BEAM DYNAMICS SIMULATION CODE FOR THE OAK RIDGE SPALLATION NEUTRON SOURCE RING[*]

J. Beebe-Wang, M. Blaskiewicz, A.U. Luccio, BNL, Upton, NY 11973, USA
J. Galambos, J. Holmes, D. Olsen, ORNL, Oak Ridge, TN 37831, USA

Abstract

We describe the principles of a computer code to simulate in 6 phase space dimensions the dynamics of a high intensity particle beam. The immediate purpose is to study the 1 GeV proton accumulator ring for the Spallation Neutron Source to be built at Oak Ridge.

INTRODUCTION

There is a joint effort between National Laboratories in the USA to study and build at Oak Ridge an accelerator based pulsed neutron spallation source, SNS[1]. The accelerator system of the facility consists of a 1 GeV proton linac followed by an accumulator ring. The design beam power is 2 MW, corresponding to a very intense circulating current in the accumulator (2.10^{14} ppt). Uncontrolled beam losses should be limited to 10^{-4}.

Ring beam dynamics is being studied by two teams, at Oak Ridge and Brookhaven. For some time we used ACCSIM[2], then we decided to write a new code, SAMBA (Sensible Analysis Model for Beams in Accelerators). SAMBA is written in C++, for two main reasons: (i) to achieve a more natural parallel code development (C++ is the standard in Industry, where teams of programmers work on the same code), (ii) use the supervision of a SuperCode allowing the coexistence of compiled and interpreted modules.

6-dimension simulation codes, including space charge, are essential tools for the study of high intensity beams. The real problem is how to implement the known physical formalism to obtain credible results in reasonably short computer run times.

The accelerator descriptor is MAD[3], a well developed and maintained code able to produce first and second order orbit transport maps and to handle lattice errors.

PIC MODEL

The beam is represented by random macro particles (typically 10^5) in 6 dimensions. For orbit transport, the lattice is subdivided in a few sections. First order **R** 6x6 matrices and second order **T** maps for each section are produced by MAD. For distorted closed orbit due to applied orbit bumps, typically at injection, or to lattice errors, the matrices are accordingly displaced.

Transverse Motion

In the transverse phase space, space charge kicks are applied to the macros. Betatron tune distribution in the beam is calculated. With y the vertical, x the radial and s the longitudinal coordinate of a macro, transverse motion is in (x, p_x, y p_y). To calculate transverse space charge kicks, at the end of each machine section the beam charge distribution ρ is obtained by binning and counting in a regular mesh. From ρ, the space charge force is calculated

$$\vec{F}(\mathbf{P}) = \frac{\mu_0 e c^2 N e}{4\pi\gamma^2}\vec{g}, \quad \vec{g} = \int \frac{\rho(\mathbf{Q})}{r^3}\vec{r}\,d\mathbf{Q} \qquad (1)$$

on an individual macro at **P** by action of an element of a continuum at **Q** at a distance

$$r^2 = r_\perp^2 + \gamma^2 r_\parallel^2 \qquad (2)$$

with a relativistic longitudinal dilation factor γ^2. The longitudinal part of the integration is almost a constant unless we are close to either end of the bunch. Factoring out the longitudinal charge density as

$$\rho(\mathbf{Q}) = \rho_\parallel \rho_\perp \qquad (3)$$

the integration loses one dimension and we treat the beam more as a bundle of spaghetti than a cloud of points.

At the end of a machine section of length L a space charge transverse kick is applied to every macro at **P**

$$\delta\mathbf{p}_\perp = \int_L \vec{F}(\mathbf{P})dt \approx \check{F}\frac{L}{\beta c} \qquad (4)$$

The betatron tune distribution in the beam is calculated as the local derivative of the force. It gives indication on the transverse stability limit, but is not directly used.

The calculation of the transverse space charge force is the single most time consuming algorithm. Many strategies to cut this time all derive from the observation that, at least during one turn, the general shape of the charge profile would not change much, leading to the concept of "rubberbanding" of the distribution -a given shape calculated once per turn is stretched or compressed by radial scaling factors from one interface to the next-

Scaling can be derived (i) from a statistical evaluation of the emittance, (ii) by proportionality to the square root of the Twiss beta function, (iii) by using the envelope equation integrated in a Core model (see later), (iv) by fitting the calculated force profile and transferring the coefficients to the next interface.

Transverse space charge forces introduce coupling between the radial and vertical motion. The betatron oscillation differential equations

$$\begin{cases} x'' + K_x^2 x = f_x = \dfrac{F_x(x,y)}{m_0\gamma\beta^2c^2} \\ y'' + K_y^2 y = f_y = \dfrac{F_y(x,y)}{m_0\gamma\beta^2c^2} \end{cases} \qquad (5)$$

[*] Sponsored by the Division of Materials Science, U.S. Department of Energy, contract number DE-AC05-96OR22464

are coupled. The tune spectrum in one mode will contain side bands due to the other mode[4]. Polynomial expansion of the function at the r.h.s of the equation yields, to first order, a tune shift

$$\Delta\nu = -\frac{r_o N}{\beta^2\gamma^3\nu}\frac{g_r}{2r} \tag{6}$$

with r_0 the proton classical radius. For unimodal charge distributions (say, Gaussian), Eq. (6) gives the maximum tune shift in the center of the beam. To higher orders, Eqs. (5) lead to higher order nonlinear tune terms and coupling.

SAMBA statistically calculates the emittance of the beam as the determinant of the covariance matrix

$$\varepsilon^4 = \det\begin{pmatrix} \langle x^2\rangle & \langle xp_x\rangle & \langle xy\rangle & \langle xp_y\rangle \\ \langle p_x x\rangle & \langle p_x^2\rangle & \langle p_x y\rangle & \langle p_x p_y\rangle \\ \langle yx\rangle & \langle yp_x\rangle & \langle y^2\rangle & \langle yp_y\rangle \\ \langle p_y x\rangle & \langle p_y p_x\rangle & \langle p_y y\rangle & \langle p_y p_x\rangle \end{pmatrix} \tag{7}$$

The radial emittance -and similarly the vertical- are defined as

$$\varepsilon_x^2 = \det\begin{pmatrix} \langle x^2\rangle & \langle xp_x\rangle \\ \langle p_x x\rangle & \langle p_x^2\rangle \end{pmatrix} = \langle x^2\rangle\langle p_x^2\rangle - \langle xp_x\rangle^2 \tag{8}$$

It is then $\varepsilon^4 = \varepsilon_x^2\varepsilon_y^2 + \text{coupling terms}$

If there is no coupling, horizontal and radial emittances should be individually monitored during tracking.

Longitudinal Space. Impedance Budget

The linear part of transport is done with the last two rows and columns of **R**. The equations of synchrotron motion deal with the non linear part

$$\begin{cases} \phi_{n+1} = \phi_n + \tau\omega_{RF}\left(\dfrac{\delta p}{p}\right)_{n+1} \\ \delta p_{n+1} = \delta p_n + \dfrac{q}{\beta^2 c}\displaystyle\sum_{h=1}^{H} V_h\big(\sin(h\phi_n) - \sin(h\phi_s)\big) \end{cases} \tag{9}$$

In particular we studied the RF barrier voltages produced with two frequencies[8].

In the longitudinal, other than RF voltage action we have space charge kicks calculated via an impedance budget

$$\delta p_{SC,n} = \frac{q}{\beta^2 c}\sum I_k Z_k \tag{10}$$

where I_k are the Fourier components of the beam current and Z_k the longitudinal impedances. Commonly, the largest is an imaginary impedance for cylindrical perfectly conducting walls, proportional to $\partial I/\partial s$, the longitudinal current gradient in the beam[5]. This gradient can be numerical very noisy if calculated from the distribution and some smoothing is needed. Fourier filtering is naturally provided by the already done FFT of the beam.

Foil. Losses

Injection in the SNS ring is accomplished via multi-turn injection of H^- ions stripped in a foil. The acceptance is painted by moving the equilibrium orbit at the foil with a collapsing magnetic orbit. SAMBA simulates orbit bumps as well as ion conversion and scattering in the foil. Lost particles by interactions with walls or collimators, or not being converted at the foil are accounted for and subtracted from the cycle.

CORE MODEL.

To be trustworthy, the calculated beam behavior must agree with experimental data and with established theories. Thus the validation of the code is an essential exercise. Validation can be done e.g. if we believe that the beam shows a Core behavior plus a Halo. A core calculation based on the solution of the differential equation for the envelope would show the main feature of beam propagation, so, the results of a CORE model [6] should continuously guide and control the PIC model results. PIC itself will best describe the details and the formation of Halo. The integration of CORE and PIC is one of the future development considered for SAMBA.

PROGRAMMING

SAMBA is written in C++, and operates within the SUPERCODE[7] driver shell. It is useful to explain the relationship between user provided "physics" modules and the driver shell. Conventional scientific programs have a prescribed flow of logic originating in a main program. Often there are some program flow choices, governed by appropriate choice of input variable settings. The program execution remains in compiled code until program completion. If the user desires some new flow logic, an edit-compile (debug) programming cycle is required.

On the other-hand, the SUPERCODE is a programmable driver shell, which can execute interpreted script files as well as compiled "physics" modules. Generally, runs are done by reading in a "script" file, that is more than a typical "namelist" stream which simply assign values to variables. In SUPERCODE, calling sequences to compiled code can be customized within a script "input" file. In fact there is no fully compiled set of logic to do a run. Rather, only the "general purpose" physics modules are compiled, and the calling sequences, looping, initial variable settings etc. are done through the shell. Routines can be entered on-the-fly in script files.

Both interpreted and compiled code have their place. Compiled code is in general faster. Compiling everything however leads to code clutter (i.e. one-off numerical experiments, or scans that are never used again, along with control variables). The general philosophy for the separation of interpreted / compiled code is: (i) code that is execution intensive, and/or general purpose is compiled. Examples of compiled code would be particle transport through a matrix multiplication operation. (ii) Code that is problem specific and generally not called often during a run is interpreted. Interpreted code examples are variable initialization, setting up parametric scans, etc.

The procedure for actually constructing "physics modules" to be run with the SUPERCODE shell is described in Ref.[7]. When adding a new module in practice, it's usually sufficient to mimic the method of an existing physics model. The Module Descriptor File is a useful place to get information about a module. It lists all the variables and routines that the Driver shell knows about in that particular Module. This is the place where these quantities should be documented. Variable quantities in these files can be manipulated from the Shell. The routines in the Module descriptor files can also be called from the Shell. In fact, putting together a set of variable initializations, and routine calls in a script file is how a "run" is typically done.

The driver shell can not directly access class members. To access class members from the Shell (manipulate, view, etc.), a Module routine must be written (and compiled) to perform the appropriate manipulation. Examples where this may be done would be a routine to create a macro-particle, or a routine to dump macro-particle information to a stream. These compiled routines could then be called from the Shell, and the actual class member manipulations would be performed when the compiled module is called.

Class Hierarchy for Ring and MacroParticles

For tracking of macro-particles around a ring, two main ideas come to mind: the macro-particles and the Ring elements. These are each represented as classes. The actual user implementation of these classes is done via "modules" that contain the user interaction mechanisms for instantiating objects, performing member function calls, etc. The Macro-Particle class is a simple container class to hold information specific to each macro-particle. A macro-particle object contains a reference to a synchronous- particle object. (There is a separate Sync-Part class to hold the synchronous particle information). The synchronous particle must be instantiated before any macro-particles can be created.

As the macroparticles circulate, operations will be performed on/with them. E.g. they may undergo a matrix transfer, or a space-charge update, or a dump of information. Each operation will be performed at a "node" of the Ring. The Node class represents the common set of features such operators have. As this is an abstract class, no Node objects will be created, but rather this class will be inherited by sub-classes that will have actual objects.

Each node can do some operation with the macro-particles. Two Node member function hooks are provided for this: (i) a node-Calculator, and (ii) an update-Part-At-Node routine. The first is a place where preliminary calculations are done, which depend on more than one macro-particle. The second operates on an individual macro-particle. For example, a "Transfer matrix" node could advance a single macro-particle through a matrix.

Examples of Node sub-classes are: (i) a Transfer-Matrix class for transporting macro-particles around the Ring, (ii) a L-Space-Charge class to give longitudinal space charge kicks, and (iii) a RF-Cavity class to give RF voltage kicks. When a Node sub-class is added, a constructor should be supplied. Also, at least one calculator should be supplied (either a node-Calculator or update-Part-At-Node). With these class members defined and declared, we know both "when" each node should be called as we work our way around the Ring and "what" it should do.

MODULES

A macro-Particle module contains the interfaces between the user/shell and the macro-Particle class. Member functions in this module offer a window into the macro-Particle class. The add-Macro-Particle routines allow direct addition of a macro-particle with specified values. It can be called directly from the shell or from other modules.

The Ring module is a general module which controls the execution flow for particle tracking. Although it has no classes itself, it contains initialization routines to determine the order in which the Node class actions are performed, and the looping routines that actually perform the calls to the Node class calculators.

Modules for Derived Node Classes

Modules includes capabilities for adding a foil, and automatically injecting particles at the foil at each turn. They contain a Foil class which includes information about the foil. Transfer matrices are a fundamental mechanism used to transport macro-particles from one point in a ring lattice to another. A Trans-Matrix class is provided to contain transfer matrix information. The closed orbit of the Ring can be artificially altered anywhere in the Ring by introduction of an ideal bump. The Node sub-class used to contain the bumps is the Ideal-Bump class. A module is provided to add RF cavities. The Node sub-class is the RF-Cav.

Miscellaneous calculations are performed by specialized modules. Presently, the only calculation done here is to find the emittance of the macro-particles, Eq. (7).

REFERENCES

[1] SNS Collaboration, "The National Spallation Neutron Source", NSNS/CDR-5 1997
[2] F.W. Jones, "Users Guide to ACCSIM ", TRI-DN-90-17, June 1990.
[3] F.Ch. Iselin, "The MAD Program, Version 8.7", CERN/SL/92(AP), July 17, 1992
[4] A.U.Luccio, "Numerical Calculation of the Tune Spread Induced by Transverse Space Charge in a Synchrotron", BNL/NSNS TechNote 023, Jan. 29, 1997
[5] A.W. Chao,"Physics of Collective Beam Instabilities in High Energy Accelerators", Wiley, New York 1993
[6] J.Holmes, J.D.Galambos, D.K.Olsen, and S.Y.Lee, "An RMS Particle Core Model for Rings", Proc. Shelter Island Workshop, May 2-5, 1998
[7] S.W. Hanoi, "Using and Programming the SUPERCODE", July 21, 1995
[8] M.Blaskiewicz, M.Brennan, Proc. EPAC'96, p.2373