Proc. Workshop on Accelerator Orbit and Particle Tracking Programs, BNL, May 3-6, 1982

BNL 31370

CONF-820574--4

LILA: The Long Island Lattice Analogue*

J. Niederer and B. Morris
Brookhaven National Laboratory

DE82 017273

Introduction

LILA is a BNL adventure to create a particle orbit and tracking program ensemble for large storage ring accelerator design and also controls operation. The accelerator physics parts are based largely on the PATRICIA program of H. Weidemann, as enhanced by S. Kheifets in a later version with multipole effects. We have emphasized the data base aspects of the tracking problem, as modern storage rings contain thousands of distinct lattice items, each with perhaps up to fifty parameters of its own. We have also introduced the general and flexible program structures long familiar to high energy physics event analysis, by which an event is reconstructed in steps from points into lines, projections into tracks, tracks to vertices and the like. Thus, LILA is a modern amalgam of the original PATRICIA, a relational data base and memory management mechanism, and a number of enhancements for treating nonlinear forces.

The Basics of Orbit Tracking

The major part of PATRICIA and similar programs tends to be input and data structure related, and miscellaneous overheads. The coding is only about 10-20% orbit physics related. Thus in a programming sense, the physics parts are a perturbation superposed upon a data base problem. Furthermore, the variety of the physics is rather small, as it describes the life cycle of a particle in a ring, which is a bit of a bore. The particle drifts, or its path may be bent, or it may be kicked, or otherwise tormented. The particle traverses a very limited and dreary landscape: magnets, deflectors, etc. Its social life is similarly dull. It moves in a small pipe, it endures beam-beam or random gas encounters, or it may be nudged about by synchrotron oscillations and magnet irregularity influences. This repertoire of activities contains only about twenty items, all of which have the form of a series of drifts and forces acting over local sections of orbit. The modelling problem is clearly dominated by the thousands of elements of the lattice at which these several processes occur, rather than by the handful of processes.

These elementary deflection processes are tabulated in LILA by exercises that compute closed orbits, adjust sextupoles for desired chromaticity, measure effective aperture, track particles for many turns, display phase space plots of the tracking, and Fourier analyze the plots for harmonics and instabilities. The papers of S. Kheifets and F. Dell of this Workshop describe these PATRICIA calculations in detail. LILA reorganizes these procedures to be more flexible and self-contained, and couples them to a comprehensive data base structure. This flexibility, for example, enables Fourier analysis to be done repeatedly during a simulated storage cycle to analyze the buildup of particle harmonics in time.

*Work performed under the auspices of the U.S. Department of Energy.

Tracking and the Lattice Data Base

LILA is designed to ease the task of assembling elaborate beam element lattices from a set of simpler magnet and force components. This is accomplished in part by an input language of familiar accelerator terms, and by a modular data and program structure to be outlined later in this paper. The lattice description within the program may progress through several stages of increasing complexity and storage size. The program reexpresses this data in forms which yield maximum execution speed. In the simplest stage, which is adequate for central orbits of on-energy particles, only a few standard linear elements need to be involved. Input data are converted to appropriate transfer matrices for these. As most analyses will also consider the effects of small energy or magnet strength fluctuations upon the beam, the first and second derivatives of these matrices are also computed during the setup step. In an intermediate stage, orbits are subjected to nonlinear forces, and a representative multipole set may be interspersed among the basic linear elements. Typically this will involve about ten varieties of pole set, placed at a few hundred lattice points. For this stage, LILA combines sequences of adjacent linear elements lying between nonlinear ones into equivalent matrices for speeding up tracking and orbit loops. Use of pointers and linked lists enables the program to reuse the same basic matrices and pole sets, minimizing storage and setup complications. More detailed tracking usually involves a lattice with realistically misplaced magnets, fields with a distribution of errors about assumed tolerances, and time dependent variations of particle energy and perhaps fields. For these LILA uses the constructs of the basic beam elements to assemble a modular representation where every element acquires a unique matrix, or pole set, with the necessary fluctuations expressed through randomization or other criteria. This final stage requires a large increase in memory. As the number of distinct elements seen by the program is now increased by a hundred fold, updating for energy changes becomes lengthy when done exactly. Updates are therefore handled by small change approximations using the derivatives of the original element matrices. The various stages can be written or recalled as disc snapshots using commands in the input stream. The program contains checks to insure that the stage of the data matches the operations needed. This hierarchy also helps with fitting the program into smaller computers.

Modular Structures

Both data and coded procedures are organized into self-contained modular units in LILA. This emphasis on structure leads to a program that is very flexible with respect to inputs and program changes, and one which tends to go to completion, leaving a trace of diagnostics, regardless of how erratic its inputs may be. Tracking is a major procedure that illustrates the structural principles of the program. Tracking can be viewed as a straightforward tabulation of a series of effects by forces and drifts on the orbit of a particle. The details of the forces are contained in entities called data modules. A data module resembles a packet, with a header (envelope) to tell what it contains and what is to be done with it, and a data or table part as contents. LILA uses a pattern mechanism to describe the order in which the drifts and forces are encountered. A section of code, called a process, corresponds to each of the elementary force or drift variants. The tracking procedure thus employs a set of processes in code, each of which operates upon a unique type of data module, in an order specified by a pattern. Tracking over an orbit consists of reading a pattern, computing an

indicated deflection or displacement using the data of the module, and then returning to fetch the next item of the pattern, and repeating. There is a marked analogy to the way proteins are built from patterns coded into DNA. The main procedures of LILA all have the Read, Branch, Compute, Repeat form. This structure is aided by standard service routines which relate names of lattice elements to places in memory, and create data modules, patterns and the like.

Using LILA

The inputs to LILA are based upon the same organization of procedures, patterns, and processes keyed to data modules. They provide data for the menu of basic forces, and for the patterns. Commands initiate procedures once the data corresponding to a lattice have been submitted. These features provide an extremely powerful and flexible mechanism for exploring accelerator design options.

All data for a beam element is contained in a single data module. The first appearance of data for a named element, such as

$$B1 = BEND(L = 1.,S = 2., \ldots)$$

creates a module for that element. The data may be changed, or expanded during the course of a run, but the module itself continues as the sole container of information about B1. In this example, B1 is the name of the lattice element, BEND is a keyword that refers to the general type of element, and the brackets enclose a list of attributes, length (L), strength (S), etc. B1 is a base element which describes an elementary force; it cannot be subdivided into simpler components. A data module may be rewritten during the various stages of the more complicated procedures, but its purpose, name, and linkage remain the same.

A second type of input includes information about the patterns which describe the order of elements in the lattice. A pattern is simply a list of names, for which the order is its most important property. A pattern may include names of other patterns, and many levels of nesting can be handled. The pattern must eventually reduce to a series of names of base elements containing attributes. An ISABELLE cell, for example, might be described as

$$C1 = CELL(QF,D1,B1,D2,B2,D3,B3,D1,QD,D1,B3, \ldots)$$

where the names are those of magnets and drifts. A group of pattern modules will usually be expanded into a map consisting entirely of base elements, by a routine that works very much like the loader of a computer operating system. It essentially links the names in the various pattern modules with the locations of base data modules in a memory pool.

Commands to LILA are of a pushbutton nature, such as computing an orbit, adjusting sextupoles, or tracking. Commands instruct the program to operate on the data base which results from lattice inputs. A command normally names a region of the lattice, and may also include parameters and references to lists which help describe the type of action intended. These features permit tuning sextupoles to differing criteria over different sections of the ring, for example. A command may require that modules of the data base be modified to a new stage. It may also require a pattern map for the region named. A command can operate over the full lattice, any part of it, or even mix-match combinations of tentative design options.

A final category of inputs is a group of quantities which help to parameterize commands, such as number of turns for tracking, program switches, and starting values of injected particle tracks.

```
SWITCH(22) = 12, 13,,15(sets switches 22, 23, and 25)
```

A simple minded characterization of these input data line types is "NOUN" for a pattern or other name list, "ADJECTIVE" for basic data modules, (attribute) "VERB" for command, and "ADVERB" for parameter. This view leads to an impressively clean separation of function and modularity.

The LILA Language

The LILA input language is free field and user friendly. It is very similar to that of the MAD package described by C. Iselin at this Workshop. The vocabulary is familiar: MAGNETs are joined into a CELL, STRING, GROUP or INSERTION, CELLs into a SECTOR, SECTORs into a PERIOD, and PERIODs into a RING. The idea of a complicated lattice assembled from repeated simple structures is fundamental; the names can be changed to taste. A MAGNET can be represented as a combination of a DRIFT, an EDGE lens, a BEND, a QUARDUPOLE, a SEXTUPOLE, and a MULTIPOLE as basic elements. Actually, LILA should accept most MAD inputs with a few statements added at the head of the input stream to reconcile the slightly differing vocabularies. Given the unusually thorough specification for MAD, it seems prudent for us to include all of the force options treated by MAD into future versions of LILA, and also to minimize differences in input descriptions so that common documentation can be used for the two sets of programs.

An example of the statements needed to randomize multipole sets in magnets illustrates the power of this approach to lattice design.

Multipoles MP1 and MP2 are to be randomized at their various occurrences within region THIRDA of the complete ring:

```
BLUE = RING(THIRDA, THIRDB, ...)
```

Base element multipoles occur within the pattern:

```
MP1 = MULTIPOLE (.....)
```

Pole values are to be varied within a set of tolerances:

```
TLIST = TOLTAB(1., 2., 3., ...)
```

Tolerance lists are matched to the magnets involved:

```
CODEA = TOLERANCE(TLIST, MP1, MP2, ...)
```

A command initiates the procedure for randomizing multipoles:

```
RANDOM(CODEA, THIRDA)
```

Similar routines carry out a second example, the updating of the current in a group of magnets connected to a common power supply.

```
BH1 = BEND(L = 3., S = 5., ...)     (Basic Bend)
P1  = SUPPLY(CH = 0.01, ...)        (1% increase in current)
PLIST = POWER(P1, BH1, BH2, ...)
RIPPLE(PLIST, ASECTOR)
```

Magnets of kind BH1, BH2, etc. connected to supply P1 have strength increased by 1% throughout the region ASECTOR.

Input Handlers

Input handlers resemble those of conventional compilers. While much of this handler is overbuilt for a tracking application, it is intended to be the common interface between the accelerator operator and a number of system data bases, such as plumbing and power, as well as lattice simulation. An input line is assembled into groups of related characters called tokens. Types of tokens include names, numbers, strings of characters, relational (=, <, >, $\leq$, $\geq$, :, = -) and mathematical operators, ignores (i.e. blanks), punctuation, and characters whose role can be specially assigned. A name begins with a letter or special character and ends with punctuation, an operator, or an otherwise defined break character. Each symbol of the character set has an index, which may be changed as desired to redefine the function of the symbol. Thus while we choose to ignore blanks, a blank may be redefined to be a part of a name or as punctuation. Tokens and their function tags are grouped into a temporary data module.

A parsing procedure sorts tokens of an input line into keywords, names, operators, and lists. Keywords are used to distinguish the kind of line: lists of names for patterns, lists of attributes for basic lattice elements, and program commands and parameters. The name of a keyword may be redefined during the course of a run. The name of an element already linked to a keyword may be used in place of a keyword (equivalence). A keyword has an associated template for building data modules from a line of tokens. A template may include names of attributes, range limits for checking valid data values, a scale factor, and password protection if relevant. The parsing step recognizes new lines, equivalences, continuations, and changes and replacements to existing lines (modules). It relates names to the memory storage and pooling scheme, using a directory, and linked lists of modules. Diagnostics are provided. The forgiving style tries to get through all of the input data despite errors.

Parsed results are passed to storage routines that assign token lines to pool memory or a specific common. These routines act like the FORTRAN READ and DATA statements. Storage indices are checked against limits set in common blocks or data modules. Data values are checked against limits if preset in templates of keyword or common modules. Finally acceptable new and continuation modules are linked into the directory and memory pooling scheme. Errors result in diagnostics.

A keyword set is predefined in BLOCK DATA statements, and additional keywords can be entered in the input stream. The keyword mechanims relates user names to the storage scheme, marks type of line, and contains the template mechanism. Any number of keywords, which are expressed as separate keyword modules, may be used, subject to available memory. Similarly the attribute or data part of a modules, whose structure is defined by means of the template in the keyword, may be of any length, and may include arrays. Its items may be names, strings, or numbers. This mechanism in effect maps the free form of the inputs into the necessarily position oriented data structures needed by conventional programs. Definitions and defaults for program parameters are also preset in BLOCK DATA.

Controls and Program Formats

Controls requirements have strongly influenced the construction of LILA. The code produced is exceptionally fast, robust, modular, flexible, and general. All of these factors lead to important advantages for storage ring design applications as well. They do lead to more complication, and take more code to do a given job than a conventional single author tracking program. Yet modern programming techniques and tools have readily borne this additional burden. The code added for efficiency is insignificant in memory needs when compared with the typical 100 000 word data bases involved. The memory management scheme, essential to flexibility, allows a physicist to expand a lattice of 2000 elements in one representation to a more complicated one of 50 000 by changing a single memory size number in two places in the MAIN program.

LILA is coded in FORTRAN, and is currently meshed with the CDC UPDATE program library mechanism. A migration to a PATCHY form should be attempted if LILA gains acceptance. Conversion from its present 60 bit CDC 7600 form to that appropriate to 32 bit machines has been anticipated somewhat in the coding, but the indexing and pointer mechanisms are more effective on the larger word size machines, and will need attention during translation. The most effective translation scheme appears to be via PATCHY or a similar program library that is portable, and allows for conditional compilations. Thus lines of code needing translation can be written in place in the form needed for a particular computer, and the library sorts them out at compile time.

Status

Coding and most debugging is complete. Timing depends mainly upon the number of multipole locations and number of terms used. An extreme case of 540 multipole sets of 15 poles in a ring yields timings of about 25 milliseconds per turn. Improvement factors of about two seem possible from machine language coding the multipole algorithm, and more from more reasonable lattice models, so numbers in the range of 5-10 ms seem likely for ISABELLE-size configurations. A few of the more complicated energy variation procedures remain for debugging.

Acknowledgment

LILA was prepared by a programming team approach. T. Clifford prodded us to more user friendly approaches, and helped test some of the front end concepts by applying them to an enhanced PERT-like management monitoring system. G. Smith helped with the nonlinear force presentations. P. Taibbi handled the data entry and program library chores with quite phenomenal accuracy and diligence. Our debt to our SLAC colleagues for the originals has been noted, and S. Kheifets has been most generous in helping adapt PATRICIA for BNL use.