GPU Simulations of Nonlinear High-current Beam Dynamics

García Gil, Pablo
University of Vigo (Spain), pablogarciagil16@gmail.com

The aim of this project is to obtain a new tracking simulation for the heavy-ion synchrotron SIS100 based on the recent xsuite library. With this we intend to obtain a new more universal simulation that can run on more devices thanks to the use of OpenCL instead of CUDA as a communication API to the GPU, allowing computation on Nvidia and AMD graphics cards as well as on CPUs.

1 Introduction

The tracking simulation of a synchrotron consists of the update of the position and momentum coordinates of particles as they pass through every element of the accelerator. This process is known as tracking the lattice, the set of accelerator elements. In addition to this lattice tracking, the space charge effects should be added. These effects consist of the Coulomb repulsion force between particles because of their charge and is simulated by adding between the elements of the lattice a new element called space charge node (SC node).
The stable version of the simulation uses the library SixTrackLib [1] for the lattice tracking and the library PyHEADTAIL [2] for the space charge effects. PyHEADTAIL uses CUDA as GPU communication API which only allows the use of Nvidia cards for computing. The aim is to develop a new version of the simulation using the xsuite [3] library that gives the same results as the stable version. Switching to xsuite would provide greater versatility by allowing the use of any type of GPUs and CPUs thanks to the use of OpenCL as API. This change allows to run simulations on the new cluster of 400 AMD Radeon Instinct MI100 available at GSI with great computational power. The original simulation script has been developed to analyse the space-charge limit of the FAIR synchrotron SIS100 [4].

2 Results comparison

It is necessary to compare both codes to make sure that both give the same results. The space charge effects and lattice tracking have been analysed independently and are finally put together in one single script. This way it is easier to locate and measure the differences found.

2.1 Tracking effects

This script has as a starting point the files generated for MAD-X [5] that define the accelerator lattice. From these we have to import to xsuite the set of elements that represent as faithfully as possible the accelerator. Both simulation scripts have the same MAD-X files to ensure that both lattices are exactly equal. The same particle beam distribution was used in both codes, which were found to be identical before tracking.

2.1.1 Difference in the definition of the longitudinal coordinate ζ between codes

Xsuite uses from version 0.14 on-wards a ζ variable that represents the particle longitudinal position which has a different definition from the ζ found in SixTrackLib. The definition of the variables are

$$\zeta_{stl} = s - \beta ct \tag{1}$$

$$\zeta_{xsuite} = s - \beta_0 ct \tag{2}$$

where s is the absolute position in the accelerator, c is the speed of light and β represents the particle velocity divided by the speed of light while β_0 represents the reference velocity of the bunch divided by the speed of light. This change must be taken into account using the following conversion

$$\zeta_{stl} = \frac{\beta}{\beta_0}\,\zeta_{xsuite} \tag{3}$$

2.1.2 Comparison between tracking codes

The positional coordinates and moment coordinates after every lattice element in both codes have been compared to look for differences in the tracking of any element.
The figure 1 shows the difference of results in the x-coordinate of a random bunch particle over one complete revolution and the position in x of the same particle at the different points calculated by tracking. As it can be seen in the figure, the difference is negligible. The relative difference is on the order of magnitude of 10^{-30}, which is even less than double floating point machine precision, so it has been demonstrated that both codes perform equivalent tracking. The maximum difference found was 3.312×10^{-29}.

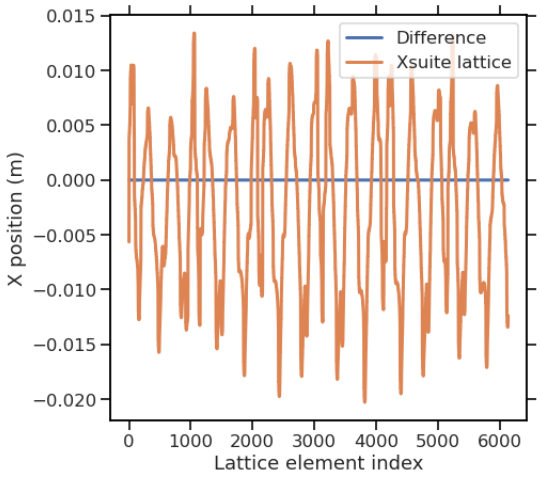

Fig. 1: *Tracking differences along the lattice*

2.2 Space charge interactions

There are different algorithms to apply the space charge effects in the simulation. These simulations use the PIC (particle-in-cell) algorithm. PIC gives results more similar to real behaviour but is more computationally expensive.

It must be taken into account that in the creation of the mesh needed for PIC algorithm PyHEADTAIL places a cell boundary at the origin where xsuite places a cell center. This generates a change in the mesh that could mean a small change in the results, especially in this testing phase where we are working with few particles to speed up the program. To remedy this we need to subtract the distance of a cell to the right in xsuite to counteract this internal effect.

With this consideration it is possible to achieve the same mesh for the calculations. This gives the same conditions to both codes. This difference in definition does not mean that one code is less reliable than the other. The fact of forcing the same mesh is to ensure that the rest of the algorithm is the same. Comparing the results the maximum difference found was $2,233 \times 10^{-06}$ which represents an error of 0.09%. This difference may be caused by the way the two codes interpolate onto the mesh but does not make a substantial difference.

2.3 Tracking and SC interactions integration

In the development of the complete final script some modifications were made to obtain a better performance. Xsuite proposes the application of the space charge interactions by the PIC method by first applying nodes of a different method and then replacing them with PIC nodes. This method also creates a node for each location which implies generating the same mesh for all nodes as many times as there are space charge nodes. This can be a waste of computational power creating many nodes replicating the same mesh multiple times.

The solution that has been proposed to this problem consists of a function that searches for the appropriate locations for each node. It then places directly on the nodes a new object that we have called SpaceChargeWrapper which contains the desired length of the SC node and the SC node object itself which is common to all. This object has a track function that changes the length of the PIC SC node and calls the track function of this object. This allows to do the usual tracking of the SC nodes with a single object for different distances. A small flowchart of all this is shown in figure 2.

3 Run times comparison

In this section the run times of both codes on different platforms are compared in order to understand in which situation it is better to use each code. It is also analysed which part of each code is the slowest.
To monitor execution times we use the python function cProfile which allows us to obtain detailed times for each function and instruction.

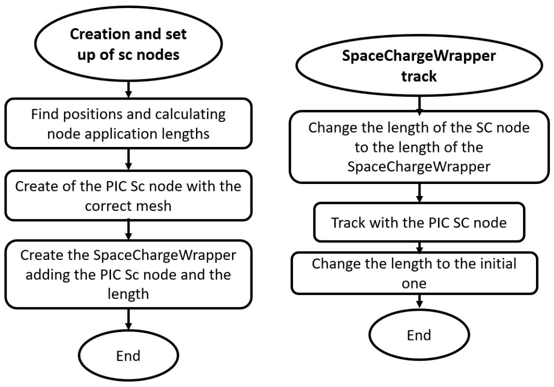
Creation and set up of sc nodes
Find positions and calculating node application lengths
Create of the PIC Sc node with the correct mesh
Create the SpaceChargeWrapper adding the PIC Sc node and the length
End
SpaceChargeWrapper track
Change the length of the SC node to the length of the SpaceChargeWrapper
Track with the PIC SC node
Change the length to the initial one
End

Fig. 2: *SpaceChargeWrapper creation and tracking*

3.1 Computing time of the main parts of the code

The execution times of the main functions of the code have been compared in order to find out which are the most time-consuming. It has been observed that even with a low number of turns, tracking takes most of the time. Already in a simulation of only 10 turns for a million particles and a thousand SC nodes it has been observed that in both libraries the tracking exceeds 70% of the total run time. It is most logical as a measure of the speed of the codes to focus on the time it takes for a turn around the accelerator lattice. The difference in these two time values will be noticeable in simulations with a high number of turns, where time is crucial.

3.2 Time measurement

We have obtained the times for different number of turns for each code in a set of platforms and we have seen that there is a clear linear relationship between the number of turns, as expected. From a linear fit we have obtained the time versus the number of laps so that we can compare the slope of these straight lines which represents the cost in time of a turn without taking into account the setup preparations. The slope and the R^2 of the linear fits are shown in the table below and a graphic representation is shown in the figure 3.
Simulation times as reported in Table 1 were taken for one million particles and 1000 SC nodes. The hardware devices used to obtain the results were the GPUs NVIDIA GeForce RTX 2080 Ti and the AMD Radeon Instinct MI50 as well as the CPU AMD Ryzen Threadripper 1950X. The GPUS have run the code with either CUDA or openCL while the CPU has run on a single core.

Tab. 1: *Timing linear fits*

Code - device	Time/turn(s)	R^2
Pht+stl - RTX	9.130	0.999
Xsuite - RTX	12.719	0.999
Xsuite - MI50	21.448	1.000
Xsuite - CPU	728.736	1.000

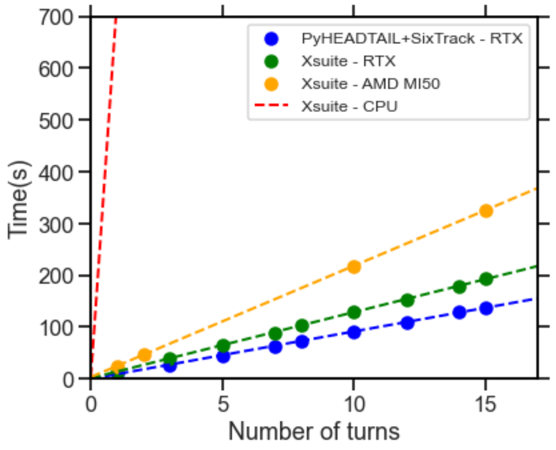

Fig. 3: *Different codes and devices timing*

As a first conclusion, GPU usage is much faster than CPU usage, as expected. This is due to the fact that the GPU allows parallelisation of computation and in the case of the type of operations needed for the simulation it is possible to take a big advantage of this.
The second point is that comparing both codes in the same device (the NVIDIA GeForce RTX 2080 Ti) the stable versión of the simulation performs better than the new xsuite one with a difference of more than 3 seconds per turn. Also the new code on the AMD MI50 gets slower results than using the Nvidia with a relatively big difference.

3.3 Timing of the difference between tracking lattice and space charge effects

The execution times of a lap on different platforms have been taken to analyse the times of the sub-processes in the different platforms. Simulation times were also taken for one million particles and 1000 SC nodes. The devices used were the same like in the previous section, adding the GPU NVIDIA Tesla V100. The results are shown in figure 4.

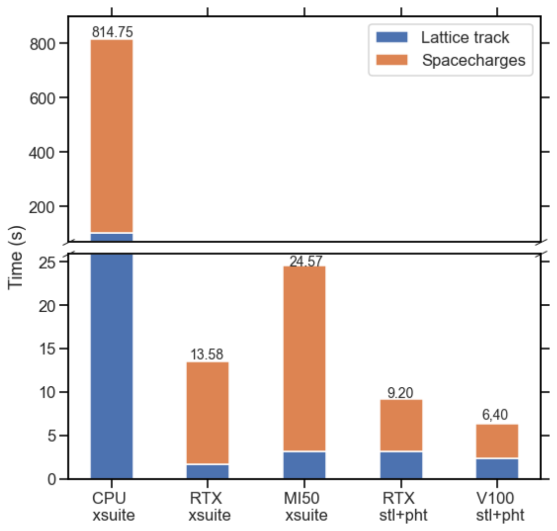

Fig. 4: *Tracking parts timing*

What can be observed in this graph is that in the case of stable code, the percentage of space charge effects time is lower than in the case of new code. This means that the xsuite space charge effects code appears less efficient than PyHEADTAIL. Furthermore, once again it is observed that AMD cards perform slower than Nvidia cards in the new code. However, the time difference is small enough that exploiting the AMD GPU cluster is still a big advantage.

4 Conclusions

The goal of establishing a new simulation script for SIS100 running on AMD GPU cards has been achieved. Using xsuite provides greater versatility and now allows to use the AMD GPU cluster at GSI for future simulation studies. However, it is clear that there is room for improvement in performance: the xsuite based script is up to a factor 4 slower on high-end GPUs when comparing to the stable version of PyHEADTAIL and SixTrackLib, especially the PIC algorithm for space charge computations.

5 Script and complementary information

The simulation script as well as the different scripts and jupyter notebooks of the main tests are available at the following github project:

`https://github.com/pgarciagil/xsuite_benchmarks_and_code`

Acknowledgments

I would especially like to thank my tutor Adrian Oeftiger for all the help and time spent and for giving me this opportunity to learn and be curious about every detail. I must also give special thanks to Conrad Caliari, Dmitrii Rabusov and Vadim Gubaidulin for all the help and patience. You have made this project an even better experience. Last but not least I want to thank my partner Thiti Aungcharoen for all the support. I will never forget this great experience.

References

[1] R. De Maria et al. “SixTrack Version 5: Status and New Developments”. In: *Proc. 10th International Particle Accelerator Conference (IPAC'19), Melbourne, Australia, 19-24 May 2019* (Melbourne, Australia). International Particle Accelerator Conference 10. https://doi.org/10.18429/JACoW-IPAC2019-WEPTS043. Geneva, Switzerland: JACoW Publishing, June 2019, pp. 3200–3203. ISBN: 978-3-95450-208-0. DOI: `doi:10.18429/JACoW-IPAC2019-WEPTS043`. URL: `http://jacow.org/ipac2019/papers/wepts043.pdf`.

[2] *PyCOMPLETE/PyHEADTAIL: CERN PyHEADTAIL macro-particle code for simulating beam dynamics in particle accelerators with collective effects.* URL: `https://github.com/PyCOMPLETE/PyHEADTAIL`.

[3] *Xsuite — Xsuite 0.2.1 documentation.* URL: `https://xsuite.readthedocs.io/en/latest/index.html`.

[4] Adrian Oeftiger et al. “Simulation study of the space charge limit in heavy-ion synchrotrons”. In: *Physical Review Accelerators and Beams* 25 (5 May 2022). ISSN: 24699888. DOI: `10.1103/PhysRevAccelBeams.25.054402`.

[5] *MethodicalAcceleratorDesign/MAD-X: MAD-X repository.* URL: `https://github.com/MethodicalAcceleratorDesign/MAD-X`.