Interim report

Software development for electron cooling calculation

The code development in the frame of Attachment#2

I.Meshkov, A.Smirnov, A.Sidorin, G.Trubnikov,
JINR, Dubna, Russia

January 31, 2005

Abstract

The report describes modifications in the BETACOOL code developed during the first stage of the work according to the Attachment#2 to the Agreement on a Software Development for Electron cooling Calculation. The purpose of modifications was to improve models for Intrabeam Scattering (IBS) simulation, to develop numerical procedures for beam-beam parameter evaluation, to provide simulation of the electron cooling using results of numerical calculation of the friction force. At this stage of the work a version of the BETACOOL program for UNIX operation system was prepared and tested. The procedures required for integration of the BETACOOL algorithms into Unified Accelerator Library (UAL) were prepared. Required modifications of interface part of the program were performed. New model of electron bunch was introduced and some modifications in the code related to the luminosity and IBS calculation were done.

Introduction

Within the framework of the Attachment#2 JINR is obligated to carry out software development, provide a manual with a complete description of the code and perform numerical simulation required to compare prediction of different models of general effects. General goals of new software development are

- to perform simulations of the ion distribution function evolution in time using real electron distribution function calculated with external program,
- to improve the "core – tail" model of intrabeam scattering process simulation on the basis of theory by G.Parzen,
- to perform simulations of intrabeam scattering at coupled transverse motion of the ions,
- to perform benchmarking of the UNIX version of the program,
- to prepare a version of the program for multi processor calculations,
- to provide intrabeam scattering growth rate calculation using Molecular Dynamics technique at different ion distribution functions,
- to integrate the BETACOOL algorithms into ion dynamics simulation with UAL,
- to begin simulations using results of numerical calculation of the electron cooling friction force obtained with another programs,
- to develop numerical procedures for beam-beam parameter evaluation and include diffusion due to beam-beam effect into simulations.

It was planned to carry out the work in two stages. Each of them includes the code development and integration of BETACOOL algorithms into UAL (both further development and UAL integration are provided in parallel).

This report describes results of the first stage of the work during which the following code development was performed:

1.
1.1. Improvement of the "core – tail" model of IBS simulation on the basis of bi-Gaussion approximation of the ion distribution function.
1.2. Improvement of algorithms for luminosity calculation and development of the electron beam model.
1.3. Preparation the procedures and algorithms for simulation of the cooling process using numerical results of the friction force calculation.
1.4. Development of the algorithms for beam-beam parameter calculation and preparation for simulation of diffusion due to beam-beam effect.

2.
2.1 Preparation and benchmarking of UNIX version of BETACOOL.
2.2. Compiling the BETACOOL program as a library of procedures, preparation of required adapter procedures for integration of BETACOOL under UAL framework.
2.3. Comparison of the intrabeam scattering simulations in the frame of Model Beam algorithm and tracking based on Molecular Dynamics technique.
2.4. Testing and benchmarking of major cooling models under UAL.

In this report we present general results of the code modification. The development of the algorithms for luminosity calculation is related to introduction of a model of hour-glass effect estimates using Model Beam (MB) algorithm, and it will be described in detail in the final report. The electron cooling simulation was modified in order to permit input of the parameters of electron distribution function in the form of the electron bunch emittance and momentum spread. New model of the electron beam "uniform bunch" was developed in addition to Gaussian bunch and Gaussian cylinder to cover all typical electron distributions in the co-ordinate space. To support all the models the required modifications in input file and interface part of the program were done. In Fig. 1 an example of new visual forms in the interface dedicated to input of electron beam model parameters is presented. The input parameters for the uniform bunch model are the following: the bunch dimensions in the transverse planes (the model presumes uniform density in the transverse direction and Gaussian distribution in the longitudinal degree of freedom), rms bunch length, distance in longitudinal direction between electron and ion bunch centres and number of electrons in the bunch. The other forms containing parameters for electron cooling simulation were rebuilt in accordance with the electron cooling model described in [1]. The modifications of the electron cooling are aimed to provide the simulation of the cooling process using real electron distribution at the second stage of the work and their detailed description will be one of the topics of the final report.

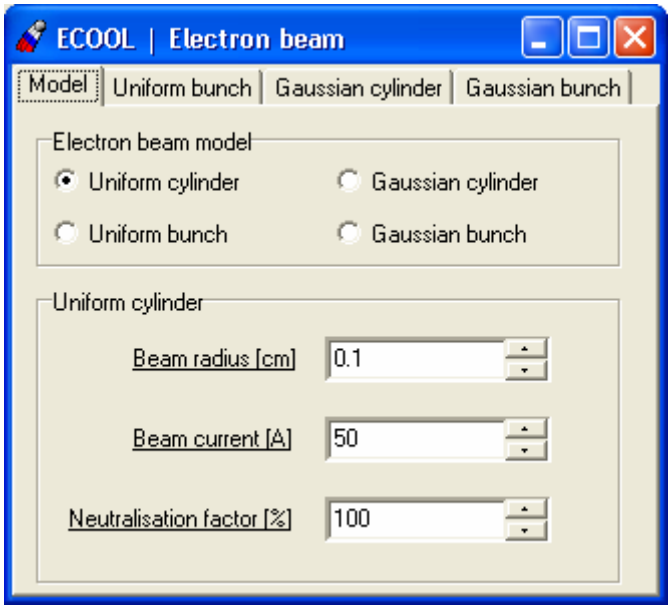


Fig. 1. New form for choice of the electron beam model.

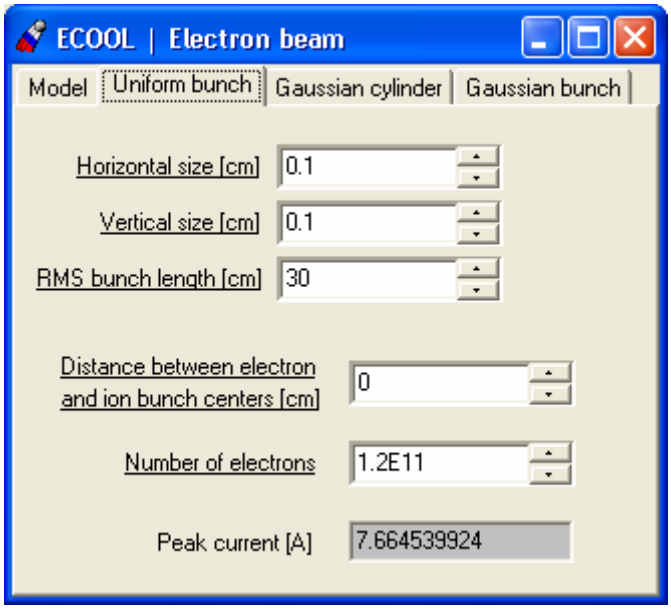


Fig. 2. The form for input the electron bunch parameters at uniform bunch model.

The developed algorithms for beam-beam parameter calculation, the procedures and algorithms for simulation of the cooling process using numerical results of the friction force calculation, description of the "core-tail" model using bi-Gaussian approximation of the ion distribution function, results of preparation and benchmarking of UNIX version and integration of BETACOOL under UAL framework are presented in the independent parts of this report.

1. Beam-beam parameter calculation

Diffusion power due to beam-beam interaction in the collision point is calculated as a function of beam-beam parameter which has a meaning of linear part of betatron tune shift due to beam-beam collision. As a first step in beam-beam effect simulation a few algorithms for beam-beam parameter calculation were developed and tested in the BETACOOL code. In this part the analytical theory and different numerical algorithms for the beam-beam parameter calculation are described.

1.1. Analytical formulae for beam-beam parameter

We start consideration of beam-beam effects with calculation of an increment of transverse particle momentum after crossing the opposite bunch (see Fig. 1.1). Consider "strong-weak" approximation to beam-beam interaction. In this model it is assumed that particles of a weak-beam (index 2) are influenced by a strong electromagnetic field of the opposite bunch (index 1), while the strong bunch does not feel any field from the weak bunch.

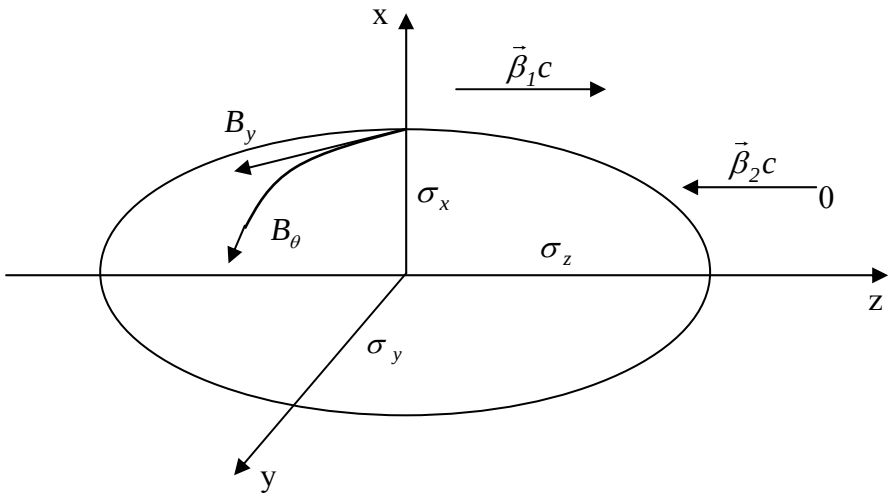


Fig 1.1. Interaction of test particle with strong opposite bunch

Assume that opposite bunch with N_1 particles has the Gaussian space charge density distribution with rms bunch size σ_x, σ_y, σ_z:

$$\rho(x,y,z,v_1,t) = \frac{q_1 N_1}{(2\pi)^{3/2}\sigma_x\sigma_y\sigma_z} exp\left(-\frac{x^2}{2\sigma_x^2} - \frac{y^2}{2\sigma_y^2} - \frac{(z-v_1 t)^2}{2\sigma_z^2}\right). \quad (1.1)$$

Let us come to moving frame of coordinates (noted as prime coordinate system). Longitudinal position in moving frame is

$$z' = \gamma(z - v_1 t). \quad (1.2)$$

In the moving frame the space charge density is

$$\rho'(x,y,z') = \frac{q_1 N_1}{(2\pi)^{3/2}\sigma_x\sigma_y(\sigma_z\gamma)} exp\left(-\frac{x^2}{2\sigma_x^2} - \frac{y^2}{2\sigma_y^2} - \frac{z'^2}{2\sigma_z^2\gamma^2}\right). \tag{1.3}$$

Electrostatic potential of the Gaussian bunch is:

$$U'(x,y,z') = \frac{q_1 N_1}{4\pi^{3/2}\varepsilon_0}\int_0^\infty \frac{exp\left[-\frac{x^2}{(2\sigma_x^2+w)} - \frac{y^2}{(2\sigma_y^2+w)} - \frac{z'^2}{(2\sigma_z^2\gamma^2+w)}\right]}{\sqrt{(2\sigma_x^2+w)}\sqrt{(2\sigma_y^2+w)}\sqrt{(2\sigma_z^2\gamma^2+w)}}dw. \tag{1.4}$$

Transversal components of electrostatic field in the self frame are attained by differentiation of potential (1.4) $E'_x = -\partial U'/\partial x$, $E'_y = -\partial U'/\partial y$:

$$E_x = -\frac{q_1 N_1 \gamma}{2\pi^{3/2}\varepsilon_0}\int_0^\infty \frac{exp\left[-\frac{x^2}{(2\sigma_x^2+w)} - \frac{y^2}{(2\sigma_y^2+w)} - \frac{\gamma^2(z-v_1 t)^2}{(2\sigma_z^2\gamma^2+w)}\right]}{(2\sigma_x^2+w)^{3/2}\sqrt{(2\sigma_y^2+w)}\sqrt{(2\sigma_z^2\gamma^2+w)}}dw, \tag{1.5}$$

with analogous expression for E_y component. Moving bunch of charged particles creates magnetic field (see Fig. 1.1)

$$B_x = -\beta_1\frac{E_y}{c},\ B_y = -\beta_1\frac{E_x}{c} \tag{1.6}$$

Equations of transverse test particle motion are

$$\begin{aligned}\frac{dp_x}{dt} &= q_2\left[E_x - (-v_2 B_y)\right] = q_2 E_x(1+\beta_1\beta_2),\\ \frac{dp_y}{dt} &= q_2\left[E_y - (-v_2 B_x)\right] = q_2 E_y(1+\beta_1\beta_2).\end{aligned} \tag{1.7}$$

To define a change of particle momentum after crossing an opposite bunch, equations (1.7) have to be integrated along the time of integration. Assume that particle position and Lorentz force are not changed during test particle crosses the opposite bunch (thin lens approximation). Longitudinal equation of motion of test particle $z = -v_2 t$ has to be substituted into expression for field (1.5). In the adopted approximations the change of transverse particle momentum is

$$\Delta p_x = q_2\left(1+\beta_1\beta_2\right)\int_{-\infty}^{\infty} E_x dt = -\frac{q_1 q_2 N_1 (1+\beta_1\beta_2)}{2\pi^{3/2}\varepsilon_0\left(v_2 - v_2\right)} x \int_0^{\infty} \frac{exp\left[-\frac{x^2}{\left(2\sigma_x^2+w\right)} - \frac{y^2}{\left(2\sigma_y^2+w\right)}\right]}{\left(2\sigma_x^2+w\right)^{3/2}\sqrt{\left(2\sigma_y^2+w\right)}} dw, \qquad (1.8)$$

similar for Δp_y. Consider linear approximation to increment of particle momentum. Integral in Eq. (1.8) can be evaluated analytically:

$$\int_0^{\infty} \frac{\exp\left[-\frac{x^2}{\left(2\sigma_x^2+w\right)} - \frac{y^2}{\left(2\sigma_y^2+w\right)}\right]}{\left(2\sigma_x^2+w\right)^{3/2}\sqrt{\left(2\sigma_y^2+w\right)}} dw \approx \int_0^{\infty} \frac{dw}{\left(2\sigma_x^2+w\right)^{3/2}\sqrt{\left(2\sigma_y^2+w\right)}} = \frac{1}{\sigma_x\left(\sigma_x+\sigma_y\right)}. \qquad (1.9)$$

Therefore, linear approximation to increment of particle momentum is

$$\Delta p_x = -\frac{q_1 q_2 N_1\left(1+\beta_1\beta_2\right)}{2\pi\varepsilon_0\left(\beta_2+\beta_1\right)\sigma_x\left(\sigma_x+\sigma_y\right)} x. \qquad (1.10)$$

Let us introduce the value of beta-function at the interaction point β_x^*, β_y^*. Then the change of slope of particle trajectory in linear approximation can be written as follow:

$$\Delta\frac{dx}{dz} = \frac{\Delta p_x}{p_z} = 4\pi\frac{\xi_x}{\beta_x^*}x, \ \Delta\frac{dy}{dz} = \frac{\Delta p_y}{p_z} = 4\pi\frac{\xi_y}{\beta_y^*}y,$$

where ξ_x, ξ_y are beam-beam parameters, which have a meaning of linear part of betatron tune shift due to beam-beam collision:

$$\xi_x = N_1\frac{\beta_x^*}{4\pi}\frac{q_1 q_2}{4\pi\varepsilon_0 m_2 c^2}\frac{\left(1+\beta_1\beta_2\right)}{\gamma_2\beta_2\left(\beta_1+\beta_2\right)}\frac{2}{\sigma_x\left(\sigma_x+\sigma_y\right)},$$

$$\xi_y = N_1\frac{\beta_y^*}{4\pi}\frac{q_1 q_2}{4\pi\varepsilon_0 m_2 c^2}\frac{\left(1+\beta_1\beta_2\right)}{\gamma_2\beta_2\left(\beta_1+\beta_2\right)}\frac{2}{\sigma_y\left(\sigma_x+\sigma_y\right)}. \qquad (1.11)$$

In the case of arbitrary density distribution in the opposite bunch the beam-beam parameters can be calculated in accordance with the definition:

$$\xi_x = \frac{\beta_x^*}{4\pi p_z}\lim_{x\to 0}\frac{\Delta p_x}{x}.$$

The momentum variation is calculated with the same formula (1.8)

$$\Delta p_x = q_2\left(1+\beta_1\beta_2\right)\int_{-\infty}^{\infty} E_x dt\,,$$

but the electric field component is calculated as a solution of Poisson equation at given distribution shape. For *y* direction the formulae are similar.

The other way is to use the same formulae (1.11) but to calculate the beam size as dimensions of the region containing given percent of particles or as a Full Width on Half Maximum (FWHM) of the beam distribution in co-ordinate space. In this case the formulae (1.11) have to contain additional normalizing factor, which provides coincidence of the different formulae in the case of Gaussian distribution.

For a bunch of a round shape of cross-section ($\sigma_x, = \sigma_y = \sigma$) the term $\frac{N}{\pi\sigma^2}$ can be interpreted as a particle local density in the centre of the bunch integrated along *z* co-ordinate. Such approach permits to calculate the beam-beam parameter without solution of Poisson equation. At the current stage of the software development the algorithm for beam-beam parameter calculation on the basis of different emittance definition and local density evaluation are realized.

1.2. Numerical algorithms for beam-beam parameter calculation

For gold-gold collisions at RHIC the beam-beam parameter is calculated for a collision of two identical bunches. In the frame of Rms dynamics algorithm the analytical formulae (1.11) for beam-beam parameters can be rewritten in the following form:

$$\xi_x = \frac{Z^2e^2}{Amc^2}\frac{N\beta_x^*}{4\pi\sigma_x\left(\sigma_x+\sigma_y\right)}\frac{\left(1+\beta^2\right)}{\gamma\beta}, \tag{1.12}$$

$$\xi_y = \frac{Z^2e^2}{Amc^2}\frac{N\beta_y^*}{4\pi\sigma_y\left(\sigma_x+\sigma_y\right)}\frac{\left(1+\beta^2\right)}{\gamma\beta}. \tag{1.13}$$

here mc^2 = 932 MeV is nucleon rest energy, *A*, *Z* are the ion atomic and charge numbers, *N* is the ion number, β, γ are relativistic parameters which are equal for both bunches. The rms beam dimensions - $\sigma_{x,y}$ – are calculated from the beam rms emittance under assumption that the dispersion and alpha functions in the collision point are equal to zero:

$$\sigma_{x,y} = \sqrt{\varepsilon_{x,y}\beta_{x,y}^*}\,. \tag{1.14}$$

In the frame of Model Beam algorithm the beam dimensions can be calculated in accordance with different emittance definitions or the formulae (1.12, 1.13) can be rewritten using local particle density in the central part of the bunch.

When the emittance is calculated from the model particle array as statistic rms values the beam-beam parameter is calculated using the same formulae (1.14, 1.12, 1.13).

At the beam dimensions definition via FWHM of the model particle distribution the beam-beam parameter in each plane is calculated using the particle number inside the FWHM. For this in the formulae (1.12, 1.13) instead of the ion number N one uses the value

$$N_{x,y} = \frac{N_{FWHM,x,y} N}{N_{MB}}, \tag{1.15}$$

where N_{FWHM} is the model particle number inside the area corresponded to FWHM beam dimension in the corresponding plane, N_{MB} is the total particle number in the model beam.

When the emittance is defined as a square of ellipse containing given percent of the ions, the beam rms dimensions for the formulae (1.12, 1.13) is calculated with the normalization factor

$$\sigma = \frac{\sqrt{\varepsilon_\eta \beta^*}}{2\ln\left(\frac{1}{1-100\eta}\right)}. \tag{1.16}$$

where η is the percent of particles using for emittance definition. At Gaussian distribution the normalization (1.16) provides independence of the beam-beam parameter on the emittance definition.

The beam-beam parameter can be calculated also via local density of the model particles in accordance with the formula:

$$\xi_{x,y} = \rho \frac{Z^2 e^2}{Amc^2} \frac{N\beta^*_{x,y}}{4N_{MB}} \frac{\left(1+\beta^2\right)}{\gamma\beta}, \tag{1.17}$$

where ρ is the model particle local density in cm^{-2}. The local density is calculated along the trajectory of the ion, located at the equilibrium orbit in the center of the bunch. Algorithm of the local density calculation is the following:

- for each model particle its distance from the equilibrium orbit in radial direction is calculated in accordance with

$$r_i = \sqrt{x_i^2 - y_i^2}, \tag{1.18}$$

- from the obtained array one choices N_{count} ions having minimum distance from the equilibrium orbit, N_{count} is input parameter for the algorithm (Fig. 1.3),

- mean square distance of the ion from axis is calculated

$$\left\langle r^2 \right\rangle = \frac{\sum_{i=1}^{N_{count}} r_{i,\min}^2}{N_{count}} \tag{1.19}$$

- finally, the local density is calculated as

$$\rho = \frac{1}{2} \frac{N_{count}}{\pi \left\langle r^2 \right\rangle}. \tag{1.20}$$

The multiplier ½ provides equivalence of the formula (1.17) to formulae (1.12, 1.13) in the case of Gaussian distribution. The described algorithm is similar to algorithm for luminosity calculation via local ion density.

1.3. Interface description and preliminary results of benchmarking

The beam-beam parameter is calculated during dynamics simulation in the case when the effect "Collision Point" is active. Correspondingly all input parameters in the BOLIDE interface are located in the form "Effects | Collision Point" in additional tab sheet (Fig. 1.2). For calculation via local density the particle number N_{count} is the same as for luminosity calculation via local density (Fig. 1.3).

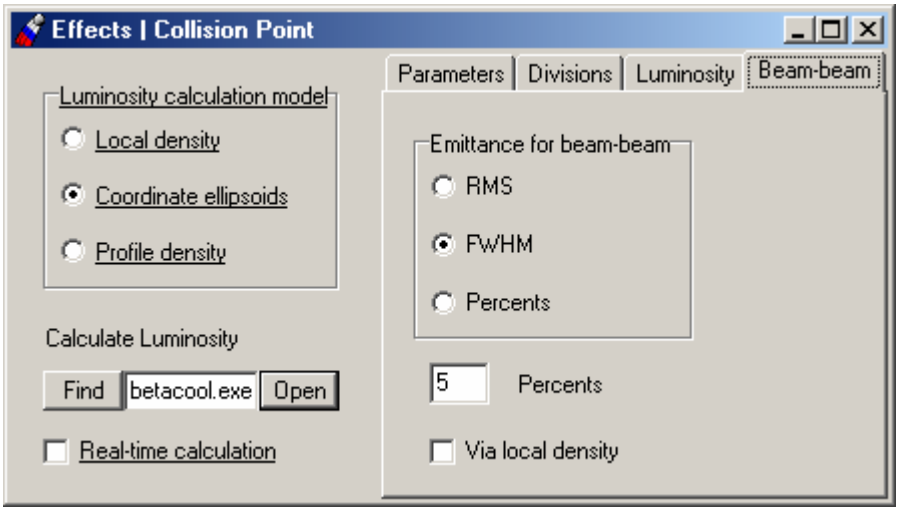


Fig. 1.2. Tab sheet for choice of the model for beam-beam parameter calculation in the frame of Model Beam algorithm. The beam-beam parameter is calculated via local density when the check-box "Via local density" is checked, in the opposite case the calculations are provided in accordance with chosen emittance definition.

Example of the beam-beam parameter calculations using different algorithms are listed in the Table 1.1. The beam parameters at the calculations were: horizontal and vertical emittances are $2.5 \cdot 10^{-8}$ π·m·rad, momentum spread 0.001 and the ion number is 10^9. Number of the model particles is 2000. Maximum fluctuation of the result takes a place for calculations via local density due to

pure statistics: rms spread of the beam-beam parameter value from realization to realization is about 6%, that is approximately equal to $\frac{1}{\sqrt{N_{count}}}$. In the other cases the spread of results is determined by the model particle number and it is about 1 – 2 %.

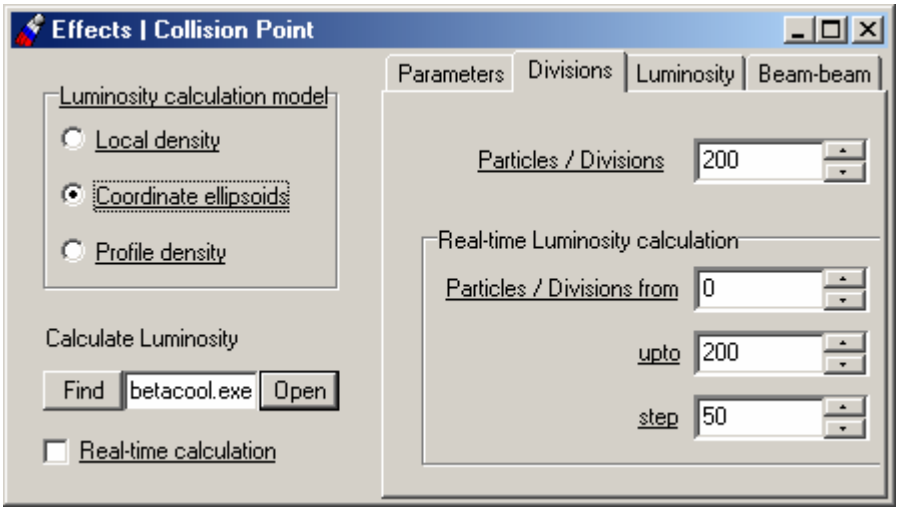


Fig. 1.3. The number of ions for beam-beam parameter calculation via local density N_{count} is input in the counter "Particles/Divisions".

Table 1.1. Example of beam-beam parameter calculation for Gaussian bunch.

	ξ_x	ξ_y
RMS Dynamics algorithm	0.001441	0.001441
Model Beam algorithm		
Rms emittance	0.001439	0.001441
FWHM emittance	0.001499	0.001542
30% emittance	0.001393	0.001426
40% emittance	0.001441	0.001373
50% emittance	0.00139	0.001453
60% emittance	0.001385	0.001425
Via local density, $N_{count} = 200$ (different realizations of the model particle array)	0.001497 0.001456 0.001332 0.001459 0.001438 0.001384 0.001424 0.001339 0.001547 0.001276 0.001314 0.001394	

2. Electron cooling simulation using tabulated friction force values

As soon as no any existing model for friction force calculation from the electron cooling effect satisfies and truly describes influence of cooling electrons on ions, the model for the electron cooling with direct calculation of binary collisions between real particles was proposed. In this case the result of simulation is a table of friction force values for different ion transverse and longitudinal velocities. The initial parameters for the modeling must be number of particles, range of studied velocities and step over velocities. Calculation of such a tabled is planned with VORPAL code developed in Colorado Univ. This code is intended to be calculated in large cluster farm.

In order to optimize the simulation process it was proposed to use BETACOOL code for the definition of the table of friction force values and the fastest interpolation method.

Scheme of investigation was the following: with a known analytical model for friction force the table was generated and it was assumed as a tabulated one obtained from the direct binary collision calculation. Then this table saved to file was used in BETACOOL as one of the model of friction force calculation and was compared in rms dynamics simulation to the original analytical one. Result obviously depends on the velocity range and number of steps over velocity. So the goal of optimization was obtaining the similar results with tabulated and analytical models varying with range (or number of ranges) and splitting number in every range, and kind of interpolation over the table, simultaneously to minimizing calculation recourses.

2.1. Expected shape of the friction force

General problem of the cooling process simulation using analytical friction force formulae can be simply illustrated on example of a semi-empirical formula by Parkhomchuk. In accordance with this formula the friction force in magnetized electron beam has a symmetry form for both longitudinal and transverse components and can be presented by the following vector equation:

$$\vec{F} = -\vec{v}_i \frac{4Z^2 e^4 n_e L_P}{m} \frac{1}{\left(v_i^2 + \Delta_{e,eff}^2\right)^{3/2}}, \tag{2.1}$$

where n_e is the electron density, Δ_{eff} is the effective electron velocity spread with taking into account variations of the magnetic field line position in the transverse direction. The ion velocity v_i has two components – along and across the magnetic field line:

$$v_i = \sqrt{v_\perp^2 + v_\parallel^2} \tag{2.2}$$

The Coulomb logarithm L_p is given by the expression:

$$L_P = \ln\left(\frac{\rho_{\max} + \rho_{\min} + \langle\rho_\perp\rangle}{\rho_{\min} + \langle\rho_\perp\rangle}\right). \tag{2.3}$$

Where the minimum impact parameter is calculated in accordance with

$$\rho_{\min} = \frac{Ze^2}{m_e} \frac{1}{v_i^2 + \Delta_{e,eff}^2}, \tag{2.4}$$

$$\langle \rho_\perp \rangle = \frac{\Delta_\perp m_e c}{eB} \tag{2.5}$$

is the Larmor radius of electron rotation around the magnetic field line, where $\Delta_\perp$ is the rms transverse velocity of the electrons.

The maximum impact parameter is calculated as a minimum from three values:

$$\rho_{\max} = \min\left\{ \max\left(\rho_{sh}, \sqrt[3]{\frac{3Z}{n_e}} \right), v_i \tau, a \right\}. \tag{2.6}$$

The dynamic shielding radius is calculated using same formula:

$$\rho_{sh} = \frac{\sqrt{v_i^2 + \Delta_e^2}}{\omega_p}, \tag{2.7}$$

Second term describes the distance, which the ion passes inside the electron beam. Here τ is the ion time of flight the cooling section in the PRF:

$$\tau = \frac{l_{cool}}{\beta\gamma c}. \tag{2.8}$$

and the last term is the electron beam radius.

The formula (2.1) well describes the friction force components when the Coulomb logarithm has a value of about 10, which corresponds to “good” magnetization. The maximum impact parameter is determined by the ion velocity, and minimum one – by the transverse emittance of the electron beam and the magnetic field value. When the ratio between maximum and minimum impact parameters is about unity, the formula (2.1) as the other formulae deduced in the logarithmic approximation can not predict the friction force shape correctly. In this case one needs to provide numerical evaluation of the friction force value. To minimize the time of the numerical simulation one needs to optimize the mesh parameters in the velocity space.

For instance, the transverse component of the friction force:

$$F_\perp = -v_\perp \frac{4Z^2 e^4 n_e L_P}{m} \frac{1}{\left(v_\perp^2 + v_\parallel^2 + \Delta_{e,eff}^2 \right)^{3/2}} \tag{2.9}$$

does not depend on longitudinal component of the ion velocity if

$$v_{\parallel} << v_{\perp} \tag{2.10}$$

and depends of $v_{\parallel}$ as

$$F_{\perp} \sim \frac{1}{v_{\parallel}^3} \tag{2.11}$$

in the opposite case. Correspondingly, the mesh step over the longitudinal ion velocity can be larger in the region (2.10) and smaller in the other space. In the region (2.10) the friction force dependence on the transverse velocity component has three characteristic regions:

- linear in the range of small velocity,
- the maximum when the ion velocity is closed to effective electron velocity spread,
- $F_{\perp} \sim \frac{1}{v_{\perp}^2}$ in the region of high ion velocity.

Optimum step of the mesh along the transverse component of the ion velocity is different in all these intervals. The smallest step value has to be chosen in the range where one can expect inclination of real dependence from theoretically predicted.

In principle one can expect peculiarity of the transverse friction force component in the region of small ion velocity predicted by Derbenev-Skrinsky formula:

$$F_{\perp} = -v_{\perp} \frac{2\pi Z^2 e^4 n_e L_M}{m v^3} \frac{v_{\perp}^2 - 2v_{\parallel}^2}{v^2}, \tag{2.12}$$

(here $L_M = \ln \frac{\rho_{\max}}{\langle \rho_{\perp} \rangle}$) the force can change a sign when $v_{\perp} < \sqrt{2} v_{\parallel}$. Correspondingly, step of the mesh has to be reduced in this region.

The procedure for the mesh generation described in this report was developed as a first step of the mesh structure optimization and requires further development, which will be provided in the second stage of the work.

2.2. Interpolation methods

As the simplest interpolation method was chosen linear interpolation when the nearest node for the given value is chosen. On the one hand this way is fast enough but it is very rough and gives acceptable results when the size of table is huge and very difficult to be processed by program.

The second method is 2D interpolation – so called bilinear [William H. Press et al., Numerical Recipes in C, Cambridge university press]:
In multidimensional interpolation, we seek an estimate of y(x1, x2, . . . , xn) from an n-dimensional grid of tabulated values y and n one-dimensional vectors giving the tabulated values of each of the

independent variables x 1, x2, . . . , xn. We will not here consider the problem of interpolating on a mesh that is not Cartesian, i.e., has tabulated function values at "random" points in n-dimensional space rather than at the vertices of a rectangular array. For clarity, we will consider explicitly only the case of two dimensions, the cases of three or more dimensions being analogous in every way. In two dimensions, we imagine that we are given a matrix of functional values ya[1..m][1..n].We are also given an array x1a[1..m], and an array x2a[1..n]. The relation of these input quantities to an underlying function y(x 1, x2) is
ya[j][k] = y(x1a[j], x2a[k])

We want to estimate, by interpolation, the function y at some untabulated point (x1, x2).
An important concept is that of the grid square in which the point (x1, x2) falls, that is, the four tabulated points that surround the desired interior point. For convenience, we will number these points from 1 to 4, counterclockwise starting from the lower left. More precisely, if

x1a[j] ≤ x1 ≤ x1a[j+1]
x2a[k] ≤ x2 ≤ x2a[k+1]
defines j and k, then
y1 ≡ ya[j][k]
y2 ≡ ya[j+1][k]
y3 ≡ ya[j+1][k+1]
y4 ≡ ya[j][k+1]

The simplest interpolation in two dimensions is bilinear interpolation on the grid square. Its formulas are:
t ≡ (x1 - x1a[j])/(x1a[j+1]- x1a[j])
u ≡ (x2 - x2a[k])/(x2a[k+1]- x2a[k])
(so that t and u each lie between 0 and 1), and
y(x1, x2) = (1 - t)(1 - u)y1 + t(1 - u)y2 + tuy3 + (1 - t)uy4

Bilinear interpolation is frequently "close enough for government work." As the interpolating point wanders from grid square to grid square, the interpolated function value changes continuously. However, the gradient of the interpolated function changes discontinuously at the boundaries of each grid square.

2.3. Description of the numerical procedures and benchmarking results

For testing and comparison of all the models of calculations a special form in the BETACOOL code was created where user have a possibility to choose a model for the friction force calculation, type of interpolation, and variants of ion velocity range definition (Fig. 2.1).

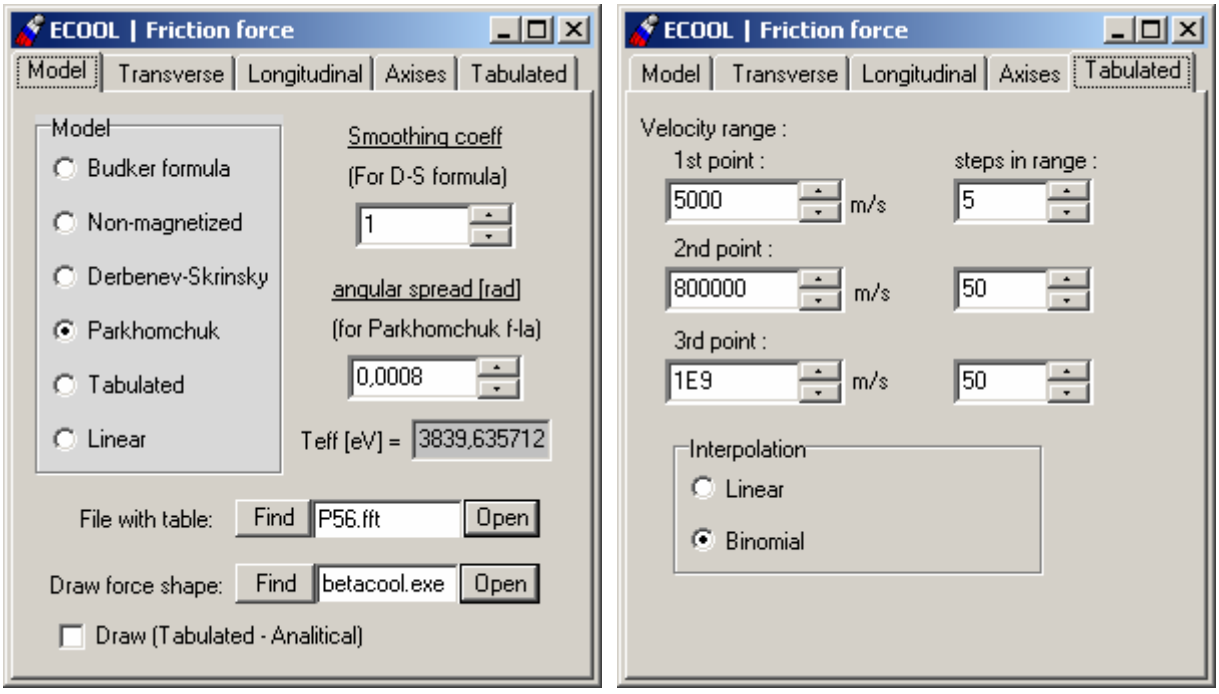


Fig. 2.1. Friction force Form.

File with tabulated force values is saved in specialized format. User can define borders for several velocity ranges: 3 point mean 3 ranges – from 0 [m/s] to the 1st point, from the 1st point to the 2nd value of velocity, and from the 2nd to the 3rd one (Fig. 2.2). Inside every range the number of steps over velocity can be defined. As soon as force dependence is symmetrical of zero value we can take into account only positive range of velocities.

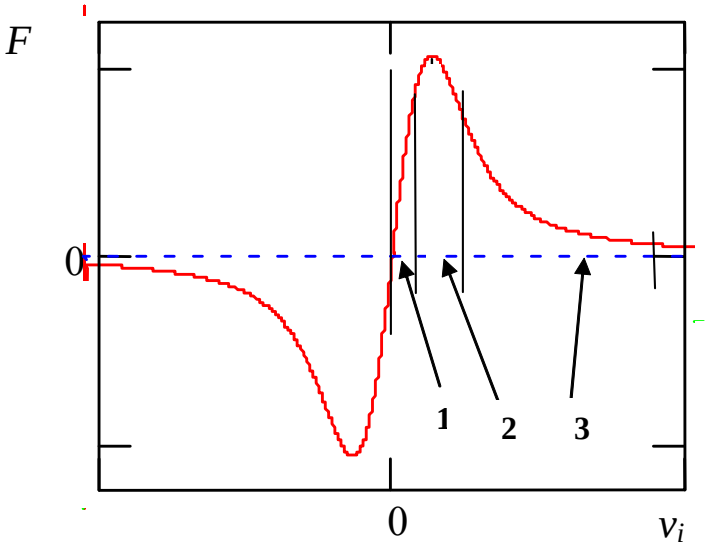


Fig. 2.2. Analytical friction force dependence and defined velocity ranges.

At first such a splitting was intended to make a detailed analysis and splitting in the range 2, where there is a peak, which can be incorrectly interpretec with interpolation. This peak theoretically corresponds to the so-called effective temperature o the ions and can be easily interpreted in

velocity units [m/sec]. For the RHIC parameters this value is about $7 \cdot 10^5$ m/sec. The dependence for the range 1 (as V) and range 3 (as $1/V^2$) were well known. In this way we tried to make about 5-10 splits in the 1st range (from 0 – to $5 \cdot 10^5$ m/s) and ($9 \cdot 10^5$ – to the maximal possible ion velocity ~ $3 \cdot 10^8$ m/s). But for the second range maximal possible splitting was proposed (from 50 to 500 – depends on the PC power).

After some investigations the correct tendency for the approach was found: maximal splitting must be not in the "uncomfortable" region of the peak, but in the range where the most ion velocity are found for the machine parameters, namely transverse ion velocities. For the investigated RHIC parameters this region for transverse ion velocity is about $1 \cdot 10^6$ m/s. Here the contribution of the velocities is largest to the total friction force value. As soon as transverse velocities are larger than longitudinal (~ by two orders of magnitude) they dominate in the final friction force value (both velocities contribute to the friction force formula with the same weights).
Criteria for good coincidence of friction force values obtained with interpolation from table and analytically calculated one can use comparison of 3D diagrams for beam r.m.s. growth rate dependence in the working range of transverse and longitudinal emittances for RHIC parameters.

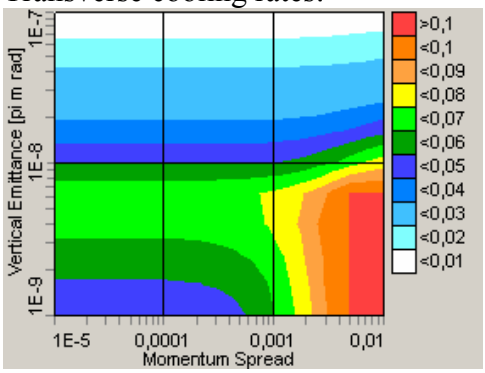
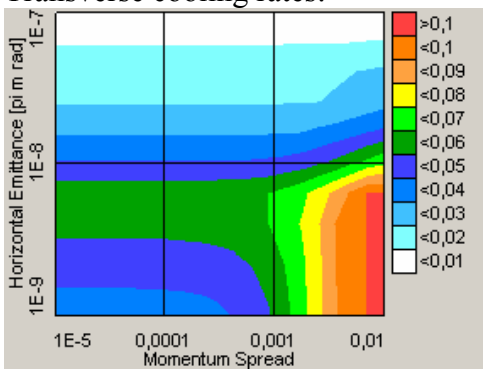
During optimization table splitting and velocity range were minimized. As a result reasonable conditions for the direct simulation of friction force table were achieved:

1st range of velocities: from 0 to $5 \cdot 10^3$ m/sec with 5 splits
2nd range of velocities: from $5 \cdot 10^3$ to $8 \cdot 10^5$ m/sec with 20 splits
3rd range of velocities: from $8 \cdot 10^5$ to $5 \cdot 10^6$ m/sec with 50 splits.

At least for the tested parameters, this suggests that the friction force can be accurately represented by approximately 70 x 70 numerical arrays. In other words, about 5000 velocity point calculation will be needed with the VOPRAL codes for each set of parameters. To decrease this number one needs to develop more appropriate algorithm of the mesh formation. First of all the number of divisions and step have to be different in longitudinal and transverse degrees of freedom.

Mentioned above 3D plots for comparison are presented in the Fig. 2.3.

Uniform cylinder

Analytical formula (Parchomchuk)	Bilinear interpolation from the table (par31.fft)
Transverse cooling rates: 	Transverse cooling rates: 
Longitudinal cooling rates:	Longitudinal cooling rates:

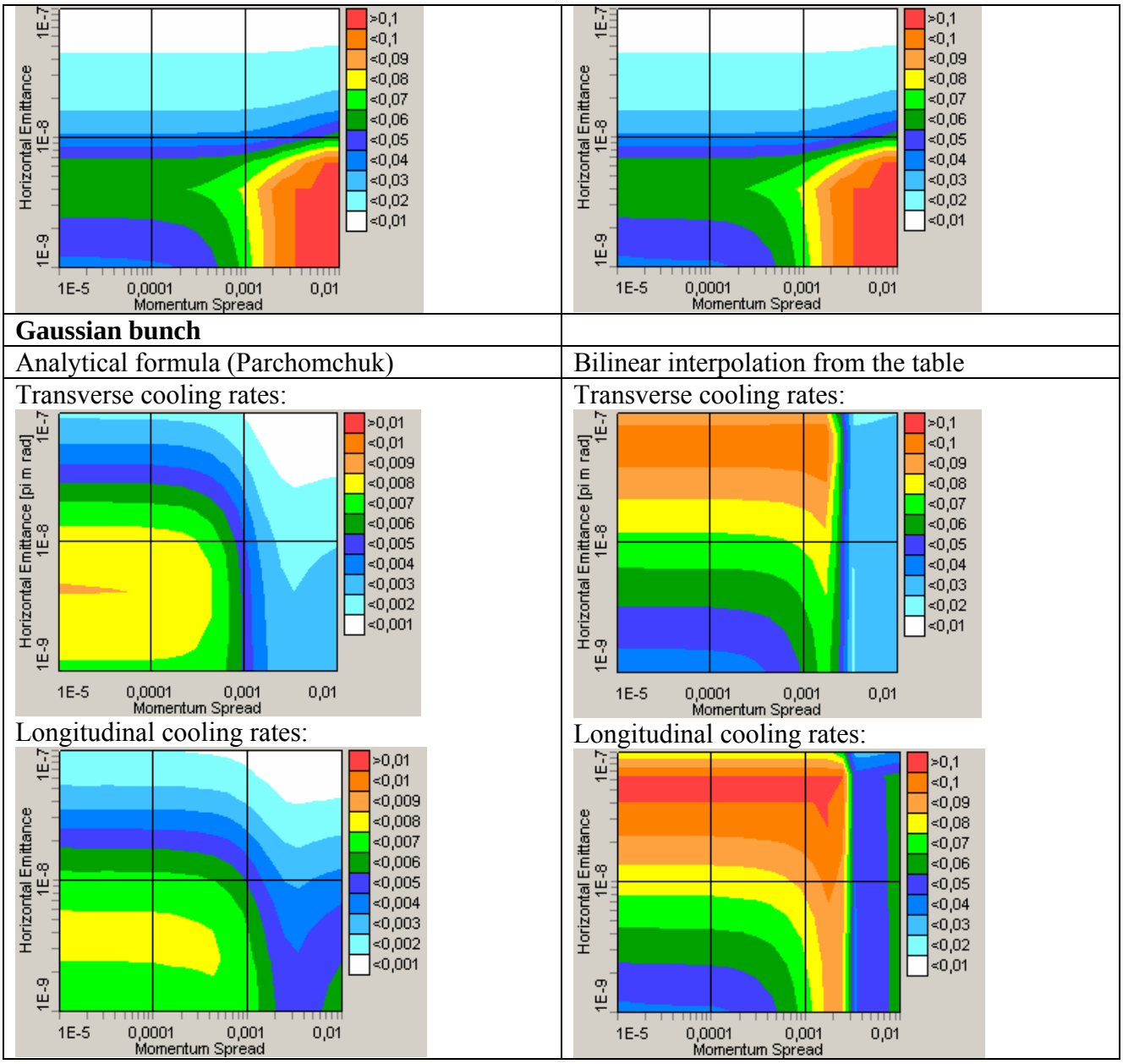


Fig. 3.3. Comparison of cooling rates calculated using analytical formula and interpolation of tabulated values calculated with the same formula.

At chosen parameters of the mesh the coincidence is good for the electron beam model as an uniform cylinder. Difference in the cooling rates at Gaussian bunch reflects the fact, that the friction force is the function of transverse co-ordinates in this case. The friction force table was calculated at electron density in the centre of the electron bunch, but the density averaged over the ion betatron and synchrotron oscillations is sufficiently less. To avoid overestimation of the cooling force one needs to calculate the friction force table at some effective electron density, which can be estimated by comparison of the results calculated at different densities with the analytical formulae.

3. Bi-Gaussian model of IBS

As a first step in realization of intrabeam scattering process simulation on the basis of theory by G.Parzen the "core – tail" model was improved using bi-Gaussian approximation of the ion distribution function. In the frame of the "core-tail" model in the Model Beam algorithm the core parameters were calculated using FWHM characteristics of the beam profile, the tail was determined as an rms profile width. In general case the model particle distribution function can be presented as a sum of two Gaussian functions for core and tail particles:

$$f(x) = f_{tail}(x) + f_{core}(x) = a_{tail} \exp\left[-\frac{1}{2}\left(\frac{x}{\sigma_{tail}}\right)^2\right] + a_{core} \exp\left[-\frac{1}{2}\left(\frac{x}{\sigma_{core}}\right)^2\right], \tag{3.1}$$

where a_{tail}, a_{core}, σ_{tail}, σ_{core} – amplitudes and widths of beam profile. These parameters can be calculated using mean square method. This method minimizes the deviation between the model beam profile and Bi-Gaussian distribution:

$$\sum_{i=1}^{N} \left[y_i - f(x_i)\right]^2 \to Minimum\,, \tag{3.2}$$

where (x_i, y_i) – points of the beam profile distribution, N is total number of profile histogram divisions. The optimization problem is solving in the program using Powell method. The same procedure is realized for all degrees of freedom: for momentum distribution, horizontal and vertical profiles. RMS parameters of the ion beam $\vec{E}_{RMS}\left[\varepsilon_x, \varepsilon_y, \Delta p / p\right]$ are calculated in the suggestion that distribution of model particles has Gaussian shape. Beam parameters (emittances, momentum spread) and particle number of Bi-Gaussian distribution are:

$$\vec{E}_{tail} = \vec{E}_{RMS}\sigma_{tail}^2, \quad \vec{E}_{core} = \vec{E}_{RMS}\sigma_{core}^2, \quad N_{core} = \left[1 + \frac{\sigma_{tail}}{\sigma_{core}} \cdot \frac{a_{tail}}{a_{core}}\right]^{-1}, \quad N_{tail} = 1 - N_{core}\,. \tag{3.3}$$

The IBS growth rates for core and tail parts of Bi-Gaussian distribution are calculates with standard procedure on the base of choosing IBS model for Gaussian distribution:

$$\begin{aligned} \vec{R}_{tail} &= IBS.Rates(\vec{E}_{tail}, N) \\ \vec{R}_{core} &= IBS.Rates(\vec{E}_{core}, N) \end{aligned}. \tag{3.4}$$

Model particles in the tail get the kick with standard procedure using of $\vec{R}_{tail}$ growth rates. Particles in the core get the kick from core and tail heating rates:

$$\vec{R}_{core} N_{core}^2 + \vec{R}_{tail} N_{tail}^2 \frac{\vec{E}_{tail}}{\vec{E}_{core}}\,. \tag{3.5}$$

4. Integration of BETACOOL under UAL framework

The Unified Accelerator Libraries (UAL) provides a modularized environment for applying diverse accelerator simulation codes. At this time the object-oriented programs included in UAL are: PAC (Platform for Accelerator Codes), ZLIB (Numerical Library for Differential Algebra), TEAPOT (Thin Element Program for Optics and Tracking), ACCSIM (Accelerator Simulation Code), ORBIT (Objective Ring Beam Injection and Tracking Code). Modules that are partially supported and are under active development are ICE (Incoherent and Coherent effects), AIM (Accelerator Instrumentation Module), SPINK (tracks polarized particles in circular accelerator), and TIBETAN (longitudinal phase space tracking). The Application Programming Interface (API), written in Perl, provides a universal shell for integrating and managing all project extensions.

BETACOOL as a program calculating evolution of r.m.s beam parameters in presence of heating and cooling effects was successfully implemented into UAL. As soon as BETACOOL has a wide kit of different models for heating effects (intrabeam scattering, interaction of two beams, scattering on residual gas, etc) and cooling effects (electron cooling, stochastic cooling) all these effects have rather complex structure. The task was to organize a simple way to call any model in the frame of UAL and to integrate and transfer all internal parameters which BETACOOL operates with including initial parameters into UAL global scope.

BETACOOL uses a lot of parameters for any effect model, they are initial parameters of the beam(s), ring structure, lattices, parameters for the effect calculation itself, etc., and a kit of internal parameters which are temporary calculated. It is necessary to integrate them into UAL or initialize some of them from UAL variables.

The obvious advantage of BETACOOL implementation is object oriented method of the program code. In order to use some procedures and functions from BETACOOL effects calculation was proposed library-like scheme. Program code is compiled and linked to the library and special program-adaptor must be created using UAL interface which will contain so called “bridge” functions which let to use BETACOOL possibilities for effects simulation in the UAL framework.

4.1. Library realization and adaptor scheme

BETACOOL library is created as shared library ***libUalBetacool.so*** under Linux OS with a help of gcc compiler. This library is created in a standard way, we used a make utility.

To use all the BETACOOL functions it is necessary to make a linkage to BETACOOL header files where all the objects and functions are declared (here ***#include*** is to the folder where all BETACOOL headers are):

```
// Betacool classes
#include "..\include\xdynamic.h"
```

we made ***#include*** only to one BETACOOL header file because all others are included from it.

Two adaptors are created for now. One of them is declared and described in files ***Ring.hh, Ring.cc***.

This adaptor connects all the BETACOOL parameters with UAL internal variables. It creates an object of specially declared class Ring. When this object is created it uses a standard BETACOOL input file (of *.bld type) as a parameters of initialization, where all the simulation parameters are described. User must edit this file and set parameters in accordance with simulation task.

```
// Constructor
BETACOOL::Ring::Ring(std::string& fileName)
{
  char fn[120];
  strcpy(fn, fileName.c_str());
  xData::Get(fn); // read file of parameters and initialization of Betacool objects
  xData::Set(fn); // more initialization of Betacool objects and save file of parameters
}
```

This constructor uses standard BETACOOL functions ***xData::Set()*** and ***xData::Get()*** for setting internal parameters. These functions load input file, organize a special data array inside BETACOOL framework and set its elements in accordance with a parameter file, where the coincidence between BETACOOL data array and parameters name is specialized.

Calling of any BETACOOL parameters is possible via global BETACOOL objects, which are declared in ***xdynamic.h*** file. Here is a list of these objects:

iDraw
iBeam
iRing
iTime
iTaskRates
iDynamics
iEcool
iForce
iEbeam
iIBS
iLosses
iHeat
iTarget
iRestGas
iColl
iStochastic

A special function is foreseen in the described adaptor file, inside class ***Ring***. It lets to fill the ring lattices in the BETACOOL array of variables with lattices from UAL which are calculated by one of the program in the UAL framework. This function creates an array of optic elements from the optic structure calculated in UAL.

```
/* Calculates twiss parameters and sets Ring containers */
```

```
void build(const char* latticeName);
```

When tracking of lattices is completed by UAL tool and they are stored in dedicated array (here in example ***vtwiss***), they can be transferred to BETACOOL ring elements setting the kit of lattices for every element in accordance with the lattices calculated by another library of UAL (for example TEAPOT):

```
// Start the initialization of Betacool_Ring

 iRing.Number(vtwiss.size()-1);
 iRing.Arc = 0;
 for(int j = 0; j < vtwiss.size()-1; j++){
  iRing[j].EL_LATTICE = true;
  iRing[j].Length = m_lattice[j+1].getLength();
  iRing.Arc += iRing[j].Length;
  iRing[j].Lattice.dist  = iRing.Arc;
  iRing[j].Lattice.betax = vtwiss[j].beta(0);
  iRing[j].Lattice.betay = vtwiss[j].beta(1);
  iRing[j].Lattice.alfax = vtwiss[j].alpha(0);
  iRing[j].Lattice.alfay = vtwiss[j].alpha(1);
  iRing[j].Lattice.Dx = vtwiss[j].d(0);
  iRing[j].Lattice.Dy = vtwiss[j].d(1);
  iRing[j].Lattice.Dpx = vtwiss[j].dp(0);
  iRing[j].Lattice.Dpy = vtwiss[j].dp(1);
```

The second adaptor file (***CompositeTracker.cc, CompositeTracker.h***) contains description of few functions, which let to change beam parameters due to active effects influence using BETACOOL simulation models.

Here special class ***CompositeTracker*** is organized. The main function described here makes tracking of the particle beam through effects:

```
void BETACOOL::CompositeTracker::propagate(PAC::Bunch& bunch)
{
 readBunch(bunch);
 iBeam.Emit = iBeam.Get_Emit(iRing.LATTICE);
 iDynamics.Drawing(iRing.LATTICE);
 xLattice Lattice2 = iRing.LATTICE;
 for (int i = 0; i < xEffect::ACount; i++)
 {
  if (xEffect::AItems[i]->Use)
  {
   transRotate(Lattice2, xEffect::AItems[i]->Lattice);
   addKick(i);
   Lattice2 = xEffect::AItems[i]->Lattice;
  }
 }
```

```
  transRotate(Lattice2, iRing.LATTICE);
  longRotate();
  ++iTime;
  writeBunch(bunch);
}
```

It uses the following functions, which are created in adaptor file and use internal BETACOOL procedures:

- ***readBunch()*** – creates a bunch of particles (BETACOOL object) as an array, in accordance with number of model particles, and fills coordinates of every particles from real coordinates, calculated in and taken from UAL;

- ***iBeam.Get_Emit*** – calculates beam emittance accordingly to obtained coordinates and matched to the current element lattices;
- ***addKick()*** – calculates momentum deviation from the active effect (if such) in accordance with BETACOOL algorithm of Model Beam;

- ***transRotate()*** – propagates particles through optic element – changes particle transverse coordinates with the help of the element transformation matrix. Transformation matrix is calculated in the area between lattices, which are parameters of the function;

- ***longRotate()*** – rotates bunch in longitudinal phase space;

- ***write bunch ()*** – takes back changed array of particles to the UAL framework.

Two additional functions are also declared in adaptor file.
First of them allows setting lattices for any point of "BETACOOL ring" from the UAL array of lattices:

```
  ModelBeamTracker.setLattice(twiss);
```

The second function has two parameters: name of effect, and lattices in the point where this effect is located. Function checks if such an effect is considered in BETACOOL initial parameters and if found sets lattices there in accordance with UAL calculation:

```
  ModelBeamTracker.registerEffect("XADDHEAT", "clock8");
```

Here in adaptor file user can add in accordance with class ***CompositeTracker*** syntax any other function where can be used BETACOOL tools.

4.2. User program and usage of adaptors and library

To use adaptor functions with BETACOOL library user must follow next 2 steps:

1. Compile and link his own code with include of path to the library: ***libUalBetacool.so*** and BETACOOL header files. It can be done during the assembly of the executable user file with the following instructions in Makefile:

Include library instruction: -L ../lib/ -lUalBetacool
Include instruction: -I. -I../lib/

2. In user's own program code it is necessary to make #include instructions to BETACOOL headers and to the header file of adaptor:

```
#include "CompositeTracker.hh"      // Adaptor reference
#include "Ring.hh"                  // Adaptor reference
#include "xdynamic.h"               // Betacool reference
```

Then the usage of adaptor and library to calculate effects from any of BETACOOL models is simply calling for functions:

```
//---------------------------------------------------------------------------
  std::cout << "START BETACOOL " << std::endl;
//---------------------------------------------------------------------------

// read BETACOOL input file
BETACOOL::Ring& theRing = BETACOOL::Ring::getInstance("rhicfodo.bld");

// set BETACOOL Lattices from UAL
 theRing.build(ring.c_str());
 std::cout << "Lattice Number = " << iRing.Number() << ", Circumference = " << iRing.Circ()
<< std::endl;

// calculate growth rates from BETACOOL Effects
vectorU Rates = xEffect::Summary(iTime, iBeam, iRing);
 std::cout << "Rates [Ex, Ey, dP/P, N] = " << Rates[0]() << ", " << Rates[1]() << ", " <<
Rates[2]() << ", " << Rates[3]() << std::endl;

// Constructor for beam propagation object
BETACOOL::CompositeTracker ModelBeamTracker;

// set time step
ModelBeamTracker.setTimeStep(iDynamics.IntegrationStep());

// set lattice in start point
ModelBeamTracker.setLattice(twiss);

// reset Lattice of Betacool Effect from UAL twiss
ModelBeamTracker.registerEffect("XADDHEAT", "clock8");
```

```
// Overwrite UAL bunch with the Gaussian distribution from Betacool
iDynamics.Distribution(iRing.LATTICE, false); ModelBeamTracker.writeBunch(bunch);

int istep = 0;

while(true)
  {
  // set time step
  ModelBeamTracker.setTimeStep(iDynamics.IntegrationStep());

// kicks from Betacool Effects (IBS, Ecool, ...)
  ModelBeamTracker.propagate(bunch);
}
```