Simulation of Longitudinal Beam Dynamics Problems in Synchrotrons
Lecture 2

Numerical Challenges & Limitations of Longitudinal Beam Dynamics

Helga Timko

CERN, BE-RF

Inverted CERN School of Computing, 23-24 February 2015

Contents

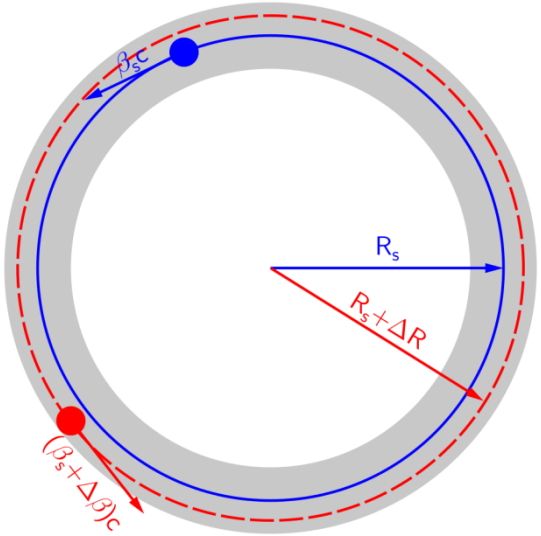
- **Setting the scene**
- **Kick & drift**
  - Approximations done and validity range
- **Collective effects**
  - Pros and cons of frequency vs time domain
  - Challenges of multi-bunch modelling
- **Multi-bunch modelling**
  - In view of collective effects
- **Performance optimisation**
  - Typical bottlenecks

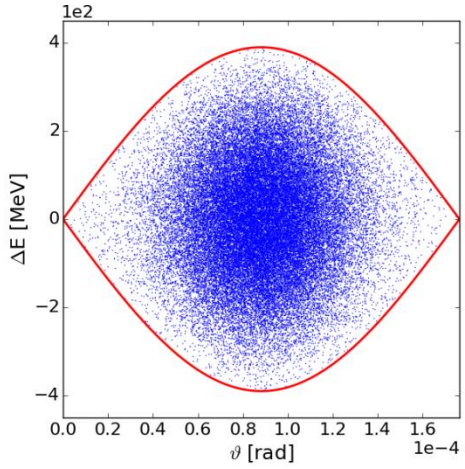
Coordinate system and code structure

SETTING THE SCENE

Synchrotron model

- **Bunches are described in $(\varphi, \Delta E)$ phase space**
  - φ = RF phase when the particle arrives to the cavity
  - ΔE = energy offset from the synchronous energy
- **Centre of coordinate system is always defined by the synchronous particle**
  - Non-inertial system
  - Possible change of system each turn (time step)





Building blocks of numerical code

- **Bunch coordinates (2D)**
- **Tracker:**
  - Energy kick: RF kick from cavities and magnetic ramp
  - Drift: phase slippage during one turn
- **Collective effects**
  - Impedances and/or wake fields
  - Slicing of the bunch
- **Feedback loops, frequency and phase corrections**
- **I/O, statistics, and plotting**

Approximations in the Equations of Motion

KICK & DRIFT

Energy kick

- **Natural choice: 1 turn = 1 time step, since also measurements are restricted to a certain spot in the machine**
- **In general, the energy change from one turn to another (or one station to another) for K RF systems with harmonics h_k is**

$$\Delta E^{n+1} = \Delta E^{n} + \sum_{k=1}^{K} eV_k{}^{n} \sin\left[\frac{h_k{}^{n}}{h_0{}^{n}}\left(\varphi_k{}^{n} - \phi_{\text{offset}}{}^{n}\right)\right] - \left(E_s{}^{n+1} - E_s{}^{n}\right)$$

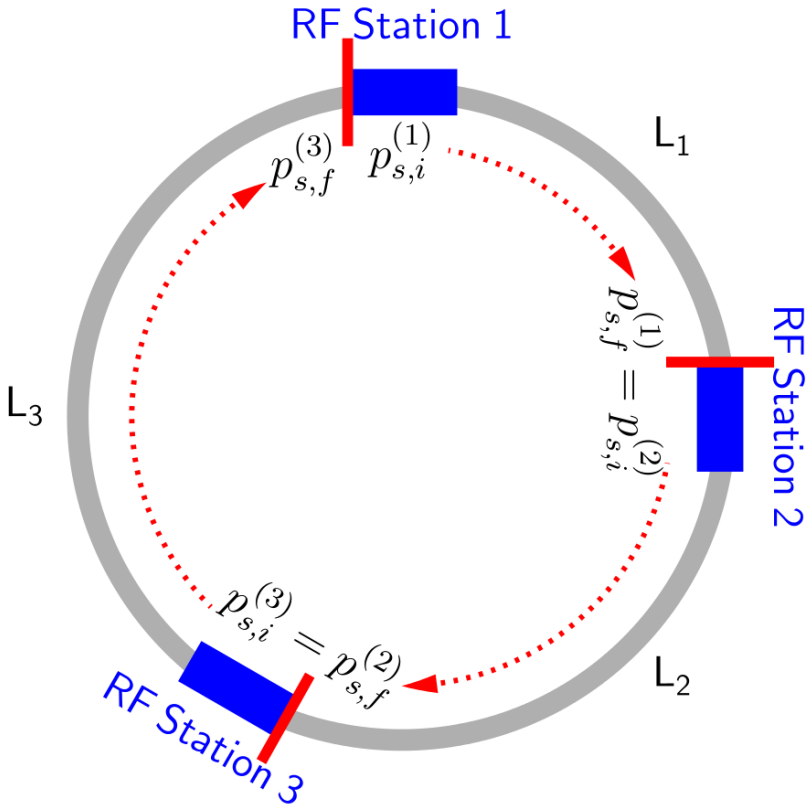
RF acceleration

$\propto$ Magnetic ramp

- **This equation is (formally) exact. And for CERN machines, it is sufficient to consider a single RF station, i.e. to approximate different RF systems as placed in the same spot.**
- **In general, if E_s changes significantly from one station to another, the time step has to be 'sub-cycled'**

Sub-cycling of time step

- **Kick and drift done for each RF station separately**
- **Need to know the synchronous energy (momentum) at each station for each turn**
- **N.B. complications: there is different impedance in different places of the ring ⇒ calculations of collective effects need to be split down as well**



Phase drift

- **The corresponding phase drift from one station to another (defined w.r.t. to the 'main' harmonic):**

$$\Delta\varphi^{n+1} = \frac{{f_{RF}}^{n+1}}{{f_{RF}}^{n}}\Delta\varphi^{n} + 2\pi h\frac{L_i}{C}\left(\frac{1}{1-\eta(\delta^{n+1})\delta^{n+1}} - 1\right)$$

- **Formally exact, but in practise we need to approximate** $-\eta(\delta)\delta \approx -(\eta_0 + \eta_1\delta + \eta_2\delta^2 + \cdots)\delta$
  - η_i are machine-dependent constants; calculated from the magnetic 'lattice' properties
  - In CERN machines $\delta = \sigma(10^{-3} - 10^{-4})$, so $\eta(\delta)\delta \approx \eta_0\delta$ is typically good enough
    - To obtain the same accuracy with $\delta = \sigma(10^{-2})$, next term needed, too

- **N.B. drift becomes much more complicated with LLRF loops modelled, as they add corrections (usually from the previous turn) to phase and frequency, while the coordinate system keeps changing…**

Hidden approximation in the time step

- What we know about the synchronous particle is that it passes the cavity with energy $E_s{}^n$ at moment t^n and returns to the same spot with energy $E_s{}^{n+1}$ at moment t^{n+1}
- What we don't know is how exactly the energy changed meanwhile $E_s(t)$, and thus, how much time one turn actually took $\Delta t = \int_{E_s{}^n}^{E_s{}^{n+1}} \frac{dE_s(t)}{\dot{E}_s(t)}$
- But the EOMs use phase and energy offsets, so why do we care about time? In fact, the time-phase conversion is hidden in the drift equation through the frequencies.
- Since acceleration is adiabatic, a linear approximation is fine. But what to use: $\dot{E}_s(t) \approx \frac{E_s{}^{n+1} - E_s{}^n}{T_0{}^n}$, $\frac{E_s{}^{n+1} - E_s{}^n}{T_0{}^{n+1}}$, or $2\frac{E_s{}^{n+1} - E_s{}^n}{T_0{}^n + T_0{}^{n+1}}$?

Interaction of the beam and its surroundings

COLLECTIVE EFFECTS

Recall the physics

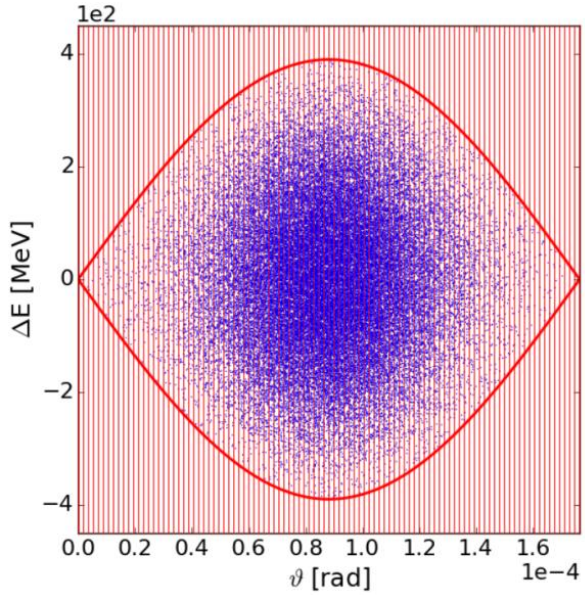
- **The energy kick of particle k (arriving at time t) depends on the voltage induced by the particles i in front of it (arriving at time $\tau < t$)**
  - In ultra-relativistic limit, the particle also sees half of the voltage induced by itself ($W(0) = \frac{1}{2} \lim_{t \to 0^+} W(t)$)

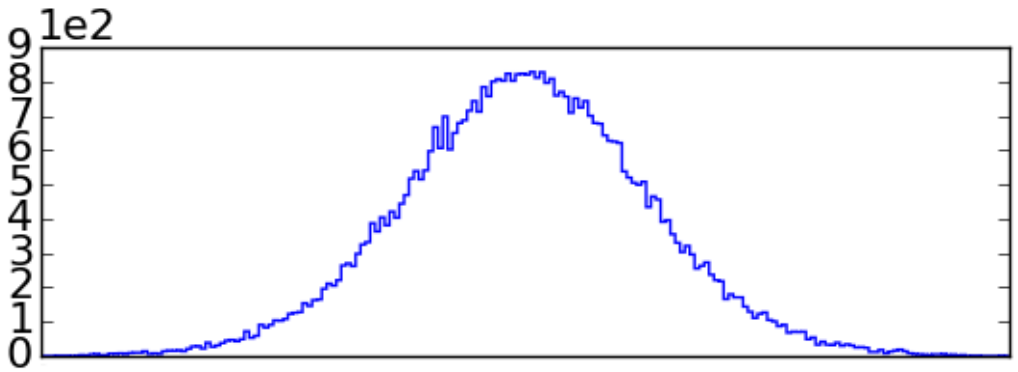
$$\Delta E(t) \propto \int_{-\infty}^{t} d\tau \, \lambda(\tau) W(t - \tau)$$

- **Or in a discretised way:**

$$\Delta E_k \propto \sum_{i=1}^{k} \lambda_i W_i$$

Bunch ‘slicing’





- **But what resolution to take?**
- **N.B. histogram also useful for statistics: in a real machine, often only the bunch profile, i.e. histogram, can be measured**
  - When we compare bunch lengths, same fit should be applied!

Induced voltage in time domain

- **The convolution calculation becomes quickly expensive as it scales as $\propto \frac{N_{sl}(N_{sl}-1)}{2}$ (N_{sl} = no. of slices)**
- **In addition, the knowledge we have about the machine impedance is also limited to a certain frequency $f_{\max} \Rightarrow$ makes no sense to calculate more precise than $\Delta t \sim \frac{1}{2 f_{\max}}$**
- **If we are not interested in phenomena above a given frequency $f < f_{\max}$, we can reduce the slice resolution further $\Delta t \sim \frac{1}{2f}$**
- **This is just a rule of thumb, the actual resolution should be determined via convergence studies '$N_{sl} \to \infty$', '$N_p \to \infty$'**
  - Careful: too many slices can introduce numerical artefacts (high-frequency bunch oscillations) $\Rightarrow$ divergence!

Induced voltage in frequency domain

- **The 'expensive' convolution calculation can be avoided by going to frequency domain**
  - Multiplication: $\propto N_{sl}$, 2 FFTs: $\propto 2N_{sl} \log_2 N_{sl}$
- **The energy kick in frequency domain can now be calculated through a simple multiplication between impedance $Z_n(f)$ and bunch spectrum $S_n(f)$:**

$$\Delta E_n(f) \propto S_n(f) Z_n(f)$$

  - Necessary frequency resolution determined by sharpest peak in $Z_n(f)$
- **But nothing comes for free; every turn, two FFTs have to be performed:**
  - Bunch profile to spectrum: $S_n(f) \propto \sum_k \lambda_k(t)\, e^{-2\pi i f_n t_k}$
  - Energy kick in time domain: $\Delta E_k(t) \propto \sum_n \Delta E_n(f) e^{+2\pi i f_n t_k}$

Pros & cons of f domain calculation

- **Multiplication faster than convolution (details in next slide)**
- **But using the forward and backward FFT, we perform actually a circular convolution**
  - $\Rightarrow$ need to take care that the induced voltage (energy kick) is properly decayed behind the bunch, otherwise we can get an unphysical induced voltage in front of the bunch
- **There is no 'ultimate recipe' how to circumvent this**
  - Zero padding of the impedance works effectively, but makes the FFTs very expensive
    - Computation speed comparable if not worse than in time domain
  - Applying pass-band filters on the induced voltage is more effective in terms of CPU time
    - But suitable filter depends on physics problem!

Runtime comparison

Frequency domain

- $\propto N_{sl}\,(1 + 2\log_2 N_{sl})$
- **Single bunch, 100 slices: ~1400 operations**
- **Multi-bunch, 10^6 slices: ~4×10^7 operations**

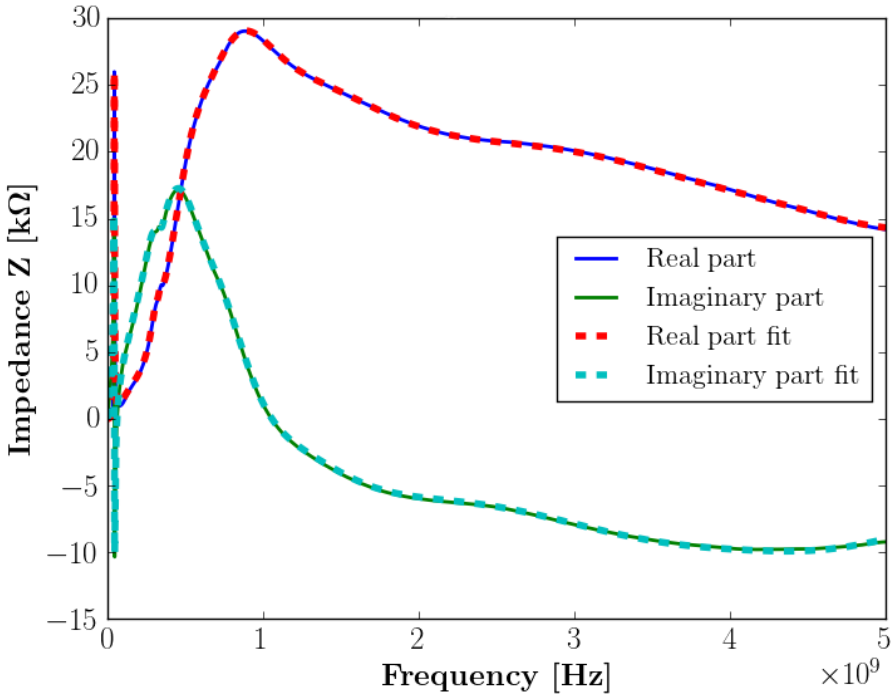
Time domain

- $\propto \frac{N_{sl}(N_{sl}-1)}{2}$
- **Single bunch, 100 slices: ~5000 operations**
- **Multi-bunch, 10^6 slices: ~5×10^{11} operations**

Careful with FFTs

- **Example: the SPS kicker impedance is known until 5 GHz, where the impedance is not decayed**
  - 'Brute force' FFT will not work due to the step function at 5 GHz
- **Alternative: model as a sum of broad-band resonators**
  - Can be calculated analytically
  - Decays nicely in infinity

SPS kicker impedance fit with 11 resonators



Interlude: broad-band resonator

- **Similar to an RLC-circuit**

$$\frac{1}{Z} = \frac{1}{R} + \frac{i}{\omega L} - i\omega C$$

- **A broad-band resonator impedance is modelled as**

$$Z = \frac{R}{1 + iQ\left(\frac{\omega_r}{\omega} - \frac{\omega}{\omega_r}\right)}$$

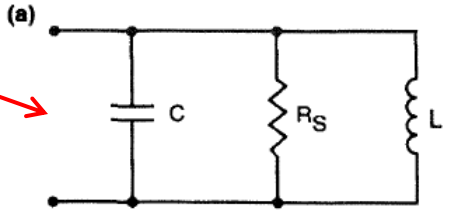
  - Quality factor $Q = R\sqrt{C/L}$
  - Resonant frequency $\omega_r = 1/\sqrt{CL}$

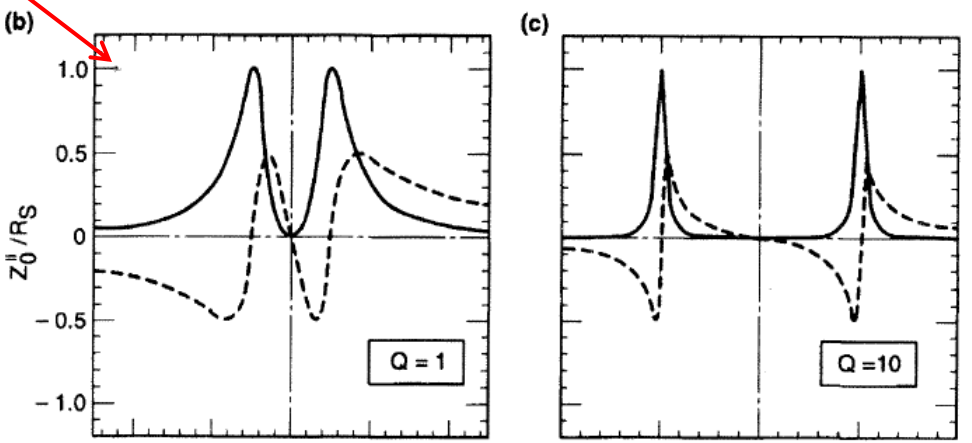
- **Corresponding wake field:**

$$W(t \geq 0) = \begin{cases} \alpha R, \text{ if } z = 0 \\ 2\alpha R e^{-\alpha t}\left(\cos\tilde{\omega}t - \frac{\alpha}{\tilde{\omega}}\sin\tilde{\omega}t\right), \text{ if } t > 0 \end{cases}$$

with $\alpha = \omega_r/(2Q)$, $\tilde{\omega} = \sqrt{{\omega_r}^2 - \alpha^2}$

Broad-band resonator impedance with different widths (Q-factors)

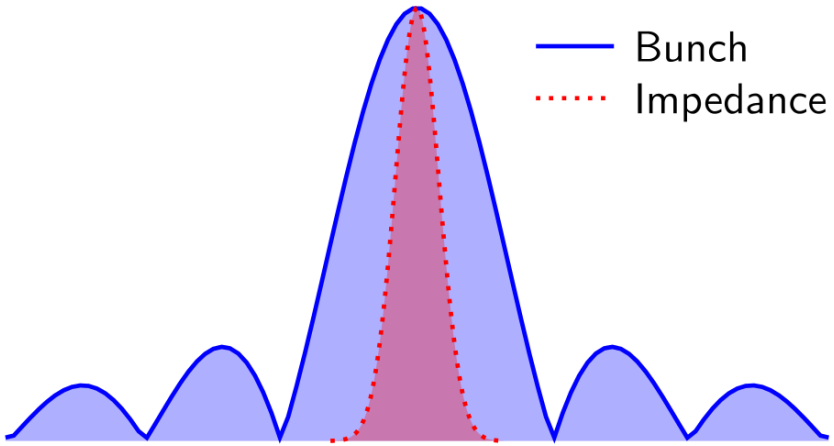




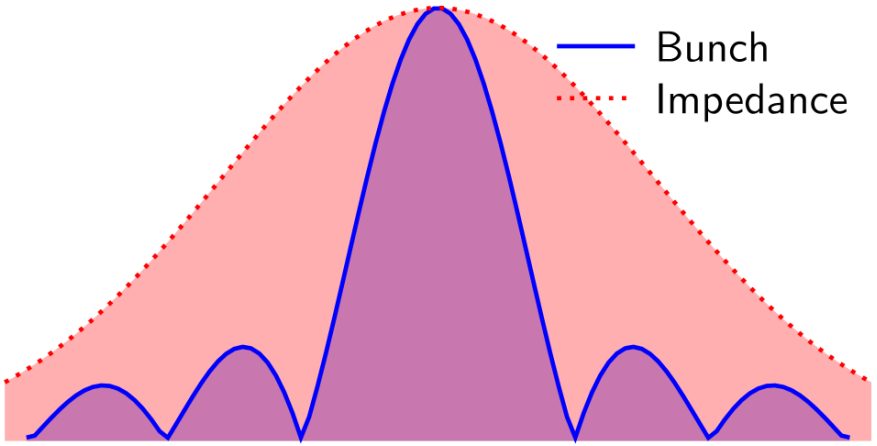
Solid line: real part
Dashed: imaginary part

Numerical simulation of instabilities

- **No general recipe, but what you should ask yourself is**
  - What is the machine impedance? What is the bunch spectrum? What type of coupling do we expect?
  - Bunches ‘sample’ the impedance:



‘Short’ bunch, narrow-band impedance
The integral of the impedance (R/Q) dominates



‘Long’ bunch, broad-band impedance
The peak of the impedance (R) dominates

Issues with space charge

- **Calculating space charge (SC) effects is a field by itself**
- **Space charge is usually modelled via impedance. The energy kick received by a particle due to space charge is**

$$\Delta E \propto \frac{Z}{n}\frac{d\lambda}{dt}, \text{ where } n = \frac{f}{f_0}$$

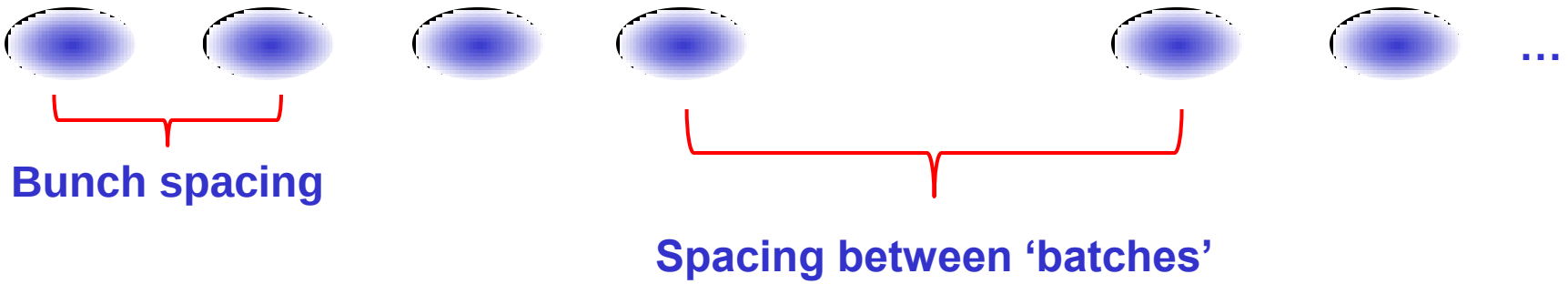
- **Analytic or simulation models are used to determine $\frac{Z}{n}$, which depends on beam pipe geometry and transverse beam size**
  - Ideally, the exact machine layout should be taken into account!
- **The longitudinal derivative of the line density $\frac{d\lambda}{dt}$ poses numerical challenges: noisy even using millions of particles**
  - Numerical artefact: high-frequency components
  - Smoothing of λ using pass-band filters can help

Another layer of complexity

MULTI-BUNCH MODELLING

Multi-bunch modelling

- **In reality, we can have many bunches along the machine, with different spacing between the bunches**



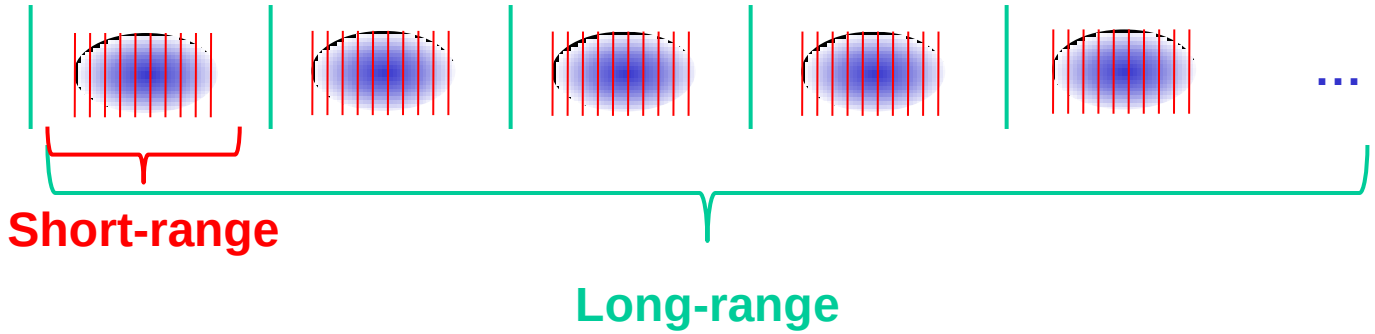
- **From the tracker point of view, nothing changes; the single bunch case is trivially extended to multiple bunches. Particles are treated independently, so parallelisation is trivial, too.**

Complication: coupled-bunch effects

- **Collective effects, however, couple multiple bunches and sometimes even multiple turns (own wake from past turns)**
- **Calculation of induced voltage quickly becomes computationally heavy and parallelisation non-trivial**
  - Need to reduce no. of slices as much as possible
- **Again, no general recipe ☹. Physics should lead us!**
- **E.g. in a machine like LHC, where there is just one RF system and the bunch train has a simple pattern (1 full + 4 empty buckets), we can avoid slicing empty buckets**
- **But in general a 'bucket' is not always well-defined…**
  - Can you think of an example?

A possible physics solution

- **Separate long-range and short-range wakes in the problem**
  - If you're lucky, you can ☺
  - But if the long-range interaction is due to a high-frequency impedance, you'll still need high resolution
- **Treat short-range wakes with finer, and long-range wakes with rougher slicing**
- **Then the induced voltage calculation can be done in parallel on different groups of bunches**



Bottlenecks of longitudinal beam dynamics codes

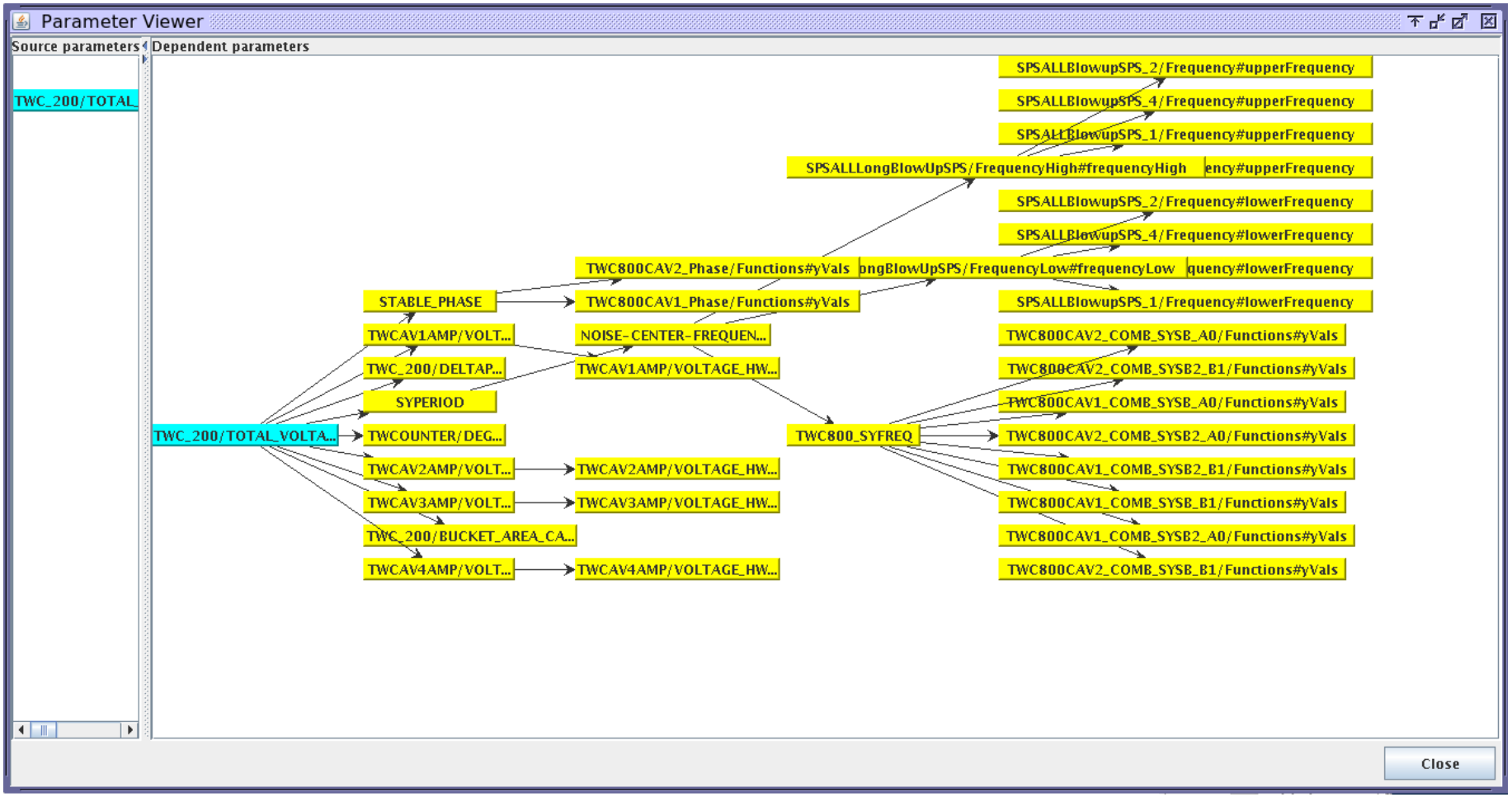
PERFORMANCE OPTIMISATION

Code architecture

- **Modular architecture required to be able to versatile simulations according to our needs**
- **BLonD: python structure with C++ routines**
- **Main building blocks like tracker and collective effects are well defined**
- **But apart from these main building blocks, many other features might be required, depending on the task (machine to be modelled)**
- **Thus, in general, parameter hierarchy can become very complex with all LLRF loops, statistics, etc.**
  - Imagine a code that could simulate all CERN synchrotrons ☺

Real life parameter dependencies…

- **SPS parameters that depend on the 200 MHz voltage**



Bottlenecks

- **Main limitation: runtime**
  - Already the simulation of LHC ramp with single bunch (~8.7 million turns) can easily take ~1 week w/o optimisation
- **RAM memory can become a problem for multi-bunch case**
  - Issue is total size (see later)
- **To build a multi-bunch code, we need both**
  - Optimisation
  - Parallelisation
- **What do you think, what are the main performance bottlenecks of a longitudinal beam dynamics code?**

Bottlenecks in runtime

- **Language choice itself has its bottlenecks**
  - E.g. python: faster development, but longer runtimes
- **Tracker has expensive operations**
  - Kick: sinusoidal RF potential
    - Trigonometric functions needed also in many other places
  - Drift: many divisions, expansion in off-momentum
- **Collective effects**
  - Slicing method
  - In general slow both in frequency and time domain
  - Non-trivial parallelisation due to coupling between particles
- **Communication between modules**
  - Overhead can be significant

Python examples: histogram (1)

- **Python built-in functions can be handy but also expensive**
- **E.g. numpy.histogram always does a sorting**
  - For N_p macroparticles and N_{sl} slices the routine scales linear in N_p and N_{sl} for $N_p > 65536$

```
Numpy.histogram: ( a = coordinate array, bins =e dge array, M = len(a),
                   S = len(bins)-1 )

block = 65536
for i in arange(0, len(a), block):
    sa = sort(a[i:i+block])
    n += np.r_[sa.searchsorted(bins[:-1], 'left'), \
        sa.searchsorted(bins[-1], 'right')]

n = np.diff(n)
```

$$\sim 2^{16} \log_2 2^{16} \times \frac{N_p}{2^{16}}$$

$$\sim N_{sl} \log_2 2^{16} \times \frac{N_p}{2^{16}}$$

$$\sim N_{sl}$$

$$\Sigma = 16\, N_p \left(1 + \frac{N_{sl}}{2^{16}}\right) + N_{sl}$$

Python examples: histogram (2)

- **In comparison, the 'hand-made' histogram that uses no sorting is linear in N_p, but independent of N_{sl} (because access time doesn't depend on array length)**

```
cpp_histogram: ( input = array of coordinates, output = array of edges,
                 M = len(input), S = len(output) - 1 )

inv_bin_width = n_slices / (cut_right - cut_left);          ~ c1

    for (i = 0; i < n_macroparticles; i++) {
        a = input[i];                                       ~ c2 × Np
        if ((a < cut_left)||(a > cut_right))                ~ c3 × Np
            continue;                                       ~ c4 × Np
        fbin = (a - cut_left) * inv_bin_width;              ~ c5 × Np
        ffbin = (uint)(fbin);                               ~ c6 × Np
        output[ffbin] = output[ffbin] + 1.0;                ~ c7 × Np
    }
```

$$\Sigma = c_1 + CN_p$$

Python examples: sine function (1)

- **Trigonometric functions are present not just in the tracker, but in any analytic calculation of potential well, separatrix, etc.**
  - Other than the tracker, a discretised solution is more effective
- **In python, the numpy.sin function can be up to 17× slower than the math.sin function**
  - Both use look-up tables
- **Even more optimised: the CERN VDT library**
  - VDT = **v**ectorise**d** ma**t**h for exp, log, sin, cos, …
  - Scalar and double precision
  - Sine function based on Pade polynomials ⇒ vectorisable
- **Runtime-critical parts like tracker should be in C++…**

Python examples: sine function (2)

- **C++ kick using VDT fast_sin()**

```
#include "sin.h"
…
extern "C" void kicks(const double * __restrict__ beam_theta, …){
…
    beam_dE[i] = beam_dE[i] + voltage[j] * fast_sin(harmonic[j] *
                 beam_theta[i] + phi_offset[j]);
…}
```

- **C++ kick embedded in python tracker module**

```
import ctypes, copy, sys, thread
from setup_cpp import libfib
…
def track(self, beam):
    v_kick = np.ascontiguousarray(self.voltage[:, self.counter[0]])
    …
    libfib.kicks(beam.theta.ctypes.data_as(ctypes.c_void_p),…)
```

An example of potential speed-up

- **LHC ramp w/ feedbacks, no collective effects, single bunch**
- **Profiled with $N_p = 50000$, $N_{sl} = 100$, $N_{\mathrm{turns}} = 1000$ on a PC**

numpy.histogram, python tracker

Function/Module	Total Time	Local Time
track	4.510	0.008
slice_constant_space_histo...	3.477	0.006
histogram	3.476	0.048
beam_coordinates	0.001	0.001
gaussian_fit	1.024	0.014
convert_coordinates	0.001	0.001
plot_long_phase_space	3.064	0.001
track	1.877	0.008
kick	1.456	1.455
drift	0.385	0.216
kick_acceleration	0.027	0.027

C++ histogram, C++ tracker

Function/Module	Total Time	Local Time
track	1.191	0.008
gaussian_fit	0.993	0.014
slice_constant_space_histo...	0.188	0.167
data_as	0.048	0.016
__init__	0.026	0.026
beam_coordinates	0.001	0.001
__getattr__	0.000	0.000
convert_coordinates	0.002	0.002
loadtxt	1.113	0.335
track	0.747	0.611
drift	0.082	0.067
data_as	0.048	0.016
__init__	0.026	0.026
ascontiguousarray	0.016	0.006
__getattr__	0.000	0.000

Input/output

- **What we typically want to store:**
  - Bunch coordinates or bunch profiles for post-processing
  - And/or already processed statistics, perhaps even turn-by-turn
- **Binary format preferred to reduce file size. Some options are:**
  - HDF5 data model; HDF = **h**ierarchical **d**ata **f**ormat
    - Portable
    - Parallel
    - But quite a parameters to optimise: buffer size, compression rate, etc.
  - Input/Output module of the CERN Root library
  - …

Data processing

- **To compare with measurements, the simulation data has to be processed in the same way as in 'real life'**
- **This data processing can take significant runtime as well**
  - Online processing often better alternative to storing lots of data
  - Optimisation might be required
- **Some typical data processing done**
  - Statistics: means, averages, emittances (phase-space area)
  - Bunch profile (histogram), bunch spectrum (FFT)
  - Bunch length using various fits to the profile (e.g. Gaussian)
  - 'Simulated measurement' – reproduction of experimental signal processing
  - Plotting

RAM memory footprint

- **A double-precision floating-point number takes 8 bytes**
  - Footprint reduction by reducing data structure size (double → float) ⇒ faster computation as caches are filled faster
- **There are several 'dimensions' in which RAM memory is potentially an issue**
  - RF parameters (voltage, momentum, harmonic…)
    - With up to 10 million turns, and 10-100 arrays ⇒ ~1-10 GB
    - It's not a 'must' to store these (even though preferable e.g. in python)
  - Particle coordinates ($\varphi, \Delta E, \delta$)
    - With up to 1 million particles per bunch and 100 bunches ⇒ ~1 GB
  - Collective effects: induced voltage calculation
    - In case of high-frequency effects, i.e. very accurate slicing, and multiple bunches, also here arrays of 1-10 million entries are possible
- **In general, memory-heavy simulations can be avoided**

Parallelisation needs

- **To summarise what we've said about parallelisation:**
- **Absolutely necessary for multi-bunch simulation**
- **Tracker trivially parallelisable**
  - Vectorised kick and drift using polynomial sine function
- **Parallelisation of collective effects is tricky due to coupling**
  - GPU might be preferred to CPU
- **With optimised tracker and induced voltage calculation, simulation-dependent features will dominate the runtime**
  - LLRF loops, data processing, etc.
- **Task parallelisation important but parameter hierarchy is very complex if we want to build a general code**

Runtime estimate continued…

POTENTIAL OPTIMISATION GAIN

A (very) rough runtime estimate (1)

- **How long would it take to simulate 'just' the LHC full beam with intensity effects?**
- **A reasonable simulation that was done:**
  - Single bunch
  - 50,000 particles, 100 slices
  - Acceleration ramp: 8,700,000 turns (11 minutes real time)
  - Phase loop and noise injection for controlled emittance blow-up
  - No intensity effects
  - Runtime: **3 days on a single CPU**

A (very) rough runtime estimate (2)

- **Intensity effects (might even need much more particles/slices)**

  $\times 2 \rightarrow$ **6 days**

- **Full LHC cycle: injection (1 h) + ramp (11 min.) + physics (8 h)**

  $\times 50 \rightarrow$ **300 days ~ 1 year**

- **In physics: beam-beam interaction, 6D phase space**

  $\times$ 2-10? $\rightarrow$ **2 years**

- **Full beam**

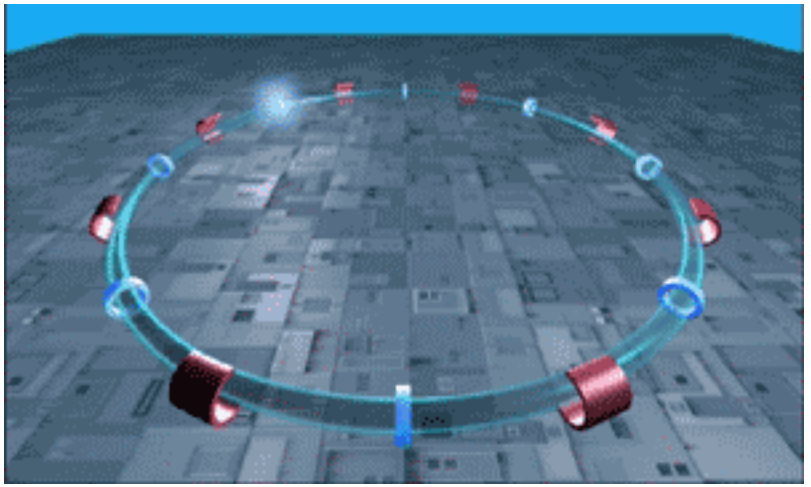
  $\times 2808 \rightarrow$ **5600 years**

- **Coupled bunch effects**

  $\times$ 2-10? $\rightarrow$ **>10000 years ($\gg$ LHC lifetime!)**

A (very) rough runtime estimate (3)

- **Even if we'd have 10,000 processors and our code would perfectly scale, the above simulation would still take 1 year**
  - So modelling such complexity of physics is still unrealistic
- **On top, we can guess that only a small fraction of the code will scale nicely with the number of processors**
- **What we'll need for sure are:**
  - Versatile parallelisation methods
  - Lots of optimisation
  - … and some creativity!



Take-home messages

- **What is the preferred coordinate system to describe beam dynamics?**
- **What are the two options we have to describe collective effects? Which method is faster?**
- **What types of parallelisation methods might be useful to speed up our calculations? What are the difficulties?**

References

- CERN BLonD Beam Longitudinal Dynamics code: http://blond.web.cern.ch
- Wakes, impedances, and broad-band resonator: A.W. Chao: Physics of collective beam instabilities in high energy accelerators, Wiley, New York, 1993
- CERN VDT library: https://svnweb.cern.ch/trac/vdt
- HDF5 data model: http://www.hdfgroup.org/HDF5/
- CERN Root Input/Output module: https://root.cern.ch/drupal/content/inputoutput