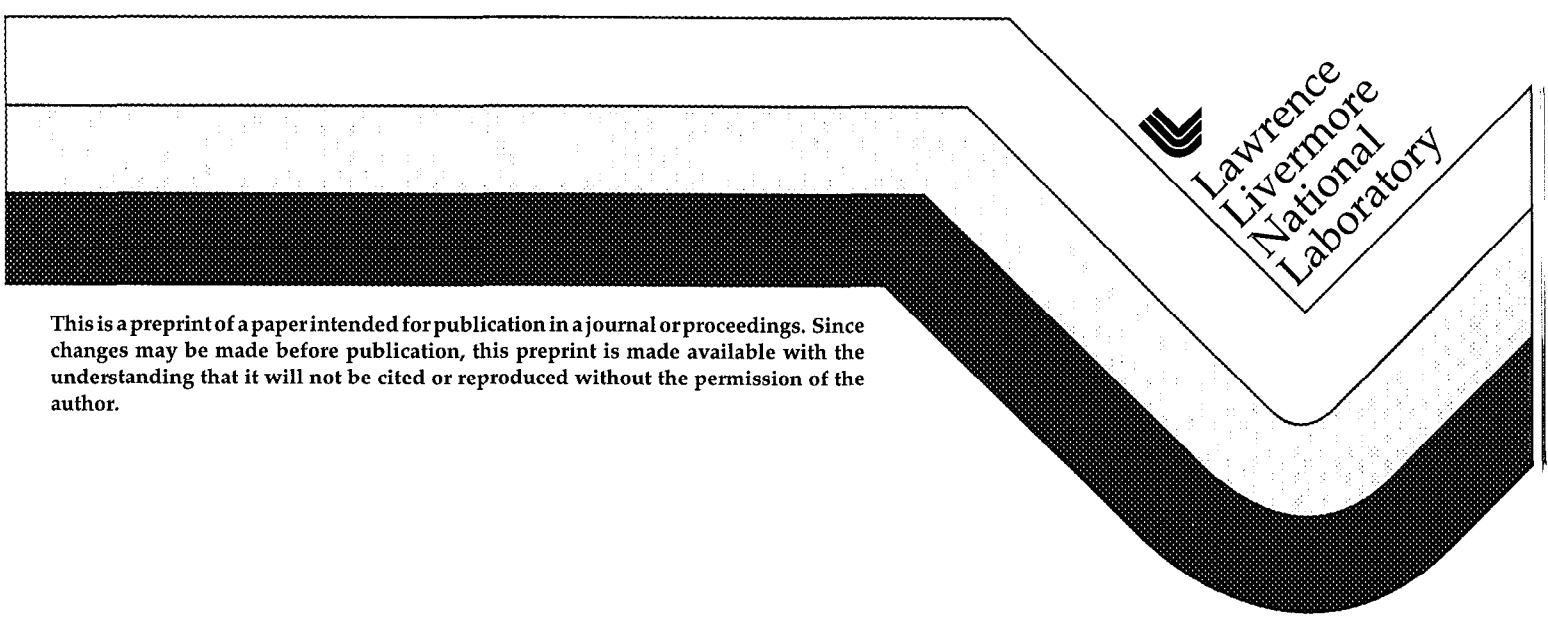
UCRL-JC-129415
PREPRINT

New Methods in WARP, a Particle-in-Cell Code for Space-Charge Dominated Beams

D. P. Grote
A. Friedman
I. Haber

This paper was prepared for submittal to the
16th International Conference on Numerical Simulation of Plasmas
Santa Barbara, CA
February 9-12, 1998

January 12, 1998



New Methods in WARP, a Particle-in-Cell code for Space-Charge Dominated Beams

David P. Grote and Alex Friedman, LLNL
Irving Haber, NRL

Introduction

The current U.S. approach for a driver for inertial confinement fusion power production is a heavy-ion induction accelerator; high-current beams of heavy ions are focused onto the fusion target. The space-charge of the high-current beams affects the behavior more strongly than does the temperature (the beams are described as being "space-charge dominated") and the beams behave like non-neutral plasmas. The particle simulation code WARP[1] has been developed and used to study the transport and acceleration of space-charge dominated ion beams in a wide range of applications, from basic beam physics studies, to ongoing experiments, to fusion driver concepts.

WARP combines aspects of a particle simulation code and an accelerator code; it uses multi-dimensional, electrostatic particle-in-cell (PIC) techniques and has a rich mechanism for specifying the lattice of externally applied fields. There are both two- and three-dimensional versions, the former including axisymmetric (r-z) and transverse slice (x-y) models. WARP includes a number of novel techniques and capabilities that both enhance its performance and make it applicable to a wide range of problems. Some of these have been described elsewhere[2].

Several recent developments will be discussed in this paper. A transverse slice model has been implemented with the novel capability of including bends, allowing more rapid simulation while retaining essential physics. An interface using Python as the interpreter layer instead of Basis has been developed. A parallel version of WARP has been developed using Python.

Transverse Slice Model

For some induction accelerator studies the longitudinal effects can be small, and so simulating a transverse slice of a beam is often sufficient. In fact, the earliest particle codes developed for space-charge dominated beam simulation were slice codes. Since slice codes already existed and since WARP was originally developed to provide new functionality (a three-dimensional and an axisymmetric code), a slice version was not originally created. Now that the WARP3d and WARPrz codes are mature, a slice code, called WARPxy, was developed that takes full advantage of the functionality of the existing codes and introduces new capabilities beyond those offered by earlier codes.

The slice code uses the longitudinal position, s, for the independent variable, tracking the slice from location to location. (this is different than the 3D and RZ codes, which use time as the independent variable.) At each slice, the transverse position and velocity are known, as well as the longitudinal velocity. Since there may be a longitudinal velocity spread, each particle has its own time step size which is inversely proportional to its longitudinal velocity.

One of the new capabilities of WARPxy is that it allows a changing longitudinal velocity; this is important when there are axial electric fields, such as in an accelerating gap, and when there are transverse magnetic fields, such as in a magnetic quadrupole element. On each time step, since the longitudinal velocity may change, a new time step size must be calculated for each particle. The new time step size must be calculated consistently with the longitudinal velocity so that the particle travels exactly to the next slice location. Though the time step size can in some cases be expressed analytically, an iterative scheme is used instead for generality.

An additional (and related) capability of WARPxy is that it can follow a slice through a bent accelerator lattice. Instead of representing the beam as slices of a cylinder, in a bend the beam is represented by slices of a torus. Here, as with a changing longitudinal velocity, the time step size for each particle must be recalculated on each time step since the distance to the next slice location varies across the beam. The same iteration loop used to account for a changing longitudinal velocity is used for bends. The bent geometry is accounted for in the solution of the field by solving Poisson's equation in cylindrical coordinates. On entrance or exit of a bend, sub-steps are taken so that the particles land exactly on the edge of the bend so that the transformations and time step size corrections can be done exactly.

The particle advance is done using the split leap-frog scheme as shown in figure 1. When the longitudinal velocity is changing and/or when the length of the step varies transversely in a bend, the time step size, dt, is not known in advance. The first two steps of the leap-frog advance are iterated over to calculate the correct time step size. On each iteration, the time step size is scaled by the ratio of the desired step size, ds, and the distance actually traveled by the

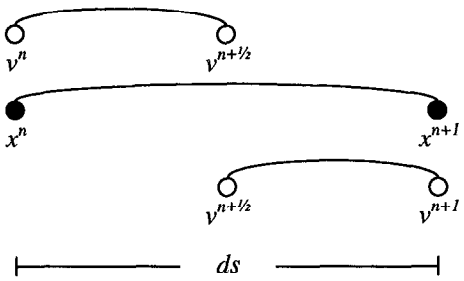


Figure 1. The particle advance is done using a split leap-frog scheme. The first two steps are iterated over until $v_z^{n+1/2}dt = ds$.

particle. In a straight lattice, that distance is $v_z^{n+1/2}dt$. In a bent lattice, the distance is calculated as an angle traversed around the bend. This iteration converges rapidly; three or four iterations are generally sufficient.

WARPxy was created from WARP3d and takes advantage of its capabilities. It can use the full lattice description and all of the diagnostics and post-processing developed for WARP3d. Additionally, the 3-D field solver can be used in situations where internal conductors are present; any longitudinal forces from the conductors are included in the particle advance. This close connection to WARP3d allows a "thick slice" model, where longitudinal self-fields of a finite length beam are preserved but all of the particles are assumed to be at the same longitudinal position relative to the lattice of external fields. With the thick slice model, beam dynamics (especially oscillatory modes) that have a longitudinal scale length similar to the beam or pipe radius can be more efficiently simulated.

Numerous tests have been carried out examining the slice code, all giving good results. A stringent test has been simulation of a recirculating induction accelerator with magnetic quadrupoles, a system with changing longitudinal velocity and bends. It has been shown (analytically and in simulations with WARP3d) that a beam with a longitudinal velocity spread will undergo emittance growth after the transition from a straight lattice to a bent lattice[3]. The emittance growth in simulations with WARPxy of the transition into our recirculator experiment has shown excellent agreement with analytical estimates and WARP3d results, both in the magnitude and rate.

Since its development, WARPxy has seen extensive use. Among other uses, it has been used to help understand ongoing experiments, to study resonances and resonance crossing in repeating lattices, and to act as a test bed for alternative, higher order advance schemes[4].

Python

WARP was originally built on top of the Basis[5] system, which allows interactive (run time) control of the code and access to its internal database. Over the last several years, other interpreters have been developed and have become of interest. One of the more interesting ones is Python[6], which is a fully object-oriented interpreter that can be easily extended with compiled code. Its advantages are that its kernel is small and portable (it is written in standard C) and that it has a large base of developers who are producing extensive and useful packages. What makes Python especially interesting is that it can readily be used on a parallel architecture. Because of this, an effort has been made to connect WARP into Python.

The one big drawback of Python is that it currently does not have an automated process that provides access to compiled routines and data. With Basis, the code developer creates a file in which the variables and routines that are to be accessed from the interpreter are listed. This file is used by Basis to create the code that allows the access. Since these variable description files have already been created for WARP, it is desirable that this mechanism be maintained when using another interpreter. We have had to develop a variable description file parser and wrapper generator (written in Python, of course) that extracts the necessary information from the variable description files and creates the interface between Python and the compiled code (all Fortran in our case), allowing access to the compiled code from the interpreter.

Basis was designed so that there is a direct connection between interpreter layer variables and the compiled data (which is located in Fortran common blocks). So, when compiled code is called, it has access to all of the user-inputted data through the common blocks instead of requiring all of the data to be passed down through argument lists. This allows a model where data is allocated and set from the interpreter and then directly used within the compiled code. The interpreter layer of Basis acts like a Fortran main routine that has all of the common blocks included.

Though Python variables are fundamentally different from Basis variables, the above model can still be used. Everything in Python is an object and all variables are references to objects. On an assignment, a variable will refer to the new object and loses any information about what it was previously referring to. Because of this, a direct connection cannot be made between a Python variable and compiled data, since reference to the data is lost on assignment to the variable. However, compiled data can be accessed through variable attributes (which are the same thing as class data and methods of C++). The setting (and getting) of variable attributes is different than assignment since it is done through routines which are defined as part of the specification of the variable type.

The interface to the compiled code is generated by creating a new Python type for each of the WARP packages. The compiled data is then accessed through attributes of the types. The only drawback of this mechanism is that it requires more typing, needing the package name as a prefix to all compiled variables. For example, the time step size, dt, which is in the package named "top" is referred to by "top.dt" in Python instead of simply "dt" as it is in Basis. This drawback can be alleviated for arrays, though, since assignments can be done using array slicing which copies the data instead of rereferencing the variable name. Care must be exercised not to lose the reference, though.

The Python interface has been used and tested on a number of workstations and on a Cray T3E (see next section). It must be said that the interface is not complete. Some sizable issues still need to be addressed, such as restartable data dumps, and much cleaning up and refinement is still needed.

Parallel WARP

While many accelerator problems relevant to heavy-ion fusion can be realistically simulated on modern day serial machines, there are many problems of interest and importance that require significantly greater computing power. For example, high-resolution simulations of the LBNL electrostatic quadrupole injector[7] and multi-lap simulation of the small recirculator experiment[1] each require on the order of one CPU-day on a Cray C90. While single runs are not extravagant, many runs are needed to sufficiently characterize the systems – that becomes difficult to deal with. Full simulations of a driver scenario would require one or more orders of magnitude more computational time. The need for these larger simulations has led us to develop a parallel version of the three-dimensional code WARP3d.

For parallelization, the domain decomposition is done along the longitudinal axis of the beam. Since the beam is generally longer than it is wide, that decomposition is the clear choice. While decomposition of the transverse plane would be an improvement, it is not essential since the systems will generally be long enough to make good use of many processors with the one-dimensional decomposition. The particles are divided using the same decomposition. Timings of WARP3d have shown a speedup that is nearly linear in the number of processors. Table 1 compares the timings of WARP3d on several different machines and with several different numbers of processors on a Cray T3E900.

Table 1. The runtime for two different sizes of a typical simulation is compared for various machines. The numbers are the time in seconds to complete one time step. The pentium 2 is a state-of-the-art PC running Linux. The J282 and Alpha 440 represent high-end workstations. The two Crays are the supercomputers. For both simulations, the supercomputers offer superior performance. The PC is showing impressive performance and has the highest performance per dollar for our applications.

	Pentium 2 300 MHz	HP J282	Alpha 440	Cray C90	Cray T3E 900		
					16	32	64
100K particles 64x64x256 grid	4.24	2.05	1.96	0.69	0.21	0.11	0.09
1M particles 128x128x512 grid	52.4	19.0	25.5	5.03	3.17	1.61	0.82

The most difficult aspect of the development of the parallel version of WARP3d has been the user interface. WARP was tied to Basis, which has not been ported to a parallel machine, and so something different had to be done. The initial attempt was to have WARP, running with Basis on a local serial machine, act as a master that spawned Basis-free WARP processes in the parallel environment. The user would interact with the code via the locally running WARP, which in turn would communicate with the parallel processes via message passing. While that method appears elegant, it is hampered by two significant weaknesses. The first weakness makes the code hard to use. The user does not easily get access to the full runtime database since only variables for which special data passing routines are written are accessible. The second makes the code unusable in some environments. The method relies on having remote execution privileges to start up the parallel processes and interactive use of the computer. Some computer centers do

not give remote execution privileges and limit the interactive use of the computer. Also, requiring a remote connection has the problem of code crashes if there are network problems.

To satisfy the need for full access to the runtime database and running directly on the machine with the option of using batch, an interpreter that can be used on the parallel machine must be used. This led to the use of Python. The work described above creating an interface to WARP with Python can be directly carried over to a parallel environment with one additional piece of work. The user interaction with multiple processors must be dealt with. For that, code developed by others was used[8]. An interface was developed where only one processor handles all input and broadcasts it to the other processors. The user controls which processors can create output.

With this work, the WARP code now has a flexible and powerful interpretive interface that is the same on all machines, serial and parallel. One important aspect of the interface is having the same plotting capabilities on the parallel machine as on the serial machine.

Conclusion

WARP has proven itself to be a valuable tool in the examination and understanding of the behavior of space-charge dominated beams. While WARP has become in some ways a mature code, it is still continually evolving. As discussed in this paper, we have added a lower-dimensional slice model that allows for faster scoping and examination of problems while retaining important physical phenomena. WARPxy has already proven its usefulness. An additional part of the development is keeping the code up to date with available tools and machines and looking toward the future. We have developed the interface between WARP and the Python interpreter. This has freed us from complete dependence on Basis and has given us an interpretive interface on a larger variety of environments. Specifically, it has given us an interactive interface in a parallel environment, giving us access to substantial computational power.

References

1. D. P. Grote, et. al., "Three-Dimensional Simulations of High Current Beams in Induction Accelerators with WARP3d", Fus. Eng. & Des., 32-33 (1996) 193-200.
2. D. P. Grote, A. Friedman, I. Haber, "Methods used in WARP3d, a Three-Dimensional PIC/Accelerator Code", Proceedings of the 1996 Computational Accelerator Physics Conference, AIP Conference Proceedings 391, p. 51.
3. J. J. Barnard, H. D. Shay, S. S. Yu, A. Friedman, and D. P. Grote, "Emittance growth in Heavy Ion Recirculators", 1992 Linear Accelerator Conference Proceedings, Ottawa, Canada, vol. 1, p 229, AECL Research, (1992).
4. A. Friedman, et. al., these proceedings.
5. P. F. Dubois, The Basis System, LLNL Document M-225 (1988).
6. The best source is the web page, http://www.python.org/
7. S. M. Lund, et. al., "Numerical Simulation of Intense-Beam Experiments at LLNL and LBNL", 12th International Symposium on Heavy-Ion Inertial Fusion, to be published in Nuclear Instruments and Methods A.
8. The primary authors are D. Beazley and T. B. Yang.

Author:

David P. Grote
LLNL L-645
P.O. Box 808
Livermore, CA 94550
grote1@llnl.gov

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48.

Technical Information Department • **Lawrence Livermore National Laboratory**
University of California • Livermore, California 94551