

Synergia: a hybrid, parallel 3D space charge code with circular machine modeling capabilities

James F. Amundson and Panagiotis Spentzouris

Fermi National Accelerator Laboratory, P.O. Box 500, Batavia, Illinois, 60510

Abstract. We describe Synergia, a hybrid code developed under the DOE SciDAC-supported Accelerator Simulation Program. The code combines and extends the existing accelerator modeling packages IMPACT and mxyzptlk. We discuss the design and implementation of Synergia, its performance on different architectures, and its potential applications.

E-mail: amundson@fnal.gov, spentz@fnal.gov

1. Introduction

Synergia is the product of a project funded by the Department of Energy's SciDAC Advanced Accelerator Modeling Project. The mission of the project is to create a distributable code that builds upon existing accelerator physics software, as opposed to writing new software from scratch. As such, Synergia is a hybrid code built out of existing, but possibly modified, components.

By utilizing existing components, we have managed to produce a code with extensive capabilities in a relatively short time. Synergia is currently capable of modeling both linear and circular machines. It can read machine descriptions from Methodical Accelerator Design (MAD) files. It also has a full three-dimensional space charge model and is fully parallelized. This functionality was all derived from the existing components. Synergia also has a powerful and flexible user interface, based on Python, which is original to Synergia. Synergia is a work in progress, however, we have already used it to model a real machine.

We start by describing the two main existing components that comprise the physics content of Synergia. We then discuss the synthesis of the existing codes into Synergia. Finally, we discuss applications of the code to circular machines including the performance on various computer architectures.

2. Components

Synergia a hybrid code; most of its physics capabilities are derived from existing components. These codes are described in detail elsewhere, so we only discuss their features which are used in Synergia here.

2.1. IMPACT

The first component that makes up Synergia is IMPACT [Qiang]. IMPACT was originally designed to model space-charge effects in linear accelerators. The model of space-charge it implements is fully three-dimensional. IMPACT uses the split operator technique to calculate the combined effect of internal and external electromagnetic fields. The internal fields are determined by solving the Poisson-Vlasov Equation using particle-in-cell (PIC) methods. The Poisson-Vlasov Equation must be solved at fixed time. IMPACT uses a ballistic approximation to go from the longitudinal coordinate z to the temporal coordinate t ,

$$x^t = x^z - \frac{c}{\omega} p_x \psi / \gamma, \quad (1)$$

$$y^t = y^z - \frac{c}{\omega} p_y \psi / \gamma, \quad (2)$$

and

$$\delta z = -\frac{c}{\omega} \beta_z \psi. \quad (3)$$

IMPACT is implemented in Fortran 90. The implementation is fully parallel; both particle tracking and space charge calculations are parallelized. Further information on the space charge model in IMPACT is given by R. Ryne in these proceedings.

2.2. *mxyzptlk/beamline Libraries*

The second preexisting component utilized by Synergia is the *mxyzptlk/beamline* libraries [Michelloti]. This set of libraries was the first C++ package for accelerator physics. Even though they were originally written over ten years ago, they are written in a modern style, i.e., they provide real objects with encapsulation and well-considered interfaces, as opposed to some early efforts at scientific C++ which were closer to translation of Fortran codes to C++ syntax. The libraries include *basic_toolkit*, a set of useful utility classes such as Vector, Matrix, etc., *beamline*, objects for modeling elements of a beamline, *mxyzptlk*, automatic differentiation and differential algebra, *beamline*, a set of objects for modeling elements of a beam line and *physics_toolkit*, a set of classes for analysis and computation. In addition, the package includes a set of models of the accelerators at Fermilab.

In Synergia, we take advantage of the generalized propagator functors and MAD file parser available in these libraries. The MAD file parser allows us to immediately take advantage of existing MAD-based machine descriptions. The libraries can easily take an entire machine described in MAD and return a transfer map divided into an arbitrary number of slices. The libraries are flexible enough to make extracting this functionality very straightforward.

3. Synergia

Synergia is a hybrid code; the primary physics calculations are done by a combination of IMPACT and the *mxyzptlk/beamline* libraries. Combining codes which use different

paradigms and are written in different languages requires facing several problems. The first question that must be answered is the most basic: how are the different paradigms to be combined? Once that conceptual problem is solved, the technical problems of language mixing and configuration management also need to be addressed.

3.1. Paradigm Mixing

IMPACT and the mxyzptlk/beamline libraries are two very different codes. The former is a standalone program with a simple user interface based on purely numerical input. The latter is a strictly a library; performing a calculation requires with it writing a program. The user interface is therefore a set of C++ interfaces. Since the library approach is the most flexible, it is easiest to mold it into IMPACT's paradigm than the other way around.

We have expanded IMPACT to recognize a new type of beam line element, which we label simply as "external." The implementation of the external element is simply a set of hooks to glue code that, in turn, calls the mxyzptlk/beamline libraries. The external element as implemented reads in a MAD file and uses it to produce transfer maps. The details of the external element are opaque to IMPACT. IMPACT's space-charge and parallel distribution routines are then utilizable without further modification. The remaining problem is the user interface, which we address in a subsequent section.

3.2. Language Mixing

Mixing IMPACT's Fortran 90 code with the C++ code in mxyzptlk/beamline creates a technical problem. Technically, no standard exists for calls between the two languages. In practice, the problem requires determining exactly which function-mangling and type conventions are being used on a given platform/compiler combination. We have encapsulated all name-mangling and type conversion issues in preprocessor macros. As a result, the code can be compiled on multiple platform/compiler combinations without any modification to the code itself; only command-line switches are needed.

The problem of language mixing is further complicated by the fact that linking codes with elements of both languages is often non-trivial. We address this problem in the subsection on configuration management. To date we have ported Synergia to AIX and Linux. Although the space of possible unix-like platforms is large, AIX and Linux live in very separate corners of this space. We therefore have confidence that our code will be easily portable to other unix-like platforms.

3.3. Configuration Management

Configuration management is not often discussed in connection with simulation codes. It can, however, be a source of much difficulty. In situations where multiple packages are being used, configuration management can rapidly become a bottleneck. We have addressed this problem from the outset in our development of Synergia.

The configuration management problems in Synergia consist of keeping track of which components are installed where, how the code is to be built, and what options are to be given to the compiler and linker. We use the GNU Autotools for this purpose. Although the Autotools have their limitations, they do provide the most comprehensive and flexible features we are aware of.

In principle, compiling Synergia is as straightforward as typing

```
./configure && make && make install
```

in the mxyzptlk/beamline package, followed by

```
./configure && make
```

in the IMPACT package. In practice, there are several options available. We allow for alternate locations for the glib libraries (used by mxyzptlk/beamline), MPI libraries (used by IMPACT) and the mxyzptlk/beamline libraries. We also allow the user to specify the C++/C compilers and options, Fortran 90 compiler and options and additional linker options. In all cases, we attempt to determine, or at least guess, workable options on each platform.

3.4. User Interface

As we described above, the Synergia main loop is controlled by IMPACT. The IMPACT user interface consists of a text file with purely numerical parameters, many of which must be specified in the internal unit system used by IMPACT. We provide a python wrapper providing a friendlier user interface. The wrapper provides unit conversions, sensible defaults, and batch queue job creation and submission. Because each job is a python script, the full power of python is available to the user. Figure 1 shows the IMPACT input files corresponding to a sample run of the FNAL Booster. In Synergia, the user may instead enter the equivalent input shown in Figure 2. In the Synergia script all parameters are given in units of GeV and meters.

The Synergia python wrapper invokes IMPACT as a separate process instead of linking in the IMPACT code and calling it directly. The former approach is much simpler in practice because it avoids all problems with linking and calling Fortran 90 functions from Python on different platforms.

4. Applications

Although Synergia is still under development, we have already successfully used it to model the Fermilab Booster. Since IMPACT, which controls the main loop, was designed for linear machines, we have had to address some problems in the extension to circular machines. One conceptually simple problem is multiple-turn injection. In order to accommodate multiple-turn in IMPACT we have added another “beam line element,” the injection element. It allows additional particles to be injected with arbitrary offsets with respect to the beam center. We also have had to address the more complex problem of a long beams with curvature, which we discuss in the following subsection. Having done so, we describe the performance and

```

16 4
6 2746880 1 0 1
65 65 65 4 0.04 0.04 1.51692
2 0 0
0.00268186, 0.000106579, 0 1.000000 1.000000 0.000000 0.000000
0.00268186, 0.000106579, 0 1.000000 1.000000 0.000000 0.000000
0.0940268, 0.000427895, 0 1.000000 1.000000 0.000000 0.000000
0.042 4e+08 9.38272e+08 1.000000 2.01e+08 0.000000
  0 0 81 -2 270.000000/
  474.2 100 1 91 1.0 0. 0.040000 0. 0. 0. 0. 0. /
  0 0 83 -2 270.000000/
  474.2 100 1 91 1.0 0. 0.040000 0. 0. 0. 0. 0. /
  0 0 84 -2 270.000000/

```

etc.

Figure 1. IMPACT input example.

```

p = impact_parameters.Impact_parameters()
ip.processors(16,4)
ip.space_charge_BC("trans finite, long periodic round")
ip.input_distribution("6d gaussian")
ip.pipe_dimensions(0.04,0.04)
ip.kinetic_energy(0.400)
ip.scaling_frequency(201.0e6)
ip.x_params(sigma = .004 , lam = 1.0e-4)
ip.y_params(sigma = .004 , lam = 1.0e-4)
pz = ip.gamma() * ip.beta()*ip.mass_GeV
ip.z_params(sigma = 0.10, lam = 3.0e-4 * pz)
ip.particles(2700000)
ip.space_charge_grid(65,65,65)
booster = impact_elements.External_element(length=474.2,
                                           kicks=100, steps=1, radius=0.04,
                                           mad_file_name="booster.mad")

for turn in range(1,11):
    ip.add(booster)

```

Figure 2. Synergia input example.

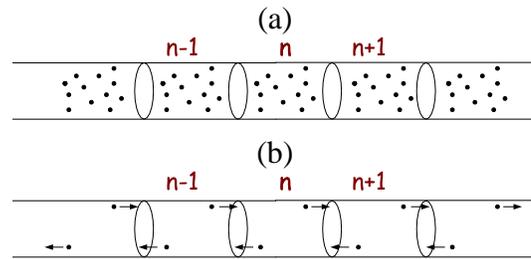


Figure 3. Periodic boundary conditions. (a) n^{th} period and neighbors. (b) Particles entering and leaving n^{th} period and its neighbors.

scaling behavior of the code on various architectures. We discuss the comparison of Synergia results with studies of the Fermilab Booster in a separate contribution to these proceedings.

4.1. Long Beams

In order to apply Synergia to long beams in circular machines, we slice the beam into short slices. We can easily do this in the case of beams with periodic structure. (Of course, a uniform beam can be considered a limiting case of a periodic beam.) A natural choice for the periodicity of a synchrotron is the period of the rf structure inherent to the injected beam. We have used this choice in our modeling of the Fermilab Booster.

Instead of modeling an entire long beam, we model a single slice in the midst of the beam. We use periodic boundary conditions to incorporate the effects of the portions of the beam outside the slice. As illustrated in Figure 3, one effect of the boundary conditions is to cause particles leaving the beam slice from the leading face to enter through the trailing face. The net effect of this procedure is to allow us to use a short beam slice with negligible curvature to model a beam that is long on the scale of the radius of a circular machine.

4.2. Performance

We have run benchmarks for Synergia on four machines of different configuration. We use the GNU C++ compiler, g++, on all machines. For the purposes of benchmarking all code was compiled with the flags “-g -O2”.

The first machine we used is Seaborg, the AIX machine at NERSC. Seaborg has 2,944 375 MHz POWER3 processors. We used the native IBM Fortran 90 compiler on this machine. At the other end of the spectrum, we ran the code on abacus, an 800 MHz Pentium III laptop running Linux, where we used the Intel Fortran compiler. We also ran on two Linux clusters. The first is NERSC cluster Alvarez, which has 80 dual 866 MHz Pentium III processors connected with Myrinet networking. We used the Portland Group Fortran compiler on Alvarez. The second cluster is the Fermilab Beams Theory Group’s Heimdall, 32 dual 1.4 GHz Athlon processors. At the time we ran these benchmarks Heimdall was connected with 100 Mbit/s ethernet. In the meantime, part of Heimdall has been updated to Gigabit ethernet. We used both the Portland Group compiler and the Intel compiler on Heimdall.

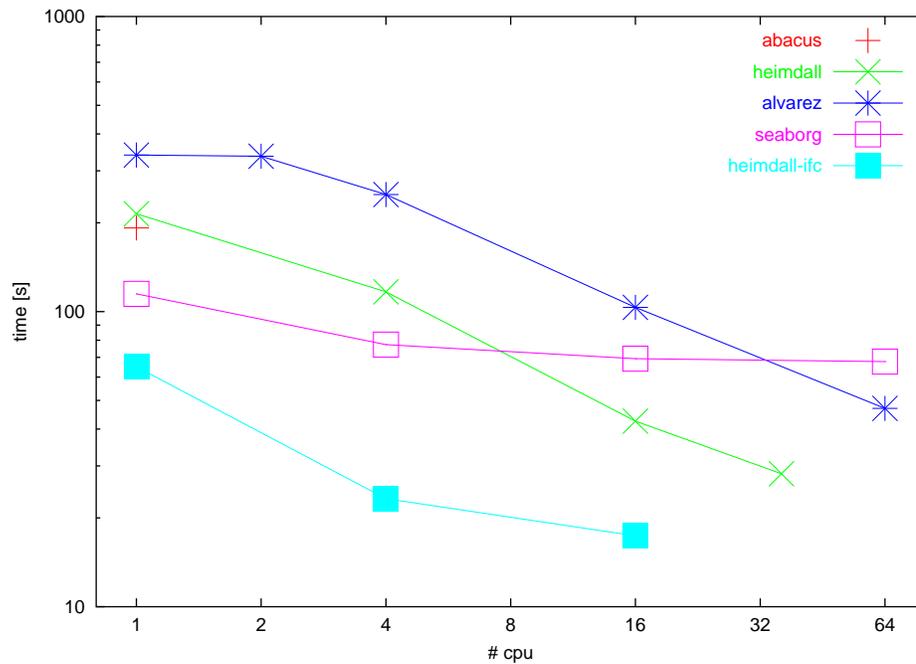


Figure 4. Performance of various configurations on an example without space charge.

In Figure 4 we show the results of running a single turn of the Fermilab Booster without space charge effects. The simulation contained 100,000 particles and 200 individual transfer maps. The surprising result was the difference in performance between the Portland Group and Intel compilers. We were able to obtain somewhat better performance out of the Portland Group compiler by adjusting the optimization flags, but we were not able to match the performance of the Intel compiler with any combination. The poor scaling on Seaborg is not entirely understood, but we did not deem the issue worth pursuing because the scaling is really important only when the much more computationally-intensive space charge calculations are included.

In Figure 5, we show the results of running a single turn of the Fermilab booster including space charge effects. In this case we included 2.7 million particles and 100 space charge kicks with the corresponding 200 transfer maps. Several features are noticeable in the results. Since the space charge calculation requires intensive inter-processor communication, the effects of networking are very important. The expensive Myrinet networking in Alvarez shows itself to be very effective in maintaining scalability up to 64 processors. The complex internal networking of supercomputer Seaborg also shows excellent scaling behavior. Although Heimdall did very well in the case without space charge, we see that the slow networking limits scaling behavior at higher processor numbers. The newer Gigabit networking on Heimdall should improve the situation.

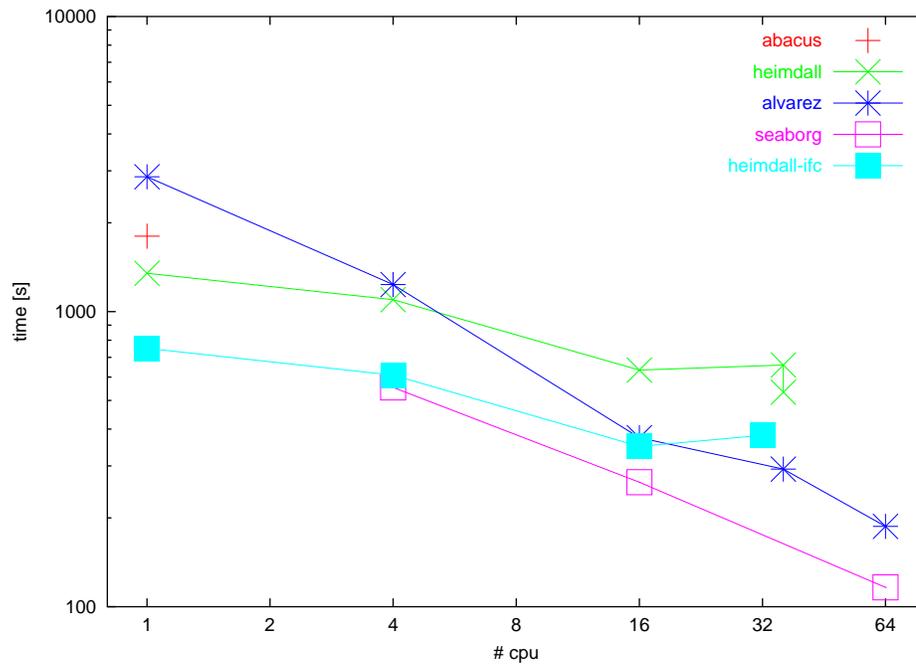


Figure 5. Performance of various configurations on an example with space charge. The two points for 36 processors on Heimdall reflect different geometries for the parallelization.

5. Conclusions

Synergia is a parallel three-dimensional space charge code for modeling both linear and circular machines. It is primarily a combination of the existing IMPACT code and the mxyzptlk/beamline wrapper. In addition to providing a way to combine the capabilities of these existing codes, it provides a Python interface that is both human-friendly and powerful. Although the code is already in use for studies of the Fermilab booster, Synergia is still a work in progress. The code can be obtained by contacting the authors.

References

- [Qiang] J. Qiang, R. D. Ryne, S. Habib and V. Decyk, *J. Comput. Phys.* **163**, 434 (2000).
- [Michelloti] L. Michelotti, FERMILAB-CONF-91-159 *Presented at 14th IEEE Particle Accelerator Conf., San Francisco, CA, May 6-9, 1991.*
- L. Michelotti, FERMILAB-FN-535-REV.
- L. Michelotti. Published in Conference Proceedings: *Automatic Differentiation of Algorithms: Theory, Implementation, and Application*. Society for Industrial and Applied Mathematics. First International Workshop on Computational Differentiation. 1991.
- L. Michelotti. Published in Conference Proceedings: *Advanced Beam Dynamics Workshop on Effects of Errors in Accelerators, their Diagnosis and Correction*. Corpus Christi, Texas. October 3-8, 1991. American Institute of Physics: Proceedings No.255. 1992.