A quick introduction to qv3d

John Farmer

August 2, 2021

1 Introduction

Particle-in-cell (PIC) codes are now ubiquitous in the study of plasma physics. The majority of these models solve for the electromagnetic fields using a FDTD implementation of Faraday and Ampere's laws, e.g. the Yee solver. While such schemes offer the most complete treatment of the plasma response, they are inefficient for applications where the characteristic timescale of the system is much longer than the characteristic length scale. This is due to FDTD methods being subject to the Courant condition for stability, with the maximum timestep linked to the cell size.

Consider a relativistic electron beam propagating in plasma. The transverse two-steam instability will modulate the beam with a characteristic length scale $\sim c/\omega_p$. The characteristic timescale for the beam evolution is $1/\omega_\beta = \sqrt{2\gamma}/\omega_p$. The disparity becomes even greater when considering a proton beam - for the 400 GeV SPS beam used in AWAKE, the betatron frequency is ~ 1250 times smaller than the plasma frequency. The constraint on the timestep therefore results in a large computational overhead.

One solution to this problem is to make use of the quasistatic approximation. If the drive beam and plasma response evolve slowly in the co-moving frame of the beam, they may be treated as being in equilibrium. The fields are then electro/magnetostatic, and may be solved as a boundary value problem, integrating backwards from the upstream boundary. Without the constraint of the Courant condition, an orders-of-magnitude speedup can be achieved.

This document serves as a brief introduction to using the `qv3d`, built on the framework of the fully electromagnetic VLPL code. For a detailed discussion of the model, consult A. Pukhov, Particle-In-Cell Codes for Plasma-based Particle Acceleration, CERN Yellow Reports 1, 181 (2016).

2 The input deck

Most quantities in the input deck are reasonably obvious from their names. Some options are really only for specific applications, so don't worry if you don't understand everything.

Length and time in the code are normalized to the characteristic plasma wavenumber/angular frequency, i.e. plasma oscillations have a period of 2π. Densities are relative to the corresponding plasma density, mass is given in AU, momenta are normalised to mc and external magnetic fields are normalised to Tesla per $1/k$. The characteristic plasma length/time/density is chosen by setting `Wavelength` (given in centimetres in `Domain`). The plasma physics will scale, but it is important to set the correct plasma wavelength for modelling synchrotron emission/radiation damping and ionization. The characteristic wavelength should be chosen such that the plasma a density ~ 1, although this seems to have very little impact.

The various species in the simulation are designated as being either a beam - modelled in 3d phase space - or part of the background plasma - modelled as a series of 2d slices through which the beam will propagate. The first species, "electrons", is always electrons of the background plasma. Additional species should be numbered sequentially, with the total number set in `Nspecies` in `Domain`. The different distributions available are in `plasma.cpp`. The list has grown organically as we needed things, so I'm afraid there's no reason to it. Distribution 1 is Gaussian, which is probably a good place to start (but note the radii correspond to $\sqrt{2}\sigma$).

The simulation domain has a size `Xlength`×`Ylength`×`Zlength` with cell size `hx`×`hy`×`hz`. The simulation window propagates in the x-direction. The plasma sheets wash over the driver, advanced with a "timestep" of `hx/NxSplit`. The beam species are then advanced with timestep `Ts`. The (beam) timestep only needs to to resolve evolution of the system in the co-moving frame - in many cases, it's sufficient to chose `Ts` sufficiently small to resolve betatron oscillations. Further gains can be achieved using subcycling - updating the beam several times for each timestep of the bulk plasma. The simulation ends once `PhaseStop` has been reached.

3 Output

The code has several different outputs. `SavePeriod` is how often the beam particles should be saved. The particle save can be turned off on a per-species basis using `SkipSaveFlag=1` in the species definition. In the save files, stored at `h5files/vs***_3d_particles.h5`, particle positions are in centimetres, momenta are normalised to mc, and weight is normalised to the funamental charge unit.

The code also saves specific fields, so-called movies. They can be the full 3d fields, or 2d slices through the axis at z=0. Which fields to save is specified in the `MovieHDF5` and `Movie2dHDF5` sections near the bottom of the file. Remember to set the `NMovieFramesH5` and `NMovie2dFramesH5` in `Domain` to match. The 2d saves are generally sufficient, and are obviously much smaller and so can be saved more often. Setting `FieldsFlag3dH5=1` or `FieldsFlag2dH5=1` creates an additional FieldsFrame file every movie save, containing $E_{x,y,z}$ and $B_{x,y,z}$.

Alternatively, these can be saved in the mframe itself. A typical list could be

```
&MovieHDF5
  Frame0 = n0
  Frame1 = n1
  Frame2 = ex
  Frame3 = ForceY
  Frame4 = Al2
/
```

which would save the densities of the electron species and `Specie1`, the longitudinal field, the transverse force $E_y - cB_z$, and the normalised laser intensity. The axis data in the movie saves is normalised to $1/k$, densities to the characteristic density, and fields to the corresponding cold nonrelativistic wavebreaking limit. Movies are saved to `h5files/v3d_mframe_*****.h5` and `h5files/v2d_mframe_*****.h5` for 3d and 2d, respectively. The optional full field saves are stored at `h5files/v*d_FieldsFrame_*****.h5`.

The first movie frame is saved after the first timestep, whereas the first particle saves occurs after `SavePeriod` by default. This behaviour can be altered by setting `FirstSaveTime`.

We include a small set of scripts, `vlp3` (virtual laser plasma postprocessing), to make postprocessing a little easier for new users. This include scripts to visualise movie files and to calculate various statistics (RMS radius, emittance, average energy) of a beam. The collection will continue to be extended, and contributions are welcome.

4 Parallelization

Parallelization is achieved by dividing up the simulation domain between different processors, with information exchanged via MPI. The product of `Xpartition` $\times$`Ypartition`$\times$`Zpartition` should be the number of tasks you allocate (likely using either mpirun or srun, depending on your setup). As discussed above, the fields are solved as a boundary-value problem from the upstream edge of the simulation box. Parallelization in x therefore results in a cascade - the first (upstream) slice solves the first timestep, then the second slice solves the first timestep while the first solves the second timestep, and so on. This is an important consideration for parallelization - if you only make 50 steps, there's no gain parallelizing more than 50 times in x, and even this will mean half your processors are empty at any given time. The parallelization in y and z is limited by the FFTW parallelization, and so using more cores isn't necessarily faster. Play around with it a bit, but it may be beneficial to run several simulations at once, rather than using all cores on a single job. The associated speedup of the quasistic treatment typically far outweighs the limitations in parallelization imposed by the model.

The parallelization of the code no longer forces all domains to be the same size. This is convenient when trying to run the same simulation on different numbers of processors (note that the random seed will still be different). The mesh geometry for each processor is saved in the particles file. This lets you do e.g. binning using the cell number (given cell number is for the local mesh), which is much faster.

5 Advanced topics

5.1 Domain size

For large simulations, it is worth checking the size of the whole simulation domain (reported near the start of v*.log). The FFTW algorithm is fastest if the lengths of the FFT domain are the product of small primes. For RFFT with quadratic or NGP interpolation, the FFT domain is equal to the simulation domain in y and z. For linear interpolation, it is the domain size $+1$ (particles in cells $0..N-1$ deposit charge to cell boundaries $0..N$). Accidentally choosing a prime value for the FFT range can significantly increase computation time ($\sim 30\%$).

Short example - I have a simulation domain of $1000 \times 400 \times 400$. I'm using trilinear interpolation, so the FFT domain is 401×401. 401 is a prime number, so my simulations are needlessly slow. I decrease the transverse domain size by 1 cell, simulations are faster. Bear in mind that floating point rounding can give weird results, so check the domain size given in the log!

5.2 Particle seeding

The typical behaviour in many PIC codes is to initialise a set number of particles spaced regularly within each cell. In `qv3d`, this grid loading is the default. Beam particles are modelled in 3d, so you should choose a cube for the number of particles per cell. The plasma is modelled as a 2d sheet, so here you should choose a square. These are important, as other values will lead to the same incomplete filling of the grid for every cell, which results in a bulk anisotropy and can give nonphysical results. The particle weight is proportional to the value of the density function at centre of the cell.

Grid loading can be turned off for beams by setting `RandomSeed = 1` in the species section. In this case, particles are initialised with a random position inside the cell.

Since charged beams in plasma undergo betatron oscillations, having a fixed number of particles per cell may not always be appropriate. For a cylindrical beam, the number of particles initialised at radius R would be proportional to R. After a quarter betatron period, the many low-weight macroparticles from the beam edge will have migrated to the centre, while the few high-weight macroparticles in the centre will have migrated to the edge. This results in a reduction in the resolution of the phase space distribution at the beam edge, which could act to seed instabilities. Rather than greatly increasing the number of particles per cell, one alternative is to use macroparticles of fixed weight, acheived by setting `EqualWeights = 1`. This initialises more particles in the high-density beam regions, so that the resolution of the phase space distribution should remain unchanged while undergoing betatron oscillations. The weight is determined such there are `P_perCell` particles per cell at the peak beam density. A non-integer particle number uses probabilistic rounding, so 6.1 particles has a 90% chance of being 6, and a 10% chance of being 7, and a 0% chance of anything else. It's possible to get good results using very few particles per cell, but this will also depend on your resolution. As ever, check if the results converge. `P_perCell` is still given as an integer in the input deck. Since this will result in a different number of particles in different cells, it should be used in conjunction with `RandomSeed = 1`.

`PhaseSpaceFillFlag` is now a simple binary, which gives an option for a "cold start" beam. In most cases, users should set this to 0.

For beam densities which vary rapidly on the scale of the grid, their initial size will be a convolution of the desired value and the cell size. In these cases, it may be preferable to initialise the beam externally using some script, and import the resulting distribution into `qv3d`. This is also the preferred approach for initialising a non-Gaussian transverse momentum distribution, e.g. moving the beam focus. See section **??** on how to do this.

5.3 Resume

It is possible to resume a simulation, for example after reaching `PhaseStop` or being killed by the scheduler. The quasistatic model requires only the beam particles to resume, i.e. fields and plasma particles are not required. To resume, set `Reload = 1`, and choose `Nwrite` as the as the save number to load. The code will load the `vs<savenumber>pe<tasknumber>_3d.dat` files, as well as `vs<savenumber>_3d_particles.h5` from the present working directory, which typically means making a soft link from the `h5files` directory. Note that the parallelization must be identical. When resuming from a completed run, remember to increase `PhaseStop`.

5.4 ContinueBack

As the code uses a moving window propagating at the vacuum speed of light, causality is entirely from the upsteam boundary along the length of the beam. The code provides the functionality to save all fields at, and particles crossing, a fixed point in the co-moving frame, allowing further simulations to be carried on from this point. This `ContinueBack` functionality is especially useful for situations where a short drive beam needs to be optimized for a given driver, as only the portion of the simulation containing the witness needs to be repeated. It also provides the ability to split a large simulation, avoiding the memory constraints which often act to limit 3d simulations.

To make use of this functionality, the base simulation should set `RearPosition`, the point at which the fields should be saved, to some positive value. This creates two additional files, `v3d_ContinueBackBeams.h5` and `v3d_ContinueBackFields.h5`.

The continue simulation can be set by choosing `ContinueBack = 1` in the `Controls` section of the input deck. The ContinueBack files are used to provide the field and create particles at the upstream simulation boundary. The continue simulation may have a smaller timestep than base simulation, but the transverse size, transverse cell size, transverse parallelization, and `hx/NxSplit` must match. The `PhaseStop` of the continue simulation cannot exceed the runtime of the base simulation. Typically, the continue simulation is started in another directory, making a soft link to the ContinueBack files.

In principle, this functionality could be extended to any PIC code using a moving window, but the saves would become prohibitively large due to the small timestep. When not using the ContinueBack functionality, `RearPosition` should be set to a negative value to avoid using disk space unnecessarily.

At present, ContinueBack does not support quadratic interpolation in the propagation direction – set `ParticlePusher = 15` for quadratic interpolation in the transverse direction, linear in the longitudinal direction. Saving ContinueBack data supports resuming a completed simulation, but not a job killed prematurely by the scheduler.

5.5 FollowParticles

The standard method for saving particles works only for beam particles. In most cases, it is the collective behaviour of the plasma which is important, for example the plasma currents `jx`, `jy`, `jz` which can be saved in the movie files. If, for some reason, you really want to look at the trajectories of individual plasma particles, there is an option for saving a 1d slice of plasma particles (taken from a single cell at $z = 0$) as it propagates over the beam. This requires some effort, so first consider whether or not you really need it.

In order to save plasma particle trajectories, set `FollowParticles = 1` in domain, and optionally `FollowParticlesSaveStride = X` to save every `X` timesteps (steps, not phase). Then, set `FollowFlag = 1` for the plasma species in question, and optionally `FollowStride = X` to save every `X`th particle. Each process saves its data in a different file, e.g. `follow8_PE123.data` for plasma particles from the eighth step from process 123. This data can be quite large, which is why the strides are included, and why there's no option to save all plasma particles.

The data will have the form: particle ID, x, y, z, p_x, p_y, p_z, weight
Of course, a particle will probably pass through several processes as it propagates over the beam, you need some way of sorting the different trajectories. The bash command `sort` is useful to combine the data from all processes into a single file, and `grep` to get a specific particle ID. You'll need to find an approach that suits you. Be aware this will likely be some work.

As ever, start with a small example and work up.

5.6 Bexternal

To add an external focusing field, you need to add an extra option `NBexternal = 1` in `Domain`.

Then create a new section, and add values for

```
&Bexternal0
  BxTesla =
  ByTesla =
  BzTesla =
  Focusing =
  Ambipolar =
/
```

default `Ywidth=Zwidth=1` TODO: clean and explain

5.7 Import from file

TODO

6 Conclusion

This is a quick introduction the code, it doesn't document all the possible things that you can do. Some of the undocumented options in the input deck have only one good value, and others have no effect at all, vestiges from when qv3d was forked from VLPL. I would suggest that you not try to understand them all – if it ain't broke, don't fix it.

I typically find the best way for new users to get to grips with the code is just to get stuck in. There are a couple of known bugs, so read release.txt first. My main advice, as with any simulation model, is to check that your results converge (e.g. halving the cell size or timestep shouldn't affect your results). What resolution you can get away with will depend on the physics that you're modelling.

If you have questions, or if you get stuck, feel free to ask. If the code doesn't have some functionality that you need, let us know. It may be that we can implement it, it may already be implemented and not yet documented, or there may be a good workaround.

Good luck, and have fun.