on three factors: (i) algorithmic improvements in solving the minimization problem that defines the M_N closure, (ii) performance improvements that leverage the available computer hardware, and (iii) scaling to extremely large problems. The work here builds on algorithmic improvements in [2, 3].[1] In the current paper, we address the other two factors. First we design and test several optimizations for the M_N algorithm that reduce the time to solution by as much as 10 times in some cases. We then explore scalability of the M_N algorithm using an explicit time integration algorithm. Using the supercomputer Titan, which is housed at Oak Ridge National Laboratory and operated by the Oak Ridge Leadership Computing Facility, we find the M_N algorithm weakly scales almost perfectly out to 17,576 compute nodes while the P_N algorithm displays an increase in time per node by a factor of 1.2x to 4x, depending on the amount of data per node. However, even with performance improvements, the time to solution of the M_N algorithm is still approximately 25 times greater than the time to solution of the P_N algorithm when both are run at full scale.

The layout of the paper is as follows. In Section 2, we briefly summarize the moment approach, discuss important implementation details, and introduce two test problems that will be used for numerical simulations. In Section 3, we introduce three improvements to the M_N algorithm: one that leverages structure in the Hessian matrix of the M_N minimization algorithm and two that use GPUs to accelerate the two most arithmetically intensive parts of the computation. In Section 4, M_N statistics and timing results are presented for the two test problems. Results of weak scaling tests for P_N and M_N are also compared. Section 5 is for conclusions and discussion. The Appendix contains useful technical details about spherical harmonics and Gaunt coefficients. Section D of the Appendix contains a glossary for the variables used throughout the paper.

[1] Although not discussed here, other efforts to solve the minimization problem have been documented in [3].

2 Moment equations

In this section, we briefly summarize the necessary background material on moment methods, give details on numerical implementation, and present two initial conditions used in the numerical examples.

2.1 Formulation

The governing equation for this study is a linear kinetic transport equation for unit speed particles in an infinite medium. This equation takes the form

$$\partial_t f + \Omega \cdot \nabla_x f = \frac{1}{4\pi}\sigma_{\mathrm{s}}\langle f\rangle - \sigma_{\mathrm{t}} f, \tag{1}$$

where (i) $x \in \mathbb{R}^3$ is a point in space, (ii) $\Omega \in \mathbb{S}^2$ (the unit sphere) is a velocity direction (velocity magnitude is 1), (iii) $t > 0$ is a point in time, (iv) $f(x, \Omega, t)$ is the kinetic density of particles with respect to the measure $dxd\Omega$, (v) $\sigma_{\mathrm{s}}(x)$ is the scattering cross section, (vi) $\sigma_{\mathrm{t}}(x)$ is the total cross section, and (vii) $\langle\cdot\rangle = \int_{\mathbb{S}^2} \cdot \, d\Omega$. In general $\sigma_{\mathrm{t}} \geq \sigma_{\mathrm{s}} \geq 0$; for the purposes of this paper, we set $\sigma_{\mathrm{t}} = \sigma_{\mathrm{s}} = 1$.

Moment equations are derived from (1). Let $\mathbf{m}(\Omega) = (m_1(\Omega), \ldots, m_M(\Omega))^T$ be a finite vector of real-valued, normalized spherical harmonics[2] (defined in Appendix A) of degree less than or equal to N with length $M = (N+1)^2$. Define the finite vector of moments with respect to Ω as $\mathbf{u}_f(x,t) = \langle \mathbf{m} f\rangle$. Then according to (1), $\mathbf{u}_f$ satisfies

$$\partial_t \mathbf{u}_f + \nabla_x \cdot \langle \Omega \mathbf{m} f\rangle = -Q\mathbf{u}_f, \tag{2}$$

where $Q = \operatorname{diag}(0, 1, \ldots, 1)$. The system (2) is not closed because the flux $\langle \Omega \mathbf{m} f\rangle$ is a linear combination of moments up to degree $N+1$ whereas $\mathbf{u}_f$ only contains moments up to and including degree N.

The system (2) is closed using an expansion operator

$$\mathcal{E} \colon D \ni \mathbf{v} \longmapsto \mathcal{E}(\mathbf{v}) \in L^1(\mathbb{S}^2), \tag{3}$$

where $D \subset \mathbb{R}^M$, and $\mathcal{E}$ satisfies the compatibility condition $\mathbf{v} = \langle \mathbf{m}\mathcal{E}(\mathbf{v})\rangle$.[3] Approximating f by $\mathcal{E}(\mathbf{u})$ in (2) yields the closed system

$$\partial_t \mathbf{u} + \nabla_x \cdot \langle \Omega \mathbf{m} \mathcal{E}(\mathbf{u})\rangle = -Q\mathbf{u}, \tag{4}$$

where $\mathbf{u}$ approximates the true moments $\mathbf{u}_f$. Equation (4) may be equivalently viewed as the following nonlinear Galerkin approximation: *Find* $\mathbf{u} \in D$ *such that*

$$\partial_t \langle \phi \mathcal{E}(\mathbf{u})\rangle + \nabla_x \cdot \langle \Omega \phi \mathcal{E}(\mathbf{u})\rangle = -Q\langle \phi \mathcal{E}(\mathbf{u})\rangle, \tag{5}$$

for all $\phi \in \operatorname{span}\{m_1, \ldots, m_M\}$.

In this paper, we focus on the entropy-based moment method M_N, but for the purposes of comparison, we also consider the classical moment method P_N. Below are their expansion operators [16]:

[2]The choice of spherical harmonics is not necessary, but it is common in radiation transport. This is because they are eigenfunctions of a more general scattering operator [32].

[3]For the P_N equations, the domain D of the expansion operator $\mathcal{E}$ is all of $\mathbb{R}^M$. For the M_N equations $D = \{\mathbf{w} \in \mathbb{R}^M : \mathbf{w} = \langle \mathbf{m}F\rangle$ for some nonnegative $F \in L^1(\mathbb{S}^2)\}$ is the set of realizable moments. More information on realizability in the context of kinetic equations can be found in [21, 23, 43]. For more general theory, see [10, 24, 44].

- P_N: $\mathcal{E}(\mathbf{u}) = \mathbf{u}^T \langle \mathbf{m}\mathbf{m}^T \rangle^{-1} \mathbf{m} = \mathbf{u}^T \mathbf{m}$;[4]
- M_N: $\mathcal{E}(\mathbf{u}) = \exp(\hat{\boldsymbol{\alpha}}(\mathbf{u})^T \mathbf{m})$, where $\hat{\boldsymbol{\alpha}} : \mathbb{R}^M \to \mathbb{R}^M$ is given by

$$\hat{\boldsymbol{\alpha}} = \operatorname*{arg\,min}_{\boldsymbol{\alpha} \in \mathbb{R}^M} \left\langle \exp(\boldsymbol{\alpha}^T \mathbf{m}) \right\rangle - \boldsymbol{\alpha}^T \mathbf{u}. \tag{6}$$

2.2 Implementation

In this section, we present an overview of several implementation details including (i) quadrature approximation in angle, (ii) finite volume discretization in space, (iii) solving the M_N minimization problem (6), and (iv) time discretization method. Since many indices and variables are used, a glossary is available in Appendix D for reference.

Quadrature approximation. For various parts of the M_N implementation, we use a quadrature to approximate integrals with respect to Ω over the domain $\mathbb{S}^2$. To this end, we use a product quadrature on the sphere [4, 46] with n_g Gauss-Legendre nodes/weights on the Ω_3-axis and $2n_g$ equally spaced nodes/weights around latitudinal circles of the sphere for a total of $Q = 2n_g^2$ quadrature nodes/weights. This quadrature has the property that it integrates spherical harmonics of degree less than or equal to $2n_g - 1$ exactly. We use the notation (Ω_q, w_q) to represent the quadrature nodes and weights, respectively, for $q = 1, \ldots, Q$. For any $G \in L^1(\mathbb{S}^2)$, integrals will be approximated as

$$\int_{\mathbb{S}^2} G(\Omega) d\Omega \approx \sum_{q=1}^{Q} w_q G(\Omega_q). \tag{7}$$

Several other quadratures exist that can integrate spherical harmonics of the same degree as the cross-product quadrature, but with fewer points. These include quadratures by Lebedev [26–30] and by Ahrens and Beylkin [1]. As the number of quadrature points increases, these quadratures are optimal in the number of nodes/weights and asymptotically use 2/3 the number of points that the product quadrature uses. See [5] for a general discussion. While we use the product quadrature for simplicity, these more efficient quadratures will likely yield a commensurate improvement in overall run-time of the M_N simulations.

Spatial discretization. For the spatial discretization, we use a finite volume method. In particular, we use a kinetic scheme [12,19,38,39]. This scheme is derived by first spatially discretizing the kinetic equation (1) using a finite volume approach and then taking moments of the discretization and applying the closure.

More specifically, we use a uniform cartesian grid with cells C_{ijk} and cell sizes $\Delta x \times \Delta y \times \Delta z$. For compactness, define

$$\begin{aligned}\boldsymbol{\delta}_{ijk} :=& \frac{1}{\Delta x} \left\langle \Omega_1 \mathbf{m} (\mathcal{E}_{i+1/2,j,k} - \mathcal{E}_{i-1/2,j,k}) \right\rangle + \frac{1}{\Delta y} \left\langle \Omega_2 \mathbf{m} (\mathcal{E}_{i,j+1/2,k} - \mathcal{E}_{i,j-1/2,k}) \right\rangle + \\ & \frac{1}{\Delta z} \left\langle \Omega_3 \mathbf{m} (\mathcal{E}_{i,j,k+1/2} - \mathcal{E}_{i,j,k-1/2}) \right\rangle, \end{aligned} \tag{8}$$

[4]In general, the matrix $\langle \mathbf{m}\mathbf{m}^T \rangle$ is diagonal since the real-valued spherical harmonics are an orthogonal basis. Furthermore, by normalizing the spherical harmonics as we do in this paper, $\langle \mathbf{m}\mathbf{m}^T \rangle$ is the identity matrix.

where $\mathcal{E}_{i\pm1/2,j,k}$ approximate the average of $\mathcal{E}(\mathbf{u})$ on the two boundary faces of C_{ijk} perpendicular to the x-axis, and $\mathcal{E}_{i,j\pm1/2,k}$ and $\mathcal{E}_{i,j,k\pm1/2}$ are defined similarly. Then, the spatial discretization is given by

$$\partial_t \mathbf{u}_{ijk} + \boldsymbol{\delta}_{ijk} + Q\mathbf{u}_{ijk} = 0, \tag{9}$$

where $\mathbf{u}_{ijk}$ is an approximation of the corresponding cell average for $\mathbf{u}$.

We use a second-order upwind scheme to calculate edge values for both P_N and M_N. For the P_N closure, the integrals in (8) can be computed exactly without explicitly calculating the edge values $\mathcal{E}_{i\pm1/2,j,k}$, $\mathcal{E}_{i,j\pm1/2,k}$, and $\mathcal{E}_{i,j,k\pm1/2}$. However, for the M_N closure, a quadrature rule is required to approximate the integrals. In addition, a double minmod limiter is used to limit slopes for M_N to ensure realizable moments as in [2,16]. More details about the discretization for both P_N and M_N can be found in [16], where the same discretization approach was used in a two-dimensional spatial setting. (Note that we abuse the notation $\mathcal{E}_{i\pm1/2,j,k}$, $\mathcal{E}_{i,j\pm1/2,k}$, $\mathcal{E}_{i,j,k\pm1/2}$, and $\boldsymbol{\delta}_{ijk}$, which may refer to values for either P_N or M_N, although the two methods yield different results.)

Minimization problem for $\mathbf{M}_N$. Given $\mathbf{u}$, the objective function for the minimization problem (6) is

$$\Phi(\boldsymbol{\alpha};\mathbf{u}) = \int_{\mathbb{S}^2} \exp(\boldsymbol{\alpha}^T\mathbf{m})d\Omega - \boldsymbol{\alpha}^T\mathbf{u}. \tag{10}$$

We use the Newton-type solver developed in [2], which requires the gradient and Hessian of Φ:

$$\mathbf{g}(\boldsymbol{\alpha}) = \langle \mathbf{m}e^{\boldsymbol{\alpha}^T\mathbf{m}}\rangle - \mathbf{u} \qquad \text{and} \qquad H(\boldsymbol{\alpha}) = \langle \mathbf{m}\mathbf{m}^T e^{\boldsymbol{\alpha}^T\mathbf{m}}\rangle. \tag{11}$$

A summary of the minimization algorithm is given in Algorithm 1. For full details consult [2][5].

Algorithm 1: Newton type optimization algorithm

```
   Input: $\mathbf{u}$, initial guess $\boldsymbol{\alpha}^{(0)}$
   Parameters: tolerance $\tau > 0$ , line search parameter $0 < \varepsilon < 1$, maxiter
 1 $\boldsymbol{\alpha} \leftarrow \boldsymbol{\alpha}^{(0)}$
 2 iter $\leftarrow 0$
 3 while iter $<$ maxiter do
 4     iter $\leftarrow$ iter $+ 1$
 5     compute the gradient $\mathbf{g}(\boldsymbol{\alpha})$ and Hessian $H(\boldsymbol{\alpha})$
 6     if $||\mathbf{g}(\boldsymbol{\alpha})|| < \tau$ then
 7         return $\boldsymbol{\alpha}$
 8     end
 9     solve for the search direction $\mathbf{d} \leftarrow -H^{-1}(\boldsymbol{\alpha})\mathbf{g}(\boldsymbol{\alpha})$
10     do a linesearch to find $t \in (0,1]$ so that $\Phi(\boldsymbol{\alpha} + t\mathbf{d}) \le \Phi(\boldsymbol{\alpha}) + \varepsilon t \mathbf{g}^T\mathbf{d}$
11     $\boldsymbol{\alpha} \leftarrow \boldsymbol{\alpha} + t\mathbf{d}$
12 end
13 return $\boldsymbol{\alpha}$
```

Since H is symmetric and positive definite, $\mathbf{d}$ is computed using a Cholesky decomposition. In the linesearch, we use $\varepsilon = 10^{-3}$, $\tau = 10^{-4}$, and `maxiter` $= 100$. In our simulations, `iter` never reached the limit `maxiter`. Typically, `iter` took on values of only 1 or 2, but a few minimizations required values as high as 25.

[5]The algorithm in [2] uses an adaptive quadrature and a regularization procedure for ill-conditioned problems. Neither of these were required for the problems in this paper.

One important implementation detail is the calculation of the objective Φ and the gradient $\mathbf{g}$, both of which can be computed from the Hessian. Given the value of the Hessian H, the computation of Φ and $\mathbf{g}$ are trivially computed since m_1 is a constant and thus

$$\Phi = \frac{1}{m_1^2} H_{1,1} - \boldsymbol{\alpha}^T \mathbf{u} \qquad \text{and} \qquad \mathbf{g} = \frac{1}{m_1} H_{:,1} - \mathbf{u}, \tag{12}$$

where $H_{:,1}$ indicates the first column of H and $H_{1,1}$ is the $(1,1)$ entry of H. Hence, calculation of the Hessian is the largest computational cost of the minimization algorithm.

Time discretization. We use the explicit Heun's method (SSP-RK2) for time integration which is 2nd order, and under the time step constraint

$$\Delta t \leq \frac{1}{2(\frac{1}{\Delta x} + \frac{1}{\Delta y} + \frac{1}{\Delta z}) + \sigma_t}, \tag{13}$$

ensures realizability for the M_N closure [2][6].

Algorithms 2 and 3 outline the approach used to calculate one time step for the M_N and P_N algorithms, respectively. The code uses one MPI task per compute node, with OpenMP or CUDA used to parallelize computations on each compute node. The four functions from lines 3 – 6 do the following

- `comm` – communicates boundary data between compute nodes via MPI;
- `min` – implements Algorithm 1 (M_N only);
- `flux_pn`/`flux_mn` – calculates $\boldsymbol{\delta}$ from (8) for P_N and M_N, respectively;
- `euler` – computes $\mathbf{u}^{(s)} = \mathbf{u}^{(s-1)} - \Delta t \boldsymbol{\delta} - \Delta t Q \mathbf{u}^{(s-1)}$, where s is the Runge-Kutta stage of the Heun method.

Algorithm 2: One time step for the M_N algorithm

```
  Input: moment at time $n$: $\mathbf{u}^n$
1 $\mathbf{u}^{(0)} \leftarrow \mathbf{u}^n$
2 for $s = 1 \ldots 2$ do
3     comm($\mathbf{u}^{(s-1)}$)
4     $\boldsymbol{\alpha} \leftarrow$ min($\mathbf{u}^{(s-1)}$)
5     $\boldsymbol{\delta} \leftarrow$ flux_mn($\boldsymbol{\alpha}, \mathbf{u}^{(s-1)}$)
6     $\mathbf{u}^{(s)} \leftarrow$ euler($\boldsymbol{\delta}, \mathbf{u}^{(s-1)}$)
7 end
8 $\mathbf{u}^{n+1} \leftarrow \frac{1}{2}(\mathbf{u}^n + \mathbf{u}^{(2)})$
9 return $\mathbf{u}^{n+1}$
```

Algorithm 3: One time step for the P_N algorithm

```
  Input: moment at time $n$: $\mathbf{u}^n$
1 $\mathbf{u}^{(0)} \leftarrow \mathbf{u}^n$
2 for $s = 1 \ldots 2$ do
3     comm($\mathbf{u}^{(s-1)}$)
4
5     $\boldsymbol{\delta} \leftarrow$ flux_pn($\boldsymbol{\alpha}, \mathbf{u}^{(s-1)}$)
6     $\mathbf{u}^{(s)} \leftarrow$ euler($\boldsymbol{\delta}, \mathbf{u}^{(s-1)}$)
7 end
8 $\mathbf{u}^{n+1} \leftarrow \frac{1}{2}(\mathbf{u}^n + \mathbf{u}^{(2)})$
9 return $\mathbf{u}^{n+1}$
```

[6]This is only proven for 1D in space in the reference, but the extension to higher dimensions is straightforward.

2.3 Test problems

Two initial conditions were used for the tests we performed: a "smooth" initial condition

$$u_\ell^{\mathrm{S}} = \begin{cases} 2\sqrt{\pi}(2 + \cos(2\pi x)\cos(2\pi y)\cos(2\pi z)), & \ell = 1 \\ 0.3\, u_1, & \ell = 2,3,4 \\ 0, & \ell > 4 \end{cases} \tag{14}$$

and a narrow "Gaussian" initial condition:

$$u_\ell^{\mathrm{G}} = \begin{cases} 2\sqrt{\pi}\max\left\{ \left(\frac{1}{\sigma\sqrt{2\pi}}\right)^3 \exp\left(\frac{-\|\mathbf{x}\|_2^2}{2\sigma^2}\right), 10^{-8} \right\}, & \ell = 1 \\ 0, & \ell > 1 \end{cases} \tag{15}$$

on a periodic domain of size $[-1,1]^3$ with $\sigma = 0.03$ for the Gaussian initial condition. Figure 1 shows $\langle f \rangle$ in the $z = 0$ plane for the two initial conditions.

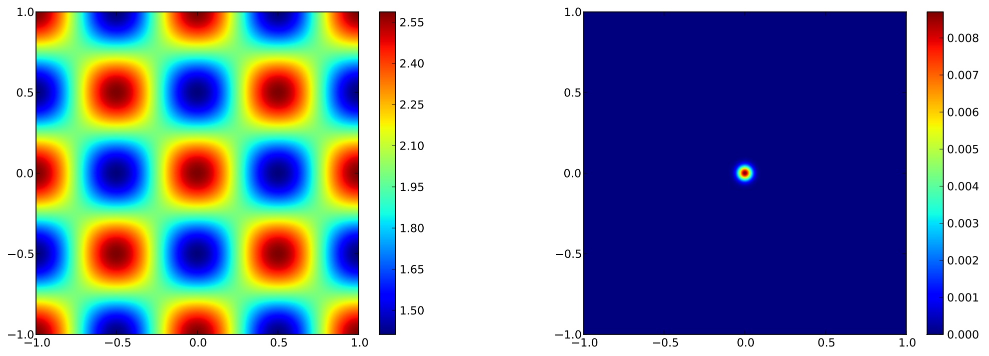

Figure 1: Left: smooth initial condition (14). Right: Gaussian initial condition (15). Figure shows $\langle f \rangle$ in the $z = 0$ plane.

3 Computational improvements for M_N

In this section, three computational optimizations for the M_N method will be detailed with results showing the efficacy for each. The results were computed using one node of the supercomputer Titan. Each node consists of a 16-core 2.2GHz AMD Opteron 6274 processor and an NVIDIA Tesla K20 GPU.

Before discussing the computational optimizations, it is necessary to know which functions comprise the bulk of the computation time. Table 1 contains timings for each of the four functions in Algorithm 2 executed on one compute node. From the table, it is clear that the bulk of the computing time is taken by the functions `min` and `flux_mn`. We also present a profile of the P_N method for comparison in Table 2. It is worth noting that the `comm` and `euler` functions have almost identical execution times when comparing P_N to M_N. This is expected since these functions are identical in both cases.

Function	Time	% Time
`comm`	0.139s	0.11%
`min`	99.764s	80.00%
`flux_mn`	24.683s	19.79%
`euler`	0.036s	0.03%
Sum	124.622s	99.94%

(a) M_3 method with $n_g = 12$. Total wall clock time was 124.702s.

Function	Time	% Time
`comm`	0.435s	0.02%
`min`	2083.208s	87.71%
`flux_mn`	290.745s	12.24%
`euler`	0.235s	0.01%
Sum	2374.623s	99.98%

(b) M_7 method with $n_g = 20$. Total wall clock time was 2375.041s.

Table 1: Profile of OpenMP code for Algorithm 2 with no optimizations. Parameters: smooth initial condition, 40^3 spatial cells, 20 time steps taken.

Function	Time	% Time
`comm`	0.133s	7.60%
`flux_pn`	1.559s	89.03%
`euler`	0.028s	1.60%
Sum	1.720s	98.23%

(a) P_3 method. Total wall clock time was 1.751s.

Function	Time	% Time
`comm`	0.428s	2.61%
`flux_pn`	15.535s	94.60%
`euler`	0.225s	1.37%
Sum	16.188s	98.58%

(b) P_7 method. Total wall clock time was 16.422s.

Table 2: Profile of OpenMP code for Algorithm 3. Parameters: smooth initial condition, 40^3 spatial cells, 20 time steps taken.

Remark 1. *For clarity in the remainder of the paper, we will use the term "optimization" to refer to computational optimizations and the term "minimization" to refer to the minimization from Equation* (6)*.*

3.1 GPU acceleration of the flux calculation

Before describing the algorithm for GPU acceleration of `flux_mn`, we describe the CPU implementation of both `flux_pn` in Algorithm 4 and `flux_mn` in Algorithm 5 for comparison. To leading order, for a single spatial cell, the calculation of $\boldsymbol{\delta}_{ijk}$ for P_N takes $6M^2$ multiplications to compute and for M_N takes $3QM$ multiplications to compute. Since $Q >> M$ in general, the calculation for P_N is much less costly than the calculation for M_N.

Algorithm 4: CPU flux algorithm for P_N

```
    Input: $\mathbf{u}_{ijk}$ for all spatial indices $i, j, k$
    Definition: $\Omega_\ell^+ = \max(\Omega_\ell, 0)$, $\Omega_\ell^- = \min(\Omega_\ell, 0)$
 1  for $i = 1 \dots n,\ j = 1 \dots n,\ k = 1 \dots n$ (parallel via OpenMP) do
 2      $\boldsymbol{\delta}_{ijk} \leftarrow 0$
 3      $\boldsymbol{\delta}_{ijk} \leftarrow \boldsymbol{\delta}_{ijk} + \frac{1}{\Delta x}\langle \Omega_1^+ \mathbf{m}\mathbf{m}^T \rangle (\frac{1}{4}\mathbf{u}_{i+1,j,k} + \frac{3}{4}\mathbf{u}_{i,j,k} - \frac{5}{4}\mathbf{u}_{i-1,j,k} + \frac{1}{4}\mathbf{u}_{i-2,j,k})$
 4      $\boldsymbol{\delta}_{ijk} \leftarrow \boldsymbol{\delta}_{ijk} + \frac{1}{\Delta x}\langle \Omega_1^- \mathbf{m}\mathbf{m}^T \rangle (-\frac{1}{4}\mathbf{u}_{i+2,j,k} + \frac{5}{4}\mathbf{u}_{i+1,j,k} - \frac{3}{4}\mathbf{u}_{i,j,k} - \frac{1}{4}\mathbf{u}_{i-1,j,k})$
 5      $\boldsymbol{\delta}_{ijk} \leftarrow \boldsymbol{\delta}_{ijk} + \frac{1}{\Delta y}\langle \Omega_2^+ \mathbf{m}\mathbf{m}^T \rangle (\frac{1}{4}\mathbf{u}_{i,j+1,k} + \frac{3}{4}\mathbf{u}_{i,j,k} - \frac{5}{4}\mathbf{u}_{i,j-1,k} + \frac{1}{4}\mathbf{u}_{i,j-2,k})$
 6      $\boldsymbol{\delta}_{ijk} \leftarrow \boldsymbol{\delta}_{ijk} + \frac{1}{\Delta y}\langle \Omega_2^- \mathbf{m}\mathbf{m}^T \rangle (-\frac{1}{4}\mathbf{u}_{i,j+2,k} + \frac{5}{4}\mathbf{u}_{i,j+1,k} - \frac{3}{4}\mathbf{u}_{i,j,k} - \frac{1}{4}\mathbf{u}_{i,j-1,k})$
 7      $\boldsymbol{\delta}_{ijk} \leftarrow \boldsymbol{\delta}_{ijk} + \frac{1}{\Delta z}\langle \Omega_3^+ \mathbf{m}\mathbf{m}^T \rangle (\frac{1}{4}\mathbf{u}_{i,j,k+1} + \frac{3}{4}\mathbf{u}_{i,j,k} - \frac{5}{4}\mathbf{u}_{i,j,k-1} + \frac{1}{4}\mathbf{u}_{i,j,k-2})$
 8      $\boldsymbol{\delta}_{ijk} \leftarrow \boldsymbol{\delta}_{ijk} + \frac{1}{\Delta z}\langle \Omega_3^- \mathbf{m}\mathbf{m}^T \rangle (-\frac{1}{4}\mathbf{u}_{i,j,k+2} + \frac{5}{4}\mathbf{u}_{i,j,k+1} - \frac{3}{4}\mathbf{u}_{i,j,k} - \frac{1}{4}\mathbf{u}_{i,j,k-1})$
 9  end
10  return $\boldsymbol{\delta}_{ijk}$ for all spatial indices $i, j, k$
```

Algorithm 5: CPU flux algorithm for M_N

```
    Input: $\boldsymbol{\alpha}_{ijk}$ for all spatial indices $i, j, k$
    Definition: $m_{q\ell} = m_\ell(\Omega_q)$
 1  for $i = 1 \dots n,\ j = 1 \dots n,\ k = 1 \dots n$ (parallel via OpenMP) do
 2      $\boldsymbol{\delta}_{ijk} \leftarrow 0$
 3  end
 4  for $q = 1 \dots Q$ (serial) do
 5      for $i = 1 \dots n,\ j = 1 \dots n,\ k = 1 \dots n$ (parallel via OpenMP) do
 6          $\mathcal{E}_{qijk} \leftarrow \exp(\sum_{\ell=1}^{M} m_{q\ell}\alpha_{\ell ijk})$
 7      end
 8      for $i = 1 \dots n,\ j = 1 \dots n,\ k = 1 \dots n$ (parallel via OpenMP) do
 9          compute $\mathcal{E}_{q,i\pm1/2,j,k}$, $\mathcal{E}_{q,i,j\pm1/2,k}$, and $\mathcal{E}_{q,i,j,k\pm1/2}$
10          for $\ell = 1 \dots M$ do
11              $\delta_{\ell ijk} \leftarrow \delta_{\ell ijk} + \frac{1}{\Delta x} w_q m_{q\ell}(\mathcal{E}_{q,i+1/2,j,k} - \mathcal{E}_{q,i-1/2,j,k})$
12              $\delta_{\ell ijk} \leftarrow \delta_{\ell ijk} + \frac{1}{\Delta y} w_q m_{q\ell}(\mathcal{E}_{q,i,j+1/2,k} - \mathcal{E}_{q,i,j-1/2,k})$
13              $\delta_{\ell ijk} \leftarrow \delta_{\ell ijk} + \frac{1}{\Delta z} w_q m_{q\ell}(\mathcal{E}_{q,i,j,k+1/2} - \mathcal{E}_{q,i,j,k-1/2})$
14          end
15      end
16  end
17  return $\boldsymbol{\delta}_{ijk}$ for all spatial indices $i, j, k$
```

Two important implementation notes are in order. First, the integrals $\langle \Omega_\ell^\pm \mathbf{m}\mathbf{m}^T \rangle$ for $\ell = 1, 2, 3$ from Algorithm 4 can be precomputed once at the beginning of the program. Second, for Algorithm 5, the outer loop in quadrature index starting at line 4 is done serially to save space in memory. This enables the variable $\mathcal{E}_{qijk}$ to be held as only one floating point number per spatial cell instead of Q floating point numbers per spatial cell. For a moment method, only $O(Mn^3)$ bytes of memory should be required for the code, but if $\mathcal{E}_{qijk}$ required Q floating point numbers per spatial cell, the memory requirement would jump to $O(Qn^3)$ bytes. This is undesirable since Q is generally much larger than M. For our simulations, there was plenty of work to do in parallel for the loops in spatial cells i, j, k, that we still got near perfect speedup, and thus parallelization in q was not necessary.

GPU acceleration of `flux_mn` is given in Algorithm 6. The bold **GPU function** in the algorithm refers to a function computed on the GPU. Hence in this implementation, the algorithm calls three separate GPU functions. (Data copies between CPU and GPU were negligible compared to the computation times on the GPU.)

Algorithm 6: GPU flux algorithm for M_N

```
   Input: $\boldsymbol{\alpha}_{ijk}$ for all spatial indices $i, j, k$
   Definition: $m_{q\ell} = m_\ell(\Omega_q)$
   Notes: Allocate all necessary memory on the GPU at the start of the program. Copy $m_{q\ell}$
          and $w_q$ from CPU to GPU at the start of the program.
1  Copy $\boldsymbol{\alpha}_{ijk}$ for all $i, j, k$ from CPU to GPU
2  Set $\boldsymbol{\delta}_{ijk} = 0$ for all $i, j, k$ on GPU
3  for $q = 1 \dots Q$ (serial) do
4      GPU function in parallel across indices $i, j, k$ do
5          compute $\mathcal{E}_{qijk} \leftarrow \exp(\sum_{\ell=1}^{M} m_{q\ell}\alpha_{\ell ijk})$
6      end
7      GPU function in parallel across indices $i, j, k$ do
8          compute $\mathcal{E}_{q,i\pm1/2,j,k}$, $\mathcal{E}_{q,i,j\pm1/2,k}$, and $\mathcal{E}_{q,i,j,k\pm1/2}$
9      end
10     GPU function in parallel across indices $i, j, k, \ell$ do
11         $\delta_{\ell ijk} \leftarrow \delta_{\ell ijk} + \frac{1}{\Delta x} w_q m_{q\ell}(\mathcal{E}_{q,i+1/2,j,k} - \mathcal{E}_{q,i-1/2,j,k})$
12         $\delta_{\ell ijk} \leftarrow \delta_{\ell ijk} + \frac{1}{\Delta y} w_q m_{q\ell}(\mathcal{E}_{q,i,j+1/2,k} - \mathcal{E}_{q,i,j-1/2,k})$
13         $\delta_{\ell ijk} \leftarrow \delta_{\ell ijk} + \frac{1}{\Delta z} w_q m_{q\ell}(\mathcal{E}_{q,i,j,k+1/2} - \mathcal{E}_{q,i,j,k-1/2})$
14     end
15 end
16 Copy $\boldsymbol{\delta}_{ijk}$ for all $i, j, k$ from GPU to CPU
17 return $\boldsymbol{\delta}_{ijk}$ for all spatial indices $i, j, k$
```

Acceleration results. As shown in Table 3, GPU acceleration of the flux calculation is significant. Results are given for the smooth initial condition with 40^3 spatial cells and 20 time steps. The times given only include time elapsed inside the `flux_mn` function. The GPU acceleration ranges from 3x to almost 8x faster depending on the number of moments. The quadrature size does not seem to have much of an impact on the rate of GPU acceleration given our implementation.

N	n_g	OpenMP	CUDA	Speedup
3	12	24.537s	8.169s	3.00x
3	20	68.289s	22.109s	3.09x
9	20	438.341s	57.113s	7.67x
9	28	858.825s	110.182s	7.79x

Table 3: Timings for the function `flux_mn`, using OpenMP vs CUDA.

3.2 Evaluating the Hessian using real Gaunt coefficients

To make the Hessian evaluation more efficient, we have employed an optimization using real Gaunt coefficients [22]. In a one-dimensional spatial setting, $\mathbf{m}$ reduces to a basis of $N + 1$ Legendre polynomials. It was shown in [25] that for this case, the Hessian can be iteratively constructed using only the first N moments by exploiting integration by parts. The same technique can be used in a Cartesian product of closed intervals (i.e. a closed rectangular prism) and a tensor product basis. However, when using spherical harmonics on the unit sphere, a similar scheme does not appear to be possible.

The entries of the Hessian matrix H from Algorithm 1 are given by

$$H_{\ell_1\ell_2} = \sum_{q=1}^{Q} m_{\ell_1 q} m_{\ell_2 q} T_q, \tag{16}$$

where $m_{\ell q} = m_\ell(\Omega_q)$, $\mathbf{m}_q = \mathbf{m}(\Omega_q) = (m_{1q}, \dots, m_{Mq})$, and $T_q(\boldsymbol{\alpha}) = w_q \exp\left(\boldsymbol{\alpha}^T \mathbf{m}_q\right)$. For fixed $\boldsymbol{\alpha}$, $T_q(\boldsymbol{\alpha})$ needs to be computed only once. Hence, evaluation of all the entries in $H(\boldsymbol{\alpha})$ using (16) requires $O(QM^2)$ flops (see Table 4).

The evaluation of H using Gaunt coefficients relies on the fact that the product of any two spherical harmonics is given by [22]

$$m_{\ell_1} m_{\ell_2} = \sum_{\ell_3=1}^{\hat{M}} \beta_{\ell_1\ell_2\ell_3} \hat{m}_{\ell_3} \ . \tag{17}$$

Here $\beta_{\ell_1\ell_2\ell_3}$ are the real Gaunt coefficients, $\hat{\mathbf{m}}$ is a vector of real spherical harmonics of degree up to and including $2N$, and $\hat{M} = (2N+1)^2$.[7] Combining (16) and (17) gives

$$H_{\ell_1\ell_2} = \sum_{\ell_3=1}^{\hat{M}} \beta_{\ell_1\ell_2\ell_3} \sum_{q=1}^{Q} \hat{m}_{\ell_3 q} T_q. \tag{18}$$

One can further optimize the calculation (18) using the fact that β is sparse. As shown in Appendix C, for ℓ_1, ℓ_2 fixed, $\beta_{\ell_1\ell_2\ell_3}$ has at most $\sqrt{\hat{M}} = 2N+1$ nonzeros. To store β sparsely, we treat β as a matrix with the indices ℓ_1, ℓ_2 giving the row index and ℓ_3 giving the column index. Then we store β in the commonly used "compressed sparse row" (CSR) format for sparse matrices [42, p. 90].

Table 4 gives the floating point operation counts for computing the Hessian using (16) and using (18) with both full and sparse formats. For the cross-product quadrature (cf. Section 2.2), typically $Q \gg M$, and the use of the full/sparse Gaunt coefficients speeds up computations considerably. In particular, evaluating the flux matrix $\langle \Omega \mathbf{m} \mathcal{E}(\mathbf{u}) \rangle$ exactly for $\mathcal{E}(\mathbf{u}) \in \operatorname{span}\{m_1, \dots, m_M\}$ requires $Q = 2(N+1)^2 \approx 2M \gg \sqrt{\hat{M}}$. Since $\mathcal{E}(\mathbf{u}) \notin \operatorname{span}\{m_1, \dots, m_M\}$ in general, it is typical to have $Q > 2(N+1)^2$. As a rule of thumb, we also use an even value of n_g (see quadrature section of Section 2.2) for symmetry concerns. We note that although the sparse Gaunt optimization always uses many fewer flops to compute the Hessian, the sparsity requires indexing into memory in a nonconsecutive way. Therefore, computational speedup is not always as large as the flop ratio.

[7]If m_{ℓ_1} has degree d_1 and m_{ℓ_2} has degree d_2, then the index ℓ_3 in the sum need only go from 1 to $(d_1+d_2+1)^2$. To have one compact notation, we set $\hat{M}$ to the maximum possible value which occurs for degree N spherical harmonics.

	No Gaunt	Full Gaunt	Sparse Gaunt
Order	$O(QM^2)$	$O(\hat{M}M^2)$	$O(\sqrt{\hat{M}}M^2)$
Sums	$QM + QM^2$	$QM + Q\hat{M} + M^2\hat{M}$	$QM + Q\hat{M} + M^2\sqrt{\hat{M}}$
Mult	$Q + QM + 2QM^2$	$Q + QM + Q\hat{M} + M^2\hat{M}$	$Q + QM + Q\hat{M} + M^2\sqrt{\hat{M}}$
Exp	Q	Q	Q

Table 4: Floating point operation counts for the three methods of computing the Hessian H. "No Gaunt" means H is computed by (16); "Full Gaunt" means H is computed by (18) without using the sparsity of β; "Sparse Gaunt" means H is computed by (18) using the sparsity of β.

Gaunt coefficients results. In Table 5, results are shown for the various Gaunt coefficients optimizations with both the CPU and GPU implementations. The timing results are for 20 time steps with the smooth initial condition on a 20^3 domain with a batch size of 300 for the GPU implementation. (Batching on the GPU is described in Section 3.3.) The times given only include time elapsed inside the `min` function. For a smaller number of moments, there is not much difference in computation time between the full and sparse Gaunt optimization. However, as the number of moments increases, the benefit of the sparse Gaunt optimization increases. This is true for both the CPU and GPU implementations. The GPU version always outperforms the CPU, particularly with a larger number of quadrature points.

		No Gaunt		Full Gaunt		Sparse Gaunt	
N	n_g	CPU	GPU	CPU	GPU	CPU	GPU
3	12	12.4s	8.6s	8.7s	7.3s	8.1s	7.1s
3	20	29.9s	12.2s	22.7s	7.9s	20.9s	7.8s
9	20	576.5s	294.0s	491.3s	138.8s	98.2s	71.5s
9	28	1107.0s	510.1s	580.3s	146.2s	167.8s	79.2s

Table 5: CPU/GPU results for various Gaunt coefficients optimizations.

3.3 GPU acceleration via batch Hessian computations

There are several possible strategies for accelerating the minimization algorithm (Algorithm 1) with the GPU. One approach is to implement Algorithm 1 completely on the GPU. However, the performance of the algorithm is degraded in this case because the minimization algorithm as a whole is not easily vectorized. A second approach is to call the GPU every time a Hessian evaluation is required, since this is the most expensive part of the calculation. Such an implementation, however, results in frequent data transfers to and from the GPU. Thus, except for the case of large numbers of moments, there is not enough work for the GPU per function call to achieve good performance. An approach that yields good performance gains for both small and large moment orders is to batch solve several Hessian calculations, associated to different moments $\mathbf{u}_{ijk}$ of the mesh, at once on the GPU.

To implement the batching strategy, we separate each Newton iteration into the evaluation of the Hessian and the remainder of the iteration. We assemble a batch of independent minimization problems—possibly in different iterations of the algorithm—that all require a Hessian evaluation. The entire batch of Hessians is evaluated in parallel on the GPU and then moved to the CPU to complete the rest of each iteration. This process is detailed in Algorithm 7 and depicted in Figure 2.

Algorithm 7: Batch optimization solver

```
   Parameters: batchsize
1  Put all the minimization problems into Queue 1 in initialized state
2  while Queue 1 and Queue 2 are not empty do
3      Compute min(batchsize, Queue 1 length) Hessians from Queue 1 on the GPU
4      Put the batch of minimization problems into Queue 2
5      foreach Minimization problem in Queue 2 do
6          Complete the Newton iteration on the CPU
7          if Minimization problem in Queue 2 meets tolerance then
8              Put the minimization problem into Queue 3
9          else
10             Put the minimization problem back into Queue 1
11         end
12     end
13 end
```

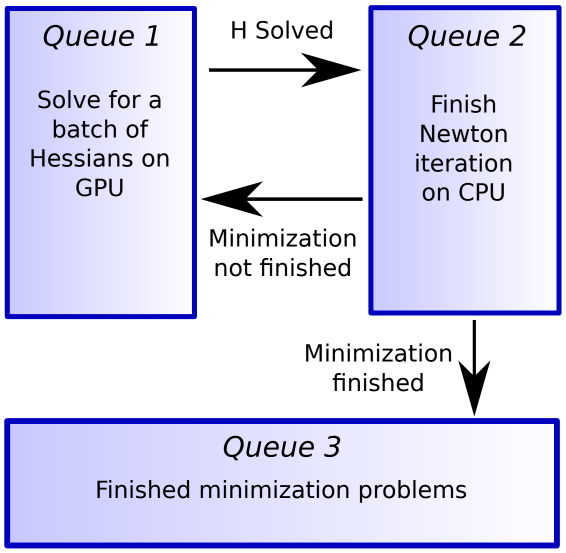
Queue 1
Solve for a batch of Hessians on GPU
H Solved
Queue 2
Finish Newton iteration on CPU
Minimization not finished
Minimization finished
Queue 3
Finished minimization problems

Figure 2: Figure describing the batch minimization algorithm with GPU acceleration.

Batch Hessian results. The batch size for the Hessian computations must be chosen large enough to give the GPU enough work to do. In Table 6, timing results are given for different batch sizes. The times given only include time elapsed inside the `min` function. Results are given for the smooth initial condition with 40^3 spatial cells and 20 time steps while using the sparse Gaunt optimization. In general, the compute time is a decreasing function of batch size.

N	n_g	CPU	GPU-25	GPU-50	GPU-100	GPU-200	GPU-400	GPU-800	GPU-1600
3	12	112.8s	122.1s	93.6s	79.4s	73.9s	69.9s	67.1s	65.7s
3	20	238.0s	167.4s	118.0s	95.2s	82.3s	75.9s	72.1s	70.2s
9	20	927.6s	999.3s	848.6s	755.1s	724.5s	714.7s	710.3s	708.3s
9	28	1620.7s	1190.7s	941.3s	846.4s	797.4s	782.6s	778.5s	774.7s

Table 6: Results using different batch sizes for calculating the Hessian on the GPU. GPU-B represents the time elapsed using the GPU with a batch size of B.

4 Statistics and scaling

In this section, we present overall program results including (i) convergence results confirming second-order space-time accuracy; (ii) composite acceleration results from the combined improvements in Section 3; (iii) statistics on the number of iterations needed to solve the minimization problems; (iv) extensive timing data for a large scale M_3 computation; and (v) weak scaling results for P_3 and M_3. All computations were performed on the supercomputer Titan.

Convergence. The convergence rate was tested using the smooth initial condition (14). The reference solution was computed using a spatial resolution of $n = 320$, and the time step used for all calculations was $\Delta t = \frac{0.9}{6}\Delta x = 0.15\Delta x$ (see (13) and note $\Delta x = \Delta y = \Delta z$).

Table 7 shows L^2 spatial convergence data at time $t = 0.3$. The table shows convergence data for moments of degree $d = 0, 1, 2, 3$. Since there are actually $2d + 1$ moments of degree d, we evaluate

$$e_d = \left(\sum_{r=-d}^{d} ||u^r_{d,320} - u^r_{d,n}||_2^2 \right)^{1/2}, \tag{19}$$

where $u^r_{d,n}$ is the degree d, order r moment of the numerical solution, computed on the spatial mesh of size n. The results in Table 7 confirm second-order temporal and spatial convergence.

n	e_0	Conv Rate	e_1	Conv Rate	e_2	Conv Rate	e_3	Conv Rate
10	1.22e-4	–	4.28e-4	–	5.46e-4	–	4.70e-4	–
20	3.79e-5	1.69	1.10e-4	1.97	1.29e-4	2.08	9.23e-5	2.35
40	1.38e-5	1.45	2.39e-5	2.20	2.86e-5	2.17	2.53e-5	1.87
80	4.35e-6	1.67	5.43e-6	2.14	5.64e-6	2.34	5.89e-6	2.10
160	7.88e-7	2.47	9.38e-7	2.53	1.04e-6	2.44	1.22e-6	2.27

Table 7: Convergence data for M_3 and smooth initial condition (14).

Algorithm acceleration. We next explore the combined improvement of the optimizations discussed in Section 3. We use both the smooth (14) and Gaussian (15) initial conditions with a 20^3 spatial mesh and a batch size of 300 for the Hessian calculations. The results are given in Table 8. The overall speedups for both initial conditions increase from 1.8x to 13x, depending on the number of moments and the number of quadrature points. As these increase, the optimizations become more effective.

IC	N	n_g	Before	After	Speedup
Smooth	3	12	15.417s	8.369s	1.84x
Smooth	3	20	38.685s	11.272s	3.43x
Smooth	9	20	598.694s	86.116s	6.95x
Smooth	9	28	1265.605s	97.635s	12.96x
Gaussian	3	12	9.146s	4.932s	1.85x
Gaussian	3	20	23.199s	7.631s	3.04x
Gaussian	9	20	354.984s	40.697s	8.72x
Gaussian	9	28	692.014s	50.728s	13.64x

Table 8: Overall improvement in time to solution for M_N simulations with smooth (14) and Gaussian (15) initial conditions.

The first row in Table 8 can be predicted given previous data in the paper. In Table 1, for M_3 with $n_g = 12$, the function `min` takes approximately 80% of the calculation time and `flux_mn` takes approximately 20% of the time. From Table 3, we expect a speedup of 3x for `flux_mn`, and from Table 5, we expect a speedup of $12.4/7.1 = 1.75$x for `min`. Therefore, the estimated total speedup is $\frac{100}{80/1.75+20/3}$ or 1.91x, which is quite close to the 1.84x speedup actually observed. The speedup results in other rows could theoretically be determined in the same manner, but an initial profile was not created for those cases.

Minimization iterations. The function `min` is the only function that may vary significantly in time to completion between different spatial cells. This happens because the number of Newton iterations needed for the algorithm to converge changes from cell to cell. To see this disparity, we plot in Figure 3 the number of iterations taken by `min` to converge for each spatial cell on the x-axis. A spatial mesh with 100^3 cells is used. For the smooth initial condition (14), 25 time steps were taken, and for the Gaussian initial condition (15), 167 time steps were taken. For the smooth initial condition, the first time step requires more iterations than any other time step because of the anisotropy of the initial condition. After this, the algorithm uses the previous data in the `min` function and only 1 or 2 minimization iterations are required for convergence. For the Gaussian initial condition, the problem is always difficult to solve on the expanding wave front, requiring in some cases more than 20 iterations for convergence of the minimization algorithm. (Note: there are twice as many Euler steps shown in Figure 3 as there are time steps since the Heun method takes two Euler steps.)

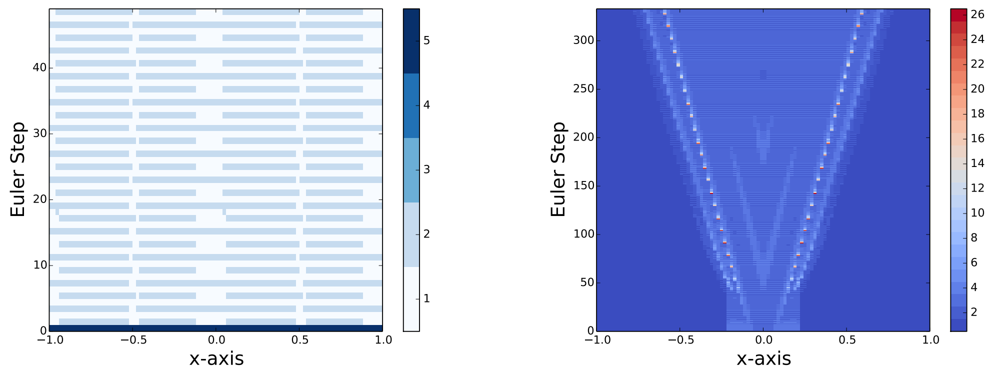

Figure 3: Plot of minimization iterations required for convergence for spatial cells on the x-axis for each Euler step. Left: smooth initial condition (14). Notice the first Euler step requires 5 minimization iterations to converge. Right: Gaussian initial condition (15). Notice the large number of minimization iterations taken on the expanding wave front.

M_3 statistics at scale. The variation in time taken by the `min` function for different spatial cells can cause load balancing problems for the M_N method that do not occur for the P_N method. To quantify this effect, we present timing data using the smooth initial condition (14) and a large parallel calculation: 26^3 nodes with 100^3 spatial cells per node. We present the elapsed time (`time`) for each of the four functions (`comm`, `min`, `flux_mn`, and `euler`) at each compute node (`node`) and each time step (`step`). The data has the form:

$$\texttt{time} = \texttt{function}(\texttt{node}, \texttt{step}). \tag{20}$$

When the stage of the time integrator from Algorithm 2 is important, these functions are separated into pairs: `comm1`/`comm2`, `min1`/`min2`, `flux1`/`flux2`, and `euler1`/`euler2`.

In Table 9, we aggregate the timing data, first summing the elapsed time for each function across time steps and then taking the mean and standard deviation of these elapsed times across nodes. We also present a normalized standard deviation calculated as standard deviation divided by mean.

Of particular importance is that the time for `comm` is not just the raw communication time; rather, it also includes the time spent waiting for data to be received. This is why the time for `comm` for M_3 is so much larger than for P_3. The standard deviation in time to compute a solution for M_3 is larger than for P_3, which causes more waiting time for receipt of data in certain nodes. A more detailed view of the timing data for M_3 with $n = 100$ is given in Figure 4, which shows the mean node timing of each function vs time step.

	n	Mean	Std	NStd
P_3	25	0.66s	0.22s	0.328
P_3	50	2.82s	0.83s	0.294
P_3	100	15.50s	7.29s	0.470
M_3	25	2.60s	1.83s	0.704
M_3	50	13.10s	12.00s	0.916
M_3	100	100.87s	88.49s	0.877

(a) `comm` timings

	n	Mean	Std	NStd
P_3	25	–	–	–
P_3	50	–	–	–
P_3	100	–	–	–
M_3	25	36.07s	1.81s	0.050
M_3	50	228.43s	12.12s	0.053
M_3	100	1643.43s	89.54s	0.054

(b) `min` timings

	n	Mean	Std	NStd
P_3	25	0.97s	0.15s	0.156
P_3	50	6.78s	0.87s	0.128
P_3	100	52.97s	7.37s	0.139
M_3	25	7.97s	0.11s	0.014
M_3	50	30.74s	0.31s	0.010
M_3	100	175.03s	0.58s	0.003

(c) `flux` timings

	n	Mean	Std	NStd
P_3	25	0.02s	0.001s	0.066
P_3	50	0.20s	0.007s	0.036
P_3	100	2.45s	0.017s	0.007
M_3	25	0.07s	0.006s	0.093
M_3	50	0.46s	0.038s	0.084
M_3	100	3.84s	0.300s	0.078

(d) `euler` timings

Table 9: P_3/M_3 timings on 26^3 nodes. (Std: standard deviation; NStd: normalized standard deviation.)

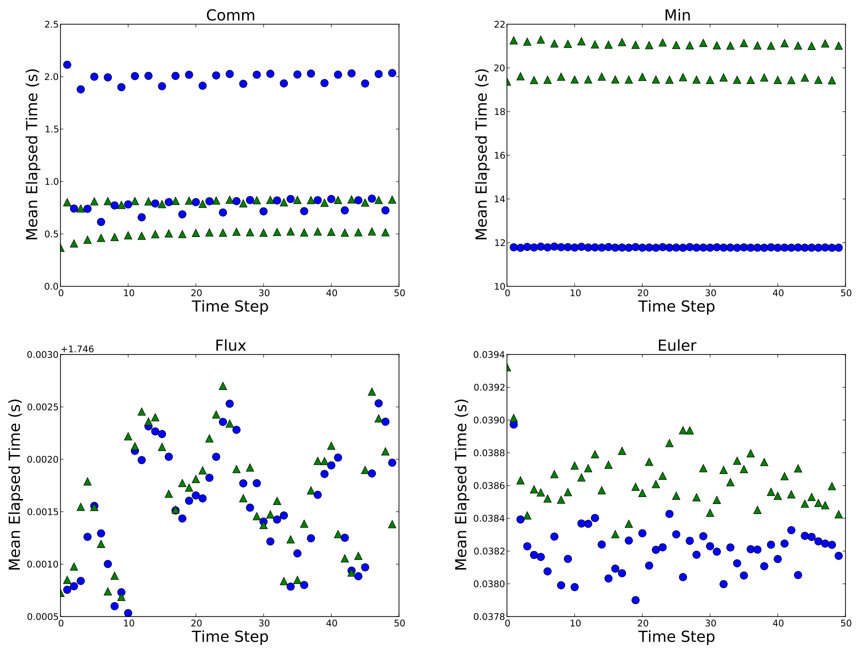

Figure 4: Mean node time is shown. Blue circles represent the first Euler step and green triangles represent the second Euler step.

The most striking feature of the data is that `comm1` takes longer than `comm2`, while `min2` takes longer than `min1`. We conjecture that there is a cause and effect relationship here in which the second `min` function takes longer to compute and more importantly has a higher standard deviation. This higher standard deviation then causes blocking of the communication to the first `comm` function. From Figure 3, we can see why `min2` takes longer to compute than `min1`. The second Euler step requires on average 2 iterations to converge whereas the first Euler step requires on average only 1 iteration to converge. This may occur because the initial guess used for the minimization algorithm at the beginning of a Heun step uses the result of the previous Euler step ($s = 1$ from Algorithm 2), which is an approximation of the solution at the current time step.

Weak scaling results. The overall goal of [2, 3] and the work presented here is to make M_N more competitive with P_N in time to solution, specifically at scale. Therefore, we performed weak scaling tests for P_3 and M_3 on Titan using the smooth initial condition (14). We ran the scaling tests for 10^3, 25^3, 50^3, and 100^3 spatial cells per node from 1 node to $26^3 = 17576$ nodes. GPU acceleration was used to speed up the M_3 calculations for both the `flux_mn` and `min` functions. A batch size of 300 was used for the Hessian computations, and sparse Gaunt coefficients were used. Figure 5 shows the weak scaling results, and Table 10 gives the difference in time to solution for a single node and for the largest scale computations. Table 11 compares the flop rates on one node for P_3 and M_3 flux computations.[8]

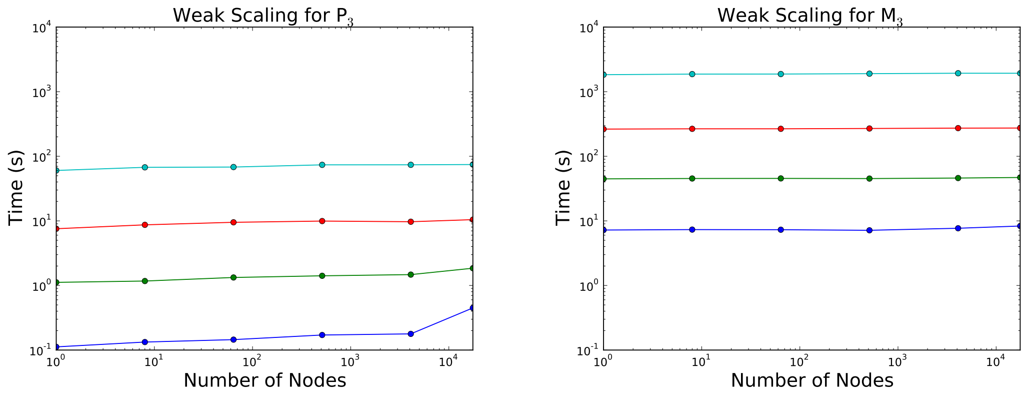

Figure 5: Weak scaling results (lines bottom to top in spatial cells per node: 10^3, 25^3, 50^3, 100^3) for (left) P_3, (right) M_3.

	1 node			17576 nodes		
Size	P_3	M_3	$\mathrm{M}_3/\mathrm{P}_3$	P_3	M_3	$\mathrm{M}_3/\mathrm{P}_3$
10^3	0.11s	7.21s	64.4x	0.45s	8.32s	18.5x
25^3	1.12s	44.68s	40.0x	1.85s	46.94s	25.4x
50^3	7.55s	263.45s	34.9x	10.44s	273.57s	26.2x
100^3	60.31s	1831.37s	30.4x	74.47s	1928.03s	25.9x

Table 10: Table comparing time to solution for 1 node and 17576 nodes. Size is the number of spatial cells per node.

[8]Flop rates are similar per node up to 26^3 nodes.

n	P_3 Flop Rate	M_3 Flop Rate
10	6.32 GFLOPS	2.00 GFLOPS
25	6.31 GFLOPS	9.57 GFLOPS
50	7.46 GFLOPS	21.83 GFLOPS
100	7.48 GFLOPS	32.44 GFLOPS

Table 11: The `flux_pn` and `flux_mn` flop rates for one node. Note the flop rates never reach peak performance because the computations are memory bandwidth limited.

We observe that the P_3 solution time does increase with the number of nodes, but only for a very small problem size (10^3 cells per node) does the simulation become communication dominated. In this case, the M_3 calculation becomes more competitive at scale, but the P_3 simulation is still about 25 times faster. Thus for the same time to solution, the P_3 simulation with the current numerical method can afford a spatial resolution that is 2–3 times finer than the corresponding M_3 simulation at the largest scale. Depending on the problem at hand, M_N may be a better closure at any scale if Gibbs phenomena in the angular approximation strongly pollutes the P_N solution and, at the same, less spatial resolution can be tolerated. We also note the disparity in the flop rates between the M_3 and P_3 implementations due to the use of CUDA for M_3 and OpenMP for P_3. Clearly, there is room for improvement in the implementation of the P_N method if GPU acceleration is used to accelerate the `flux_pn` function.

5 Conclusions and discussion

Several optimizations were shown to accelerate the implementation of the entropy-based model M_N. The use of Gaunt coefficients, in conjunction with GPU acceleration strategies, reduces the time to solution for a given test problem between 1.8x and 13x. We also computed performance statistics for M_N, and it was shown that M_N can cause some load balancing issues at scale. Finally, we compared weak scaling results of the P_N and M_N algorithms. For our test problems, the ratio of M_N to P_N in time to solution is reduced from 64x on 1 node to 18x on 17576 nodes for the smallest problem per node and from 30x on 1 node to 26x on 17576 nodes for the largest problem size per node.

We conjecture that the weak scaling results presented here provide a reasonable lower bound of what can be expected for the relative efficiency of the M_N calculation. One reason for this is the simplicity of the model. Indeed, most increases in model complexity, such as a more realistic collision operator, will require a similar increase in floating-point operations and memory resources for both P_N and M_N models, thereby decreasing the ratio in time to solution of M_N relative to P_N. A second reason is the simplicity of the explicit time integrator. Indeed, for many applications, an implicit time integration scheme is desired to allow for large time steps. Such schemes involve global communication of data between all nodes, as opposed to the communication of halo data between adjacent nodes that is characteristic of explicit schemes. An implicit time integration scheme would require a linear global solve for the P_N equations, but a global nonlinear solve for the M_N equations. Thus the communication costs for M_N are likely to be higher. However, as both methods become increasingly communication-bound, the ratio of time to solution between them will continue to decrease. In addition, the use of iterative solvers means that one could, in many cases, solve the M_N optimization problem rather coarsely in earlier iterations.

In a larger context, the scaling results in Figure 5 and Table 10 give some insight into algorithm development for emerging architectures at extreme scales. It is generally expected [37] that these new systems will be increasingly limited—both in terms of speed and cost—by memory storage and data transfer. The Titan supercomputer is just the very beginning of this evolution. New systems will lend themselves to algorithms that are arithmetically intensive, i.e., where the number of floating point operations per unit of memory is large. Thus, in order to gain accuracy in a simulation, it is natural to ask whether, at some point, it is more efficient to move from a simple, linear model to a complicated, highly nonlinear model (where more floating point operations are required) instead of simply increasing the number of unknowns (where more memory is required). The choice between P_N and M_N exemplifies this choice, and the scaling results tell us two things. First, the data transfer capabilities of Titan are still good enough to give P_N a distinct advantage in time to solution, even for full-scale simulations. Second, however, one can clearly observe the slowdown caused by memory movement at the very highest scales, particularly when the memory per node is very low. While this amount of memory per node is unreasonably low, it is important to note that the ratio of memory size to number of floating point units will continue to decrease as the system size increases.

One important comparison between M_N and P_N that has not been done here is an analysis of efficiency that compares the efficiency of the two methods, both in terms of time and energy, for realistic physics problems. For a given problem, one should determine for each method what value of N is needed to attain a prescribed level of accuracy. Based on the two values of N, one should calculate in each case the time to solution and the energy costs and then make comparisons. Such a project is currently work in progress.

A Real spherical harmonics

We used the real spherical harmonics as an orthonormal basis for $L^2(\mathbb{S}^2)$. For $d \geq 0$ and $-d \leq r \leq d$, the degree d and order r normalized, real spherical harmonic is given by

$$R_d^r(\theta,\phi) = \begin{cases} \sqrt{2}N_d^{-r}P_d^{-r}(\cos\theta)\sin(r\phi), & r<0 \\ N_d^0 P_d(\cos\theta), & r=0 \\ \sqrt{2}N_d^r P_d^r(\cos\theta)\cos(r\phi), & r>0 \end{cases} \tag{21}$$

where

$$N_d^r = \sqrt{\frac{(2d+1)}{4\pi}\frac{(d-r)!}{(d+r)!}}\,; \tag{22}$$

$$P_d^r(\mu) = (-1)^r(1-\mu^2)^{r/2}\frac{d^r}{d\mu^r}P_d(\mu), \quad r>0; \tag{23}$$

and P_d is the degree d Legendre polynomial, normalized so that $\int_{-1}^{1} P_d(\mu)d\mu = \frac{2}{2d+1}$. The spherical harmonics are packed into $\mathbf{m}$ as

$$\mathbf{m} = (R_0^0, R_1^{-1}, R_1^0, R_1^1, \ldots, R_N^{-N}, \ldots, R_N^N). \tag{24}$$

B Solving for the real Gaunt coefficients

To solve for the real Gaunt coefficients $\beta_{\ell_1\ell_2\ell_3}$, notice that (17) must be true for all points and in particular the quadrature points on the sphere described in Section 2.2. Therefore, for $q = 1\ldots Q$,

$$m_{\ell_1 q}m_{\ell_2 q} = \sum_{\ell_3=1}^{\hat{M}} \beta_{\ell_1\ell_2\ell_3}\hat{m}_{\ell_3 q}. \tag{25}$$

Taking ℓ_1 and ℓ_2 to be constant in the above equation, this is just an overdetermined linear system (assuming $Q > \hat{M}$). Denote the left hand side vector of (25) as $\mathbf{v}^{(\ell_1\ell_2)}$ (has length Q), the matrix given by $\hat{m}_{\ell_3 q}$ as A (has size $Q \times \hat{M}$), and the vector given by $\beta_{\ell_1\ell_2\ell_3}$ as $\mathbf{b}^{(\ell_1\ell_2)}$ (has length $\hat{M}$). Still with ℓ_1, ℓ_2 constant, this yields the linear system:

$$\mathbf{v}^{(\ell_1\ell_2)} = A\mathbf{b}^{(\ell_1\ell_2)}. \tag{26}$$

We solve the system as a least squares problem

$$\min_{\mathbf{b}^{(\ell_1\ell_2)}\in\mathbb{R}^{\hat{M}}} ||\mathbf{v}^{(\ell_1\ell_2)} - A\mathbf{b}^{(\ell_1\ell_2)}||_2 \tag{27}$$

using the method of computing a QR decomposition. Let $A = UR$, where U is an orthogonal matrix and R is an upper triangular matrix. Then

$$||\mathbf{v}^{(\ell_1\ell_2)} - A\mathbf{b}^{(\ell_1\ell_2)}||_2 = ||\mathbf{v}^{(\ell_1\ell_2)} - UR\mathbf{b}^{(\ell_1\ell_2)}||_2 = ||U^T\mathbf{v}^{(\ell_1\ell_2)} - R\mathbf{b}^{(\ell_1\ell_2)}||_2, \tag{28}$$

where minimizing over the last expression is trivial. Since A does not involve ℓ_1 or ℓ_2, only one QR decomposition is needed to solve the system over and over again for the different ℓ_1, ℓ_2. Also remember, the real Gaunt coefficients only need to be computed once at the beginning of the program.

C Sparsity result for the real Gaunt coefficients

For the real spherical harmonics,

$$m_{\ell_1} m_{\ell_2} = \sum_{\ell_3=1}^{\hat{M}} \beta_{\ell_1\ell_2\ell_3}\hat{m}_{\ell_3}, \tag{29}$$

where $\hat{M} = (2N+1)^2$, $\mathbf{m}$ contains spherical harmonics of degree less than or equal to N, and $\hat{\mathbf{m}}$ contains spherical harmonics of degree less than or equal to $2N$.

Proposition 1. *For fixed ℓ_1, ℓ_2, the sum in* (29) *has no more than $2N+1$ nonzero elements in it.*

Proof. The real spherical harmonics of degree d can be given in terms of the complex spherical harmonics via the unitary transformation [22]

$$\begin{pmatrix} R_d^d \\ \vdots \\ R_d^1 \\ R_d^0 \\ R_d^{-1} \\ \vdots \\ R_d^{-d} \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & & & & & & 1 \\ & \ddots & & & & \iddots & \\ & & 1 & & 1 & & \\ & & & \sqrt{2} & & & \\ & & i & & -i & & \\ & \iddots & & & & \ddots & \\ i & & & & & & -i \end{pmatrix} \begin{pmatrix} Y_d^d \\ \vdots \\ Y_d^1 \\ Y_d^0 \\ Y_d^{-1} \\ \vdots \\ Y_d^{-d} \end{pmatrix}, \tag{30}$$

where the complex spherical harmonics are defined as

$$Y_d^r(\theta,\phi) = N_d^{|r|} P_d^{|r|}(\cos\theta) e^{ir\phi}, \tag{31}$$

with N_d^r and P_d^r defined in Appendix A. The transformation (30) and its inverse can be written as

$$R_d^r = a_r Y_d^r + b_r Y_d^{-r} \tag{32}$$
$$Y_d^r = \bar{a}_r R_d^r + \bar{b}_{-r} R_d^{-r} \tag{33}$$

where

$$a_r = \frac{1}{\sqrt{2}} \begin{cases} 1, & r > 0 \\ \frac{\sqrt{2}}{2}, & r = 0 \\ -i, & r < 0 \end{cases} \quad \text{and} \quad b_r = \frac{1}{\sqrt{2}} \begin{cases} 1, & r > 0 \\ \frac{\sqrt{2}}{2}, & r = 0 \\ i, & r < 0 \end{cases} \tag{34}$$

and $\bar{a}$ denotes the complex conjugate of a.

Now, multiplying two real spherical harmonics and expanding yields

$$\begin{aligned} R_{d_1}^{r_1} R_{d_2}^{r_2} =& (a_{r_1} Y_{d_1}^{r_1} + b_{r_1} Y_{d_1}^{-r_1})(a_{r_2} Y_{d_2}^{r_2} + b_{r_2} Y_{d_2}^{-r_2}) \\ =& a_{r_1} a_{r_2} Y_{d_1}^{r_1} Y_{d_2}^{r_2} + a_{r_1} b_{r_2} Y_{d_1}^{r_1} Y_{d_2}^{-r_2} \\ &+ b_{r_1} a_{r_2} Y_{d_1}^{-r_1} Y_{d_2}^{r_2} + b_{r_1} b_{r_2} Y_{d_1}^{-r_1} Y_{d_2}^{-r_2}. \end{aligned} \tag{35}$$

Each product of two complex spherical harmonics can be written as the following linear combination of complex spherical harmonics using (complex) Gaunt coefficients [41]

$$Y_{d_1}^{r_1} Y_{d_2}^{r_2} = \sum_{\substack{d=0 \\ 2|(d+d_1+d_2)}}^{d_1+d_2} G_{d,d_1,d_2}^{r_1,r_2} Y_d^{r_1+r_2}. \tag{36}$$

In the sum (36), we use the notation $2|(d+d_1+d_2)$ to represent $d+d_1+d_2 \equiv 0 \pmod 2$. Notice how we only have to sum over very few complex spherical harmonics. Now, putting together equations (35) and (36) yields

$$\begin{aligned} R_{d_1}^{r_1} R_{d_2}^{r_2} = \sum_{\substack{d=0 \\ 2|(d+d_1+d_2)}}^{d_1+d_2} \Big(& a_{r_1} a_{r_2} G_{d,d_1,d_2}^{r_1,r_2} Y_d^{r_1+r_2} \\ & + a_{r_1} b_{r_2} G_{d,d_1,d_2}^{r_1,-r_2} Y_d^{r_1-r_2} \\ & + b_{r_1} a_{r_2} G_{d,d_1,d_2}^{-r_1,r_2} Y_d^{-r_1+r_2} \\ & + b_{r_1} b_{r_2} G_{d,d_1,d_2}^{-r_1,-r_2} Y_d^{-r_1-r_2} \Big). \end{aligned} \tag{37}$$

Label the four parts of the summand from (37) as (I), (II), (III), and (IV). Then expanding the right side of equation (37) back into real spherical harmonics using (33) and using the fact

$$G_{d,d_1,d_2}^{r_1,r_2} = G_{d,d_1,d_2}^{-r_1,-r_2} \tag{38}$$

yields

$$\begin{array}{ll} (I): & G_{d,d_1,d_2}^{r_1,r_2} (a_{r_1} a_{r_2} \bar{a}_{r_1+r_2} + b_{r_1} b_{r_2} \bar{b}_{r_1+r_2}) R_d^{r_1+r_2} \\ (II): & G_{d,d_1,d_2}^{r_1,-r_2} (a_{r_1} b_{r_2} \bar{a}_{r_1-r_2} + b_{r_1} a_{r_2} \bar{b}_{r_1-r_2}) R_d^{r_1-r_2} \\ (III): & G_{d,d_1,d_2}^{-r_1,r_2} (b_{r_1} a_{r_2} \bar{a}_{-r_1+r_2} + a_{r_1} b_{r_2} \bar{b}_{-r_1+r_2}) R_d^{-r_1+r_2} \\ (IV): & G_{d,d_1,d_2}^{-r_1,-r_2} (b_{r_1} b_{r_2} \bar{a}_{-r_1-r_2} + a_{r_1} a_{r_2} \bar{b}_{-r_1-r_2}) R_d^{-r_1-r_2}. \end{array} \tag{39}$$

The coefficients of R in (I), (II), (III), and (IV) yield the real Gaunt coefficients except when $r_1 + r_2 = 0$ or $r_1 - r_2 = 0$. In these cases, either (I)+(IV) or (II)+(III) yields the real Gaunt coefficient.

Comparing the expressions $a_{r_1}a_{r_2}\bar{a}_{r_1+r_2}$ and $b_{r_1}b_{r_2}\bar{a}_{-r_1-r_2}$ from (I) and (IV) respectively, if $r_1 + r_2 \neq 0$, then exactly one of these expressions is real and the other one is purely imaginary and nonzero. The same is true for $b_{r_1}b_{r_2}\bar{b}_{r_1+r_2}$ and $a_{r_1}a_{r_2}\bar{b}_{-r_1-r_2}$. Since the coefficients of $R_d^{r_1+r_2}$ and $R_d^{-r_1-r_2}$ must be real, only one of (I) and (IV) is nonzero. In the case $r_1 + r_2 = 0$, $R_d^{r_1+r_2} = R_d^{-r_1-r_2} = R_d^0$ and the coefficients are added to form the real Gaunt coefficient. This same parity occurs for (II) and (III). Hence, (I), (II), (III), and (IV) represent at most two nonzero real Gaunt coefficients.

The maximum possible number of nonzero real Gaunt coefficients occurs when $d_1 = d_2 = N$, which gives the permissible ℓ values in sum (37) of

$$d = 0, 2, \ldots, 2N. \tag{40}$$

For $d = 0$, the only possible value of m is zero yielding at most 1 real Gaunt coefficient. For $d = 2, 4, \ldots, 2N$, there are two possible nonzero real Gaunt coefficients, which yields a total of $2N + 1$ possible real Gaunt coefficients. This possibility is achieved by the product $R_N^1 R_N^1$. $\square$

D Variables

n	-	Number of spatial cells in a Cartesian direction (n^3 total cells).
N	-	Maximal degree of moments.
M	-	Number of moments ($M = (N+1)^2$).
$\hat{M}$	-	Number of moments needed in the computation of Gaunt coefficients ($\hat{M} = (2N+1)^2$).
Q	-	Number of quadrature points in angle.
n_g	-	Number of Gauss-Legendre quadrature points on Ω_3-axis ($Q = 2n_g^2$).
$\mathbf{m}$	-	Vector of real spherical harmonics up to and including degree N.
$\hat{\mathbf{m}}$	-	Vector of real spherical harmonics up to and including degree $2N$.
i, j, k	-	Indices for spatial coordinates.
q	-	Index for quadrature in angle.
ℓ	-	Index for entry in a vector.
d, r	-	Degree and order indices respectively for spherical harmonics.

Acknowledgments

This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

This research is sponsored by the Office of Advanced Scientific Computing Research; U.S. Department of Energy. The work was performed at the Oak Ridge National Laboratory, which is managed by UT-Battelle, LLC under Contract No. De-AC05-00OR22725.

Research sponsored by the Laboratory Directed Research and Development Program of Oak Ridge National Laboratory (ORNL), managed by UT-Battelle, LLC for the U. S. Department of Energy under Contract No. De-AC05-00OR22725.

References

[1] Cory Ahrens and Gregory Beylkin. Rotationally invariant quadratures for the sphere. *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 465(2110):3103–3125, 2009.

[2] G. W. Alldredge, C. D. Hauck, and A. L. Tits. High-order entropy-based closures for linear transport in slab geometry II: A computational study of the optimization problem. *SIAM J. Sci. Comput.*, 34(4):B361–B391, 2012.

[3] Graham W. Alldredge, Cory D. Hauck, Dianne P. O'Leary, and André L. Tits. Adaptive change of basis in entropy-based moment closures for linear kinetic equations. *Journal of Computational Physics*, 258(0):489 – 508, 2014.

[4] Kendall Atkinson. Numerical integration on the sphere. *J. Austral. Math. Soc. Ser. B*, 23:332–347, 1982.

[5] Kendall Atkinson and Weimin Han. *Spherical Harmonics and Approximations on the Unit Sphere: An Introduction.* Springer-Verlag, Berlin Heidelberg, 2012.

[6] C. Berthon, M. Frank, C. Sarazin, and R. Turpault. Numerical methods for balance laws with space dependent flux: Application to radiotherapy dose calculation. *Communications in Computational Physics*, 10(5):1184, 2011.

[7] Thomas A. Brunner. Forms of approximate radiation transport. Technical Report SAND2002-1778, Sandia National Laboratories, 2002.

[8] Thomas A. Brunner and James Paul Holloway. One-dimensional Riemann solvers and the maximum entropy closure. *J. Quant Spect. and Radiative Trans*, 69(5):543 – 566, 2001.

[9] Jean-François Coulombel, François Golse, and Thierry Goudon. Diffusion approximation and entropy-based moment closure for kinetic equations. *Asymptotic Analysis*, 45(1):1–39, 2005.

[10] Raúl E. Curto and Lawrence A. Fialkow. Recursiveness, positivity and truncated moment problems. *Houston Journal of Mathematics*, 4:603–635, 1991.

[11] B. Davison. *Neutron Transport Theory.* Clarendon Press, Oxford, England, 1957.

[12] S. M. Deshpande. Kinetic theory based new upwind methods for inviscid compressible flows. In *American Institute of Aeronautics and Astronautics, New York*, 1986. Paper 86-0275.

[13] B. Dubroca and J.-L. Feugeas. Étude théorique et numérique d'une hiérarchie de modèles aux moments pour le transfert radiatif. *C.R. Acad. Sci. Paris*, I. 329:915–920, 1999.

[14] B Dubroca, J-L Feugeas, and M Frank. Angular moment model for the fokker-planck equation. *The European Physical Journal D*, 60(2):301–307, 2010.

[15] R. Duclous, B. Dubroca, and M. Frank. A deterministic partial differential equation model for dose calculation in electron radiotherapy. *Physics in medicine and biology*, 55(13):3843, 2010.

[16] C. Kristopher Garrett and Cory D. Hauck. A comparison of moment closures for linear kinetic transport equations: The line source benchmark. *Transport Theory and Statistical Physics*, 42(6-7):203–235, 2013.

[17] Thierry Goudon and Chunjin Lin. Analysis of the M1 model: Well-posedness and diffusion asymptotics. *Journal of Mathematical Analysis and Applications*, 402(2):579–593, 2013.

[18] Clinton Groth and James McDonald. Towards physically realizable and hyperbolic moment closures for kinetic theory. *Continuum Mechanics and Thermodynamics*, 21(6):467–493, 2009.

[19] A. Harten, P. D. Lax, and Van Leer. On upstream differencing and Godunov-type schemes for hyperbolic conservation laws. *SIAM Rev.*, 25:35–61, 1983.

[20] C. D. Hauck. High-order entropy-based closures for linear transport in slab geometry. *Comm. Math. Sci.*, 9:187–205, 2011.

[21] Cory D. Hauck, C. David Levermore, and André L. Tits. Convex duality and entropy-based moment closures: Characterizing degenerate densities. *SIAM J. Control Optim.*, 47(4):1977–2015, 2008.

[22] Herbert H.H. Homeier and E. Otto Steinborn. Some properties of the coupling coefficients of real spherical harmonics and their relation to Gaunt coefficients. *Journal of Molecular Structure: THEOCHEM*, 368(0):31 – 37, 1996. Proceedings of the Second Electronic Computational Chemistry Conference.

[23] M. Junk. Maximum entropy for reduced moment problems. *Math. Models Meth. Appl. Sci.*, 10(7):1001–1025, 2000.

[24] Samuel Karlin and Lloyd S Shapley. *Geometry and Moment Spaces.* RAND Corporation, 1952.

[25] Jean B Lasserre. Semidefinite programming for gradient and hessian computation in maximum entropy estimation. In *Decision and Control, 2007 46th IEEE Conference on*, pages 3060–3064. IEEE, 2007.

[26] V.I. Lebedev. Quadratures on a sphere. *USSR Computational Mathematics and Mathematical Physics*, 16(2):10 – 24, 1976.

[27] VI Lebedev. Spherical quadrature formulas exact to orders 25–29. *Siberian Mathematical Journal*, 18(1):99–107, 1977.

[28] V.I. Lebedev. A quadrature formula for the sphere of 59th algebraic order of accuracy. *Russian Acad. Sci. Dokl. Math.*, 50:283–286, 1995.

[29] VI Lebedev and DN Laikov. A quadrature formula for the sphere of the 131st algebraic order of accuracy. In *Doklady. Mathematics*, volume 59, pages 477–481. MAIK Nauka/Interperiodica, 1999.

[30] V.I. Lebedev and A.L. Skorokhodov. Quadrature formulas of orders 41, 47, and 53 for the sphere. *Russian Acad. Sci. Dokl. Math.*, 45:587–592, 1992.

[31] C. D. Levermore. Moment closure hierarchies for kinetic theory. *J. Stat. Phys.*, 83:1021–1065, 1996.

[32] E. E. Lewis and W. F. Miller, Jr. *Computational Methods of Neutron Transport.* John Wiley and Sons, New York, 1984.

[33] M. González, E. Audit, and P. Huynh. Heracles: a three-dimensional radiation hydrodynamics code. *Astronomy & Astrophysics*, 464(2):429–435, 2007.

[34] James McDonald and Manuel Torrilhon. Affordable robust moment closures for CFD based on the maximum-entropy hierarchy. *Journal of Computational Physics*, 251(0):500 – 523, 2013.

[35] Philipp Monreal and Martin Frank. Higher order minimum entropy approximations in radiative transfer. *arXiv preprint arXiv:0812.3063*, 2008.

[36] Ph. Nicolaï, J.-L. Feugeas, C. Regan, M. Olazabal-Loumé, J. Breil, B. Dubroca, J.-P. Morreeuw, and V. Tikhonchuk. Effect of the plasma-generated magnetic field on relativistic electron transport. *Phys. Rev. E*, 84:016402, Jul 2011.

[37] Department of Energy Scientific Grand Challenges Workshop Series. Architectures and technology for extreme scale computing, 2009.

[38] B. Perthame. Boltzmann type schemes for gas dynamics and the entropy property. *SIAM J. on Numer. Anal.*, 27(6):1405–1421, 1990.

[39] B. Perthame. Second-order Boltzmann schemes for compressible Euler equations in one and two space dimensions. *SIAM J. Numer. Anal.*, 29(1):1–19, 1992.

[40] G. C. Pomraning. *The Equations of Radiation Hydrodynamics.* Pergamon Press, New York, 1973.

[41] J. Rasch and A. Yu. Efficient storage scheme for precalculated Wigner 3j, 6j and Gaunt coefficients. *SIAM Journal on Scientific Computing*, 25(4):1416–1428, 2004.

[42] Yousef Saad. *Iterative Methods for Sparse Linear Systems.* Society for Industrial and Applied Mathematics, Philadelphia, PA, second edition, 2003.

[43] J. Schneider. Entropic approximation in kinetic theory. *Math. Model. Numer. Anal.*, 38:541–561, 2004.

[44] James Alexander Shohat and Jacob David Tamarkin. *The Problem of Moments.* American Mathematical Society, New York, 1943.

[45] R. Turpault. A multigroup M1 model for radiation hydrodynamics and applications. In A.D. Ketsdever and E.P. Muntz, editors, *23rd internationnal symposium on rarefied gas dynamics.* American Institute of Physics, 2004.

[46] W. Walters. Use of the Chebyshev-Legendre quadrature set in discrete-ordinate codes. Technical Report LA-UR-87-3621, Los Alamos National Laboratory, 1987.

[47] D. Wright, M. Frank, and A. Klar. The minimum entropy approximation to the radiative transfer equation. *Proc. Symp. Appl. Math.*, 67:987–996, 2009.