SLAC-PUB-12895

Applications of parallel computational methods to charged-particle beam dynamics ☆

A. Kabel[a,*,1], Y. Cai[a,1], M. Dohlus[b], T. Sen[c], R. Uplenchwar[a,1,2]

[a] *Stanford Linear Accelerator Center, 2575 Sand Hill Road, Menlo Park, CA 94025, USA*
[b] *Deutsches Elektronen-Synchrotron DESY, Notkestr. 85, D-22603 Hamburg, Germany*
[c] *Fermi National Accelerator Laboratory, Batavia, IL, USA*

Abstract

The availability of parallel computation hardware and the advent of standardized programming interfaces has made a new class of beam dynamics problems accessible to numerical simulations. We describe recent progress in code development for simulations of coherent synchrotron radiation and the weak–strong and strong–strong beam–beam interaction. Parallelization schemes will be discussed, and typical results will be presented.

1. Coherent Synchrotron Radiation: `TraFiC`[4]

Coherent Synchrotron Radiation (CSR) occurs when short bunches travel along strongly bent trajectories, leading to a tail-head interaction of the bunch with itself due to retardation effects. Such configurations typically occur in the bunch compression sections of Free Electron Laser Facilities; its impact can be macroscopic, leading to growth of the transverse projected or slice emittance, thus degrading FEL performance, or microscopic, leading to induced longitudinal short-wavelength density modulations in the beam, which may get amplified by the subsequent compression and transport mechanism.

The code `TraFiC`[4] was developed to handle the first class of problems; it models the bunch by a collection of spatially extended, non-compressible weighted macro-particles. These particles are tracked through the magnetic lattice in the laboratory frame of reference; CSR fields are calculated from first principles by storing the history of every macro-particle and using a retardation method on these histories. The fields of all particles are calculated for each particle applied at a single location for each time step, using a split-operator approach. Obviously, this algorithm is $O(N^2_{\text{particles}})$; in practice, in all but the simplest setups, parallelization is required to get manageable running times for simulations.

Among other things, `TraFiC`[4] has been used in the design of bunch compression sections for the DESY TESLA Test Facility [1], the LCLS [2], and for the simulation of a dedicated CSR experiment at the CERN CLIC Test Facility [3]. It has been benchmarked against other codes, using different approaches and/or simplified models for CSR.

Recent improvements of the code include a complete rewrite of the tracking part in C++; extensive documentation and class structure documentation; a very flexible

☆ Invited talk presented at the 8th International Computational Accelerator Physics Conference, Saint Petersburg, Russia, June 29–July 2, 2004.

* Corresponding author.
E-mail address: akabel@slac.stanford.edu (A. Kabel).

[1] Work supported in part by the U.S. Department of Energy under contract number DE-AC02-76SF00515.

[2] Support by the U.S. Department of Energy's SciDAC program is gratefully acknowledged.

mechanism of creating bunch populations by means of applying functional operators to pre-defined distributions or distributions read from files; MPI 'stub' libraries to allow for compilation and running of TraFiC^4 on single processor machines; more efficient storage of trajectory histories; dynamic load-balancing; and a basic checkpoint/restart mechanism.

Parallelization, so far, has been relying on a crude, but effective mechanism: the nth of N nodes would run a complete replica of a single-node TraFiC^4 instance, reading the same input file. It would, however, only calculate the fields, due to *all* particles P, onto a subset of particles $P_n, P = \bigcup_i P_i, P_i \cap P_k = \{\}$. It would then gather the fields onto $P \backslash P_n$ from the other nodes in a collective synchronization step, apply them to the trajectories of P, and go to the next time step. This method, however, required storing the history of all particles on every node.

In our new approach, we only store the trajectories of particles P_n on node n. For a given lab time t, it will broadcast a field calculation request for their positions $\vec{x}_i(t)$ to the other nodes, gather calculation requests for fields due to P_n from the other nodes, process the requests, and then scatter its results and aggregate the other nodes' results, thus obtaining the total field. Most of these operations can be done asynchronously. This scheme involves more administrative overhead, however, there is no need to store any trajectories besides those of P_n. For typical problem and cluster sizes, the required memory per node could thus be reduced from 1 GB to some tens of MB.

An application of the improved TraFiC^4 code is a parameter study for the bunch compression section of the LCLS facility. In this study, more details of which can be found in Ref. [4], we vary the compression ratio of the bunch compressor to include bunch lengths well below the design values, while using nominal LCLS parameters for the other parameters. As the non-gaussian character of the initial distribution and the non-linear part of the initial energy distribution are of crucial importance, the new bunch population capabilities of TraFiC^4 were essential in this study.

The resulting final macro-particle distribution of each run was sorted into longitudinal bins; further post-processing removed the correlated energy spread. The resulting binned distributions are evaluated with respect to FEL figures of merit (namely, saturation length and saturation powers). The results show a gain in FEL performance with decreasing bunch length; even at 9.6 μm, the last bunch length investigated, we do not reach a break-even point.

The study was run on 512 processors per run on the NERSC facility. As the required longitudinal resolution was very high due to the high compression ratio and low natural energy spread, the number of macro-particles used was 7000, the highest number of macro-particles used so far in any TraFiC^4 run. A synopsis of resulting FEL figures of merit is shown in Figs. 1 and 2.

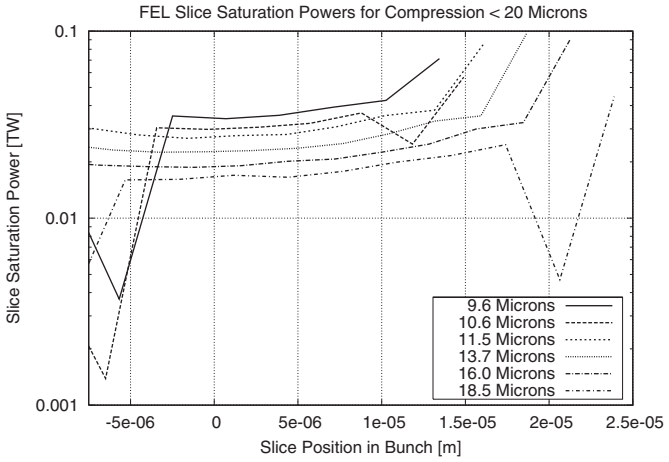

Fig. 1. Slice saturation powers for different bunch lengths.

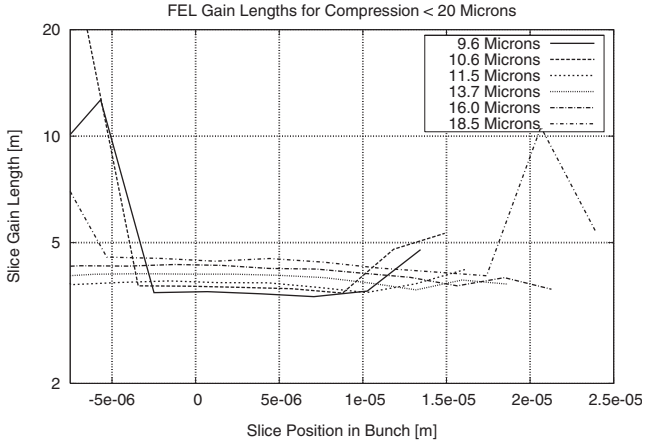

Fig. 2. Slice saturation length for different bunch lengths.

2. The weak–strong beam–beam effect: DUMBBB

The beam–beam interaction plays a crucial role in design and operation of colliding storage rings. It will limit luminosity, determine equilibrium emittance, and can affect beam lifetimes due to diffusion processes. For lack of an effective damping mechanism, the last item is especially important for hadron machines such as the Tevatron or the LHC. We can distinguish two realms of the beam–beam effect: strong–strong and weak–strong. In the former case, both beams' transverse fields affect each other significantly, while in the latter case the beams' charges differ strongly, so one ('strong') beam can be assumed to be unaffected by the other ('weak') beam. If the equilibrium distribution of the strong beam (as determined by the lattice and initial conditions) is known, the problem reduces to a single-particle dynamics problem in the presence of a highly non-linear force.

With current Tevatron operation parameters, there are 72 weak–strong beam–beam interactions affecting the dynamics of the weak (anti-proton) beam. It can be expected that the resulting, highly non-linear one-turn

map affects beam lifetime due to incoherent resonances or diffusion processes. To study these effects, we have developed a C++ code DUMBBB for fast tracking of single particles in the presence of weak–strong beam–beam interactions (for both parasitic and design interactions, i.e. off- or on-center). Being a single-particle code, parallelization reduces to the task of running many instances of the same code acting on different parts of a huge particle ensemble; communications is only required for calculation of collective quantities such as particle loss rates or emittances.

The code models the beam–beam interactions as a synchro-betatron mapping [5], the beam–beam kick itself is calculated from the Bassetti–Erskine [6] formula, using the Chiarella–Matta–Reichel approximation [7] for the evaluation of the complex error function. In a hadron machine, it is important to avoid all sources of numerical noise; the Chiarella algorithm is implemented as a templated C++ function with accuracy selectable at compile time. We find a 10^{-6} relative accuracy sufficient for turn numbers in the 10^5 range.

The weak-beam part of the machine is modeled by a concatenation of beam–beam elements, linear 6×6 transfer maps between non-linear elements (obtained by having a Perl script run MAD8 on a optics description file), a noise-inducing element to model emittance growth due to scattering processes, and a energy-dependent tune advance element to introduce total ring chromaticity.

The code allows for full coupling. All element transfer functions are templated with respect to the type of phase-space variables; in particular, they can operate on differential-algebraic quantities. This allows for finding exact solutions for the linear part of the one-turn map at start-up and constructing invariant initial weak and strong 6×6 distributions, matched to measured emittances. Beam–beam elements have specialized functions depending on whether or not the strong beam shows hourglass effect, tilting, or position-depending tilting during an interaction. Also, they are templatized with respect to the number of slices used in the synchro-betatron mapping.

The aggregated lattice is repeatedly applied to real-value phase-space vectors of the initial weak distribution, which can be "de-cored" to remove particles from the core of the distribution, which are not expected to contribute to diffusive or resonant particle losses. Care must be taken to keep this operation invariant with respect to the one-turn map. The particles' excursions in action-angle space are recorded; once every few thousand turns, the J_x, J_y space is swept and particles beyond a certain action aperture are counted. This way, we obtain a plot of particle loss vs. time for different assumptions about the limiting aperture of the machine. We typically run 10^{10} particle turns. The resulting dependencies are fitted with respect to τ against $\exp(-t/\tau)$ and $\exp(-\sqrt{t/\tau})$ particle loss behaviors, which are the limiting cases of solutions of the diffusion equation with absorbing boundary conditions for small and large-aperture boundaries. Due to the uncertainties of the diffusion model, the real aperture, and the simplifications in the model, the resulting lifetime should not be viewed as an absolute prediction, but as a figure of merit establishing signatures of the real lifetime of the machine.

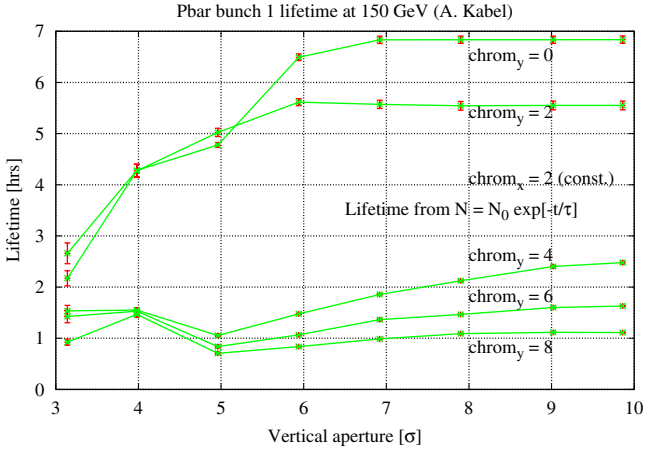

Fig. 3. Lifetime vs. vertical aperture and vertical chromaticity for Tevatron at injection.

We have done a series of parameter studies for the Tevatron at injection (150 GeV, 72 parasitic crossings, modeled as single-slice interactions). A typical result for varying chromaticity is shown in Fig. 3. Other parameter studies included sweeps of helix separations; weak-beam emittances, strong-beam charges, and two different bunch train schemes for 18 bunches on each of the bunch trains, resulting in lifetime differences of factors of two depending on deleting the odd- or even-numbered interactions; the latter result was checked independently with resonance strength studies.

To model non-linear effects due to the lattice, we have implemented a method for high-speed evaluation of multivariate polynomials. The method relies on the recursive definition of P_n^v, a v-variate homogeneous polynomial of degree n, as a direct sum $P_n^v = P_{n-1}^v \oplus P_n^{v-1}$ and a recursive evaluation algorithm $P_n^v(x_1,\ldots,x_v) = x_1 P_{n-1}^v(x_1,\ldots,x_v) + P_n^{v-1}(x_2,\ldots,x_v)$. Using C++'s templated data structure mechanisms for the definition of P and inlining for the definition of the polynomial evaluation, the method effectively generates explicit expressions for Horner's scheme for any order at compile time. The method is easily generalized to inhomogeneous polynomials. We observe floating point efficiencies of >0.85 on Intel hardware and a speed gain of a factor of 4 as compared to standard implementations; still, we would need to gain another factor of 10 in speed to use 10th order polynomial transfer maps between beam–beam interactions.

3. The strong–strong beam–beam effect: NIMZOVICH

In the strong–strong realm, the colliding bunches influence each other substantially. Little is known analy-

tically about the resulting equilibrium distributions. Numerical methods used to determine them have converged on using PIC methods, modeling the collision process as a series of synchro-betatron mappings of test particles in the presence of two-dimensional field distributions (Ref. [8] and references therein). We have developed a code using this principle adapted to high longitudinal resolution, extreme aspect ratios, and the presence of parasitic crossings resulting in multi-bunch effects.

3.1. 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 *Bunches*. 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:

(1) 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.
(2) 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 $N_{\text{Slabs}}N_{\text{Slices}}$ 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.

3.2. Field calculation

Point charges are deposited on a cartesian grid with typical dimensions of $N_x = 64 \ldots 512 \otimes N_y = 64 \ldots 512$, 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 $G(r) = 1/4\pi \log r^2$. 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 [9] package. Free boundary conditions are implemented by using the Hockney trick [10] 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

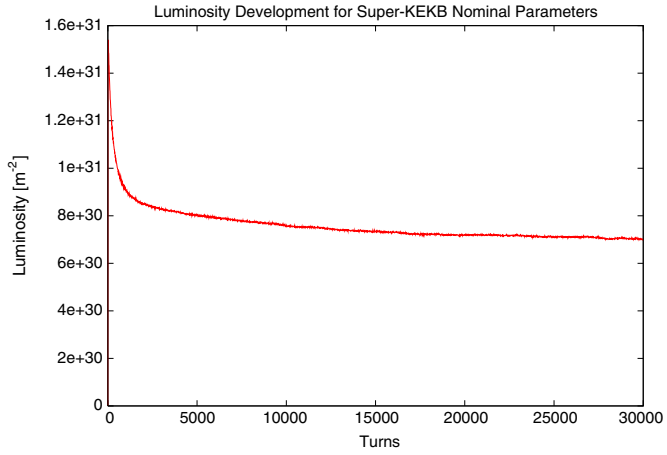

Fig. 4. Luminosity vs. time for NIMZOVICH SuperKEKB benchmark example.

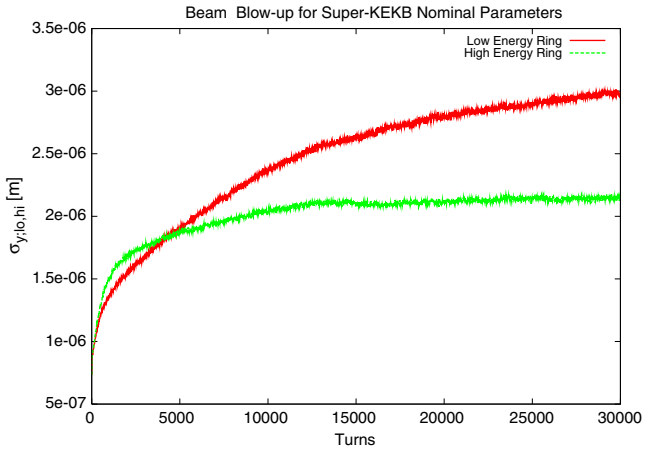

Fig. 5. Beam size vs. time for NIMZOVICH SuperKEKB benchmark example.

$2N_x$ 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.

3.3. Slice-to-slice interaction and adaptive slices

The longitudinal domain decomposition makes use of Hirata slicing [5]. For field calculation, we make use of a convex interpolation method proposed by Ohmi [11] to avoid field discontinuities close to slice boundaries (at the cost of having to do two field calculations for each timestop). We use the same scheme to calculate the luminosity with high accuracy by sampling the opposing bunch's charge density with the macroparticles.

Using this scheme, each slice will execute grid operations (sampling fields or depositing particles) on four 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.

3.4. Results

We present a typical result, a single-bunch luminosity simulation for Super-KEKB with parameters as given by Ohmi et al. [8]. We observe good agreement with the results in Ref. [8], obtained by other codes (Figs. 4 and 5).

References

[1] TESLA Technical Design Report, 2001.
[2] Linac Coherent Light Source (LCLS) conceptual design report, SLAC-R-593.
[3] H.H. Braun, et al., Phys. Rev. ST Accel. Beams 3 (2000) 124402.
[4] A.C. Kabel, P. Emma, Peak current optimization for LCLS Bunch Compressor 2. Proceedings of the 9th European Particle Accelerator Conference (EPAC 2004).
[5] K. Hirata, H. Moshammer, F. Ruggiero, Part. Accel. 40 (1993) 205.
[6] M. Bassetti, G. Erskine, CERN ISR TH/80-06, 1980.
[7] F. Matta, A. Reichel, Math. Comp. 25 (1971) 339.
[8] K. Ohmi, M. Tawada, Y. Cai, S. Kamada, K. Oide, J. Qiang, Phys. Rev. Lett. 92 (2004) 214801-1.
[9] M. Frigo, S.G. Johnson, FFTW 2.15 User's Manual, ⟨http://www.fftw.org/fftw2_doc/⟩
[10] R.W. Hockney, J.W. Eastwood, Computer Simulation Using Particles, Bristol and Philadelphia, 1988.
[11] K. Ohmi, in: Proceedings of the 2003 IEEE Particle Accelerator Conference.