SLAC–PUB–5302
SSCL–305
July 1990
(A)

SSCTRK: A PARTICLE TRACKING CODE FOR THE SSC*

D. RITSON
Stanford Linear Accelerator Center,
Stanford University, Stanford, CA 94309, USA
and SSCL,† Dallas, TX 75237–3946, USA

1. Introduction

Critical to the SSC performance is an understanding of the factors controlling the dynamic aperture of the main storage ring. The most critical problems are expected to arise at injection time prior to ramp-up to full energy. At injection the beam will coast for up to one hour. This long time is required to fill sequentially the two rings. It is known that the effective aperture of a proton storage will progressively degrade with storage time.

The SSC storage time at injection is $\sim 1/2$ hr or $\sim 7 \times 10^6$ turns around the machine. The SSC aperture will be 4–5 cm, approximately half that of the Tevatron. The multipole field errors, both random and systematic are thus substantially greater than those of the Tevatron. The tune of the SSC will be about five times that of the Tevatron and the sensitivity to systematic tune shift will then be worse by a factor of 5. Therefore it cannot be argued that success of the Tevatron guarantees success for the SSC.

While many indirect methods are available to evaluate dynamic aperture there appears at this time to be no reliable substitute to tracking particles through realistic machine lattices for a number of turns determined by the storage times. Machine lattices are generated by "Monte Carlo" techniques from the expected rms fabrication and survey errors. Any given generated machine can potentially be a lucky or unlucky fluctuation from the average. Therefore simulation to serve as a predictor of future performance must be done for an ensemble of generated machines. Further,

* Work supported by Department of Energy contract DE–AC03–76SF00515.
† Operated by Universities Research Association, Incorporated, for the US Department of Energy contract DE–AC02–89ER40486.

Invited talk at the Workshop on Nonlinear Problems in Future Particle Accelerators, Capri, Italy, April 19–25, 1990.

several amplitudes and momenta are necessary to predict machine performance. Thus to make Monte Carlo type simulations for the SSC requires very considerable computer resources. Hitherto, it has been assumed that this was not feasible, and alternative indirect methods have been proposed or tried to answer the problem.

We reexamined the feasibility of using direct computation. Previous codes have represented lattices by a succession of thin elements separated by bend-drifts. With "kick drift" configurations, tracking time is linear in the multipole order included, and the code is symplectic. Modern vector processors simultaneously handle a large number of cases in parallel. Combining the efficiencies of kick drift tracking with vector processing, in fact, makes realistic Monte Carlo simulation entirely feasible.

SSCTRK uses the above features. It is structured to have a very friendly interface, a very wide latitude of choice for cases to be run in parallel, and, by using pure FORTRAN 77, to interchangeably run on a wide variety of computers. We describe below the program structure operational checks and results achieved.

2. Description of Program

The code splits naturally into two parts. A Monte Carlo generator to provide one (or more) machine lattices and tracking code to examine the long term stability of particles launched into the generated machine lattice(s).

The Monte Carlo generator sets up a nominal thin lens FODO lattice chromatically corrected by sextupoles to provide the desired numbers of elements, tune values and radius. Files of orbit errors and multipole errors are then generated from specified rms and systematic field and orbit errors. These are combined together (including multipoles caused by orbit error feed down) into effective multipoles centered around the equilibrium orbit. The program then makes user specified corrections, employing multipole families, to provide a final file that should correspond to that to be attained operationally during the commissioning of the accelerator.

The generator is fast, and it provides lattices with the expenditure of seconds or less of cpu time. The generator can be used to sequentially generate specified lattices or cases and to accumulate them into an array indexed by the case number that is to be used for parallel (vector) processing.

Kick drift tracking then tracks the particles over the required number of turns through the set of lattices provided by the generator. The tracking simulation can include synchronous motion, ramping in energy, and power supply ripple for the focusing quadrupoles relative to the bend magnets.

At each turn, quantities such as averages and rms deviations of the Courant and Snyder invariants are accumulated. After n turns (a specified number), these quantities are dumped into an output minifile. The minifile data is then put through a postprocessor to provide graphics or other outputs. The final orbit data are also dumped to permit resumption of running if so desired.

3. Physics Inputs

We represent the lattice by a succession of thin elements. This naturally provides sympleticity and, in view of the fact that the focusing quadrupoles only occupy 5% of the arcs of the lattice, it is a reasonable approximation. At present the IRs are represented as unit transformations. This is a reasonable approximation under injection conditions, but it will require modification to simulate the low beta interaction optics and future code will include IP effects. The RF acceleration is modeled by a single "thin" RF cavity located in a dispersion-free location. The drift-bend transfer matrices are approximated with a standard six-by-six linear transfer matrix acting on the vertical, the horizontal coordinates, and the longitudinal coordinate and momentum.

A SSC half-cell contains a quadrupole, a chromatic sextupole corrector and five bend magnets. All of these components may be incorrectly placed or contain systematic or random multipole errors.

To economize cpu time, we use the Simpson Rule procedures of Neuffer and of Forest and Peterson to concatenate lattice errors into the ends and cell centers. Essentially, this procedure redistributes the errors in the five dipoles in a half-cell into three positions. The procedure takes a series of multipoles of order n s_{in} at points t_i, where t_i is the displacement from cell center and $2t_0$ is the half-cell length, and to replace them with three values S_{1n}, S_{2n}, S_{3n} lumped at the beginnings, midpoints and ends of the half-cells.

The S_{in}, $i = 1, 2, 3$ are determined for each half-cell by the three conditions

$$S_{1n} + S_{2n} + S_{3n} = \sum s_{in} \tag{1}$$

$$-t_0.S_{1n} + t_0.S_{3n} = \sum t_i.s_{in} \tag{2}$$

$$t_0^2.S_{1n} + t_0^2.S_{3n} = \sum t_i^2.s_{in} \quad . \tag{3}$$

This procedure takes account of the variations of beta and eta values over the cells and cancels the deviations of the these parameters to second order for random errors and to third order for systematic errors. The efficacy of these procedures for random errors is detailed in the *1988 Rome Particle Accelerator Conference,* by Peterson and Forest, page 827. The results are good to 5–10%. We have compared this to a noncorrelated concatenation that assigns random errors to the beginning, center and end of the cells in the ratio of $1 : \sqrt{2} : 1$. Both schemes give results equivalent on the 5% level. Etienne Forest analytically finds both schemes to be nearly equivalent. For systematic errors in the SSC, we have compared analytic first-order predictions of tune shifts in the distributed case with errors lumped according to the Neuffer procedure for both octupole and decupole systematics, and we have found agreement to a few percent. We therefore assign an upper bound to the error

on the dynamic aperture of $< 10\%$ smaller than that associated with predicting tolerances to be achieved in practice.

4. Code Checks

As is standard, the code is double precision. Detailed below are checks or procedures used to ensure integrity of the code.

4.1 Overall Organization

A potential source of problems is the unavoidable complexity of code written to model a real machine with all its underlying warts and blemishes. To minimize the complexity SSCTRK was designed as a "linear" code. The main program steps sequentially through subroutine calls that take the specifications and translate theme into a design lattice structure. The program assembles this structure into a "real" object flawed by systematic and random errors. It applies corrections to this object and finally checks this object by tracking to find the maximum usable aperture. Each of these steps is done using main subroutine calls. The subroutines are simple, perform general tasks and contain an absolute minimum of internal branches or special purpose assumptions. Any operation or piece of code occurs only once in the program. The only subsidiary subroutines or functions called by these main subroutines are mathematical functions or subroutines such as a random number generator.

Tricks or shortcuts in coding, equivalence statements, and backward "goto" statements are deliberately avoided, and at all stages the code is designed to be explicit and to conform fully to FORTRAN 77 protocol. To further ensure full transportability, no external system specific routines are invoked.

The complete SSCTRK code (excluding comments) has less than one thousand lines of FORTRAN. This amount of code is easy to debug, edit and transfer between machines.

Identical running on IBM, CRAY, SX2 and SUN computers is already a strong check on the generality of the code.

4.2 Standard Code Checks

We have made use of many of the standard techniques currently available to ensure error free coding. These include:

(1) The "IMPLICIT NONE" convention. This convention forces the programmer to explicitly assign integer, real etc. types to all variables. The compiler automatically flags any variable that is not explicitly typed (spelling errors for instance) or whose type conflicts with a previously typed variable of the same name.

(2) Load options to check for improper initialization of variables. These options per-

mit loading of memory banks with zeros or alternately with "overflow" numbers. Incorrectly initialized variables are flagged by the program running differently with different load options or abending with error messages.

(3) Quadruple precision compilation. The code can be compiled with a quadruple precision option and compared to the standard double precision results to ensure that round off precision is indeed sufficient.

(4) The debug option. All sections of the code were stepped and looped through under the "debug" option to ensure that operations were indeed proceeding as expected.

4.3 Redundancy checks

Central to the integrity of the code is the central tracking routine. Kick code can be written very simply in complex notation. As the tracking routine dominates the cpu usage, there is a high premium on optimizing its performance. In practice, therefore, this routine utilizes real quantities and is arranged to minimize the multiplication and additions required for evaluation. For operation with the CRAY 2, where the FORTRAN compiler was less than optimal, an assembly coded routine was used (this, in fact, gained a factor of 2 in speed). These substantially more complicated and opaque routines were checked against the results from a slower but transparent tracking routine implemented in complex notation, and they gave identical results within the double precision roundoff errors.

The fact that the code runs "identically" on a variety of different computers has already been alluded to.

Twiss parameters and tunes are calculated directly from the lattice characteristics. They are independently evaluated from the orbits obtained in tracking. Agreement of these Twiss parameters and tunes obtained from orbit data provide an important redundancy check on the code.

Chromatic tune shifts resulting from systematic errors are obtainable analytically to first order, and it has been checked that the code produces correct results for these terms.

Finally, intensive useage by a number of people over an extended period is an excellent tool for uncovering potential flaws in the underlying structures.

4.4 Reverse tracking

An interesting code check is provided by taking a particle through a number of turns, reversing its direction and returning it through the same number of turns to ensure that it returns to the identical starting amplitude. This test did indeed return the particle to the identical starting pointing within the precision expected from the previously determined roundoff errors.

5. Long-Term Running

A standard worry in long-term simulation is whether artificial damping or beam blowup is caused by nonphysical artifacts of the code.

In principle the code is fully symplectic. There are, however, a number of approximations that have been used.

Fully accurate tracking involves a "finite momentum" correction. This correction is of the order of v_s/v where v_s is the longitudinal component of velocity, and v, the magnitude of the velocity. This is a very small correction indeed but it results in a loss by a factor of 2–3 in tracking speed. Most codes do not include this correction, and we have also omitted it for most running. We have, however, checked with a million turn thirty-two case run that inclusion of this factor did not effect the dynamic aperture determination.

We have made long runs with linear lattices that show no growth or change in the Courant Snyder invariants. This demonstrates that roundoff errors that cause departures from symplecticity are sufficiently small in double precision simulations as to not cause problems. It also provides a check that coding artifacts do not appear to be producing spurious problems.

6. Computation Requirements

6.1 Memory Usage

Considerable care was taken to minimize the memory requirements of the program. Typically, memory requirements are dominated by the array used to describe the lumped multipole content around the machine plus, of course, a standard overhead of a few megabytes. An initial parameter statement sets the size of this multipole array.

For a lattice of eight hundred quadrupoles there will be, including midcell locations, sixteen hundred sets of multipoles. If multipoles up to sixth order are included for each lumped element the required storage is twelve REAL*8 numbers per location or a total of $\sim 154,000$ bytes per case. The program therefore is relatively modest in its demands for memory, even when used in a vector mode.

6.2 Speed

SSCTRK has made extensive SSC simulation studies. To economize super computer use we have assembly coded the actual short tracking routine. We give below in Table 1 typical cpu tracking times required for a model machine with eight hundred half-cells, lumping multipole errors into the center and ends of the half-cells, and multipole order up to six included.

Current technologies are rapidly evolving, and it can be expected that the above performance figures will improve by substantial factors. Usually most of the required information can be obtained running between 32 to 64 cases for a 10^5 turns each

Table 1: Benchmark computer speeds for SSCTRK.

Scalar processors	
SUN SPARC workstations	15 particle turns/sec
IBM 3080	80 particle turns/sec
IBM 6000 workstation	100 particle turns/sec
Vector processors	
CRAY 2	1,000 particle turns/sec
CRAY YMP	2,000 particle turns/sec
SX2	3,000 particle turns/sec

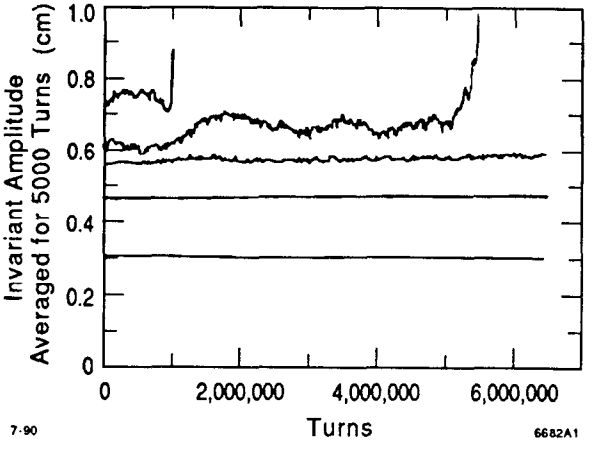

Figure 1: The evolution of the root of Courant and Schnyder invariant for individual particles averaged for 5,000 turns plotted against turn number out to six million turns.

requiring $\sim$ 1 hr of supercomputer time. At complete evaluation of evaluations of dynamic aperture using 32 parallel cases and 5×10^6 turns per case can be run in about one day of cpu time.

7. Typical Results

Figure 1 shows the behavior of tracks launched at different amplitudes. Plotted is the root of the Courant and Schnyder invariant averaged over 5,000 turns versus turn number. At small amplitudes well within the dynamic aperture the amplitude is unchanging. At larger amplitudes the motion begins to show chaotic irregularities and beyond the dynamic aperture the particle motion is chaotic and loss occurs after 150,000 turns. An informative display of results is provided by a "loss-plot." Figure 2 shows the history of a group of particles at four closely spaced amplitudes injected into an ensemble of eight Monte Carlo generated lattices for 4 cm aperture machines. The particles at amplitudes well above the dynamic aperture. After relatively few turns the particles are lost from the machine aperture. If a particle is lost it is restarted at a smaller initial amplitude. This process continues until it

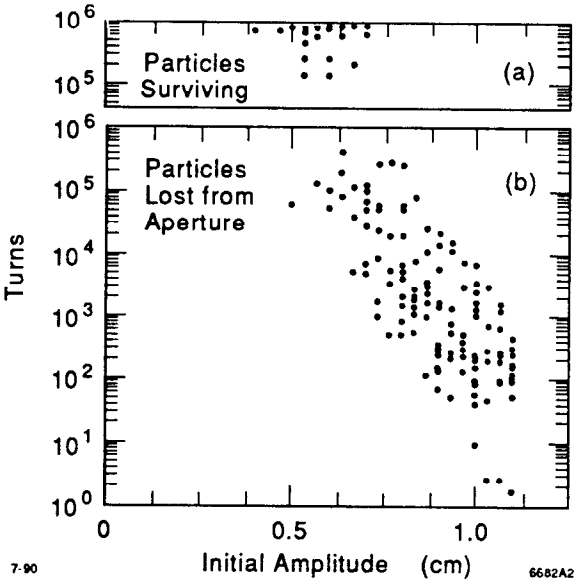

Figure 2: Particles initially launched at 4 amplitudes close to 1.1 cms into eight Monte-Carlo generated lattice structures. After a particle leaves the aperture it is restarted at 1 mm in. Plotted is the logarithm of the number of turns prior to loss versus aperture. (a) Surviving particles and (b) lost particles.

is injected at an amplitude just within the long-term dynamic amplitude. Plotted in Fig. 2(a) is the logarithm of the number of turns made by the surviving particles versus aperture; and in Fig. 2(b), the logarithm of the number of turns before loss versus aperture. The envelope of the curve in Fig. 2(b) effectively delineates the edge of the dynamic aperture as a function of turns. Typically, an increase in the number of turns required, by a factor of 30, will decrease the dynamic aperture by 1 mm.

8. Conclusions and Future Directions

On the basis of the above SSCTRK appears to provide a sound reliable basis to predict the SSC performance. This has been achieved via straightforward computer tracking. Computer capabilities are increasing at a remarkable speed and over the next few years one can predict with confidence that tracking simulations for long storage times will become increasingly simple.

For the future, detailed IP configurations and operational simulation capability will be added to the code.