Proceedings of the Federated Conference on Computer Science and Information Systems pp. 381–385
DOI: 10.15439/2020F130
ISSN 2300-5963 ACSIS, Vol. 21

Parallel implementation of a PIC simulation algorithm using OpenMP

Alin Suciu, Anca Hangan, Anca Marginean, Marius Joldos
Technical University of Cluj-Napoca, Romania
Computer Science Department
Email: {alin.suciu, anca.hangan, anca.marginean, marius.joldos}@cs.utcluj.ro

Gabriel Voitcu, Marius Echim
Institute of Space Science, Magurele, Romania
Email: {gabi, echim}@spacescience.ro

***Abstract*—Particle-in-cell (PIC) simulations are focusing on the individual trajectories of a very large number of particles in self-consistent and external electric and magnetic fields; they are widely used in the study of plasma jets, for example. The main disadvantage of PIC simulations is the large simulation runtime, which often requires a parallel implementation of the algorithm. The current paper focuses on a PIC1d3v simulation algorithm [1][2] and describes the successful implementation of a parallel version of it on a multicore architecture, using OpenMP, with very promising experimental and theoretical results.**

I. Introduction

PARTICLE-IN-CELL (PIC) simulations are extremely useful to model self-consistently plasma phenomena at kinetic scales [3][4]. Such kind of simulations focus on the individual trajectories of a very large number of particles in self-consistent and external electric and magnetic fields. One class of important phenomena that can be modeled with PIC simulations are the high-speed plasma jets observed within Earth's magnetosheath (see, for instance, [5] and references therein). This topic is highly relevant for the geomagnetic environment (e.g. [6][7]), but it is also of great importance in other astrophysical and space science contexts, like, for instance, the interaction of the planetary/magnetospheric plasmas with solar and stellar winds or the propagation of astrophysical relativistic jets (e.g.[8][9]). More generally, the topic of high-speed jets is relevant to transport phenomena, kinetic processes and discontinuities in collisionless magnetized plasmas.

The interaction of high-speed plasma jets with non-uniform magnetic fields has been investigated over time with multiple fluid and kinetic approaches (see, for instance, [10] for a review on this topic). Nevertheless, the electromagnetic PIC approach is the most suitable tool to address this issue since it allows for the simultaneous investigation of key physical effects as self-polarization, finite Larmor radius effects and electromagnetic processes. Indeed, [11][12][13] used for the first-time such kind of simulations in a 3D geometry to investigate the interaction of high-speed plasma jets with non-uniform magnetic fields in a simplified magnetopause-like configuration typical for a northward interplanetary magnetic field.

The electromagnetic PIC approach provides a fully kinetic description of plasma by considering both self-consistent electrostatic and electromagnetic effects at microscopic level. The time-step and grid spacing used in electromagnetic PIC simulations must fulfill very restrictive stability conditions in order to avoid undesired numerical effects that could arise due to the discretization of space and time [3]. Thus, the very fine spatial and temporal resolution resolves even the smallest scales, i.e. Debye length and plasma frequency, but also leads to large simulation runtimes.

To surpass this limitation, parallelization is often used for implementing PIC simulation algorithms, and parallel algorithms for PIC simulations are present throughout the scientific literature. The solution presented in [14] uses a distributed memory model to simulate large scale systems. In their approach, parallelism is achieved through geometrical domain decomposition, each process being responsible for evolving its own sub-domain. The authors of [15] have a similar approach of domain decomposition, while investigating load-balancing strategies for both distributed memory and shared memory models.

Finally, the authors of [16] achieve parallelism by dividing particles among threads according to their positions, on a shared memory machine, while taking advantage of cache reusability. A hybrid approach is presented in [17], in which the author uses MPI for communication between processes and OpenMP to parallelize the loops inside the processes. This way, the implementation takes advantage of the fact that the processing nodes have a multi-core architecture. To obtain better performance, in some approaches [18][19], parallel implementations are optimized using techniques that take advantage of the physical parallel machine characteristics.

In this paper, we use an explicit and relativistic 1d3v electromagnetic PIC algorithm developed for one-dimensional kinetic simulation of fully-ionized collisionless magnetized plasmas. This approach considers simulation geometries having a single dimension in the configuration space and all three dimensions in the velocity space. Such an algorithm can be used to study, for instance, the formation, structure and evolution of one-dimensional tangential discontinuities, a topic of great importance for understanding the physics of high-speed jets in space plasmas (e.g. [20][21]). Given the sequential algorithm, we describe how we developed its parallel version, while discussing the parallelization strategy. To the best of our knowledge, no other parallelization of the discussed algorithm can be found in the literature.

Algorithm 1 Algorithm PIC1d3v

```
Begin simulation
// read input parameters
0. read(m, np, nt,...);
// initialize data
1. init(...);
// execute main loop
for i=1 to nt do
    Begin
        // half-advance of magnetic field
        2. bfield(...);
        // push particles (one time-step)
        3. mover(...);
        // half-advance of magnetic field
        4. bfield(...);
        // current density computation
        5. current(...);
        // full-advance of electric field
        6. efield(...);
        // apply periodicity for particles
        7. period(...);
        // compute energy density
        8. energy(...);
        // write data to files (optional)
        9. write_files(...);
    End loop
End simulation
```

The rest of this paper is organized as follows. Section II describes the sequential PIC1d3v algorithm. Section III describes the parallelization strategy, by first analyzing the execution time of each step and investigating the formula for the overall speedup. Section IV presents the experimental results. Section V concludes the paper.

II. The PIC1d3v algorithm

The PIC1d3v algorithm that we are focusing on is based on the proposed algorithm from [1][2] and raises a series of challenges that will be detailed below, following the pseudocode of the algorithm, step by step. (see Algorithm 1 below). Moreover, the flow of the simulation is visually described in Fig. 1.

Step 0– Read input parameters

During this step of the simulation the parameters are read from the input file. The most important parameters, those who control the size of the simulation are:

1) m – the number of cells; particles are distributed randomly across these cells at the beginning of the simulation.
2) np – the number of particles, meaning that np electrons and np ions are used in the simulation.
3) nt – the number of iterations; the main loop is executed nt times.

For the range of problems studied in our research group, concrete values for m range from $1,000$ to $10,000$ cells, values for np range from $100,000$ to $1,000,000$ particles and the values for nt range from 1 to $10,000,000$, but technically these values could be increased if needed. Let us focus a bit on the largest possible simulation consisting of $10,000,000$ iterations for $1,000,000$ particles spread across $10,000$ cells. The internal memory requirements for such a simulation are not difficult to fulfill for a common serial computer available today in any research lab. However, the execution time required for such a simulation (for a common serial computer) is impressive: approximately 450 days (that is 1.267 years)! The main goal of this paper, as stated above, is to reduce this execution time as much as possible, using a multicore CPU architecture, which is very commonplace nowadays.

Step 1 – Initialize data

During this step of the simulation the position, velocity, fields, currents and energy data is initialized.

Steps 2 to 9 – The main loop

The main focus of the algorithm is the main loop, which is also clearly the most time consuming part, and therefore the focus of the parallel optimization. We will shortly describe each step of the main loop below.

Steps 2 and 4 – Half-advance of the magnetic field

In these steps, the components of the magnetic field (according to Faraday's law) are computed, at a given time, by knowing their values at the previous time-step. At each step, the magnetic field advances only for a half time-step.

Step 3 – Push particles over one time-step

This step of the algorithm computes the electric and magnetic fields in the actual position of each particle. Then, it moves the particle to its new position after one time-step and recomputes its velocity.

Step 5 – Current density computation

At this step, current density is computed for particles, while applying periodic boundary conditions and a smoothing procedure.

Step 6 – Full-advance of the electric field

This step of the algorithm computes the components of the electric field from Ampere's law, at a given time, by knowing their values at the previous time-step.

Step 7 – Apply periodicity for particles

This step of the algorithm applies periodicity for each particle.

Step 8 – Compute energy density

This step of the algorithm computes the energy density of the system and does not interfere with the simulation. This step is used for diagnosis purposes throughout the simulation.

Step 9 – Write data to files (optional) For each iteration, computed data can be saved in binary output files, if intermediate data is required by the user.

III. Parallel implementation of the PIC1d3v algorithm

When considering a parallel implementation, the first question that needs to be addressed refers to the targeted parallel

hardware architecture (multicore, GPU, cluster, etc.). For the purpose of our simulation needs we considered a multicore architecture as our target, so the natural choice for the software platform, given that our serial implementation was done in C/C++, was OpenMP.

With regard to the parallelization strategy/methodology, a closer look at Fig. 1 and Algorithm 1 reveals data dependencies among the steps of the sequential algorithm (e.g. to compute the electric field we need to compute the magnetic field and the currents first) which prevent loop iterations to be executed in parallel; so the only hope that remains is to concentrate our efforts on parallelizing the individual steps, with a focus on the main loop. We will discuss below the parallel approach taken for each step of the main loop of the algorithm.

A. Parallelization analysis

In order to compute the speedup of the main loop, as each step of the computation could be inherently sequential or maybe have some degree of parallelism, we need to apply Amdahl's law. Amdahl's law fits perfectly our scenario of having a fixed size problem that we want to solve in as little time as possible (also, processors have the same architecture, frequency, cache size, etc.). Therefore, we need to know the fractions of the computation, each fraction corresponding to one step of the main loop (in order to be able to apply Amdahl's law). We have to compute f_2 to f_9, the fractions of the total sequential execution time spent by each step of the main loop. To do so, we need to measure the total sequential execution time of the main loop (denoted as t_{loop}) and the sequential execution time for each step of the loop: $t_2, t_3, ..., t_9$. We then compute:

$$f_i = \frac{t_i}{t_{loop}}, \text{for } i = \overline{2,9}. \tag{1}$$

In the following, we analyze each step of the main loop, in the maximal simulation scenario ($m = 10.000$, $np = 1.000.000$), in order to identify the hotspots and the bottlenecks. We then concentrate on parallelizing the hotspots, as this will give us the highest overall speedup. We also provide concrete values for each fraction of the main loop, in this scenario, so that we can identify the most significant steps and concentrate the parallelization efforts there. The concrete values for each fraction mentioned below are taken from Table I below, to illustrate the impact of each step of the algorithm.

Steps 2 and 4 – Half-advance of the magnetic field

These steps account (each) for $f_2 = f_4 = 0.01\%$ of the execution time of the main loop, due to the fact that it mainly consists of a single "for" loop of m iterations. We did not parallelize these steps because the overhead induced by the parallelization (thread creation and management, synchronization) would only lead to a slowdown instead of a speedup of the execution.

Step 3 – Push particles over one time-step

This step accounts for $f_3 = 74.83\%$ of the execution time of the main loop, and we were able to parallelize this step with the help of the "parallel for" directives. No data dependencies were detected so the parallelization was straightforward.

Step 5 – Current density computation

This step accounts for $f_5 = 8.39\%$ of the execution time of the main loop. Unfortunately this step requires a lot of synchronization among threads, so only minor parts of it were parallelizable. The "parallel for" directive was used where possible.

Step 6 – Full-advance of the electric field

This step accounts for $f_6 = 0.04\%$ of the execution time of the main loop, due to the fact that it mainly consists of a single "for" loop of m iterations. We did not parallelize this step because the overhead induced by the parallelization (thread creation and management, synchronization) would only lead to a slowdown instead of a speedup of the execution.

Step 7 – Apply periodicity for particles

This step accounts for $f_7 = 0.54\%$ of the execution time of the main loop; although it accounts for a very small fraction of the main loop, it has an embarrassingly parallel structure, being in fact just one loop controlled by the np parameter, so we used a "parallel for" directive to parallelize it.

Step 8 – Compute energy density

This step accounts for $f_8 = 16.170\%$ of the execution time of the main loop, and we were able to parallelize this step with the help of the "parallel for" directives. No data dependencies were detected so the parallelization was straightforward.

Step 9 – Write data to files (optional)

This step is optional so we will ignore it for now, especially since it represents I/O time and therefore it cannot be improved by parallelism. However, experiments performed at the maximum size of the simulation show that the impact of this step is negligible.

B. Computing the speedup

Given that we know the fractions of the main loop, f_2 to f_8, and assuming that we can compute (for a certain scenario, on a certain architecture, with a certain number of processors) the maximum speedup for each fraction, s_2 to s_8, then the overall speedup for the main loop is given by the following formula, derived from Amdahl's law:

$$s_{loop} = \frac{1}{\sum_{i=2}^{8} \frac{f_i}{s_i}} \tag{2}$$

One can notice that for steps 2, 4 and 6 of the main loop, the speedup is 1, since these are the serial fractions of the loop; on the other hand, all the other fractions will have a speedup larger than 1, but we expect different speedups for different steps, due to different degrees of parallelization that are possible for each of them. If we take a look at the initialization steps (steps 0 and 1), we notice that step 1 is inherently sequential but step 2 has some potential for parallelization; let us assign fractions f_0 and f_1 to these steps too, and also the speedups $s_0 = 1$ and $s_1 > 1$ (s_1 will be

determined experimentally). Then the total speedup in this scenario is, by a similar formula:

$$s_{tot} = \frac{1}{\frac{f_0}{s_0} + \frac{f_1}{s_1} + \frac{f_{loop}}{s_{loop}}} \tag{3}$$

The problem however is that these fractions are not constant, they depend upon the parameter nt; let us consider first $nt = 1$, then we can measure t_{01}, t_{11} and t_{loop1} respectively, the execution time for the first two steps and the loop, with one execution of the main loop ($nt = 1$). The corresponding fractions are then:

$$f_{01} = \frac{t_{01}}{t_{01} + t_{11} + t_{loop1}} \tag{4}$$

$$f_{11} = \frac{t_{11}}{t_{01} + t_{11} + t_{loop1}} \tag{5}$$

$$f_{loop1} = \frac{t_{loop1}}{t_{01} + t_{11} + t_{loop1}} \tag{6}$$

and in general for nt iterations of the main loop, we have:

$$f_{0nt} = \frac{t_{01}}{t_{01} + t_{11} + nt * t_{loop1}} \tag{7}$$

$$f_{1nt} = \frac{t_{11}}{t_{01} + t_{11} + nt * t_{loop1}} \tag{8}$$

$$f_{loopnt} = \frac{nt * t_{loop1}}{t_{01} + t_{11} + nt * t_{loop1}} \tag{9}$$

Therefore, to compute the respective fractions, we only need to measure the execution time of one iteration of the main loop, which is feasible in practice and quite fast; based on that, considering the impact of the initialization steps, we can compute the total speedup of the algorithm as:

$$s_{tot} = \frac{1}{\frac{f_{0nt}}{s_0} + \frac{f_{1nt}}{s_1} + \frac{f_{loopnt}}{s_{loop}}} \tag{10}$$

We may notice that if the parameter nt grows, the impact of the initialization steps, as expected, becomes more and more negligible, as f_{loopnt} approaches 1 and thus s_{tot} approaches s_{loop}.

IV. Experiments and results analysis

We performed a series of experiments on a multicore (quad core) computer server (Intel®Core2™ Quad CPU - Q8400 @ 2.66GHz processor, 8GB RAM).

First, we tested the limits of the simulation with regards to the size of the problem, and we encountered no problems and no limitations whatsoever, the only limit being the execution time, which grows in direct proportion with the growth of the nt parameter. To estimate the achievable speedup we performed a test with the following parameters (maximum size simulation scenario): $m = 10,000$; $np = 1,000,000$; $nt = 10$.

We computed the average execution time for each step of the main loop both for the serial execution (1 processor, 1 thread) and the parallel execution with the maximum number of processors (4 processors, 4 threads).

Thus, we could compute the speedups for each of the parallelized steps, and we obtained the results presented in Table I. Following equation (2), we compute the main loop speedup as:

$$s_{loop} = \frac{1}{0.377496} = 2.649 \tag{11}$$

One can notice that the computed formula matches exactly the value from the lower right corner of Table 1, so the theoretical prediction perfectly matches the experimental result; the impact of the serial fractions of the problem is concentrated in step 5 (current) and we were able to obtain an overall speedup for the main loop of 2.649 on a quad core computer. Considering that the impact of the initialization steps is negligible for a very large number of iterations, we may conclude that the overall speedup is approximately equal to the loop speedup:

$$s_{tot} \approx s_{loop} = 2.649 \tag{12}$$

Since the number of processors used for the parallel execution was $n = 4$, we can also compute the overall efficiency as:

$$e_{tot} = \frac{s_{tot}}{n} = \frac{s_{tot}}{4} = 66.23\% \tag{13}$$

A simple computation shows that this will reduce the execution time of the maximal simulation from 450 days to 170 days on the tested architecture. However, the number of cores can be increased even further and thus, with no changes to the implementation the execution time will be reduced even further. Another advantage of this approach is that we know in advance how much time the parallel simulation will take, so we can decide to launch it or not, perhaps adjust the setup (e.g. increase the number of cores, decrease the umber of iterations, etc.), and then we can re-compute the speedup and so on.

An important issue that we considered is the precision of the obtained results, given the fact that the algorithm uses a lot of floating point computations, and it is well known that even addition is not associative in floating point arithmetic. Thus we performed a series of experiments where we measured the maximal difference between the serial values and the parallel values, and the experiments showed that parallelization did not have a significant impact on the final and the intermediary results. The maximal difference was of order 10^{-9}, which is acceptable for the considered PIC simulation scenarios.

V. Conclusion

PIC simulations play an important role when simulating the individual trajectories of a very large number of particles in self-consistent and external electric and magnetic fields. We used here an explicit and relativistic 1d3v electromagnetic PIC algorithm developed for one-dimensional kinetic simulation of fully-ionized collisionless magnetized plasmas.

Starting from a PIC1d3v serial algorithm whose execution, for the largest simulation scenario) takes over a year, we were

TABLE I
EXECUTION TIMES, FRACTIONS AND SPEEDUPS FOR THE MAIN LOOP

Step no. (name)	Serial time (ms)	Fraction(%)	Parallel time (ms)	Speedup
2 (bfield)	0.57	0.0147	0.57	1.000
3 (mover)	2904.26	74.8314	868.60	3.344
4 (bfield)	0.56	0.0144	0.56	1.000
5 (current)	325.68	8.3915	305.64	1.066
6 (efield)	1.58	0.0407	1.58	1.000
7 (period)	20.77	0.5352	10.04	2.069
8 (energy)	627.65	16.1721	278.10	2.257
LOOP	3881.07	100	1465.09	2.649

able to significantly reduce the execution time by using a multicore CPU architecture, which is common place nowadays, and the clever use of the OpenMP directives, with a minimal modification of the original serial C/C++ code.

We performed a thorough analysis of the sequential algorithm and identified each step, each fraction of the problem, each dependency, computed the speedup for each fraction and finally computed the overall speedup, which shows promising results and enables even more speedups to be obtained on a multicore architecture with more cores, with no changes to the parallel code. Based on the promising results of the PIC1d3v parallel simulation presented here, we plan to move to a full PIC3d parallel simulation, following the same fundamental ideas presented in this paper.

ACKNOWLEDGMENT

This work was supported by the Romanian Ministry of Research and Innovation via a PCCDI grant (PCCDI Project no. 18 PCCDI/2018). Gabriel Voitcu and Marius Echim acknowledge support from the Romanian Ministry of Education and Research through the Core Programme LAPLAS VI/2020.

REFERENCES

[1] Omura, Y., Matsumoto, H. “KEMPO1: Technical Guide to One-dimensional Electromagnetic Particle Code", in Computer Space Plasma Physics: Simulation Techniques and Software, edited by H. Matsumoto and Y. Omura, pp. 21-65, Terra Scientific Publishing Company, Tokyo, 1993.

[2] Voitcu, G. "Kinetic simulations of plasma dynamics across magnetic fields and applications to the physics of planetary magnetospheres”, PhD thesis, University of Bucharest, Romania, 2014.

[3] Birdsall, C. K., Langdon, A. B.“Plasma physics via computer simulation”, Boca Raton: CRC Press, 1991, doi:10.1201/9781315275048.

[4] Hockney, R. W., Eastwood, J. W. “Computer simulation using particles”, Boca Raton: CRC Press, 1988, doi: 10.1201/9780367806934.

[5] Plaschke, F., Hietala, H., Archer, M., Blanco-Cano, X., Kajdic, P., Karlsson, T., Lee, S. H., Omidi, N., Palmroth, M., Roytershteyn, V., Schmid, D., Sergeev, V., Sibeck, D. Jets Downstream of Collisionless Shocks, Space Science Reviews, 214, 81, 2018, doi:10.1007/s11214-018-0516-3.

[6] Hietala, H., Partamies, N., Laitinen, T. V., Clausen, L. B. N., Facsko, G., Vaivads, A., Koskinen, H. E. J., Dandouras, I., Reme, H., Lucek, E. A. “Supermagnetosonic subsolar magnetosheath jets and their effects: from the solar wind to the ionospheric convection”, Annales Geophysicae, 30, 33, 2012, doi:10.5194/angeo-30-33-2012.

[7] Archer, M. O., Hietala, H., Hartinger, M. D., Plaschke, F., Angelopoulos, V. “Direct observations of a surface eigenmode of the dayside magnetopause”, Nature Communications, 10:615, 2019, doi:10.1038/s41467-018-08134-5.

[8] Karlsson, T., Liljeblad, E., Kullen, A., Raines, J. M., Slavin, J. A., Sundberg, T. “Isolated magnetic field structures in Mercury’s magnetosheath as possible analogues for terrestrial magnetosheath plasmoids and jets”, Planetary and Space Science, 129, 61, 2016, doi:10.1016/j.pss.2016.06.002.

[9] Nishikawa, K.-I., Frederiksen, J. T., Nordlund, A., Mizuno, Y., Hardee, P. E., Niemiec, J., Gomez, J. L., Peer, A., Dutan, I., Meli, A., Sol, H., Pohl, M., Hartmann, D. H. “Evolution of global relativistic jets: collimations and expansion with kKHI and the Weibel instability”, The Astrophysical Journal, 820:94, 2016, doi: 10.3847/0004-637X/820/2/94.

[10] Echim, M. M., Lemaire, J. F. “Laboratory and numerical simulations of the impulsive penetration mechanism”, Space Science Reviews, 92, 565, 2000, doi:10.1023/A:1005264212972.

[11] Voitcu, G., Echim, M. “Transport and entry of plasma clouds/jets across transverse magnetic discontinuities: Three-dimensional electromagnetic particle-in-cell simulations”, Journal of Geophysical Research - Space Physics, 121, 5, 4343-4361, 2016, doi:10.1002/2015JA021973.

[12] Voitcu, G., Echim, M. „Tangential deflection and formation of counterstreaming flows at the impact of a plasma jet on a tangential discontinuity”, Geophysical Research Letters, 44, 12, 5920-5927, 2017, doi:10.1002/2017GL073763.

[13] Voitcu, G., Echim, M. „Crescent-shaped electron velocity distribution functions formed at the edges of plasma jets interacting with a tangential discontinuity”, Annales Geophysicae, 36, 1521-1535, 2018, doi:10.5194/angeo-36-1521-2018.

[14] Bart, G, Peltz, C, Bigaouette, N, Fennel, T, Brabec, T, Varin, C. Massively parallel microscopic particle-in-cell. Computer Physics Communications. 2017 Oct 1;219:269-85.

[15] Miller, KG, Lee, RP, Tableman, A, Helm, A, Fonseca, RA, Decyk, VK, Mori, WB. Dynamic load balancing with enhanced shared-memory parallelism for particle-in-cell codes. arXiv preprint arXiv:2003.10406. 2020 Mar 23.

[16] Shah, K, Phadnis, A, Shah, M, Chaudhury, B. Parallelization of the Particle-In-Cell Monte Carlo Collision (PIC-MCC) Algorithm for Plasma Simulation on Intel MIC Xeon Phi Architecture. In proceedings of International Conference for High Performance Computing 2017.

[17] Sáez, X., Soba, A., Cela, J.M., Sánchez, E., Castejón, F. Particle-In-Cell algorithms for Plasma simulations on heterogeneous architectures. In 2011 19th International Euromicro Conference on Parallel, Distributed and Network-Based Processing 2011 Feb 9 pp. 385-389, doi:10.1109/PDP.2011.42

[18] Marszałek, Z., Woźniak, M., Połap, D., Fully flexible parallel merge sort for multicore architectures. Complexity, 2018, doi: 10.1155/2018/8679579

[19] Palkowski, M., Bielecki, W., "Parallel cache-efficient code for computing the McCaskill partition functions," 2019 Federated Conference on Computer Science and Information Systems (FedCSIS), Leipzig, Germany, 2019, pp. 207-210, doi: 10.15439/2019F8.

[20] Roth, M., De Keyser, J., Kuznetsova, M. M. “Vlasov theory of the equilibrium structure of tangential discontinuities in space plasmas”, Space Science Reviews, 76, 251, 1996, doi:10.1007/BF00197842.

[21] Echim, M. M., Lemaire, J. F., Roth M. “Self-consistent solution for a collisionless plasma slab in motion across a magnetic field”, Physics of Plasmas, 12, 072904, 2005, doi: 10.1063/1.1943848.