TRACKING CODE DEVELOPMENT FOR BEAM DYNAMICS OPTIMIZATION*

L. Yang†, BNL, Upton, NY 11973, USA

Abstract

Dynamic aperture (DA) optimization with direct particle tracking is a straight forward approach when the computing power is permitted. It can have various realistic errors included and is more close than theoretical estimations. In this approach, a fast and parallel tracking code could be very helpful. In this presentation, we describe an implementation of storage ring particle tracking code `TESLA` for beam dynamics optimization. It supports MPI based parallel computing and is robust as DA calculation engine. This code has been used in the NSLS-II dynamics optimizations and obtained promising performance.

INTRODUCTION

`TESLA` is an accelerator lattice calculation and single particle dynamics tracking code. It starts as an engine for linear lattice optimization at ALS, LBNL and dynamic aperture (DA) tracking at NSLS-II, BNL. Recently this code was used with multi-objective evolutionary algorithm to optimize DA at various momentum and positive chromaticities [1] and satisfied performance are obtained.

The input file format for `TESLA` is quite similar to MAD8 but the hierarchy of elements and beamlines are kept internally. Expressions and references are available in nature form and common mathematical functions are supported. The input files can be nested by "include" command.

In `TESLA`, the physics model of magnetic elements are mainly symplectic integrator and the default is Yoshida scheme [2]. The dipole model is based on exact solutions using exact Hamiltonian [3] and this is much faster than the usual integration over slices. Insertion device such as damping wiggler is modeled by kickmap as usual.

Dynamic aperture (DA) tracking is done in both x–y and δ–x plane in a multiparticle way to improve the speed. Frequency map analysis (FMA) is implemented as part of the external toolkits, both serial and MPI based parallel versions are available. Visualization is done in accompanying Python scripts.

The strategy of `TESLA` development is that it is a library first then a standalone application. The library form can be called by scripting languages and serve as computing engine for parallel applications on clusters. The later is only for users interested in tools easy to learn and to use on their PC.

* Work supported by DOE contract number DE-AC02-98CH10886
† lyyang@bnl.gov

LATTICE FORMAT AND GENERAL PARSER

The input file is parsed by a module called GLPS (general lattice parser). It is designed for parsing the accelerator layout description file and outputs simple flat form. The flat form will have only numbers instead of mathematical expressions or references to variables and the beamline will be expanded into element list from a multiplication and reverse. Because it does not depend on keywords, this is general enough to be a parser for other accelerator simulation codes.

The input file of `TESLA` has three types of statements

1. Element and beam line definition. This is quite same as MAD8, but internally in the parser, hierarchy is kept in the beamline definition.

```
QF: Quad,L =0.2, K1=0.2, shift=(3e-6,5e-6,0),
   method="Yoshida4";
BD: Bend, L=2.0, angle=pi/60, e1=pi/120;
FAKE: LINE = (2*QF, BD, -BD, -2*(QF,QF,BD,BD));
```

2. Expression: `var = expression`, the expression can be single value or a vector.

```
speed_of_light = 3e8;
A = (0, .2, 0);
QF.K1 = 0.2; ! K1 field of QF element
QD.K2 = -QF.K1;
```

   Variables can be referred in the expressions. Common mathematical expressions and constants are included, e.g. `sin`, `exp`, `pi`.

3. Action. It starts with an action name, then a list of key and value pairs.

```
set, title="new lattice";
set, energy=3.0;
track, method="matrix", orbit=(1e-6,0,0,0,0,0);
```

The comments start with "`!`" for single line and are inside "`{}`" for one block. By using an action "`include, ''layout.lat''`", it is possible to nest one lattice file in the other and it is a good practice to put the lattice layout in one file while include it in different "action" file for different simulations.

GLPS (general lattice parser) can be used by any simulation code which has an input grammar as the following

```
statement: exprs | element | actions | line ;
exprs: var = expr
actions: action, exprs
element: family, exprs
line: line=(expr, ...)
```

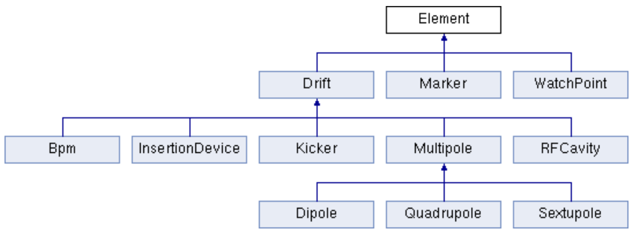

Figure 1: Class hierarchy of common elements

The input file is parsed to a linked list of key-value pairs, a flat form, and a simulation code will go through this list to construct elements. In this linked list, the multiplication, nesting and reverse of beamlines are expanded into list of elements, but the nesting structure is kept in its "full name". For example, a beamline

```
BL1: LINE=(D,B);
BL2: LINE=(BL1, D, 2*(B,F), -(C,E,-(F,G)));
```

will be expanded as BL1: (BL1.D, BL1.B) and BL2: (BL2.BL1.D, BL2.BL1.B, BL2.D, BL2.B, BL2.F, BL2.B, BL2.F, BL2.F, BL2.G, -BL2.E, -BL2.C). The new element name (full name) in the beamline has "domain" information, and its order is the order of nesting. The element which is defined in one place but put in several places in a beamline has different "full name", e.g. BL2.BL1.B, BL2.B. The multiple reverse sign (-) are merged according to "double reverse is identity" rule, but the parser can also keep the numbers of reverse sign. This approach has been more powerful and general than using pointer or a family name to represent elements from same definition.

OBJECT AND PHYSICS MODEL

The TESLA library is written in C++ in an object oriented way. The common storage ring elements are abstracted as different C++ class inherited from a root class Element(see Fig. 1). The Element class is called abstract class and defines the interface for inherited classes. e.g. track, matrix are the common routines that every element should implement. This makes the concept more clear when expanding, e.g. track, to a set of DA vector or DA map from simply particle coordinate.

The class hierarchy is shown in Fig. 1. Two possibly different classes are Bpm and Kicker, which are inherited from Drift. In this way they can have finite length. Dipole inherits from Multipole to use the symplectic integrator when the multipole components exist (errors or real field).

The physics model for elements are matrix concatenation and symplectic integrator. The former follows linear and second order matrix theory as other codes did. The symplectic integrator adapts Yoshida scheme with a default 4th order [2]. Fig. 2 illustrates the drift-kick patterns for this 4th order integrator. An example phase plot of NSLS-II lattice is shown in Fig. 3, where most of the magnets are using symplectic integrator and damping wigglers are using

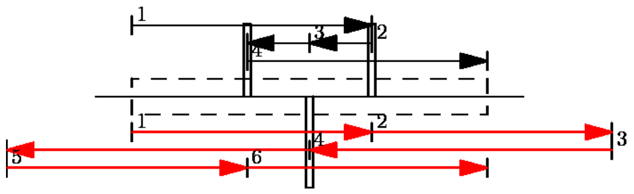

Figure 2: Drift-kick illustration of Yoshida scheme 4th order symplectic integrator. The top(black) arrows are reduced form of particle flow path, the bottom(red) arrows are equivalent full flow path.

kickmap. This plot is not meant to prove the symplecticity but to show good stability in both planes for 2048 turns of tracking.

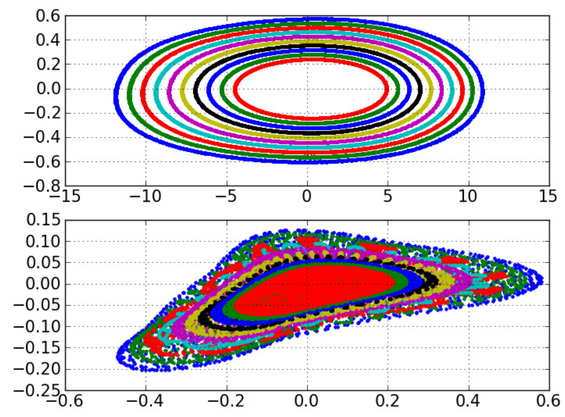

Figure 3: Phase plot of an example NSLS-II lattice (in the unit of mm and $mrad$). Quadrupoles, sextupoles and damping wigglers are included. The top and bottom plots are for x–p_x and y–p_y plane.

Slicing is allowed when the element is long or a higher orbit precision is required. The elements modelled by kickmap are also implemented [4]. Radiation in dipoles is implemented using classical formula. The RF effects are also included.

Errors including misalignment are implemented as an external class. This makes various kind of error distributions available and in a central way.

TOOLKIT

TESLA is developed first as a library for single particle dynamics. It focuses on fast and defailted particle trackings. For simulations where intensive tracking required, TESLA acts as an low level engine. The parallelization and physics optimization are on top of this tracking engine. Two examples are parallel FMA (frequency map analysis) and dynamic aperture optimization. The author believed that the parallization above the level of single particle tracking, or above the TESLA library could be more flexible and efficient, especially when no interaction between particles are concerned most of the time.

There are NAFF (numerical analysis of fundamental frequency) routine in TESLA and two classes FrequencyAnalyzer and FrequencyMap are implemented for frequency analysis based on turn by turn orbit data. A parallel FMA toolkit is developed with these routines. It includes tracking, analysis and visulization. Tracking and frequency analysis part are done in TESLA library. On a MPI based cluster, particles are distributed into different nodes, where serial tracking and frequency analysis are done. The results are merged to one HDF5 file for postprocessing with Python scripts. Since the output data format is standard and accepted by commercial tools like Matlab and Mathematica, the postprocessing tools have more choices.

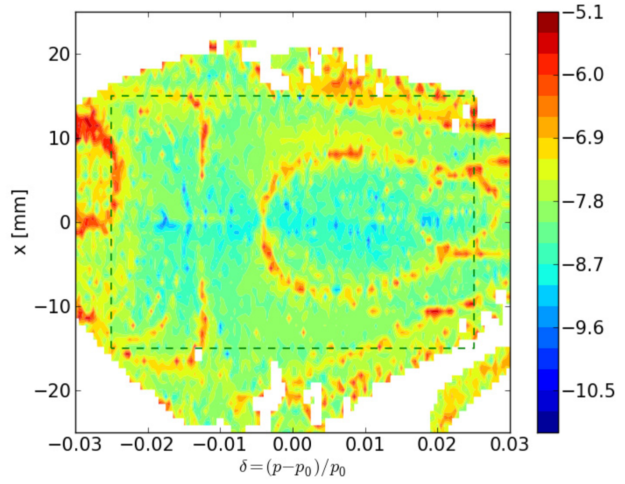

Figure 4: An example FMA on x–δ plane. 14460 particles on a regular grid are tracked on 80 CPUs.

An example of FMA is shown in Fig. 4, where 14460 particles on regular x–δ grid are tracked on 80 CPUs for 1024 turns. The NSLS-II lattice which has over 3000 elements in this case and about 800 meter circumference are used as an example. It took about 5 minutes to generate this plot.

The DA optimization is another parallel code using TESLA library as tracking engine. It uses multi-objective evolution algorithm [5] and runs on MPI based cluster. This optimization uses DA as objective functions by direct particle tracking, both on-momentum and off-momentum DA. Given the objective function, either the average of off-momentum DA or the stable area in x–δ plane, the optimizer analyze the results and generates a new set of input lattice, then call TESLA for a new tracking of DA. Iteratively a final set of solutions are obtained.

In TESLA, the DA searching is done along radial lines in x–y plane and also regular grid for x–δ (usually with grid size less than 0.1 mm for x and 0.1% for δ in our case). When searching along the radial lines, about 10–20 particles initially distributed on that line are tracked. From inner to the outer particles along the radial line, the bounary of live-dead particles is searched. The space between the particles is the precision of DA searching. Then 10–20 particles distributed across the boundary but in a smaller range than before are tracked. This procedure is repeated until the precision is high enough.

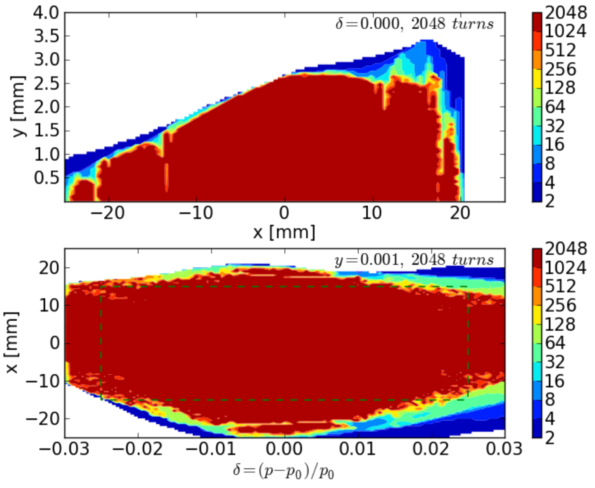

Figure 5: Survival turns of particles at x–y plane and x–δ plane. 14460 particles in total are tracked on a parallel cluster.

The survival plot of one optimization is shown in Fig. 5. Particles in two planes, $(x, y, \delta = 0)$ and $(x, y = 1\mu m, \delta)$, are tracked. The maximum survival turns are plotted on x–y and x–δ plane.

ACKNOWLEDGEMENTS

I thank the support from S.Y. Lee, D. Robin, C. Steier and S. Krinsky for their encouragement, R. Popescu for technique support on our cluster, F. Lin for being first using these tools and providing feedbacks, other collegues for their interests and collabrations.

REFERENCES

[1] L. Yang, Y. Li, W. Guo, S. Krinsky, PAC'11, March 2011, New York.

[2] Haruo Yoshida, "Construction of Higher Order Symplectic Integrators", Physics Letters A 150, 262–268, 1990.

[3] Etienne Forest, "Geometric integration for particle accelerators", Journal of Physics A: Mathematical and General 39, 5321–5377, 2006.

[4] P. Elleaume, EPAC'92, p661, 1992.

[5] K. Deb, "Multi-Objective Optimization using Evolutionary Algorithms", Wiley 2004.