BEAM-BEAM SIMULATIONS ON PARALLEL COMPUTERS[1]

A.Jejcic, J.Maillard, J.Silva
College de France, Paris, France
and
N. Dikansky, D. Pestrikov, D. Shatilov, E. Simonov
BINP, Novosibirsk, Russia

INTRODUCTION

As is known, the beam-beam simulations can be performed within the framework of the so-called weak-strong and strong-strong bunches approaches. Both of them may give valuable predictions when designing colliders.

In the weak-strong approximation the particles of the weak bunch are tracked for desired number of turns using a given fields, which are set in the Interaction Point (IP) by the strong bunch. Except for maybe studying of the time dependencies, these problems can be parallelized on many stages very easily. In the case of electron-positron colliders, the fluctuations of the particle's synchrotron radiation provides sufficient randomization of its motion. So that the simulations can be done using the ergodic assumptions (one particle tracking), or using the multiparticle tracking provided that the number of particle-turns remains the same, that gives more freedom for the code parallelism. In the case of proton (or ion) beams, the multiparticle tracking should be used as a default. The required CPU time for such problems increases linearly with the number of particle-turns.

In the strong-strong simulations the studies are focused on tracking of the particles from both colliding bunches. These problems present more difficulties even for simulations due to huge amount of necessary computations. Various schemes with multiparticle tracking are usually employed. The required CPU time for such problems increases quadratically with the number of particles and linearly with the number of turns. Computation of the beam-beam kicks in these cases uses the information about coordinates of all the macroparticles. For that reason, the parallel computations inevitable will demand inter-processor communications, decreasing the run-time performance of the code.

In this note we report our experience of the employment of parallel computing for the beam-beam simulations, both weak-strong and strong-strong. We shall consider two different computer platforms:

1. Digital's AlphaServer 4100 with 4 processors, where parallel computing can be implemented using the High Performance Fortran language (HPF), running under the Parallel Software Environment (PSE) and Digital UNIX.
2. Set up of 32 transputers T9000 connected with a Sparc station as the front-end machine. Here parallelism is achieved by explicit calls to the message-passing library. In most cases such codes can be easily transformed to PVM (Parallel Virtual Machine) codes, providing their portability.

WEAK-STRONG SIMULATIONS

Weak-strong simulation should be considered as a very good candidate for parallel processing, since the most intensive part of computations can be easily arranged in such a way that a very little (or even none) inter-processor communications will be required. To do so, each processor should track its own particle, providing that the total number of particle-turns is set to the desired value. However, to achieve the goal, one have to match the following requirements:

- Particles must be statistically independent, that can be done by setting for each processor the different initial particle's coordinates and different initial seeds for the random number's generators.
- To avoid inter-processor communications during a tracking, each processor must gather all the statistics obtained for its particle locally. After completion of the tracking, data from all the processors should be joined together, providing that the final tracking results will have the statistical reliability corresponding to the total number of particle-turns.

In the case of proton or ion beams (without cooling) the run-time randomization of the particle motion can be negligible, that presents a separate problem. Usually the goal here is not to obtain the equilibrium distribution, but to watch an evolution of the initial one. So, increasing of the number of simulated macroparticles (and processors) will be always nice, since it improves the statistical reliability of the results, but on this way we cannot reduce the tracking time.

The case of electron-positron (and also proton and ion, with cooling) colliders is quite different. Since the particles remember their coordinates for some time (usually, one damping time or something like that), the initial coordinates should be distributed according to the equilibrium, otherwise they will influence the tracking results. Since we do not know a priori this distribution (it is what we want to obtain).. the following solution is used:

[1] Work supported by INTAS, project 94-4772.

- The initial coordinates of the particles are chosen with the Gaussian (unperturbed) distribution.
- One damping time all the particles are tracked without statistics gathering, just to mix them according to the equilibrium.
- After that, the tracking is continued with accumulating of the statistics.

This algorithm obviously sets a limit for the number of particles which can be efficiently tracked (and, therefore, for the number of processors). Indeed, the required tracking time to obtain the equilibrium beam sizes is usually 1000 damping times, or like that. If we have, say, 1000 processors, it means that each one has to track its own particle one damping time for correct initialization and then one more damping time for obtaining the results (e.g. equilibrium beam sizes), that appears to be not very efficient. On the other hand, if the goal is the equilibrium distribution including the beam tails and lifetime, the tracking time should be much longer, allowing effective utilization of many thousands of processors (if they are available). However, for long-term tracking we use a special tracking technique [1] implemented in the code LIFETRAC [2,3], which allow to reduce the required CPU time by several orders of magnitude when simulating of the beam's halo. Briefly, in this technique the equilibrium distribution is built step by step, starting from the core region, toward the beam tails. The tracking time for each step is also about 1000 damping times, so that we have the same limitation for the number of processors. Summarizing the above arguments, we can say that the weak-strong simulations can be performed very efficiently in parallel, with the number of processors up to a few hundreds.

STRONG-STRONG SIMULATIONS

There are different approaches how to simulate the beams even within the strong-strong concept. The most adequate (but also most time-consuming) method, which is implemented in the tracking code TURN [4], is as follows:

- Both bunches are represented as a set of N (usually $10^3 - 10^4$) macroparticles.
- Macroparticles do not depend on their neighbors from same bunch, but do experience the nonlinear kicks from the macroparticles of the opposite bunch.
- To track any macroparticle through the IP one has to collide it sequentially with every macroparticle from the opposite bunch, thus every passage of IP will require N^2 computations of the two-particles interactions.
- Since the macroparticles are distributed longitudinally within bunches, they collide in order depending on its current longitudinal coordinates. What is important, this order must be not destroyed, since it influences the bunch's behavior. For instance, a head-tail effect can be observed in such an interaction (head of the bunch excites the head of the counter-rotating bunch, which then disturbs the tail of the first bunch). Similar effects were investigated for linear colliders, see for example [5].

We shall not discuss here the other details of such a scheme (the most important is how to avoid scattering on large angles in the two-particles collision without loosing essential details of such an interaction), since they do not depend on parallelism of the code.

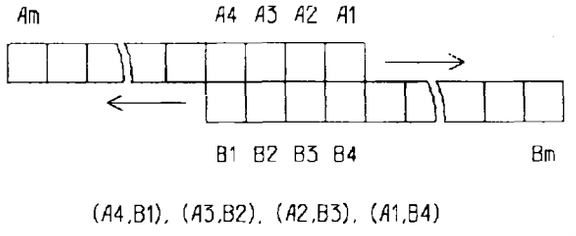

Figure 1. Two beams divided on sub-bunches, 4th step of the collision.

Since the passage through IP is the most time-consuming part of the simulation, just it should be done in parallel. In order to minimize the inter-processor communications, the following algorithm is used:

- On each turn particles are grouped into sub-bunches according to their current longitudinal coordinates. The number of sub-bunches, m, usually should be equal to the number of processors in use. So, the number of particles within sub-bunch is N/m. We shall name these sub-bunches as A_1, A_2, ..., A_m for the first beam and B_1, B_2, ..., B_m for the second one.
- The sub-bunches A_1, ..., A_m are distributed among the processors and stay there, as follows: A_1 in the 1st processor, ..., A_m in the m^{th} processor. The sub-bunches B_1, ..., B_m, on the contrary, move along the processors as is shown on **Figure 1**.
- On the first step of beam-beam collision only A_1 and B_1 sub-bunches are interacted (this occurs completely in the 1st processor). On the second step B-train is shifted, so that A_1 and B_2 sub-bunches are located in the 1st processor, A_2 and B_1 sub-bunches – in the 2nd one. These two collisions between sub-bunches can be done in parallel. On the third step already 3 processors will be used: A_1 and B_3 sub-bunches in the 1st processor, A_2 and B_2 sub-bunches in the 2nd processor, A_3 and B_1 sub-bunches in the 3rd processor. And so on, so that only on the m^{th} step all the processors will be in operation. However, the total number of required steps is not m, but 2m-1: on the last step again only one processor will work, colliding the A_m and B_m sub-bunches. Obviously, the interaction between two sub-bunches must be performed in the same order as described

above (simply replace the word 'sub-bunch' by 'macroparticle' and ignore the word 'processor').

- After completion of the IP passage, the transformation through the collider's lattice is applied to each sub-bunch, that obviously can be done in parallel easily. What is important, such a transformation affects also longitudinal coordinates of the macroparticles, so that they should be redistributed among sub-bunches in order to pass the IP again.

In general case, the number of sub-bunches can be arbitrary (but not less than the number of processors) and even different for the opposite beams. The optimum here depends on the relation between speeds of computations and inter-processor communications, and also on the number of processors and the architecture of their connections. Nevertheless, the described above particular case can be considered as a base scheme.

Now let us estimate the run-time performance of such a parallel code. The simulation time per one turn can be roughly expressed as follows:

$$T = T_L \cdot (N/m) + T_S \cdot N \cdot \log(N) + (2m-1) \cdot \{T_B \cdot (N/m)^2 + T_D + T_P \cdot (N/m)\}$$

Here the first term stays for transport through the lattice, second one – for sorting of the macroparticles according to their longitudinal coordinates and distributing them among the sub-bunches, and the last one – for the beam-beam interaction at the IP (most time-consuming). Within this last term, the factor (2m-1) is the number of steps, the first term in the braces stays for interaction between two sub-bunches, and remaining terms describe the inter-processor communications. So, effectiveness of the employed parallelism depends on many factors and should be investigated for each particular case (computer platform) independently.

As is seen, even in assumption that the inter-processor communications do not demand any time, the gain in the performance will be $\sim m^2/(2m-1) \sim m/2$. The factor of 1/2 comes from the fact that not all the processors can be always utilized. Without grouping of the macroparticles into sub-bunches (or by increasing the number of sub-bunches with respect to the number of processors) we can almost eliminate this disadvantage, but at the cost of greatly increasing inter-processor communications, that in most cases appears to be much more time-consuming.

PARALLEL IMPLEMENTATIONS OF THE CODES

We used codes [1, 4, 6] developed in the Budker Institute of Nuclear Physics (Novosibirsk) to study the beam-beam interaction for electron-positron and electron-ion colliders. We shall not discuss here their features (it is a subject for separate report within a framework of the INTAS project 94-4772), but only our experience in their parallel implementations.

On Digital's AlphaServer 4100 with SMP (installed in BINP), parallelism is achieved rather easily for Fortran codes with the help of HPF and PSE. In the case of weak-strong model, we found that the run-time performance scales almost linearly with the number of processors (we got a factor of 3.8 for 4 processors). Parallel version of the strong-strong codes now is under debugging yet, but we hope to get positive results soon.

We also tried to implement these codes on set up of 32 transputers T9000 installed in College de France (Paris), where the other parallel architecture is used. In the weak-strong case, and for some part of calculations in the strong-strong case, the radial connections master–slave are used. For the main part of calculations in the strong-strong case, the sequential connections between slaves are used to exchange sub-bunches (see **Figure 2**). However, we got some problems with porting of the codes to transputers. At the present time almost all of them are already solved, and we hope to have this scheme working soon. After that, we plan to port our codes also to PVM, that can be done rather quickly.

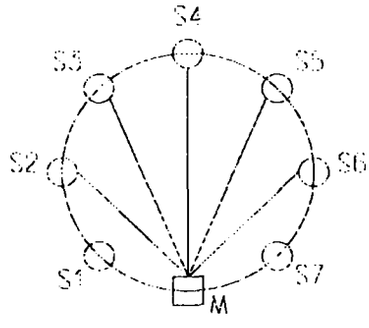

Figure 2. Architecture of inter-processor communications between master (M) and slaves (S1 – S7). Both radial and sequential communications are used.

CONCLUSION

A package of beam-beam codes have been developed, both weak-strong and strong-strong, which can be run in parallel with a great increase in run-time performance. One of the main features is their potential good portability, that opens the possibilities of their wide employment on different computer platforms.

References:

1. **D. Shatilov, Part. Acc. 52, p. 65 (1996).**
2. **D. Shatilov, KEK report 96-14 (1996).**
3. **K. Hirata, D. Shatilov, M. Zobov, Proc. of ICFA97 Workshop, Frascati, 20-25 Oct. 1997.**
4. **E. Simonov, Undocumented strong-strong code TURN.**
5. **N. Solyak, PhD Thesis, BINP, Novosibirsk, 1988.**
6. **N. Dikansky, V. Parkhomchuk, A. Skrinsky et al., http://www.gsi.de/~struck/concepts.html.**