CHEF: A FRAMEWORK FOR ACCELERATOR OPTICS AND SIMULATION*

J.-F. Ostiguy and L. Michelotti
Fermi National Accelerator Laboratory, Batavia, IL 60510, USA

Abstract

We describe CHEF, an application based on an extensive hierarchy of C++ class libraries. The objectives are (1) provide a convenient, effective application to perform standard beam optics calculations and (2) seamlessly support development of both linear and nonlinear simulations, for applications ranging from a simple beamline to an integrated system involving multiple machines. Sample applications are discussed.

INTRODUCTION

Around 1990, one of us (LM) initiated the development of a suite of libraries dedicated to accelerator simulation with an eye on non-linear dynamics [1]. The libraries would take advantage of Automatic Differentiation, a then emerging technique. High order derivatives are the backbone of perturbation analysis in nonlinear dynamics and AD can compute such derivatives to machine precision, something that standard finite difference techniques generally simply cannot deliver.

`C++` had just arrived on the scene and was selected as the implementation language because (1) by design, user-defined types can have nearly the same status as native types and (2) it provides comprehensive support for operator overloading. An additional practical consideration was that as, a superset of `C`, `C++` was well-positioned for commercial success and long term viability. This certainly turned out to be a correct assumption.

The vision was to create a framework allowing one to construct applications treating scalar and high order computations on the same footing. In principle, all code describing propagation of a particle through accelerator elements can be made to also implicitly keep track of derivatives with respect to the phase space coordinates through a simple type modification. For example, a `Particle` that would normally hold its phase space state in six `doubles` needs only to be redeclared as a `JetParticle` holding its state into six `Jet` objects. In this context, a `Jet` is simply a type which, in addition to its state coordinate, also keeps track of derivatives up to some specified order of the map involved in reaching that state.

To some extent, the initial vision was realized. Within a few years, the code base reached a relative level of maturity and the libraries were used in a number of specialized applications. Although correct results were produced, it became increasingly clear that the initial design had serious flaws. In particular, overhead associated with the utilization of a general n-th order code to perform first order computations (the basis of beam optics) was simply too high. Furthermore, for all the appeal and convenience of overloaded operators, performance was not in line with other available implementations of AD.

In mid-2003, a new development effort was initiated and CHEF was born. The objectives were to build (1) a convenient, intuitive general purpose optics tool supporting multiple popular platforms (2) a modern framework with high level components applicable to design and commisionning problems relevant to next generation machines (ILC, LHC, high intensity proton driver etc.). A decison was made to reuse a substantial portion of our existing code base. However, although the original vision remained as relevant as ever, a major overhaul, with emphasis on efficiency, was imperative. Algorithmic changes were introduced and, in the process, advantage was taken of design idioms and features of `C++` not generally available a few years ago. The details belong to another publication; suffice it to say that for first order (optics) computations, performance is now on par with codes based on conventional matrices. For high order computations, performance compares favorably to other available AD libraries. Finally, special attention paid to memory management yielded a substantially reduced dynamic memory footprint.

LIBRARY HIERARCHY

The CHEF framework is based on libraries and components organized in a hierarchical manner as shown in Fig. 1. Note that hierarchical refers to the fact that a library at a given level only depend on the libraries located below it. In this section we provide a brief description of each layer. More details are given in the following sections.

`mxyzptlk`

We can provide here a high level overview of automatic differentiation as implemented by the `mxyzptlk` library. Many interesting details are, unfortunately, beyond the scope of this article. `mxyzptlk` defines the basic types: `Jet__environment, Jet, Map` and `LieOperator`. `Jet__environment` encapsulates properties of the work environment such as dimensionality of the variable space, the maximum expansion order and the expansion reference point. In general, every `Jet` refers to a specific environment; `Jet` algebra can involve only `Jets` with compatible environments. A this juncture, it should be obvious that `Jet` is the fundamental data type that gen-

* Manuscript authored by Universities Research Association, Inc., under contract No.DE-AC02-76CH03000 with the U.S.DOE.

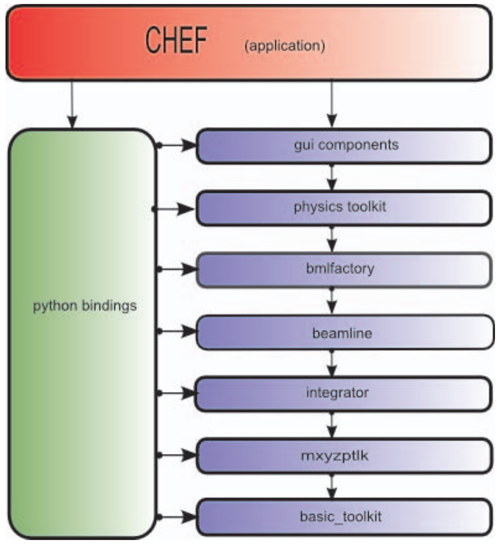

Figure 1: Hierarchical relation between CHEF and underlying libraries.

eralizes a scalar variable. Loosely speaking, it can be interpreted as a representation of a variable augmented by the coefficients of a nth-order differential form. A `Map` is simply a vector of `Jet` quantities; each component describes the corresponding coordinate transformation. Through operator overloading, `mxzypltlk` provides complete support for all basic operations on `Jets` as well as trigonometric functions, logarithms, exponential etc. In addition, the composition operator allows `Jets` and `Maps` to be concatenated, obviously an essential ingredient of accelerator related computations. The `Jet` representation in memory is sparse and ordered. Only monomonials with non-zero coefficients are stored. They are always ordered by increasing *weight* or total monomial order. The ordering is naturally preserved by the fact that basic add and multiply operations, in terms of which all other operations are ultimately decomposed, are always carried out in an ordered "register" or "scratchpad" that comprises all possible monomials. Elaborate reference counting and memory management strategies are employed to make the computational cost of instantiation and destruction of temporaries as low as possible.

`beamline`

The beamline library supports a wide variety of standard accelerator components types i.e. `sbend, rbend, quadrupole, etc` as well as the `beamline` type which is defined as a recursive to preserve hierarchical relations. Internally, the library uses true canonical coordinates, not optical coordinates. As a consequence, *computation within magnetic elements are performed using the true physical field*.

In contrast to many existing codes, *no implicit assumption is made about the the reference trajectory in individual elements*. The default propagation physics (based on either on symplectic thin kicks or exact integration) through any element can be overridden by the user. For example, to speed up tracking, one could request physics based on paraxial approximation in arc quadrupoles and a more exact formulation for low-beta quadrupoles. Elements or beamlines can be arbitrarily misaligned in three dimensional space. Note that no implicit assumption is made about the magnitude of the misalignments. Furthermore, geometric edge focusing effects can be accounted for naturally rather than through the introduction of artificial thin edge focusing elements.

`bmlfactory`

Beamlines can be instantiated from a description in an almost complete subset of the MAD8 language. Full support is provided for variable expressions. There are very few limitations; the most significant omission is a lack of support for macros. The parser is `lex` and `yacc` based and has been in use for a few years already. Very large and complex MAD8 descriptions are routinely successfully parsed. To accommodate the needs of the ILC project, a next generation parser capable of handling the `xsif` format (basically an extended version of the mad8 format suitable for both linacs and rings) is currently in development. The new parser will also lift restrictions with macros. Other parsers, including a parser for the MAD9 sequence format adopted by MAD-X exist, but are at this point, are not completely functional.

`physics_toolkit`

The `physics_toolkit` library provides a collection of standard optics and accelerator computations. Lattice functions can be computed for both periodic and non-periodic lattices. In addition to standard uncoupled lattice functions, three different methods are provided to deal with coupled lattice (1) the classical Edward-Teng functions, (2) lattice functions based on spatial eigenmodes projections or (3) a general beam envelope moment ($\sigma-$matrix) distribution. Tools for normal form analysis are also available.

`python-bindings`

A substantial fraction of the libraries public interface is available through `python`, a standard scripting language. The `boost.python` library is used to automatically generate binding code from a specification written in a simple declarative style. reminiscent of IDL.

Of all the popular scripting languages available, `python` provides one of the best "impedance match" to `C++`, that is, object-oriented features such as operator overloading, inheritance relations and virtual functions have a direct equivalent in `python`. A wide variety of existing `python` wrappers for linear algebra(e.g. LAPACK), signal processing, image processing, networking etc, can be leveraged, resulting in a very powerful facility that can be used to quickly prototype or develop accelerator simulation applications without having to deal with the complexity, time and knowledge overhead of a full-blown compilation based development environment.

CHEF, THE APPLICATION

CHEF and its underlying libraries contain virtually no platform specific code. The graphical user interface is based on Qt, a popular portable application framework available under various licenses, including the “free” QPL and GPL licenses. CHEF currently runs on the Linux, Sun/Solaris and Windows platforms; a Mac OS-X port should be available soon.

MAIN INTERFACE COMPONENTS

CHEF is a graphical application with a Multiple Document Interface (MDI), that is, all windows reside under a single parent window. As show in Fig. 2, upon launching the program, the user is presented with three main child windows: a Beamline Browser, a `python` interpreter, and a Message window. When beamlines are created, usually

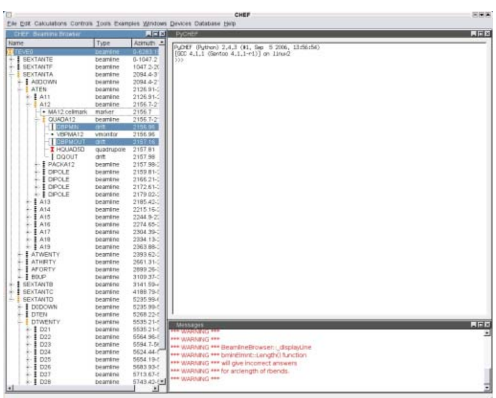

Figure 2: CHEF’s default window layout, with the Beamline Browser on the left hand side.

as a result of reading in a MAD8 format input file, they appear in the Beamline Browser window. The beamline is displayed graphically as a hierarchical tree whose branches can be expanded or collapsed with mouse clicks, in a manner analogous to the familiar interface used for hierarchical file directories. This provides a quick and efficient way of displaying the overall logical organization of a beamline or to home in on a region of interest. Beamline elements can be selected according to various criteria: pre-assigned text labels, element type, field strength, longitudinal position etc. or even a logical expression involving a combination of these criteria. Various operations can be performed on the beamlines themselves (e.g. insert monitors at specific locations); selected elements can also be used as input to another function (e.g. group a number of elements together to define a knob).

The `python` interpreter window provides an embedded `python` interpreter that is CHEF aware, that is, all the library bindings are integrated and available. Generally, the interactive interpreter is useful to run some quick tests or get status information about the CHEF application internals. In general, `python` input is not entered interactively but rather provided as separate input text file read either through a single command issued a the interactive prompt or an item in the main application menu. An important fact is that the existence of the embedded `python` interpreter enables the invocation of CHEF in pure command line mode (no GUI). Arbitrarily complex tasks (i.e. any task that can be described in the `python` language) can be performed.

Finally, the Message window logs all warning and error messages originating either from the application or from the underlying libraries.

EDITOR

Lattice files can either be read and parsed directly, or read into an editor and parsed interactively. In both cases, the parser identifies all the beamlines defined in the file; the user is subsequently presented with a dialog that allows him to select the beamline(s) he wishes to instantiate. At this point, the user is also allowed to override certain parameters such as the particle type or the beam energy. In the event the parser encounters an error, the problematic line is automatically highlighted in the editor window.

OPTICAL FUNCTIONS

As mentioned earlier, CHEF, the application, is meant to be a convenient, easy to use, general purpose tool for optics calculations. As such, it provides extensive capabilities to compute, display and save either uncoupled or coupled lattice functions for both periodic and non-periodic beamlines (e.g. transfer lines). Sample output is presented in Figs. 3 and 4.

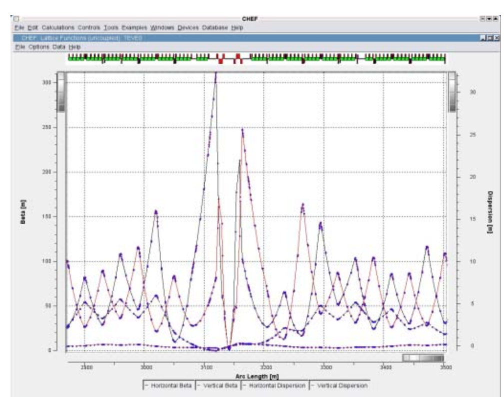

Figure 3: The standard optical function display. This sample shows the Tevatron lattice functions around the low-beta region.

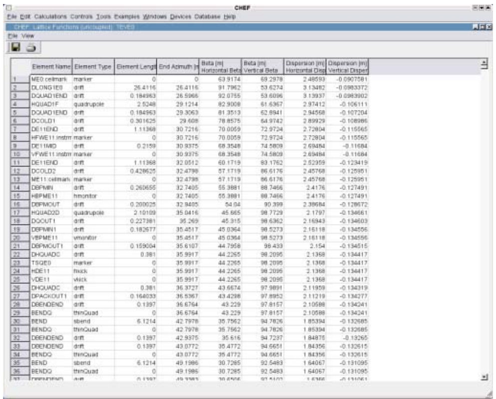

Figure 4: The optical functions presented in tabular form.

PHASE SPACE TRACKER

The phase space tracker (Fig. 5) allows one to interactively display phase space Poincaré sections. Such a display can be used to visualize the structure of resonances in a region of interest. Initial conditions of a particle can be specified either explicitly or interactively. Phase space coordinates are displayed as a persistent color-coded point each time the particle returns within the Poincaré section plane. Specifying a new initial conditions changes the point color. Nominally, the tracker provides two two-dimensional phase space cross-section displays. However, because OpenGL is used to render the phase space display window, it is also possible to generate three-dimensional phase space displays e.g. $(x, , y, p_x)$ or (x, p_x, py).

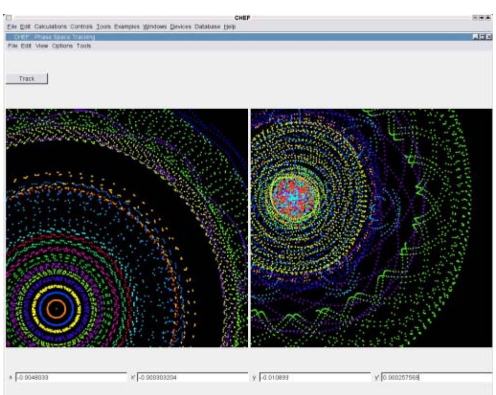

Figure 5: The phase space tracker.

TRACER

The tracer (Fig. 6) allows one to visualize the trajectory(ies) of one or more particles. In the same manner as the phase space tracker, trajectories are rendered persistently using OpenGL (useful for periodic trajectories). The use of OpenGL opens up the interesting possibility (unimplemented at this point) of visualizing beam envelopes and apertures in three-dimensions.

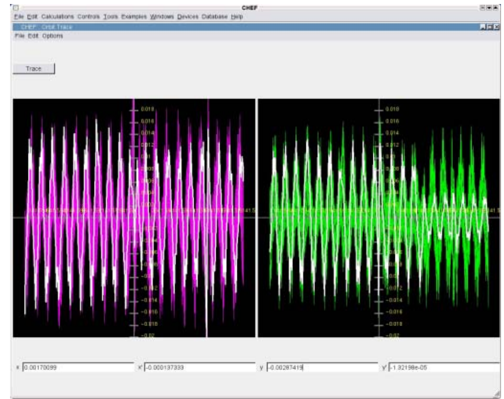

Figure 6: The trajectory tracer.

SITE VIEWER

The site viewer (Fig. 7) provides a three-dimensional display of a beamline in space. In addition, it allows one to generate and save floor coordinates for all elements.

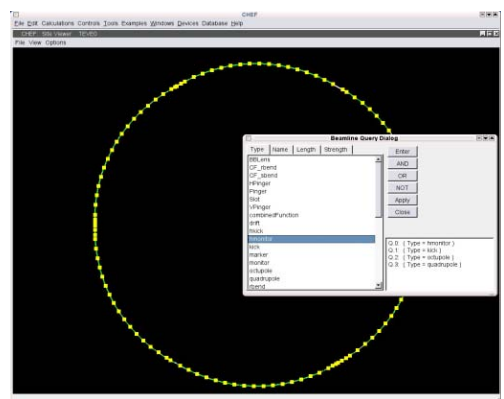

Figure 7: The site viewer. The display show a portion of the Tevatron layout. Horizontal monitors have been highlited.

CONCLUSION AND FUTURE PLANS

After much effort, CHEF and its underlying libraries have reached a point where they can be put to use and applied to real problems. In particular, the code is being used at Fermilab to study emittance preservation and beam-based alignment in the ILC. Agreement with other codes such as LIAR or Merlin is excellent and we anticipate integrating recently developed linac-specific functionality. Planned future improvements include support for physical apertures (already partially implemented) and particle loss as well as a basic matching capability.

REFERENCES

[1] L. Michelotti, "MXYZPTLK V 3.1 User's guide: A C++ Library for Automatic Differentiation and Differential Algebra. FERMILAB-FN-0535, Jan 1990.