SYNERGIA: A HYBRID, PARALLEL BEAM DYNAMICS CODE WITH 3D SPACE CHARGE

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

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 beamline/mxyzptlk. We discuss the design and implementation of Synergia, its performance on different architectures, and its potential applications.

INTRODUCTION

Synergia is an accelerator physics simulation code with a fully three-dimensional space-charge model and circular and linear machine modeling capabilities. The implementation is fully parallel. Development of Synergia has been funded by the Department of Energy's SciDAC Advanced Accelerator Modeling Project. The goals of this project include building upon existing simulations and creating distributable code. Synergia is compatible with these goals because it is a hybrid code; the primary accelerator physics components are taken from existing, although possibly modified, codes. In the interests of distributability, we have taken care to ensure that Synergia is easy to build on various architectures.

Below, we give a brief description of the components used in Synergia as well as the details involved in combining them into a single product. We pay close attention to the build system, in keeping with the "distributable" goal mentioned above. We also describe how we have taken advantage of Python to give us a flexible, humane user interface with very little effort. Since space-charge calculations are computationally intensive, we present benchmarks for our code running on various parallel clusters. Finally, we compare Synergia simulation results with results from a recent accelerator study.

COMPONENTS

The two packages at the core of Synergia are IMPACT[1] and the mxyzptlk/beamline libraries[2]. We have added glue code and a human-interface wrapper to these packages to form the Synergia package.

Impact

Synergia uses IMPACT for its parallel implementation of particle propagation, RF modeling and, most importantly, parallel space-charge calculations. IMPACT contains a fully three-dimensional space charge model utilizing the split-operator technique. The split-operator technique is applicable for Hamiltonian of the form

$$H = H_{\rm ext} + H_{\rm sc}, \qquad (1)$$

where, in our case, $H_{\rm ext}$ is the Hamiltonian for the magnetic optics part of the problem and $H_{\rm sc}$ is the Hamiltonian for the space-charge part of the problem. If the transfer maps corresponding to the individual Hamiltonians $H_{\rm ext}$ and $H_{\rm sc}$ are $\mathcal{M}_{\rm ext}$ and $\mathcal{M}_{\rm sc}$, respectively, then

$$\mathcal{M}(t) = \mathcal{M}_{\rm ext}(t/2)\mathcal{M}_{\rm sc}(t)\mathcal{M}_{\rm ext}(t/2) + \mathcal{O}(t^2) \qquad (2)$$

is the transfer map for H to leading order in t. 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. The space-charge effects in IMPACT are calculated by solving the Poisson-Vlasov Equation using particle-in-cell (PIC) methods. The magnetic optics effects vary quickly, but require little CPU time to compute. The space-charge effects vary slowly, but require a great deal of CPU time to compute. 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.

Mxyzptlk/Beamline Libraries

The mxyzptlk/beamline package is a set of C++ libraries covering a wide range of accelerator physics computations. This package was the first C++ library for accelerator physics. 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 include *basic_toolkit*, a set of useful utility classes such as Vector, Matrix, etc., *beamline*, objects for modeling elements of a beamline including a full parser for the Methodological Accelerator Design (MAD) language, *mxyzptlk*, automatic differentiation and differential algebra, and *physics_toolkit*, a set of classes for analysis and computation.

One of most important features of the mxyzptlk/beamline package for our purposes is the ability to read accelerator descriptions in the MAD language. Since MAD has become the *lingua franca* of accelerator description, being able to directly use MAD files greatly enhances the usability of Synergia. The flexibility of the beamline/mxyzptlk libraries made it easy for us to utilize the features we needed, namely the MAD parser and generalized propagator functors. Synergia passes a mad file and beamline name to beamline and beamline returns an array of transfer maps divided into an arbitrary number of slices.

[*] amundson@fnal.gov
[†] spentz@fnal.gov

SYNERGIA

Synergia is the combination of IMPACT, mxyzptlk/beamline, glue code to get the two packages talking to each other and a wrapper providing a simple, yet powerful, human interface. Figure 1 shows the relationship between Synergia components as well as the role MAD files, studies and analysis tools play in producing results.

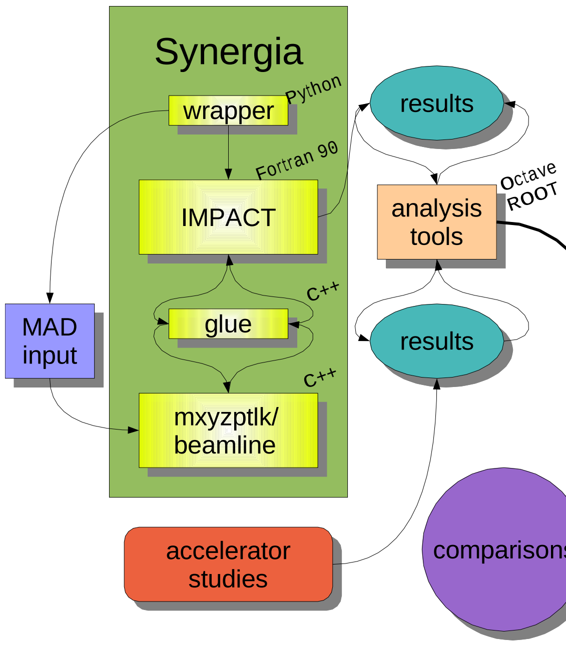

Figure 1: Synergia components and their relation to outside inputs.

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++, and either MPICH or lam. Synergia also builds without modifications on AIX, using XL Fortran, Visual Age C++ and POE. Compiling Synergia on other platforms should be a straightforward exercise.

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 2. 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 our implementation, meaning we were able to write it quickly with a minimum of opportunities for introducing bugs.

```
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: Example excerpt of a Python script showing the Synergia user interface.

Parallel Performance

We have run benchmarks of our code on four different clusters under a variety of configurations. Our benchmark is a simulation of a single revolution of the FNAL Booster (see the following section.) The simulation included 2.7 million particles undergoing 100 space-charge kicks on a $65 \times 65 \times 65$ grid.

Three of the clusters are Linux clusters: lqcd[4], heimdall[5] and Alvarez[6]. Our benchmarks include a sampling of the range of currently-available networking options for Linux: 100 Mbit Ethernet, Gigabit Ethernet and Myrinet 2000. We also compared the performance of the Intel fortran compiler (ifc) with the Portland Group fortran compiler (pgf90). For the former, the code was compiled with the optimization setting"`-O2`". For the latter the code was compiled with the setting "`-fast`". The fourth cluster we used for benchmarking was Seaborg[7], the 6,080-processor IBM SP at NERSC.

The results of our benchmarks are displayed in Figure 3. Overall, we find that Synergia scales very well up to a certain scale set by the networking used. The clear winner in scaling is the specialized configuration found in Seaborg. The fastest Linux clusters, however, showed overall superior performance. We can also see that Gigabit or Myrinet is necessary for a Linux cluster to effectively take advantage of more than a few processors. These tests were insufficient to distinguish between Gigabit and Myrinet. Somewhat surprisingly, we also see that the Intel compiler produced significantly better performance than the Portland compiler for our application.

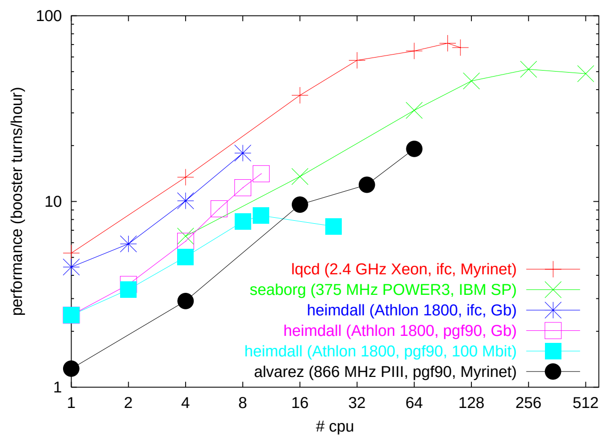

Figure 3: Performance on various parallel machines.

APPLICATION TO FNAL BOOSTER

The first important application of Synergia has been to model the FNAL Booster[3]. The Booster is an alternating gradient synchrotron with a radius of 75.47 meters. It accelerates protons from 400 MeV to 8 GeV with a typical injected current of over 450 mA. Since space-charge effects are expected to be significant in the Booster it is an excellent testing ground for Synergia.

As an example, Figure 4 shows a comparison between measured vertical and horizontal beam widths in the Booster with a Synergia simulation. Here 42 mA of current was injected in each of the 11 initial turns. For a more detailed discussion of recent FNAL Booster studies including comparisons to Synergia, see Reference [8].

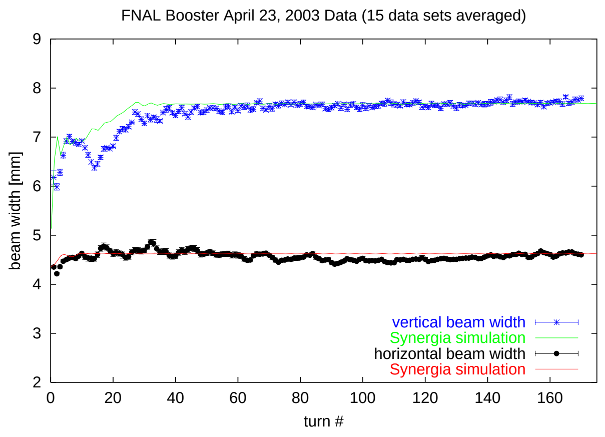

Figure 4: Synergia simulations compared to beam width measurements from the FNAL Booster.

REFERENCES

[1] J. Qiang, R. D. Ryne, S. Habib and V. Decyk, J. Comput. Phys. **163**, 434 (2000).

[2] 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.

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

[4] http://lqcd.fnal.gov/

[5] Linux cluster in the beams theory department at Fermilab.

[6] http://www.nersc.gov/alvarez/

[7] http://hpcf.nersc.gov/computers/SP/

[8] P. Spentzouris and J. Amundson, "Space charge studies and comparison with simulations using the FNAL Booster," Proc. International Computational Accelerator Physics Conference (ICAP 2002), Michigan State University, Oct. 2002; see also P. Spentzouris and J. Amundson, *these proceedings*.