A MULTI-BUNCH, THREE-DIMENSIONAL, STRONG-STRONG BEAM-BEAM SIMULATION CODE FOR PARALLEL COMPUTERS

A. Kabel, Y. Cai
Stanford Linear Accelerator Center, Stanford, CA 94309, USA*

Abstract

For simulating the strong-strong beam-beam effect, using Particle-In-Cell codes has become one of the methods of choice. While the two-dimensional problem is readily treatable using PC-class machines, the three-dimensional problem, i.e., a problem encompassing hourglass and phase-averaging effects, requires the use of parallel processors. In this paper, we introduce a strong-strong code `NIMZOVICH`[1], which was specifically designed for parallel processors and which is optimally used for many bunches and parasitic crossings. We describe the parallelization scheme and give some benchmarking results.

PARALLELIZATION

`NIMZOVICH` uses parallelization according to the SPMD (Single Program, Multiple Data) scheme. A cluster of processors is divided in two sections, called *Rings*. Each *Ring* is subdivided into several *Bunch*es. Bunches within a Ring are completely independent. Bunches in opposing Rings are independent, except if they have a design or parasitic interaction point in common, i.e., if one of their two geometric interaction points falls into a section of the ring (the *Window*) shared by both beams.

Each Bunch is divided longitudinally into several Slices. For reasons of load balancing, the slicing scheme is chosen in such a way as to have the same number of particles within each slice, assuming an initial gaussian distribution of given length. Slice borders are, however, not dynamically adapted to changed longitudinal distributions.

Given enough available processors, each Slice's portion of particles can be further subdivided. Portions of a Bunch with the same subdivision index in each slice are called a *Slab*. They do not represent any geometric subdivision.

Each processor on each Ring runs through the following sequence of steps for each Turn:

- For each Bunch in the sequence of opposing Bunches:
  - For each Slice in the opposing Bunch:
    - ∗ Deposit particles onto grid in the center of gravity of my slice
    - ∗ Solve Poisson's equation on that grid
    - ∗ Calculate electric field
    - ∗ Exchange electric field with opposing Slice in opposing Bunch
    - ∗ Kick particles
    - ∗ Advance particles to the next Slice
  - Advance particles to next opposing Bunch
- Advance particles according to one-turn map, possibly redistributing longitudinally

We assume that the bunch is longitudinally frozen during interactions, so the slice-to-slice interactions are independent and can be done in parallel. Also, bunches in the same ring are independent, their mutual interaction can be handled in parallel. Synchronization is automatic, i.e., a Bunch will see the opposing Ring's bunches in the right order, as the slice-to-slice operation constitutes a barrier synchronizing the two Rings.

When a Slice has passed the last opposing Slice of its last opposing Bunch within a Window, it is transported back to the design IP, and the one-turn map is applied to its particles. After that, a particle may fall out of its current Slice. All particles with changed Slice numbers are moved to one out of a set of send queues, and an asynchronous send operation to its new Slice initiated. The leftover particles are deposited on the Grid. Then, the process opens a receive queue for particles from backward Slices, which might be moved onto this Slice by the action of the one-turn map. The process does not have to wait for all backward slices, as the synchrotron tune is usually small and a particle is extremely unlikely to pass distances of the order of a bunch length within a single turn. The actual number of backward slices a process will wait is dynamically adapted at run time; if the number of particles received after a Slice's first interaction with the next Bunch crosses a threshold (of the order of a few particles), the waiting period is increased.

A complication arises from the fact that the longitudinal resolution required is very different for parasitic and design interactions. Thus, a Bunch will have different slicing schemes, with $N_{Slabs}N_{Slices}$ constant, for different interaction points. It is easy to see that communications due to reassignment of slices by a change of resolution can be kept at its minimum by (1) letting the numbers of slices in adjacent IPs be integer multiples (provided the bunch length does not change between IPs) and (2) have formerly neighboring slices end up in the same new slice for a resolution decrease.

FIELD CALCULATION

Point charges are deposited onto a cartesian grid with typical dimensions of $N_x = 64\ldots512 \otimes N_y = 64\ldots512$,

* Work supported by U.S. Department of Energy, contract DE-AC03-76SF00515.

[1] The code was previously named b2b3 (for beam-to-beam in three dimensions), however, 1. b2-b3..., in chess, is known as the Nimzovich-Larsen attack; also, the code runs well on 64 processors

using a 9-site stencil. As the beam pipe is usually far away, Poisson's equation on the grid can be solved using free boundary conditions. This is done by convolving with an appropriately discretized and regularized version $\hat{G}_{ik}$ of the free Green's function $G(r) = \frac{1}{4\pi}\log r^2$. The convolution is done by multiplication in momentum space; the transformation into momentum space is done by a two-dimensional Fast Fourier Transformation using the FFTW[2] package. Free boundary conditions are implemented by using the Hockney trick[1] of padding the array with zeroes to $2N_x \otimes 2N_y$.

The transformation is done by two sequences of one-dimensional transformations with a matrix transposition in between. In its parallel version, the transposition involves an expensive all-to-all communication, which might cancel the speed gains of parallelizing the transformation. In `NIMZOVICH`, the user has the choice of how finely to parallelize the solver. In our calculations, we find that the time spent in the solver equals the kick-deposit time at around 10^4 particles.

Note that this is not the optimal solution; the fact that the array was zero-padded initially allows one to get rid of $2N_x$ of $2N_x + 2N_y$ FFT's right away. $2N_x$ other transformations can be done out-of-place. Also, the parallel transposition becomes simpler, as the padding space can serve as a scratch space, so send and receive operations can be done simultaneously and asynchronously, decreasing latency. We have implemented this scheme for the special case of symmetric G functions and observe a speed gain of almost a factor of 2.

SLICE-TO-SLICE INTERACTION AND ADAPTIVE SLICES

The longitudinal domain decomposition makes use of Hirata slicing. The ith slice in the bunch $\pm$ is characterized by the longitudinal positions $t_e < t_c < t_l$ for its early boundary, center of gravity, and late boundary. Each $[t_e, t_l)$ contains the same number of particles for an initially gaussian distribution. To avoid field discontinuities at slice boundaries, we use a scheme due to Ohmi [5, 4] to evaluate the field within the ith slice in bunch $\mp$ opposing the kth slice in bunch $\pm$: Particles are deposited (by longitudinal projection) onto a grid each at times $\frac{t_e^{\pm,i}+t_c^{\mp,k}}{2}$ and $\frac{t_l^{\pm,i}+t_c^{\mp,k}}{2}$. The field of the distributions is calculated, and the resulting kick is applied to a particle μ with longitudinal coordinate $t_\mu$in bunch $\pm$ with weights $\frac{t_\mu - t_e^{\pm,k}}{t_l^{\pm,k} - t_e^{\pm,k}}$ and $\frac{-t_\mu + t_l^{\pm,k}}{t_l^{\pm,k} - t_e^{\pm,k}}$. When on-the-fly luminosity calculation is desired during an interaction (see below), a processor will transmit the ρ matrices along with the calculated $\vec{E}$ matrices. The target processor will then sample and sum the longitudinally interpolated ρ at its particles' locations along with the $\vec{E}$, thus calculating a good approximation for $\int_{slice} \rho^\pm \rho^\mp d^2x$. For N slices, $-t_e^{1,\pm} = t_l^{N,\pm} = \infty$, so the linear scheme degenerates to a t-independent kick. There are two possible prescriptions to still introduce some smoothing. One is to modify the open Hirata slicing to a closed slicing with finite (and user-selectable) cutoff $-t_e^{1,\pm} = t_l^{N,\pm} = T$. Particles with $t\mu$ without that closed interval will be discarded, resulting in a relative particle loss of $\exp(-\sigma_t/T)$, which will occur over the first $1/2\nu_{sync}$ turns, provided the beam-beam effect does not increase the longitudinal emittance. The other prescription is to use an open slicing, but use the ith center of gravity as a reference point for field interpolation, thus turning the interpolation into an extrapolation in the fringe slices.

In general, each slice will execute grid operations (sampling fields or depositing particles) on 4 different temporal positions. In a beam with a pronounced hourglass effect, the transverse dimensions of the beam might vary substantially for these times. We adapt the transverse extensions of the grids to the expected extensions of the beam, calculated from the unperturbed Twiss functions. This way, we achieve constant effective resolution across the interaction process and can use a lower-resolution grid than codes with grids of constant absolute resolution. For each slice, we have to pre-calculate two $\hat{G}$ matrices for each opposing slice, as $\hat{G}$ does not follow a simple scaling law under temporal displacement for $\beta_x \neq \beta_y$. We are currently testing a dynamic scheme in which the grid sizes are adapted to the beam dimensions as measured during the course of the simulation, which would relieve the user of having to have an estimate of beam size increase. Note that this scheme is very memory-intensive, as each Green's function on the grid has to be stored. Thus, the user can turn off this feature, losing usable resolution but gaining memory efficiency.

INPUT AND OUTPUT

The code allows for a very general description of the storage ring. The input parameters are specified as ring properties (particle masses, energies, and charges; tune advances; damping times; Twiss functions at the design IP, equilibrium emittances) for both rings, as bunch properties (emittances, particle numbers, and position in the bunch train) and IP properties (transverse offsets, crossing angles, grid resolutions). For now, we assume the IPs are distributed symmetrically around the design IP and that they are separated from the design IP by pure drift spaces and that properties like offsets and crossing angles are independent from the sequence number in the train, which is not necessarily true for non-equidistand bunch trains. We will generalize the parametrization to allow for machines such as LHC and the Tevatron.

The complete content of the particle heap will be written to disk in regular intevals, selectable by the user. The files generated are platform-independent HDF5 files [3], they comprise the complete information for all the particles, including loss flags and the internal state of the random number generators. We use the MPI parallelized version of HDF5; thus, the output will be completely independent from the computing platform or the number of processors used, thus facilitating a restart facility.

Also, each processor writes to a log file. This logfile contains collective quantities calculated on-the-fly, such as luminosity (summed over a complete turn for each single bunch), coordinate averages, and correlation matrices. All these quantities refer to coordinates of particles in the interaction point in the same turn. To calculate them, one has to stop the earliest slice in a bunch in the design IP, wait for the other slices to catch up, calculate the quantities in question, and restart the interaction process. It is easy to see that this method would decrease efficiency by a factor of 2 if applied in each turn; thus, the calculation are done at user-selectable intervals.

We also implemented an on-the-fly tune calculation. The user can specify a number of particles for which the tune is to be calculated, the particles will be selected at random from each bunch. The selected particles' coordinates will be stored for a number of turns specified by the user and determining the tune resolution. As the affiliation or particles to processors might have changed, the particles' histories will be sorted according to particle number across processors at the end of the cycle. The coordinate vectors are fast-fourier-transformed, and the tunes are determined using the Laskar algorithm and written to the log file. The stop-restart procedure described above also applies to this calculation.

BENCHMARKING

We have checked two typical cases: One is a single-bunch luminosity simulation for Super-KEKB with parameters as given by table 1. The other is a simulation of PEP-II, including the two nearest parasitic crossings, taken into account with a longitudinal resolution of 1, and for a bunch train of 4 bunches. It was not possible to do a multi-bunch simulation with sufficient resolution due to a bug in the HDF5 implementation at NERSC occuring for high number of processors. We could, however, check the code for consistency in this case. The results for Super-KEKB luminosity and rms y beam radius are given in figures 2, 1 and show good agreement with [6].

REFERENCES

[1] R. W. Hockney and J. W. Eastwood, *Computer Simulation Using Particles*. Bristol and Philadelphia, 1988.

[2] M. Frigo and S. G. Johnson, FFTW 2.15 User's Manual, `http://www.fftw.org/fftw2_doc/`

[3] F. Baker for the HDF5 project, HDF5 User's Guide, `http://hdf.ncsa.uiuc.edu/HDF5/doc/UG/`

[4] K. Ohmi, Phys. Rev. E**62**, 7287 (2000)

[5] K. Ohmi, in *Proceedings of the 2003 IEEE Particle Accelerator Conference*

[6] K. Ohmi, M. Tawada, Y. Cai, S. Kamada, K. Oide, and J. Qiang, PRL **92** (2004), 214801-1–4

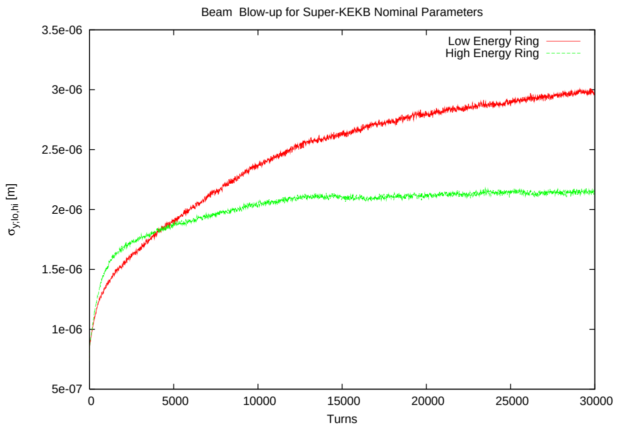

Figure 1: Beam Blow-Up for Super-KEKB.

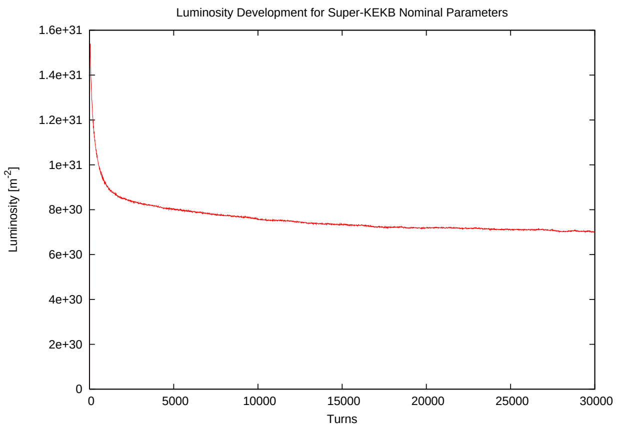

Figure 2: Luminosity for Super-KEKB.

Table 1: Benchmark Parameters. For PEP, a bunch spacing of 1.26 m is used

Symbol	PEP-II		Super KEKB		Units
	LER	HER	LER	HER	
E_0	3.1	9.0	3.5	8.0	GeV
N	7.15	4.41	12.6	5.5	10^{10}
β_x^*	0.50	0.27	0.30	0.30	m
β_y^*	10.5	11.1	3.0	3.0	mm
σ_z	10.5	11.6	3.0	3.0	mm
σ_δ	0.65	0.61	0.7	0.7	10^{-3}
ε_x	22.0	59.0	24.0	24.0	nm
ε_y	1.40	2.33	0.18	0.18	nm
ν_x	0.5162	0.5203	0.508	0.508	
ν_y	0.5639	0.6223	0.550	0.550	
ν_s	0.0270	0.0495	0.02	0.02	
$\tau_{x,y}$	9800	5030	4000	4000	Turns
τ_s	4800	2573	2000	2000	Turns