12th Int. Particle Acc. Conf. IPAC2021, Campinas, SP, Brazil JACoW Publishing
ISBN: 978-3-95450-214-1 ISSN: 2673-5490 doi:10.18429/JACoW-IPAC2021-WEPAB234

SIMULATING TWO DIMENSIONAL COHERENT SYNCHROTRON RADIATION IN PYTHON

W. Lou*, C. Mayes, Y. Cai, G. White
SLAC National Accelerator Laboratory, Menlo Park, CA, USA

Abstract

Coherent Synchrotron Radiation (CSR) in bending magnets poses an important limit for electron beams to reach high brightness in novel accelerators. While the longitudinal wakefield has been well studied in one-dimensional CSR theory and implemented in various simulation codes, transverse wakefields have received less attention. Following the recently developed two and three-dimensional CSR theory, we developed a Python code simulating the 2D and 3D steady-state CSR effects. The computed CSR wakes have been benchmarked with theory and other simulation codes. To speed up computation speed, the code applies vectorization, parallel processing, and Numba in Python.

INTRODUCTION

When an electron traverses a curved trajectory, synchrotron radiation is emitted and can give energy kicks to the other electrons in the same bunch. The low frequency components of the radiation spectrum, with wavelength on the order of the bunch length, can add coherently, and is termed coherent synchrotron radiation (CSR). CSR can cause undesired effects including increase in energy spread and beam emittance, energy loss, and potential micro-bunching instability. For a beam with high bunch charge Q and short bunch length σ_z, the effects are more severe. Since the current FACET-II chicane design at SLAC calls for extreme longitudinal beam compression ($Q \sim 2$ nC, $\sigma_z \sim 0.5\,\mu$m at the final bend), CSR effects are greatly concerned [1].

For fast computation, many simulation codes include only the one-dimensional CSR model while neglecting the 2D/3D CSR effects. The negligence is considered acceptable if the transverse beamsizes $\sigma_{x,y}$ satisfy $\sigma_{x,y} \ll \sigma_z^{2/3}\rho^{1/3}$, where ρ is the bending radius [2]. However in the middle of the FACET-II chicane compressor, this limit is not satisfied, and inclusion of the 2D/3D effects might be necessary. The steady-state (s-s) 2D/3D CSR theory has been recently developed [3], but fast numerical calculations had not been implemented. This paper shows how we developed code in Python to efficiently compute the 2D/3D s-s CSR wakes based on the theory. The benchmarking results with other CSR codes and tracking results with the FACET-II chicane are also presented here.

FROM THEORY TO CODE: NUMERICAL CONVOLUTION

For simplicity we will focus on the 2D code development first, then generalize the implementation to 3D code. The s-s longitudinal and horizontal CSR wakes solved in [3] are given as:

$$W_s(z,x) = \iint \psi_s\left(\frac{z-z'}{2\rho}, x-x'\right)\frac{\partial\lambda_b(z',x')}{\partial z'}dz'dx', \quad (1)$$

$$W_x(z,x) = \iint \psi_x\left(\frac{z-z'}{2\rho}, x-x'\right)\frac{\partial\lambda_b(z',x')}{\partial z'}dz'dx', \quad (2)$$

in which λ_b is the bunch distribution, and ψ_s and ψ_x are the longitudinal and horizontal *potential functions* solved in terms of the relativistic γ and bending radius ρ. To efficiently evaluate the double integrals, the trick is to apply 2D numerical convolution. Fortunately there are many convolutional methods available in Python. With a regularly-spaced grid, the FFT convolution method can be applied, and the codes are naturally parallelizable to allow further speed-up.

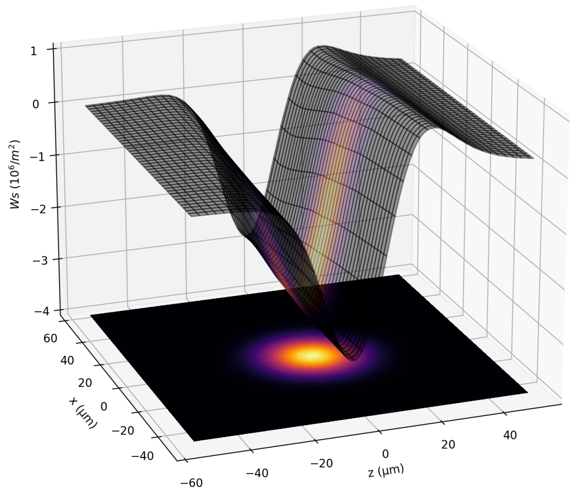

Figure 1: A typical contour plot of the longitudinal steady-state wake $W_s(z,x)$ of a Gaussian bunch distribution.

The process of computing the wake is described as follows. First, we define a 2D grid which captures all (or majority) of the bunch particles in the (z,x) space. The step sizes Δz and Δx are carefully chosen so that the potential functions are well resolved while the number of particles per grid remains reasonably large. We then apply the cloud-in-cell (CIC) charge distribution method to compute the λ_b grid and its derivative in z. Savgol filter is applied to smooth out the distribution if necessary. The ψ_s and ψ_x potential grids are computed on the *double-sized* grid as required by convolution algorithm. For instance, if the λ_b grid is of size $(N_z, N_x) = (150, 100)$, then the ψ_s grid is of size (300, 200) with the same step sizes. We then apply 2D convolution to obtain the W_s and W_x wake grids. If the computed wake grids have significant numerical noise, they are re-computed

* wlou@stanford.edu

with a newly defined grid. Figures 1 and 2 show respectively W_s and W_x wake grids of a Gaussian bunch distribution. The on-axis longitudinal wake ($W_s(x = 0)$) agrees with the 1D CSR theory. Finally, to calculate the CSR energy and horizontal momentum kicks received by the particles ($d\delta/ds$ and dx'/ds), numerical interpolation is applied using the two wake grids.

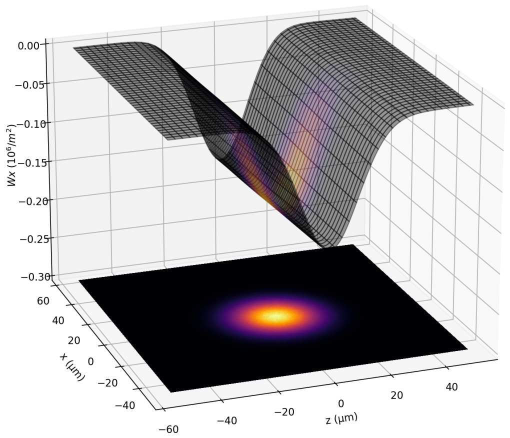

Figure 2: A typical contour plot of the horizontal steady-state wake $W_x(z, x)$ of a Gaussian bunch distribution.

NUMERICAL SPEED-UP

The main objective of our 2D CSR code is to compute the wakes *efficiently*. Besides 2D convolution, there are three additional methods we apply to further reduce the computation time. The first method is to apply *vectorization* instead of *loops* when computing the potential grids. This means that applying a function over an array of elements together is generally faster than applying the function on each element individually. For instance, to compute a 2D Gaussian distribution grid, vectorization provides ~ 500× speed-up for various grid sizes. The second method is to apply parallel computation with multiple CPUs. This can be achieved by in Python via the *ProcessPoolExecutor* module. Figure 3 shows the reduction of computation time for the ψ_s grid as the number of CPUs used increases.

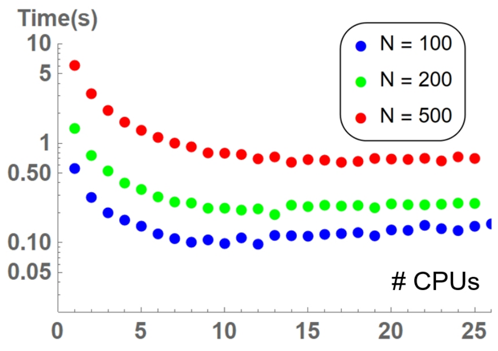

Figure 3: Timings to compute the ψ_s grid of size 2N-by-2N.

The third method is to apply Numba decorators on the applicable Python functions [4]. The decorated functions are first compiled to machine code, which allows the subsequent calls to run at native machine code speed. Simply for the λ_b grid computation, Numba provides ~ 700× speed-up. With all methods combined, the computation time is significantly reduced. Figure 4 shows the latest timings required for the four main steps to compute the CSR kicks on a million particles for different gird sizes N-by-N. The characteristic timing has been reduced to be approximately one second.

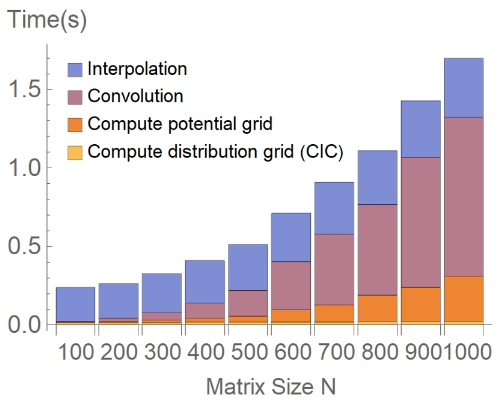

Figure 4: Timings for the four main steps to compute the W_s kicks on a million particles with a 2D grid size of N-by-N. Vectorization and Numba compiler are applied with 20 CPUs on Cori NERSC.

BENCHMARKING WITH ONE BEND

The developed 2D CSR code in Python has been named *CSR2D*. We have benchmarked CSR2D on one bending magnet with other CSR simulation codes, including Bmad (1D CSR), Elegant (1D CSR), and Lucretia MATLAB (2D CSR) [5–7]. For consistency we used the same lattice and Gaussian beam parameters as in [2], which are: $\gamma = 9804$, $\rho = 10.34$ m, $Q/e = 6.25 \times 10^9$, $\sigma_z = 20.0\,\mu$m, $\sigma_x = 23.1\,\mu$m, $L_{\text{bend}} = 0.5$ m. The initial beam has $N_p = 10^6$ macro-particles. Note that we assume s-s wake inside the entire bend. Figure 5 shows the agreement between the projected horizontal beam emittances as the bunch traverses the bending magnet. The term *energy kick only* means that only the longitudinal kick W_s from the 2D theory is applied. Note that this is physically different from 1D CSR since the off-axis wakes $W_s(x \neq 0)$ are different, but the effects can be similar as shown. The longitudinal phase space at different tracking steps have also been checked (not shown here), and agreements between the codes were observed. Similar to Fig. 5, Fig. 6 shows the results from Python and Lucretia with both longitudinal and horizontal kicks turned on. Overall the 1D and 2D Python results agree with other codes well.

SLAC FACET-II CHICANE TRACKING

To find out the effects from transverse CSR wakes, We applied CSR2D on the FACET-II chicane design lattice. The lattice and beam parameters are: $E = 30$ GeV, $\rho = 1538$ m,

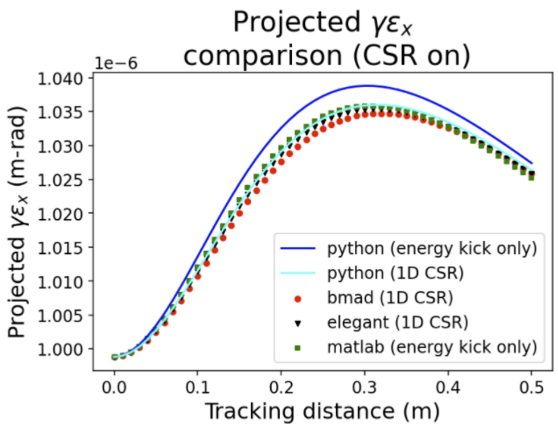

Figure 5: Projected horizontal beam emittance of a Gaussian bunch traversing a bend with steady-state CSR wake applied from different codes. With "energy kick only", only the longitudinal kicks from the 2D CSR theory are applied.

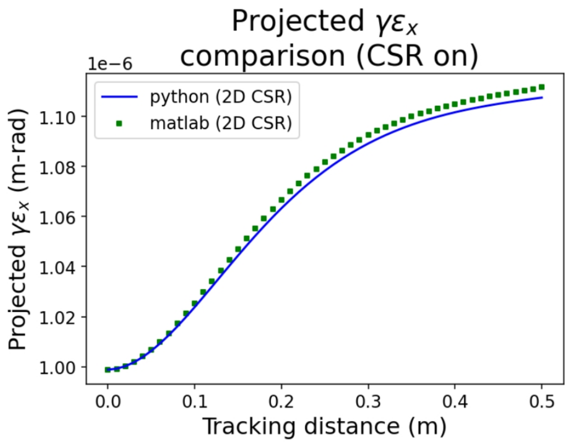

Figure 6: Projected horizontal beam emittance of a Gaussian bunch traversing a bend with 2D steady-state CSR wake applied from Python and Lucretia MATLAB.

$Q = 2$ nC, $\sigma_z = 40\,\mu$m, $\sigma_x = 134\,\mu$m, $L_{\text{bend}} = 20$ m, and $L_{\text{drift}} = 32.5$ m. The initial beam is a Gaussian with energy chirp and $N_p = 10^6$ macro-particles. Figure 7 shows the projected horizontal emittances in a log scale as the bunch traverses the chicane with different CSR settings. As expected, the final bending magnet contributes the most emittance growth since the CSR effects scale up with a compressed bunch length. Note that the horizontal CSR wake can contribute to additional increase in the beam emittance. Figure 8 shows the longitudinal and horizontal phase space plots at the end of chicane tracking, with either "energy kick only" or "both kicks on". The additional horizontal CSR wake results in significant differences.

CODE APPLICABILITY AND FUTURE STEPS

The existing s-s CSR2D code has been extended to *CSR3D* in Python, and further incorporated into Bmad via the OpenCSR package. All the codes are open-source on GitHub, and are documented with examples [8–10]. As discussed, the codes currently work with multiple CPUs on a single node, and can be extended to multiple nodes with mpi4py or Dask [11]. For potential further speed-up, parts of the code currently work with GPU via the Cupy library [12]. The next important future step for CSR2D is to include the *transient wake* computation for more physically valid tracking results. This involves computing Eq. (1) and Eq. (2) with finite boundary conditions, and is currently in progress.

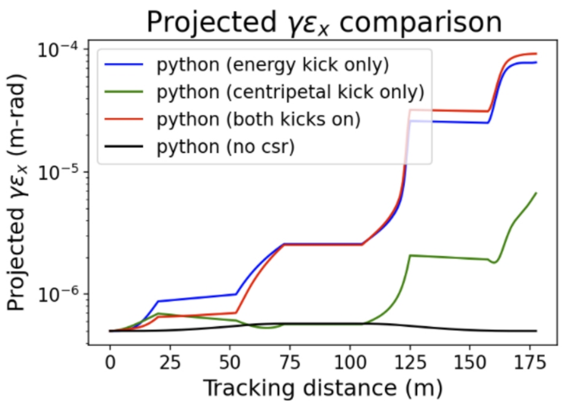

Figure 7: Projected horizontal beam emittance of a Gaussian bunch traversing the FACET-II chicane with 2D steady-state CSR wake applied from Python CSR2D.

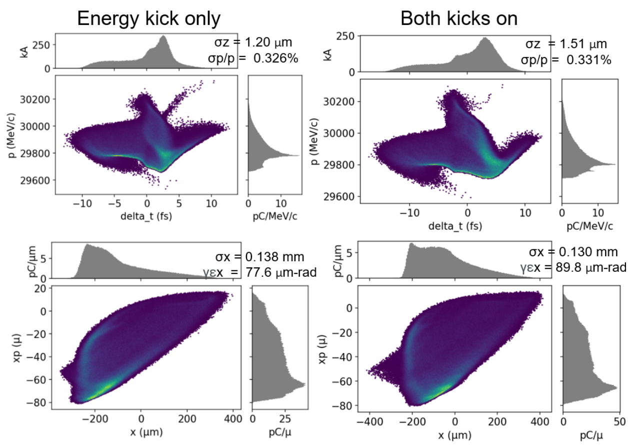

Figure 8: Longitudinal (top) and horizontal phase space (bottom) plots at the end of chicane tracking with either energy kick only (left) or both kicks on (right).

CONCLUSION

We have developed Python codes called CSR2D and CSR3D to efficiently compute the 2D/3D steady-state CSR wakes. Convolution, vectorization, multi-processing, and Numba compiler have been applied to significantly speed up the computation. The codes have been benchmarked with other CSR simulation codes for a single bend, and tracking results for the FACET-II chicane show that transverse wakes can have significant effects on the beam.

ISBN: 978-3-95450-214-1 ISSN: 2673-5490 doi:10.18429/JACoW-IPAC2021-WEPAB234

REFERENCES

[1] V. Yakimenko *et al.*, "FACET-II facility for advanced accelerator experimental tests", *Phys. Rev. ST Accel. Beams*, vol. 22, p. 101301, 2019. doi:10.1103/PhysRevAccelBeams.22.101301

[2] S. Derbenev *et al.*, "Microbunch radiative tail-head interaction", DESY, Hamburg, Germany, Rep. DESY-TESLA-FEL-95-05, Sep. 1995.

[3] Y. Cai and Y. Ding, "Three-dimensional effects of coherent synchrotron radiation by electrons in a bunch compressor", *Phys. Rev. ST Accel. Beams*, vol. 23, p. 014402, 2020. doi:10.1103/PhysRevAccelBeams.23.014402

[4] Numba, http://numba.pydata.org/.

[5] D. Sagan, "Bmad: A relativistic charged particle simulation library", *Nucl. Instrum. Meth. A*, vol. 558, pp. 356–359, 2006. doi:10.1016/j.nima.2005.11.001

[6] M. Borland, "Elegant: A Flexible SDDS-Compliant Code for Accelerator Simulation", Argonne National Lab, IL, USA, Rep. LS-287, Aug. 2000.

[7] P. Tenenbaum, "Lucretia: A Matlab-Based Toolbox for the Modeling and Simulation of Single-Pass Electron Beam Transport Systems", in *Proc. 21st Particle Accelerator Conf. (PAC'05)*, Knoxville, TN, USA, May 2005, paper FPAT086.

[8] CSR2D, https://github.com/weiyuanlou/CSR2D

[9] CSR3D, https://github.com/ChristopherMayes/CSR3D

[10] OpenCSR, https://github.com/ChristopherMayes/OpenCSR

[11] Dask: Scalable analytics in Python, https://dask.org/.

[12] Cupy: GPU accelerated computing with Python, https://cupy.dev/.