PARALLEL TRACKING IN RINGS WITH SPACE CHARGE[*]

J. Galambos[†], J. Cobb, J. A. Holmes, D.K. Olsen, ORNL, Oak Ridge TN, 37830, USA
A. Luccio, N.L. D'Imperio, BNL, Upton NY

Abstract

A parallel capability of the ORBIT [1] particle tracking code has been implemented. ORBIT is a simulation code for transverse and longitudinal beam tracking in rings, with space charge effects. The parallel implementation used here involves tracking a different set of particles on each CPU, with inter-CPU message passing required only for some initializations, space charge calculations, diagnostic calculations, and output. Macro-particles stay on the same CPU throughout the calculation. This parallel scheme fits naturally with the particle "herd" and ring "node" class structures used in ORBIT. The message passing required for the transverse space charge calculation dominates the parallel implementation efficiency. Calculations using a 128x128 node grid for the transverse space charge and using 100,000 macro-particles per CPU attain a parallel efficiency of ~90%. Full space charge cases tracking 1 million macro-particles for one thousand turns can be completed in less than one day on a small beowulf cluster consisting of 8 CPUs.

1 WHY PARALLEL

Future high intensity proton source devices such as the Spallation Neutron Source (SNS) require stringent loss criteria in order to retain hands on maintenance. For SNS a requirement is to keep losses below one part in 10^4. In order to have a confidence in multi-particle code predictions at this loss level, at least 1 to 10 million macro particles must be tracked. Further more, for SNS space charge plays an important role in the particle dynamics. Tracking this many particles for the full injection period of SNS including the effects of space charge becomes impractical for serial calculations.

We have implemented a parallel calculation in the ORBIT code to facilitate tracking of large numbers of particles. While most of the ORBIT calculations can be made parallel trivially, the space charge aspect requires more attention and is concentrated on here. We examine the efficiency of the parallel implementation used here, and also present an example of a calculation of the full 1100 turn injection. The parallel efficiency is over 80% with up to 8 processors, when a million macro-particles are used.

2 ORBIT PARALLEL IMPLEMENTATION

2.1 General Approach

ORBIT is a particle tracking code for rings which incorporates a programmable interactive driver shell. Serial runs are set up by writing small scripts (or programs) as the input file. The parallel implementation approach is similar to that of a serial run. The initial process serves as the parent. The parent spawns multiple child ORBIT processes on different CPUs, which each parse the same input script file (see Fig. 1), and perform the "same tasks".

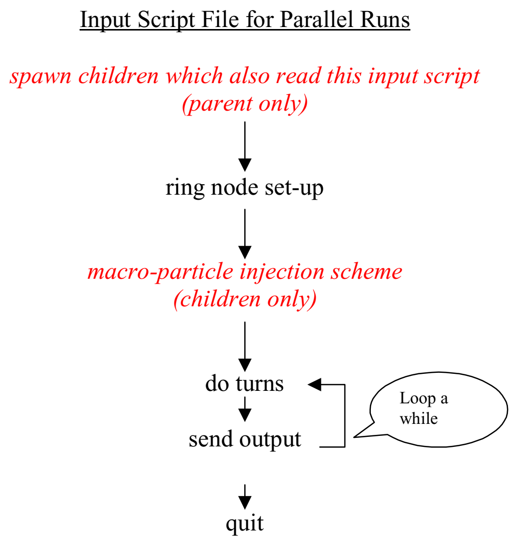
Input Script File for Parallel Runs
spawn children which also read this input script (parent only)
ring node set-up
macro-particle injection scheme (children only)
do turns
send output
Loop a while
quit

Figure 1: Parallel implementation of the ORBIT code.

The data decomposition scheme is extremely simple with this implementation, as macro-particles are never shifted from one processor to another. Most of the calculation on each parallel node proceeds independently just as if it was a stand-alone serial calculation. The exceptions requiring communication between the parallel processes are:

- Initialization of the random number seed used to generate macro-particles
- Space charge calculations
- Diagnostic calculations (e.g. overall beam moments)
- Output

[*] Work performed under the auspices of the US Department of Energy
[†] jdg@ornl.gov

Other calculations such as lattice matrix advances, aperture checks, thin lens kicks, etc. have no need for parallel communication and proceed the same as for a serial run. As such, most modules in ORBIT are unchanged in the new parallel implementation.

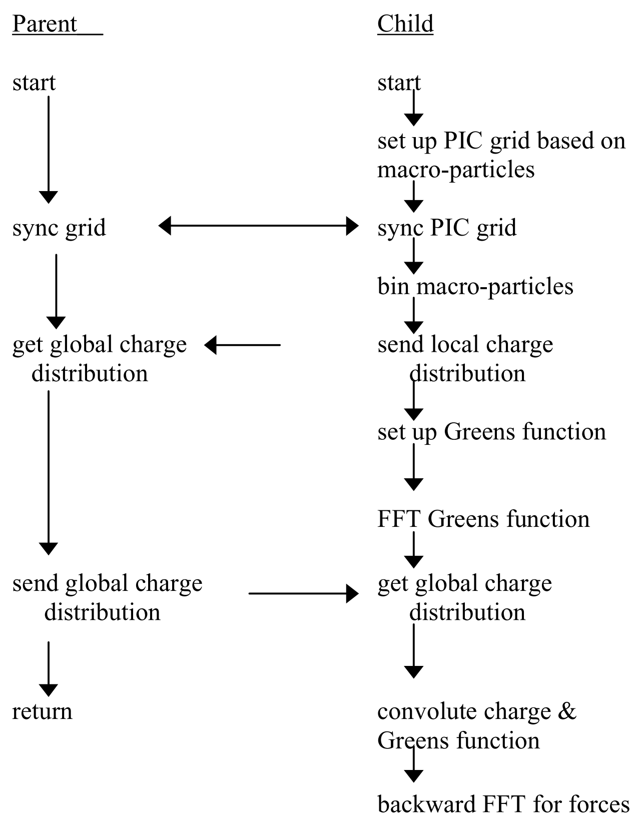
Parent
Child
start
start
set up PIC grid based on macro-particles
sync grid
sync PIC grid
bin macro-particles
get global charge distribution
send local charge distribution
set up Greens function
FFT Greens function
send global charge distribution
get global charge distribution
return
convolute charge & Greens function
backward FFT for forces

Figure 2: Parallel flow logic for the transverse space charge calculation.

2.2 *Space Charge Parallel Algorithm*

The most critical parallel message passing occurs in the transverse space charge section. As multiple herds are tracked in parallel around the ring on different processors, at each space charge kick node, the collective force from all the particles must be gathered and communicated between nodes. Typically this happens 100's of times per turn. The strategy taken to parallelize this calculation is shown in Fig. 2. Note that each child calculates its own FFT of both the Greens function and the global charge distribution[1]. This is faster than waiting for one processor to do it, and passing the results[2]. Also note that the Greens function FFT is done in-between the gather and scatter of the global charge distribution, in order to hide as much latency as possible. This calculation arrangement was arrived at for the SNS "Wonderland" beowulf linux cluster using 533MHz alpha chip processors and a 100 Mbs switched private network for communication. Using systems with higher communication / processor time ratios may make it worthwhile to calculate the charge distribution FFT on the parent in parallel while the children calculate the Greens function FFT, and subsequently pass the calculated charge distribution to the children.

[1] We repeatedly FFT the Green's function because our grid is not fixed throughout the calculation. It is allowed to grow as the particle emittances grow.

[2] Passing the charge distribution involves only ¼ of the grid points since the grid extends 2x's the particle extent to avoid false aliasing. Also passing the FFT output involves 2x's the number of grid points (real & complex).

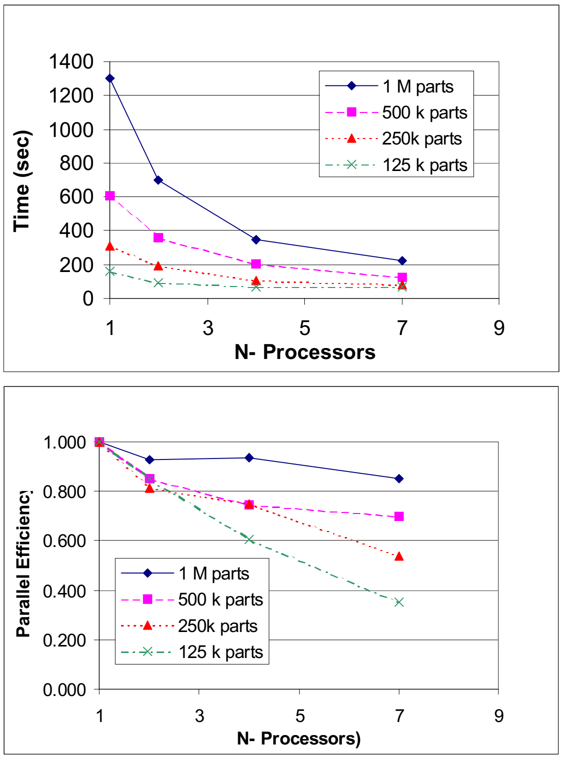

Figure 3. Parallel timings and efficiencies using linear transport matrices and a 128x128 PIC bin

3 TIMINGS, EFFICIENCIES

First some simple timings are shown to elucidate the parallel efficiency of this implementation. These runs were performed on the cluster described above. The parallel efficiency is defined to be:

$$\eta_{//} = \frac{\tau_{serial}}{\tau_{//} \times N_{cpu}},$$

where τ_{serial} is the serial CPU time, and $\tau_{//}$ is the (wall clock) time to do the same calculation in parallel with N_{cpu} parallel processing CPUs. We investigate the sensitivity of this efficiency to the number of macro-particles used and to the number of CPUs. Figure 3 shows the timing and efficiency results for an example using linear transport and a 128x128 PIC transverse PIC grid. This example calculation consists of tracking a macro-particle herd for one turn, using 480 lattice elements and space charge kicks per turn. As can bee seen, the parallel computation is > 90% efficient if there are > 10-20 particles/cell/node. Efficiencies are improved if a 64x64 PIC grid is used. Figure 4 shows timings and efficiencies for the same case, except using second order transport (here more time is spent in the non-collective particle transport calculation). While the CPU time increases, $\eta_{//}$ improves compared to the linear transport cases, when < 10 macros/cell/node are used.

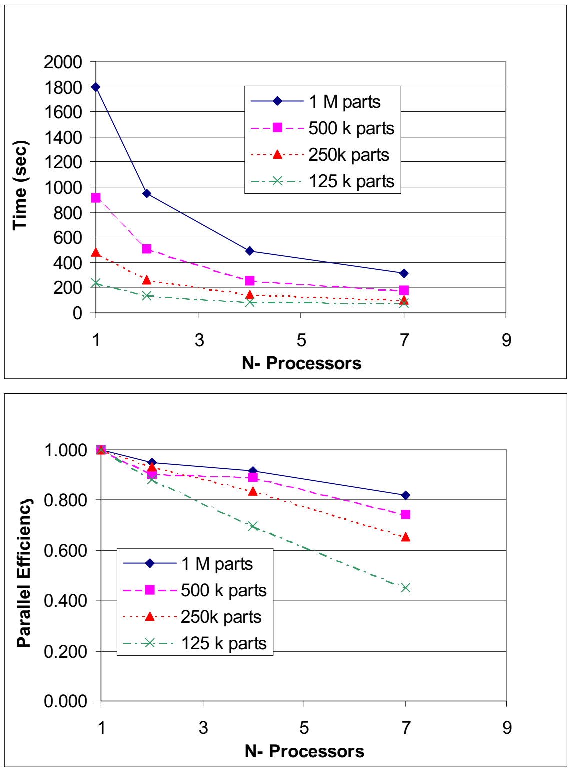

Figure 4. Parallel timings and efficiencies using second order transport and a 128x128 PIC bin.

Table 1. Efficiency impact of various transverse space charge message passing components.

Case	Time (sec)	$\eta_{//}$
Full parallel calculation	66	0.61
Mask the TSC grid synch.	63	0.63
Mask the charge distribution synch.	51	0.78
Mask all transverse space charge message passing	50	0.8
Ideal parallel case	40	1.0

With fewer particles, the calculation becomes less efficient. We investigate the inefficiency by artificially masking certain parts of the transverse space charge message passing, as shown in Table 1. An inefficient case is used for this example (125k macro-particles, 4 CPUs, a 128x128 PIC grid and linear transport). Artificially masking the grid synchronization has a minimal impact. Masking the charge distribution synchronization across the CPUs has a large impact, increasing the parallel efficiency from 61% to 78%. Finally masking all the transverse space charge message passing increases the parallel efficiency to 80%. The residual 20% inefficiency for this case is because of the fewer particles / node in the parallel case so a higher percentage of time is spent on the FFT calculation than on tracking macros.

4 EXAMPLE

Figure 5 shows an the horizontal emittance distribution at the end of the full SNS injection (1158 turns) case with correlated painting, calculated with the parallel implementation. This case uses a closed orbit bump scheme to paint up to 115 π-mm-mrad. Space charge effects cause particles to attain higher emittance. With 1 million macro particles, the distribution at the 10^{-4} – 10^{-5} level begins to be resolved.

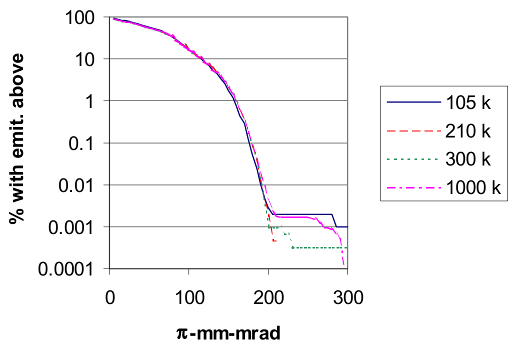

Figure 5. SNS Horizontal emittance distributions.

5 SUMMARY

A parallel implementation of the ORBIT code is available. Initial investigations on a beowulf cluster indicate favorable scaling with number of CPUs. Another version of ORBIT has also been developed using MPI message passing, and is being tested on MP machines [4]. The parallel computing approach is a natural way to proceed in order to track the required number of macro-particles to gain confidence in SNS loss predictions

REFERENCES

[1] ORBIT - A Ring Injection Code with Space Charge , J. Galambos, S. Danilov, D. Jeon, J. Holmes, D. Olsen, ORNL, Oak Ridge, TN; J. Beebe-Wang, A. Luccio, BNL, Upton, NY, PAC99, http://ftp.pac99.bnl.gov/Papers/Wpac/THP82.pdf

[2] Parallel Virtual Machine, http://www.epm.ornl.gov/pvm

[3] ORBIT User Manual, http://www.sns.gov/APGroup/Codes/Codes.html

[4] A. Luccio, N.L. D'Imperio, J. Galambos, "High Energy Accelerator Simulation and Parallel Computing", 17th International Conference on Numerical Simulation of Plasmas", Banff, Alberta, Canada, May 22-24, 2000.