PARALLEL SIMULATION ALGORITHMS FOR THE THREE-DIMENSIONAL STRONG-STRONG BEAM-BEAM INTERACTION

Andreas C. Kabel*, SLAC

Abstract

The strong-strong beam-beam effect is one of the most important effects limiting the luminosity of ring colliders. Little is known about it analytically, so most studies utilize numeric simulations. The two-dimensional realm is readily accessible to workstation-class computers (cf.,e.g.,[1, 2]), while three dimensions, which add effects such as phase averaging and the hourglass effect, require vastly higher amounts of CPU time. Thus, parallelization of three-dimensional simulation techniques is imperative; in the following we discuss parallelization strategies and describe the algorithms used in our simulation code, which will reach almost linear scaling of performance vs. number of CPUs for typical setups.

TWO DIMENSIONS

For the simplest parallelization of the strong-strong problem, both beams are represented by macro particles. We have two parallelizable tasks: the application of fields to the particles and the calculation of the fields due to the particle distribution. The latter problem is addressed by a particle-on-grid algorithm: A discretized charge density $\rho_{\vec{i}}$ is used to calculate the electric potential $\phi_{\vec{i}}$ by solving the discretized Poisson equation. A particle is deposited on $\rho_{\vec{i}}$ ($\vec{i}$, $\vec{k}$,... are two-dimensional integer vectors, corresponding to positions $x = x_0 + h_x i_1, y = y_0 + h_y i_2$ on the grid) by adding charge to the lattice sites nearest to it with distance-depending weights. We use a third-order momentum-conserving deposition scheme[3]. For the two-dimensional sub-problem, we make use of algorithms introduced and validated in [1, 2].

We divide the particles into pools local to processors. For the two-dimensional problem, a single grid is placed in the interaction point, perpendicular to the directions of motion. A parallelized solver step now is a sequence of: (1) collecting $\rho_{\vec{i}}$ from test particles (2) add all $\rho_{\vec{i}}$ from particle pools (3) distribute sum $\rho_{\vec{i}}$ to solver processors (4) calculate $\vec{E}_{\vec{i}}$ from $\rho_{\vec{i}}$ (5) distribute $\vec{E}_{\vec{i}}$ to particle pools (6) apply $\vec{E}$ to test particles (7) transport particles around the complete ring or to the next interaction point. The transport operation involves all of the particle dynamics between IPs (usually linear transport (hadron machines) or linear transport + damping ($e^{\pm}$ machines)).

* Work supported by Department of Energy contract DE–AC03–76SF00515.

THREE DIMENSIONS

For the solution of the three-dimensional problem, we divide the bunches into n_z longitudinal slices of equal lengths h_z and slice numbers $i_z \in [0, n_z)$, 0 representing the leading slice. We now need to place (for both bunches) $2n_z - 1$ grids numbered $1 - n_z \ldots n_z - 1$ at positions $\frac{1-n_z}{2}, \ldots, \frac{n_z-1}{2}$ around the IP.

For each encounter of bunches, $2n_z - 1$ steps need to be executed. numbered $s \in [0, 2n_z - 2]$. In step s, particles in slice $k \in [\max(0, s - n_z + 1), \min(s, n_z - 1)]$ are deposited on grid $\pm(k - s)$ (sign according to the direction of flight). All updated grids are then used to calculate fields, and the resulting fields are applied to the opposing bunch's particles longitudinally nearest to the respective grid. Each bunch encounter thus consists of n_z^2 slice encounters comprising 1 two-dimensional deposit/solve/kick step each.

When the bunch length is comparable to the β function in the IP, the hourglass effect becomes significant, meaning that the grids far away from the IP have to accommodate a larger bunch diameter than the grids close to it. To optimize resolution, we scale the grid resolutions according to $h_{x,y} \propto \sqrt{1 + \frac{z^2}{\beta^{*\,2}_{x,y}}}$.

Field Calculation

The field calculation is based on a convolution algorithm. The discretized charge distribution $\rho_{\vec{i}}$ is convoluted with the discretized Green's function $G_{\vec{i}-\vec{k}} \propto \log \sum_i \frac{(i_i - k_i)^2}{h_i^2}$ for the two-dimensional Coulomb problem. The convolution can be done efficiently by Fast Fourier transforming ρ, doing a point-wise multiplication with the Fourier transform $\tilde{G}$ of G, and transforming back. If we choose a lattice of dimensions $L = [0, 2h_x n_x) \otimes [2h_y n_y)$, but restrict the support of ρ to $L' = [0, h_x n_x) \otimes [0, h_y n_y)$, the $(2n_x, 2n_y)$ periodicity of $\tilde{G}$ will not modify the potential in L', i. e. the method will obtain the correct potential for open boundary conditions. This is the famous Hockney trick [3]. To avoid the singularity of the Green's function at the origin, we choose a natural smoothing prescription: we shift the Green's function by $\frac{1}{2}\vec{h}$, such that $G_0 = 0$, and evaluate the fields at a position shifted by $-\frac{1}{2}\vec{h}$.

The Green's function is pre-calculated at program start. As G obeys no simple scaling law for the case of $\beta_x^* \neq \beta_y^*$, this precalculation needs to be done for each encounter point. The method is easily generalized to non-concentric lattices, different resolutions for different beams, and dynamic rescaling of lattices, should the beam dimensions change.

Parallelizing the convolution method amounts to parallelizing the local multiplication with $\tilde{G}$, which is trivially done, and parallelizing the Fast Fourier Transform. For the latter, we use the high-performance, open-source parallel FFT library 'FFTW'[4], Calculation of the electric field is done by discretized differentiation with an appropriate weight algorithm[2]. The fields can be applied to particles by scattering particles to the appropriate slices or by gathering the fields into the particle pools; we choose the latter solution in our code.

We can make use of an additional symmetry property of the system: as there are two rings involved, we split the processors into two subgroups, each assigned to one of the bunches. The only communication necessary between these subgroups is then the exchange of the charge density $\rho_{\vec{i}}$, which can be done after collecting it to the root process of the solver. Thus, only a single pair of communicators between processes assigned to different bunches is necessary.

The advantage of this procedure is due the hardware configuration of the computer system available to us. On the IBM SP at NERSC, 16 processors share a node and can communicate via shared memory. Communications between nodes will be over a fast network, but still be substantially slower. Thus, it is advantageous to limit the distribution of the Poisson solver, which will involve a large amount of all-to-all communications, to one node. By the bisection of the problem the communications overhead penalty will start to set in at 32 processors instead of 16 processors.

Communications overhead can be further reduced by using another possible parallelization. As soon as the longitudinal slice number $n_z > 1$, each encounter will involve the independent encounters of several slices. Particle deposition and solving the Poisson equation can then be done in parallel, making it possible to keep both local to a single node by setting the number of processors in a solver to $n_{sp} < n_p/2$. However, not every encounter step involves the encounter of an integer multiple of n_{sp}, so some solvers will have been idling during one encounter. The optimum choice for n_{sp} depends on the hardware setup and the ratio of CPU time usage for solving the Poisson equation and particle-grid-dynamics, resp., so it has to be found experimentally for each given number of particles and grid size.

THE SLICE ALGORITHM

While the algorithm described above allows for a great flexibility with respect to variable computer parameters such as number of processors, number of processors in a fast sub-cluster etc., its performance for a higher number of processors is disappointing; test runs on the NERSC facility show it to go into saturation at $\approx 32 \ldots .64$ processors. This is due to the choice of a common "pool" of particles, shared among all processors, with no attempt at localization in physical space. Thus, particle localization is desirable, however, the dynamics between interaction will move a particle from one processor's responsibility into another's. Care must be taken not to lose in particle management communication what was gained by saving field communication.

Consequently, particles should be assigned to processors according to their longitudinal coordinate: The longitudinal dynamics in a storage ring usually is much slower than the transverse one, meaning that a relatively small number of particles will change processors during a single turn.

A simple equidistant slicing will lead to a very uneven distribution of particles, leaving most of the tracking work to the processors responsible for the center slices. This can be cured by a Hirata-type slicing [5], choosing borders $\zeta_i, \zeta_0 = \infty, \zeta_{z_n} = -\infty$ such that the number of particles in $[\zeta_i, \zeta_{i+1}) = N_p/n_z$; the encounter points between slices i and k are chosen at a distance $\frac{z_i - z'_k}{2}$ from the IP, where z_i, z'_k are the centers of gravity of the slices in the respective bunch. Again, the grids' resolutions are scaled according to z_i and β.

The Wraparound Algorithm with Idle Cycles

The processors assigned to the head of the bunch will be idle after the centers of the bunches have passed each other, as there are no collision partners left.

One can, however, apply the transfer map of the lattice up to the next IP (or to the beginning of the same IP) in the first of these idle steps, and do the next collision in the next step. Pparticles from trailing slices may still move into a leading slice when they are transported through the ring after they have encountered their last collision partner. This can be partially cured by inserting a 'cool-down cycle', i. e., a slice, after having been transported, waits for one or several additional idle steps, leaving CPUs unused, for particles from its trailing slices to catch up.

Assuming a matched distribution (i. e., $\rho(p,q) = \tilde{\rho}(H(p,q))$) and a quadratic Hamiltonian, and scaling the canonical variables to $\sigma_{p,q} = 1$, the number of particles moving from a slice $q_1 < q \leq q_2$ to a slice $q_3 < q \leq q_4$ in two-dimensional phasespace during a phase advance $\Delta\phi = 2\pi\Delta\nu$ (or vice versa, as the distribution is invariant under rotations) is given by the integral over the parallelogram obtained by overlapping one slice with the other, rotated slice:

$$\Delta N = N_P \int_{q_3}^{q_4} \mathrm{d}q \int_{q_1/\sin\Delta\phi + q\cot\Delta\phi}^{q_2/\sin\Delta\phi + q\cot\Delta\phi} \mathrm{d}p \rho(H(p,q)) \quad . \tag{1}$$

For a gaussian distribution, this integral has to be evaluated numerically.

The acausal leakage rate can now be calculated by use of (1), a plot of the maximum acausal leakage rate per turn vs. the number of inserted idle cycles is given in Fig. 1. The synchrotron tunes are $0.04, 0.02,$ and 0.00072 (PEP II HER, PEP II LER, and Tevatron, resp.), the number of slices is 11. For the Tevatron, the leakage rate is completely benign even for just 1 idle cycle, resulting in near-optimal CPU utilization. For the sake of clarity, we give

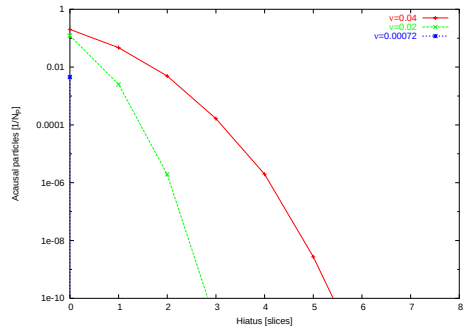

Figure 1: Acausal leakage rate for typical machines and a longitudinal decomposition into 11 slices

the wraparound algorithm in some detail for a configuration of 5 slices and a hiatus of 2. We get the following table of operations: Here, K_i stands for a kick due to slice

Table 1: Sequence of parallel operations for 5 slices and a hiatus period of 2; column number=slice number, row number=step number

0	1	2	3	4	5	6
$K_4 L$	F_0^1	$F_0^2 B_0^2$	$K_0' F_0^3$	$K_1' F_0^4$	K_2'	K_3'
K_3	$K_4 L F_0^2$	$F_0^2 B_0^3$	$F_0^3 B_1^3$	$K_0' F_0^4$	K_1'	K_2'
K_2	K_3	$K_4 L F_0^2 B_0^2$	$F_0^3 B_1^3$	$F_0^4 B_2^4$	K_0'	K_1'
K_1	K_2	K_3	$K_4 L F_0^3 B_1^3$	$F_0^4 B_2^4$	B_3^4	K_0'
K_0	K_1	K_2	K_3	$K_4 L F_0^4 B_2^4$	B_3^4	1

i of the opposing bunch, K_i' for a kick due to slice i of the opposing bunch in the next IP, L for the transport to the next IP, $F(B)_i^k$ for a forward (backward) re-shuffling operation involving slices $[i, k]$ ($k(i)$ being the originating slice. Each K operator involves the transport to the appropriate encounter point and the (un)projection on/off the slice before/after the actual kicks step. Each L operator involves the transport into the IP. The sequence expects and releases the particles in completely overlapping bunches, transported into the IP. The sequence is synchronized by collective operations and leaves no collective operation open after the encounter. The Bs' scope is different from the Fs': while one has to accept the occasional causality-violating particle being transported forward, causality violations for B can be avoided by transporting no further backwards than to the youngest slice in hiatus. Particles not belonging there will move out when it is that slice's turn to be originator of a backwards re-shuffling operation.

We have test-run the code on the NERSC facility; this time, we observe an almost linear behavior of the CPU time vs. CPU number (Fig. 2). Due to the much more favorable localization of particles on CPUs as opposed to the pool algorithm, a breakdown of this behavior will not set in before the most communication-intensive process, field solving, is distributed among more than 16 CPUs. Thus, for a typical slice number of 32, we expect this point to be reached for 1024 CPUs.

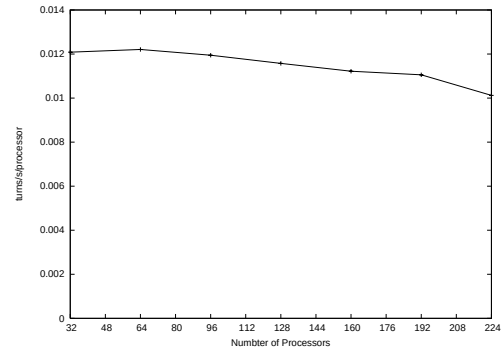

Figure 2: Scaling of a simulation run on NERSC; $n_x = n_y = 64, n_z = 16, \nu_s = 7.2 \cdot 10^{-4}$

Bunch Setup and Random Number Generation

The simulation code makes use of pseudo-random numbers during the initial setup of the particles and, in the case of electrons, for the simulation of noise induced by synchrotron radiation. The code's result have to be transparent with respect to the number of processors used; this means that initial conditions and the history of noise should be the same for a given particle, no matter what processor it is assigned to. As that assignment may change in our algorithm, a simple solution is to have each particle carry its own unique RNG. We use a 64-bit linear congruential generator. with $X_{k+1} = aX_k + p \mod m$, where a is an integer constant, $m = 2^{64}$. The generator can be made unique by chosing p the ith prime for the ith generator; this prescription will work for all particle numbers we can expect to be practically feasible.

Conclusion and Outlook

We have developed a three-dimensional strong-strong beam-beam simulation code which makes use of a novel parallelization scheme. For machines with small synchrotron tune, CPU utilization is almost optimal. The parallelization is completely transparent. We have verified the code's correctness for analytically approachable cases. We plan to use it to run precision simulation studies for PEP-II and the Tevatron.

REFERENCES

[1] S. Tzenov T. Tajima Y. Cai, A. Chao. Simulation of the beam-beam effect in e^+e^- storage rings with a method of reduced region of mesh. *PRSTA*, 4:011001, 2001.

[2] Y. Cai. Simulation of beam-beam effects in e^+e^- storage rings. *SLAC-PUB-8811*, 2001.

[3] R. W. Hockney and J. W. Eastwood. *Computer Simulation Using Particles*. Institute of Physics Publishing, 1988.

[4] M. Frigo and S. G. Johnson. Fftw: An adaptive software architecture for the fft. In *1998 ICASSP conference proceedings*, page 1381. http://www.fftw.org.

[5] H.Moshammer K.Hirata and F.Ruggiero. *Particle Accelerators*, 40:205, 1993.