Implementing a Particle-Fluid Model of Auroral Electrons

Jörgen Vedin and Kjell Rönnmark

Department of Physics, Umeå University, SE-901 87 Umeå, Sweden
{jorgen.vedin, kjell.ronnmark}@space.umu.se

Abstract. The particle-fluid model of auroral electrons that is presented in [1] is a major step forward within the field of dynamic models of the auroral generation mechanisms. The model is, however, also an example where the implementation of a physical model requires a lot of knowledge from the field of computer science. Therefore, this paper contains a detailed description of the implementation behind the particle-fluid model. We present how the particles are implemented in doubly linked lists, how the fluid equations are solved in a time-efficient algorithm, and how these two parts are coupled into a single framework. We also describe how the code is parallelized with an efficiency of nearly 100%.

1 Introduction

The aurora is created by electrons that are accelerated along the Earth's magnetic field lines before they impinge on the ionosphere and create light through excitation processes. To model the large-scale processes involved in the generation of auroras it is common to describe the electrons as a charged fluid, where the fluid equations are solved self-consistently together with Maxwell's equations. However, as the electrons are accelerated they are also heated in a process that is not properly described by a fluid model. Therefore, we have in [1] introduced a particle-fluid model of the auroral electrons, where the field solver is complemented by a particle pusher. By letting the electric field from the field solver accelerate the particles, and then using the temperature of the particle distribution as a feed-back to the field solver, we obtain a self-consistent description of the auroral electrons.

The physical description of the model is given in [1], and in this paper we concentrate on the details of the implementation. We will describe how the set of equations in the field solver are solved using an implicit algorithm in which the discretized equations are rewritten on a block-tridiagonal form to achieve good performance. We will also discuss how the particle data is implemented in linked lists to obtain a time-efficient algorithm, and how the particles and the field solver are coupled in a parallelized code. First, however, we will in the next section describe the basics of the model.

2 Model

We use a two-dimensional model where the coordinates are z along the Earth's magnetic field and x spanning different latitudes, as can be seen in Fig. 1. The dynamics of auroral electrons is in real life controlled by a generator mechanism located in the tail of the Earth's magnetosphere. To mimic this process we prescribe a generator force in our model. The generator creates an ion current perpendicular to the magnetic field lines. At the flanks of the generator region the perpendicular current is diverted into field-aligned currents connecting the generator to the ionosphere. At the ionosphere these field-aligned currents are closed by a perpendicular ion current. Thus, we have a current circuit where the upward field-aligned current is carried by downgoing electrons that create auroras.

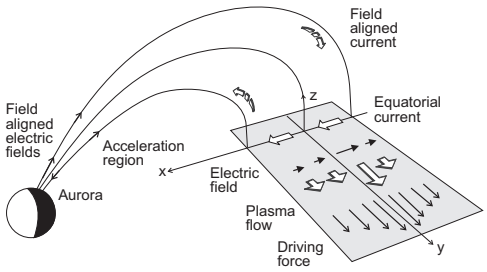

Fig. 1. The geometry of the auroral current circuit and the generator region in the equatorial magnetosphere. The curvature of the magnetic field lines is neglected in our model equations, but the convergence of the magnetic field lines is retained.

2.1 Field Solver

The electron fluid is in our model described by the equations

$$\partial_t \mathsf{E}_x = -\mathsf{A}^2 \partial_z \mathsf{B}_y - (1 - \mathsf{A}^2)\mathsf{F} \tag{1a}$$

$$\partial_t \mathsf{E}_z = \frac{B_z}{B_0} \left(\partial_x \mathsf{B}_y + \mathsf{j}_z\right) \tag{1b}$$

$$\partial_t \mathsf{B}_y = \partial_x \mathsf{E}_z - \partial_z \mathsf{E}_x \tag{1c}$$

$$\partial_t \mathsf{n} \ = -\partial_z \mathsf{j}_z \tag{1d}$$

$$\partial_t \, \mathsf{j}_z = -\frac{m_i}{m} \mathsf{n} \mathsf{E}_z - \partial_z \left(\mathsf{n} \mathsf{T}_z + \frac{\mathsf{j}_z^2}{\mathsf{n}} \right) - \mathsf{n} \mathsf{T}_\perp \frac{\partial_z B_z}{B_z} \tag{1e}$$

which are the non-zero components of Maxwell's equations together with the equation of continuity and the momentum equation. Here we use simulation variables, where E_x and E_z are the electric fields, B_y is the perpendicular magnetic

field, A is the Alfvén velocity, F is the generator force, n is the electron density, $\mathsf{j_z}$ is the field-aligned electron current, while $\mathsf{T_z}$ and $\mathsf{T}_\perp$ are the field-aligned and perpendicular temperatures. $B_z(z)$ is the geomagnetic field and B_0 is the field strength at $z = 0$ in the equatorial plane. The mass ratio in the last equation is the ion mass m_i divided by the electron mass m. For an extensive derivation of these equations and a detailed description of the physics they describe, the reader is referred to [1]. Notice that the ion dynamics is neglected in the present version of the model. This introduces constraints on the generator, and we choose its length and time scales to be larger than the ion gyroradius and the ion gyroperiod respectively.

The equation system in (1) is underdetermined since the time evolution of the temperatures $\mathsf{T_z}$ and $\mathsf{T}_\perp$ is not included. To close the equation system, we therefore introduce a particle pusher from which we can obtain temperatures that are consistent with the field solver in each time step.

2.2 Particle Pusher

The field solver that solves equations (1) is coupled to a particle pusher according to the cartoon in Fig. 2. This cycle is performed for each time step in the simulation. The particles are accelerated by the electric field from the field

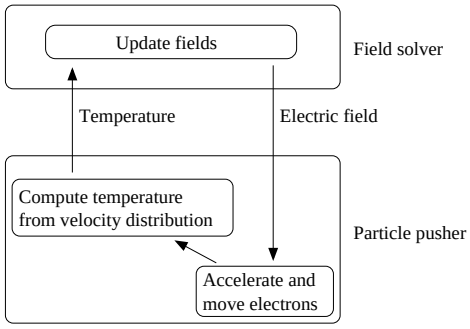

Fig. 2. Cycle performed for each time step to couple the field solver and the particle pusher.

solver, which implies that the particles move consistently with the fluid's evolution. From the velocity distribution of the electrons we can then determine temperatures that can be used in the momentum equation (1e).

The particles move along z with velocity v_z and gyrate the field line with velocity $v_\perp$. Their position and velocity are updated according to

$$d_t z = v_z \ , \tag{2}$$

$$d_t v_z = \frac{-e}{m} E_p - \mu \partial_z B_z \ , \tag{3}$$

where $-e$ is the electron charge, $\mu = mv_\perp^2/2B_z$ is the conserved magnetic moment of the particle, and E_p is the electric field that accelerates the particle. The electric field E_p is equal to the field-aligned electric field in the field solver, but with a correction described in [1] to ensure that the density and current of the particles are equal to the density and current of the fluid.

3 Implementation

3.1 Grid

The model is implemented on a two-dimensional grid of mesh size $n_x \times n_z$ illustrated in Fig. 3, where the generator boundary is at $z = 0$ and the ionosphere is on the opposite side of the simulation region. Since the z-coordinate is aligned with the magnetic field and the magnetic field lines are converging as they approach the ionosphere, the grid point separation Δx must decrease towards the ionosphere. The size of the simulation region is $L_z = 55{,}000$ km in the z-direction and $L_x = 4{,}600$ km in the x-direction at the generator boundary. The system length in the x-direction at the ionospheric boundary is merely 200 km. The grid is also chosen to be inhomogeneous for resolving the interesting features of auroral acceleration that take place mainly at altitudes below 10,000 km ($z > 45{,}000$ km) and at the field lines close to $x = 0$. The typical mesh size

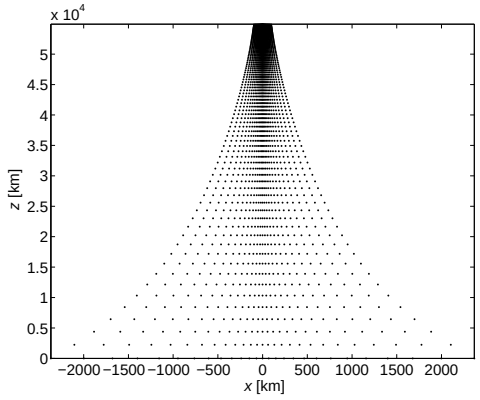

Fig. 3. The inhomogeneous grid on which the model is implemented with the generator boundary at $z = 0$ and the ionosphere on the opposite side of the simulation region.

in our simulations is $n_x \times n_z = 27 \times 100$. The value of n_x is chosen to resolve current filaments with a width of a few kilometers at the ionospheric boundary, while the value of n_z is large enough to get a good resolution of the acceleration region which has a length of a few thousand kilometers along z at an altitude centered about roughly 6,000 km.

3.2 Field Solver

The set of equations in (1) is solved by a time-centered implicit method, based on factorization of the two-dimensional spatial differential operators. Algorithms of this type are discussed by, for example, [3]. If we for notational convenience collect all the fields in a vector $\mathbf{U} = (\mathsf{E_x}, \mathsf{E_z}, \mathsf{B_y}, \mathsf{n}, \mathsf{j_z})^T$, we can after time discretization sum up (1) in the form

$$\mathbf{U}(t + \Delta t) = \mathbf{U}(t) + \Delta t \left\{\mathbf{Q}(\mathbf{U}(t + \Delta t/2)) + \mathbf{G}\right\} \quad , \tag{4}$$

where $\mathbf{Q}(\mathbf{U})$ represents the $\mathbf{U}$ dependence of the right hand side of (1) and $\mathbf{G}$ represents the inhomogeneous term involving the generator force F. We now linearize $\mathbf{Q}$ in $\mathbf{U}(t + \Delta t)$ by a Taylor expansion:

$$\begin{aligned} &\mathbf{Q}(\mathbf{U}(t + \Delta t/2)) \\ &\approx \mathbf{Q}(\mathbf{U}(t)) + \frac{1}{2}\left[\mathbf{U}(t + \Delta t) - \mathbf{U}(t)\right] \cdot \partial_{\mathbf{U}} \mathbf{Q}(\mathbf{U}(t)) \\ &= \frac{1}{2}\left[\mathbf{U}(t + \Delta t) + \mathbf{U}(t)\right] \cdot \partial_{\mathbf{U}} \mathbf{Q}(\mathbf{U}(t)) \quad , \end{aligned} \tag{5}$$

where the second step follows from the homogeneous properties of $\mathbf{Q}$. This results in a linear set of equations:

$$\left[\mathbf{I} - \frac{\Delta t}{2}\partial_{\mathbf{U}}\mathbf{Q}\right] \cdot \mathbf{U}(t + \Delta t) = \left[\mathbf{I} + \frac{\Delta t}{2}\partial_{\mathbf{U}}\mathbf{Q}\right] \cdot \mathbf{U}(t) + \Delta t\, \mathbf{G} \quad . \tag{6}$$

The operator $\partial_{\mathbf{U}}\mathbf{Q}$, which contains both ∂_x and ∂_z, is now split into two parts as $\partial_{\mathbf{U}}\mathbf{Q} = \mathbf{X} + \mathbf{Z}$, where $\mathbf{X}$ contains only ∂_x and $\mathbf{Z}$ contains only ∂_z. This decomposition is not unique, and it should be chosen in a way that makes the product $\mathbf{X} \cdot \mathbf{Z}$ as small and simple as possible. Neglecting the term $\Delta t^2/4\, \mathbf{X} \cdot \mathbf{Z} \cdot [\mathbf{U}(t + \Delta t) - \mathbf{U}(t)]$, which is of third order in Δt, we can factorize the terms in equation (6) as in an alternating direction implicit (ADI) method to find

$$\begin{aligned} &\left[\mathbf{I} - \frac{\Delta t}{2}\mathbf{X}\right] \cdot \left[\mathbf{I} - \frac{\Delta t}{2}\mathbf{Z}\right] \cdot \mathbf{U}(t + \Delta t) = \\ &\left[\mathbf{I} + \frac{\Delta t}{2}\mathbf{X}\right] \cdot \left[\mathbf{I} + \frac{\Delta t}{2}\mathbf{Z}\right] \cdot \mathbf{U}(t) + \Delta t\, \mathbf{G} \quad . \end{aligned} \tag{7}$$

Introducing $\mathbf{U}^* = [\mathbf{I} - \Delta t/2\, \mathbf{Z}] \cdot \mathbf{U}(t + \Delta t)$ as a new variable, we can solve (7) in two steps. First we solve

$$\begin{aligned} &\left[\mathbf{I} - \frac{\Delta t}{2}\mathbf{X}\right] \cdot \mathbf{U}^* = \\ &\left[\mathbf{I} + \frac{\Delta t}{2}\mathbf{X}\right] \cdot \left[\mathbf{I} + \frac{\Delta t}{2}\mathbf{Z}\right] \cdot \mathbf{U}(t) + \Delta t\, \mathbf{G} \end{aligned} \tag{8}$$

for $\mathbf{U}^*$. Then we use

$$\left[\mathbf{I} - \frac{\Delta t}{2}\mathbf{Z}\right] \cdot \mathbf{U}(t + \Delta t) = \mathbf{U}^* \tag{9}$$

to solve for the fields at $t + \Delta t$. When the operators $\mathbf{X}$ and $\mathbf{Z}$ are expressed as centered finite differences, each of these two steps consists of solving a block-tridiagonal set of equations. The number of operations needed for this scales linearly with the mesh size $(n_x \times n_z)$ and hardly requires any extra memory, which makes this algorithm very efficient. The number of operations needed for a direct integration of (6) would typically be proportional to $(n_x \times n_z)^2$.

3.3 Particle Pusher

In this model the particles are used roughly as in a regular Particle-In-Cell (PIC) code, see for example [2]. The input to the particle pusher is the electric field computed in the field solver, and the output from the particle pusher is the temperatures T_z and $\mathsf{T}_\perp$. The electric field and the temperatures are given on the grid points, but the particles can of course be located also in-between two grid points. To handle this, a PIC code utilizes a weight scheme, and in this model we use linear weights. The electric field (and also the magnetic field) used in (3) is determined from the fields at the two grid points that are closest to the particle's position, as can be seen in Fig. 4. At the particle's position z_i, the

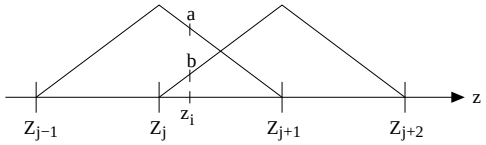

Fig. 4. To determine the electric field at the particle position z_i we use triangular shape functions. The electric field at grid point Z_j is weighted by the value of the shape function in point a and the electric at grid point Z_{j+1} is weighted by the value of the shape function in point b. These two values are then assigned to the electric field $E(z_i)$.

electric field is given by

$$E(z_i) = \left(\frac{Z_{j+1} - z_i}{\Delta z}\right) E_j + \left(\frac{z_i - Z_j}{\Delta z}\right) E_{j+1} \ , \tag{10}$$

where E_j and E_{j+1} are the electric field values at the grid points Z_j and Z_{j+1}, while Δz is the grid point separation. When the temperatures are computed, the weighting is inverted, and the temperatures at a grid point get a contribution from all particles located in the two grid cells surrounding the grid point, as can be seen in Fig. 5. If T_i is the contribution to the temperature from a particle located at z_i, then

$$T_j = \left(\frac{Z_{j+1} - z_i}{\Delta z}\right) T_i \tag{11}$$

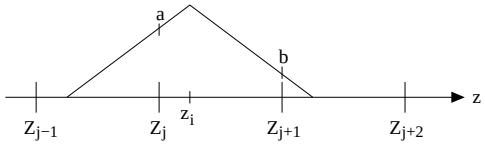

Fig. 5. A particle at position z_i will contribute to the temperatures at grid points Z_j and Z_{j+1}. If T_i is the contribution to the temperature from a particle located at z_i, then T_i will be weighted by the shape function's value at a before it is added to the temperature at Z_j and it will be weighted by the shape function's value at b before it is added to the temperature at Z_{j+1}.

is the part of T_i that is added to the temperature at grid point Z_j. Correspondingly the temperature at Z_{j+1} gets the contribution

$$T_{j+1} = \left(\frac{z_i - Z_j}{\Delta z} \right) T_i \tag{12}$$

from a particle at z_i.

During the simulation, particles that move outside the system boundaries are removed and new particles are injected at a rate determined to maintain a constant density and temperature at the boundaries. To handle this adding and removing of particles in a sufficiently fast manner the particles are implemented as a linked list. Actually, we implement the particles as six linked lists. There are two types of particles, particles of magnetospheric origin that are injected at the generator boundary and particles of ionospheric origin that are injected at the ionospheric boundary. Each type of particles are stored in three separate lists. The particles that are active in the simulation are stored in an inside-list, while the particles that have been removed from the system are stored in a used-list. Furthermore, we have an unused-list that is used as a reservoir for particles that are to be injected. In this way we need not free and allocate memory for each particle that is removed or injected. Used particles are simply transferred from the inside-list to the used-list and particles that are to be injected are transferred from the unused-list to the inside-list. When the unused-list becomes empty, we rehash the particles in the used-list to give them the desired velocity distribution and place them in the unused-list. By letting the initial unused-list hold some extra particles, we can through this procedure avoid the need for any memory allocation during the simulation. As a fallback, the program can allocate more particles if the initial unused-list turned out to be too small.

The implementation with several lists is fast, and the memory overhead is not significant. However, we need to implement the lists as doubly linked lists in order to enable particles to switch lists. This implies some overhead in the memory per particle since each particle need a pointer to both the next particle and the previous particle in the list. The total memory per particle is still only 40 bytes since each particle, apart from the two pointers, only holds three keys: z, v_z, and μ.

3.4 Particle-Fluid Coupling

The electrons are strongly magnetized which means that they are guided by the magnetic field lines and therefore only move along z, although in a gyrating motion. Hence, there are n_x independent particle pushers in the total simulation. In each particle pusher we need at least about 20 million particles to obtain reasonably good statistics when computing the temperatures from the particle distribution. For each x the particles use at least about 0.8 GB of memory, and the entire simulation uses over 20 GB, which indicates that a parallelization is needed. A huge number of particles, of course, leads to long computer times. The ratio between the time spent in a single particle pusher and the time spent in the field solver is roughly 1500 to 1.

Since almost all time is spent in the particle pushers and there are n_x independent pushers, this application is perfectly suited for parallelization on n_x processors. The code, which is written in C, is therefore parallelized using the Message Passing Interface (MPI) [4]. For each time step in the parallelized code, every processor solves (8), then each processor solves (9) and calls the particle pusher with its own value of x. Finally, before time is incremented, the fields in the vector $\mathbf{U}$, the temperatures and some help variables are synchronized on all processors using `MPI_Allgather`. This implies a message passing of about 0.2 MB for each time step.

The simulations are performed on the Sarek Linux Cluster at the High Performance Computing Center North (HPC2N) [5]. This cluster has 384 64-bit AMD Opteron 2.2 GHz CPUs in 192 dual nodes. The network, which is switched to give similar performance between each pair of nodes, has a bandwidth of about 250 MB/s and a latency of a few microseconds. With this bandwidth, the message passing will for each time step last only 10^{-3} s, while the entire time step lasts for roughly 10 s. Thus, the time for communication is negligible. If the load balancing between processors is good this would imply a parallelization efficiency of nearly 100%, and since each processor handles roughly equal amounts of particles we have no reason to expect otherwise. The Sarek cluster also has the advantage that each processor has access to 4 GB of RAM, which implies that we can use the huge number of particles that is needed for good statistics in the temperature calculations.

4 Discussion and Conclusions

According to the physical results presented in [1], the particle-fluid model produces results that are consistent with observations, and the model is a major step forward compared to previous dynamic models of auroral electron acceleration, for example [6], [7], and [8], where the variation in the electron temperatures has been neglected. The implementation is, however, much more complicated compared to implementing a pure fluid model.

The model proves to be very efficient to parallelize on the same number of processors as the number of grid points in the direction perpendicular to the

magnetic field. A further parallelization of the code can be accomplished by letting the particles at a certain field line be distributed in several linked lists and update each list on different processors. If the lengths of these lists are approximately equal, which is easily accomplished by always injecting particles into the shortest list, the load balancing of this parallelization would be perfect. Furthermore, the only communication needed in this parallelization is summing the temperatures of all lists into a single temperature along the field line, which makes this parallelization very efficient.

Acknowledgments

This research was conducted using the resources of High Performance Computing Center North (HPC2N), and was supported by the Swedish National Graduate School of Space Technology.

References

1. Vedin, J., Rönnmark, K.: Particle-fluid simulation of the auroral current circuit. J. Geophys. Res. **111** (2006), A12201, doi:10.1029/2006JA011826
2. Birdsall, C. K., Langdon, A. B.: Plasma physics via computer simulation. McGraw-Hill Book Company, New York (1985)
3. Degrez, G.: Implicit time-dependent methods for inviscid and viscous compressible flows, in *Computational Fluid Dynamics*, edited by J. F. Wendt, pp. 180–222, Springer-Verlag, New York (1992).
4. The Message Passing Interface (MPI) http://www-unix.mcs.anl.gov/mpi/
5. High Performance Computing Center North, Umeå, Sweden http://www.hpc2n.umu.se
6. Goertz, C. K., Boswell, R. W.: Magnetosphere-ionosphere coupling. J. Geophys. Res. **84** (1979) 7239–7246
7. Streltsov, A. V., Lotko, W., Johnson, J. R., Cheng, C. Z.: Small-scale, dispersive field line resonances in the hot magnetospheric plasma. J. Geophys. Res. **103** (1998), 26559-26572
8. Rönnmark, K., Hamrin, M: Auroral electron acceleration by Alfvén waves and electrostatic fields. J. Geophys. Res. **105** (2000) 25333–25344