SYNERGIA: A MODERN TOOL FOR ACCELERATOR PHYSICS SIMULATION

P. Spentzouris[*], J. Amundson, FNAL, Batavia, IL 60510, USA

Abstract

High precision modeling of space-charge effects, together with accurate treatment of single-particle dynamics, is essential for designing future accelerators as well as optimizing the performance of existing machines. Synergia is a high-fidelity parallel beam dynamics simulation package with fully three dimensional space-charge capabilities and a higher order optics implementation. We describe the computational techniques, the advanced human interface, and the parallel performance obtained using large numbers of macroparticles.

INTRODUCTION

In recent years, accurate modeling of beam dynamics in high-current low energy proton synchrotrons has become necessary because of new machines under consideration for future applications, such as the High Energy Physics neutrino program, and the need to optimize the performance of currently operating machines, such as the Spallation Neutron Source or the Fermilab Booster. These machines are characterized by high currents and require excellent control of beam losses, thus space-charge initiated halo formation is an essential component of their modeling. Beam halo causes losses which lead to irradiation of the beam line tunnel. In order to obtain accurate predictions for realistic conditions of operation, single-particle optics and self-consistent multi-particle effects must be combined in a single simulation code. Single-particle optics model the effects of the external forces on the beam, i.e. magnet and rf cavity generated forces. Space charge, a multi-particle effect, involves the interaction of the beam with itself due to its own charge. Since space charge depends on the beam distribution it must be recalculated as the beam evolves.

Several computer simulations of space-charge effects in circular accelerators using particle-in-cell techniques have been developed [1, 2, 3]. These simulations have emphasized the transverse dynamics while using a less rigorous approach for the longitudinal dynamics. Synergia [4] is a package for state-of-the-art simulation of linear and circular accelerators with a fully three-dimensional treatment of space charge, and the ability to use arbitrary order maps for the single-particle optics modeling.

Synergia is designed to be a general-purpose tool with an interface that is accessible to accelerator physicists who are not experts in simulation. Space-charge calculations are computationally intensive, typically requiring the use of parallel computers. The implementation of Synergia is fully parallel, including the particle tracking and space-charge modules. The code itself is a hybrid system based on previously developed accelerator physics codes. Synergia includes enhancements to these codes as well as new integration and interface modules. There is at least one other example of an accelerator code framework which reuses existing codes [5]. Synergia is unique in that it is designed to provide a high level framework specifically for studying 3D multi-particle dynamics in a massively parallel computing environment.

Development of Synergia has been funded by the United States Department of Energy's SciDAC Accelerator Science and Technology Project. One of the goals of the project is to create distributable codes. Since compiling hybrid code can be a complicated task which is further complicated by the diverse set of existing parallel computing environments, Synergia includes a build system that allows it to be compiled and run on various platforms without requiring the user to modify the code and/or build system.

COMPONENTS

The two packages at the core of Synergia are IMPACT [8] and the *mxyzptlk*/*beamline* libraries [9]. We have added glue code and a human-interface wrapper to these packages, together with necessary extensions of their modules, to form the Synergia package.

IMPACT

Synergia uses IMPACT for its parallel particle-in-cell (PIC) implementation, rf modeling and, most importantly, parallel space-charge calculations. IMPACT contains a suite of three-dimensional, FFT-based Poisson solvers that are invoked in the middle of each step of a split-operator-based model. We split the Hamiltonian into two pieces,

$$H = H_{\mathrm{ext}} + H_{\mathrm{sc}}, \tag{1}$$

where H_{ext} is the Hamiltonian for the external beamline element part of the problem and H_{sc} is the Hamiltonian for the space-charge part of the problem. In our case the latter is simply proportional to the scalar potential, with a proportionality constant that varies as $\frac{1}{\gamma^2}$ to account for the azimuthal magnetic field associated with the longitudinal beam current. Since the scalar potential depends only on coordinates and not momenta, the effect of H_{sc} is a change in momentum, i.e. a space-charge kick, which we denote by $\mathcal{M}_{\mathrm{sc}}$. The effect of H_{ext} is described by the transfer

[*] spentz@fnal.gov

map for the associated beamline element, $\mathcal{M}_{\rm ext}$. In the split-operator approach, an approximation to the effect of the full Hamiltonian, H, accurate through second order in the step size h, is given by

$$\mathcal{M}(h) = \mathcal{M}_{\rm ext}(h/2)\mathcal{M}_{\rm sc}(h)\mathcal{M}_{\rm ext}(h/2) + \mathcal{O}(h^3). \quad (2)$$

The problem of calculating beam propagation including space-charge effects therefore factorizes into the problem of calculating the two effects one at a time and combining them as above. A key advantage of this splitting, as opposed to one that separates the Hamiltonian into pieces involving only coordinates and only momenta, is that in our approach the rapid variation of the external fields is separated from the more slowly varying space-charge fields. In other words, we can take small steps to accurately resolve rf cavity fields, magnetic fringe fields, etc., needed to compute external transfer maps, but the separation between space-charge kicks can be large (typically a few tens of kicks per betatron wavelength). Without the factorization above, we would be forced to calculate the space-charge effects on the time scale set by the magnetic optics effects, which would be computationally prohibitive.

We have extended the original IMPACT in several ways. IMPACT now includes an injection module, allowing multi-turn injection modeling. We have extended the beam generation module to include a six-dimensional Gaussian distribution with general correlations. We have also improved the memory management, allowing for an arbitrary number of beamline elements. Finally, we have enhanced the IMPACT particle data structure to allow following individual particles throughout the simulation and calculating particle tunes.

mxyzptlk/beamline *libraries*

The *mxyzptlk/beamline* package is a set of C++ libraries covering a wide range of accelerator physics computations. Even though the original code is over 10 years old, the libraries are written in a modern style, including real objects with encapsulation and well-considered interfaces. The package includes *basic_toolkit*, a set of useful utility classes such as Vector, Matrix, etc., *beamline*, objects for modeling elements of a beamline, *mxyzptlk*, classes for automatic differentiation and differential algebra, and *physics_toolkit*, a set of classes for analysis and computation.

A desirable feature of the *mxyzptlk/beamline* package for our purposes is the ability to read accelerator descriptions in the MAD8 language [10]. Synergia passes a MAD8 file and lattice name to *beamline* which returns transfer maps for an arbitrary number of lattice slices. The MAD8 parser in *beamline* is limited to processing accelerator lattice descriptions since the Synergia interface is much more flexible than the MAD8 command language. In a generic Synergia run lattice elements from MAD8 files can be combined in arbitrary ways and even mixed with native IMPACT/Synergia elements. Synergia also takes advantage of *mxyzptlk/beamline*'s arbitrary-order transfer maps.

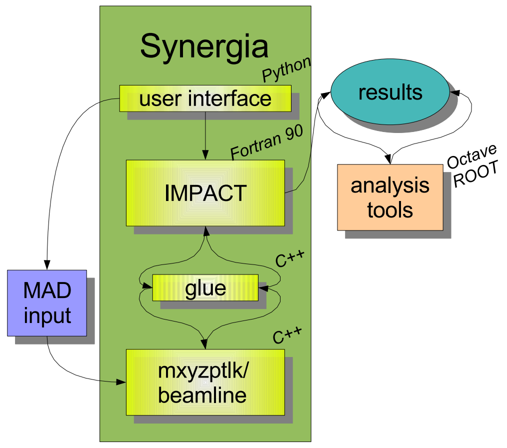

Figure 1: Synergia components and their relation to outside inputs.

The current implementation utilizes first- and second-order maps, but generalization to arbitrary orders is planned for the near future.

SYNERGIA

Synergia is the combination of IMPACT, the *mxyzptlk/beamline* libraries, glue code to get the two packages talking to each other, and a user interface wrapper providing a straightforward, yet powerful, human interface. Figure 1 shows the relationship between Synergia components, MAD8 files, and analysis tools.

Build System

Portability has been a major design concern in creating Synergia. We rely on multiple components written in multiple languages. While using multiple components allows us to quickly put together a powerful package, it also creates a configuration management problem. Multiple-language issues are particularly problematic because calling conventions vary from platform to platform. We solve the multiple language part of the problem by writing all of the inter-language wrapper code in terms of macros that can be redefined for various platforms. We solve configuration management problem by incorporating a modern build system based on the GNU Autotools to provide consistent builds on all platforms.

In principle, building Synergia is as simple as executing "`./configure && make && make install`" in the *mxyzptlk* directory followed by "`./configure && make`" in the Synergia directory. In practice, many options to configure are available. The two principles we have followed in constructing the build system are (1) modifying the source (including Makefiles) should never be necessary, and (2) all options should come with reasonable defaults. To date, Synergia builds without modifications on Linux systems using either the Portland Group F90 compiler or the Intel F90 compiler, g++ or Intel CC, and either the MPICH or LAM implementations of the Message Passing Interface (MPI). Synergia also builds without modifications on AIX, using XL Fortran, Visual Age C++ and POE.

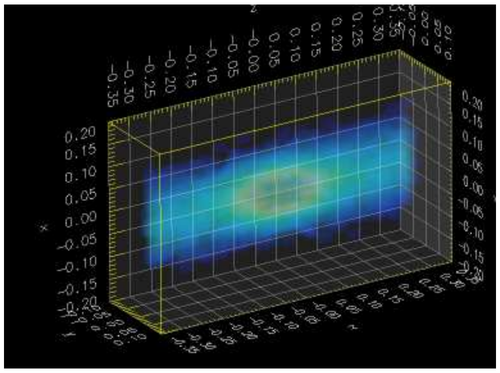

Figure 2: OpenDX visualization of a three dimensional histogram of particle density of a FNAL Booster simulation.

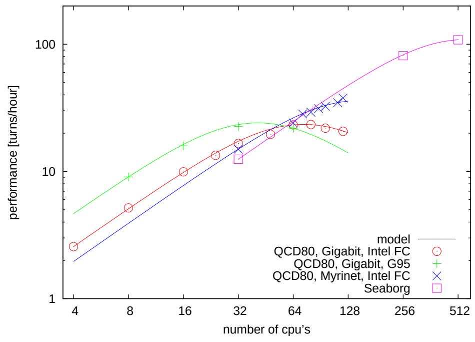

Figure 3: Synergia performance measured in simulated turns per hour versus number of cpu's for different configurations.

Human Interface

The user-level interface to Synergia consists of a set of Python classes that wrap the low-level interfaces to the code. To run Synergia, the user writes a short Python script utilizing these classes. An example script excerpt is shown in Figure 4. The use of Python has several advantages: There is no specialized syntax to learn. A user familiar with Python will be able to understand the entire interface easily. A user unfamiliar with Python will be able to copy an example script and modify it with little difficulty. Although most examples will only use Python trivially, the full power of the language is available should it be needed. Last, but not least, the use of an existing scripting language greatly simplifies the implementation, minimizing both the development time and the probability for introducing bugs.

Job Description Every Synergia job is a simple Python script. Synergia provides the class `Impact_parameters` as an interface to the internal parameters of IMPACT, including input beam, energy, space-charge parameters, etc. The accelerator lattice can be defined using elements from an external MAD8 file.

Synergia provides a simple matching module to generate matched beams, utilizing linear optics calculations from *mxyzptlk*/*beamline* for lattice function determination. We also provide an interface to our Octave utilities package that generates a matched beam in the presence of space charge by solving the r.m.s. envelope equations. The interface uses Octapy, a Python module we have developed to allow exchange of data structures between Octave and Python, as well as arbitrary Octave code execution from Python.

Job Creation and Submission Synergia jobs can be arbitrarily complex. Typically, the user will want to run several different jobs varying only a few of the many input parameters. Synergia provides several facilities to assist the user in creating, submitting and managing simulation runs.

A Python module *options* provides a simple method to write scripts accepting command-line arguments for Synergia and user-defined parameters. When a Synergia job script is run, the command-line options for that job are automatically recorded in a manner so that they can be edited and/or reinvoked. Synergia also records all job parameters in a database with a human-readable summary file.

Synergia automatically generates batch system submission scripts based on a user-supplied template. Several example templates are provided, including templates for single-processor machines, multi-processor machines, the PBS batch system and more. Optionally, jobs can be defined to run on remote machines. Synergia generates scripts to export the input files to the remote machine, submit the job, and retrieve the files from the remote machine once the job is finished.

Diagnostics A number of diagnostics are provided by default during the simulation run. In addition, we provide tools to allow users to analyze simulated data after a simulation has completed. The standard diagnostic utilities are evaluated at each split-operator step and include calculations of the second, third and fourth moments of all six degrees of freedom, two-, four-, and six-dimensional emittances, and all pairwise correlations for beam components.

For post-processing, we provide the ability to dump the entire beam, or a sampled subset of the beam, at any simulation step. Files can be dumped in plain text or HDF5 format [11]. Each particle is saved along with a unique tag so that individual particles can be tracked throughout the simulation. We provide tools for rearranging a series of particle dumps into individual tracks, both for diagnostic purposes and calculating particle tunes. The output format of the particle information dumps can be easily interfaced to visualization packages such as OpenDX [12]. An example of such visualization of a Fermilab Booster simulation is shown in Figure 2.

```
ip = impact_parameters.Impact_parameters()
ip.processors(4,16)
ip.space_charge_BC("trans finite, long periodic round")
ip.input_distribution("6d gaussian")
mad_file = "booster.mad"; mad_line = "booster"
energy = myopts.get_value("energy")
(alpha_x, beta_x, alpha_y, beta_y) = \
          madcalc.twiss_initial(mad_file,mad_line)
# Set horizontal parameters based on beam width measurement
width_x = myopts.get_value("xwidth")
eps_x = width_x**2/beta_x
(width_xprime, r_x, emittance) = \
               matching.match_twiss_width(width_x,alpha_x,beta_x)
ip.x_params(sigma = width_x, lam = width_xprime * pz)
# Set vertical parameters so that emittance_horizontal == emittance_vertical
(width_y, width_yprime, r_y) = \
          matching.match_twiss_emittance(emittance, alpha_y, beta_y)
ip.y_params(sigma = width_y, lam = width_yprime * pz)
numinjturns = myopts.get_value("numinjturns"); numturns = myopts.get_value("numturns")
x_offset= 0; y_offset= 0; phase_offset=0; output_num = 0
# Run the simulation for around the mad line for numturns turns
for turn in range(0,numturns):
    ip.add(impact_elements.External_element(
        kicks=96, steps=10, radius=0.04,
        mad_file_name=mad_file, beamline_name=mad_line))
my_synergia = synergia.Synergia(ip,sys.argv,synergia.options)
my_synergia.prepare_run(myopts.get_value("dirname"))
```

Figure 4: Example excerpt of a Synergia Python script showing lattice description from MAD8 file, beam matching utilizing Synergia's matching module, geometry setup, and simulation run.

Performance

In Figure 3 we compare performance and scaling behavior of Synergia on QCD80, a cluster of 700 MHz Pentium III machines running Linux and Seaborg, the IBM SP at NERSC. The simulation is of the FNAL Booster using a $33 \times 33 \times 257$ grid. We include a comparision with our model of parallel scaling, $t = c_0/N + c_1 N$, where t is time and N is the number of cpu's. The parameters c_0 and c_1 are related to the computational and network performance, respectively.

REFERENCES

[1] F. Jones, G. H. Mackenzie, and H. Schonauer, Particle Accelerators,vol. 31, 199 (1990).

[2] S. Machida, in Computational Accelerator Physics, edited by R. Ryne, AIP Conf. Proc. No. 297 (AIP, New York, 1994), 459.

[3] J. Galambos, J. Holmes, D. Olsen, A. Luccio, and J. Beebe-Wang, ORBIT User's Manual, Oak Ridge National Laboratory, SNS/ORNL/AP Technical Note No. 011, 1999.

[4] http://cepa.fnal.gov/psm/aas/Advanced_Accelerator_Simulation.html

[5] N. Malitsky, R. Talman, in AIP Conf.Proc. No 391 (AIP, New York, 1997), 337.

[6] Booster Staff 1973 *Booster Synchrotron* ed E L Hubbard *Fermi National Accelerator Laboratory Technical Memo TM-405*

[7] http://scidac.nersc.gov/accelerator/mli/manual.pdf

[8] J. Qiang, R. D. Ryne, S. Habib and V. Decyk,

[9] L. Michelotti, FERMILAB-CONF-91-159 *Presented at 14th IEEE Particle Accelerator Conf., San Francisco, CA, May 6-9, 1991.*

[10] F.Christoph Iselin, "The MAD program(Methodical Accelerator Design) Version 8.13/8", Physical Methods Manual, CERN/SL/92, 1992.

[11] The Hierarchical Data Format, http://hdf.ncsa.uiuc.edu/

[12] http://www.opendx.org