PARALLEL 3-D SPACE CHARGE CALCULATIONS IN THE UNIFIED ACCELERATOR LIBRARY*

N.L. D'Imperio, A.U. Luccio, N. Malitsky, BNL, Upton, NY 11973-5000, USA
O. Boine-Frankenheim, GSI, Darmstadt, Germany

Abstract

The paper presents the integration of the *SIMBAD* space charge module in the *UAL* framework. *SIMBAD* is a Particle-in-Cell (PIC) code. Its 3-D Parallel approach features an optimized load balancing scheme based on a genetic algorithm. The *UAL* framework enhances the *SIMBAD* standalone version with the interactive ROOT-based analysis environment and an open catalog of accelerator algorithms. The composite package addresses complex high intensity beam dynamics and has been developed as part of the FAIR SIS 100 project.

INTRODUCTION

Space charge calculations in beam dynamics simulation codes are often computationally expensive. The burden is magnified when one talks about simulation in three dimensions. The Unified Accelerator Libraries (UAL)[1] environment provides a module, *SIMBAD* [2], which performs parallel space charge calculations in both two and three dimensions using the Paricle In Cell (PIC) method. The Parallel 3-D implementation employed by *SIMBAD* involves dividing the beam longitudinally into numerous segments and performing 2-D transverse space charge calculations on each segment separately. In this manner parallelization consists of each process tracking only those macroparticles contained within the segments assigned to it and communication is limited to macroparticle transfer across process boundaries as a result of synchrotron motion. Since synchrotron motion is a relatively slow process, interprocess communication can be kept to a minimum. When the bunch is contained in an RF bucket, the situation becomes more complex from the perspective of parallel computing. The computational load between processes becomes unbalanced if the problem is decomposed in a naiive way. In order for all processors to be optimally utilized, some form of load balancing is necessary. *SIMBAD* uses a genetic algorithm to find the optimal configuration of segments to assign to each process taking into account both the number of segments and the number macroparticles ultimately given to each process to achieve a balanced computational load.

This paper presents some implementation details of the load balancer as well as performance analysis of the code while simulating a bunched beam in the Alternating Gradient Synchrotron (AGS).

* Work performed under the auspices of the US Department of Energy and with the support of the European Community RESEARCH INFRASTRUCTURES ACTION under the FP6 programme: Structuring the European Research Area-Specific Support Action - DESIGN STUDY (contract 515873 - DIRAC Secondary-Beams.)

UAL SIMULATION ENVIRONMENT

The *UAL* environment addresses the complex simulation tasks of modern beam dynamics studies. It offers an open collection of accelerator algorithms and a consistent mechanism for building configurable project- specific accelerator off-line models. A cornerstone of this mechanism is the *Element-Algorithm-Probe* framework which identifies the association among three major concepts. The *Element* part represents accelerator magnets and devices. In *UAL*, a hierarchical tree of accelerator components is organized as the Standard Machine Format (SMF) module. All accelerator propagators are derived from a basis class *Algorithm* and are initially separated from the accelerator elements. In applications, algorithms can be dynamically loaded and connected in accordance with the user text file written in Accelerator Propagator Description Format (APDF). *Probes* could be any obects (e.g. Bunch, Twiss function, Taylor maps, etc.) evolved by the corresponding algorithms.

In a typical simulation with *SIMBAD* trackers and Accelerator Instrumentation Module (AIM) monitors, a *SIMBAD* tracker is associated with all element types with the exception of monitors. Internally, the *SIMBAD::TSCPropagatorFFT* class is implemented as a composite model combining a space charge kick and a conventional tracker selected from a catalog of *UAL* algorithms such as the thin-lens integrator of *TEAPOT* or the Taylor map of *ZLIB*.

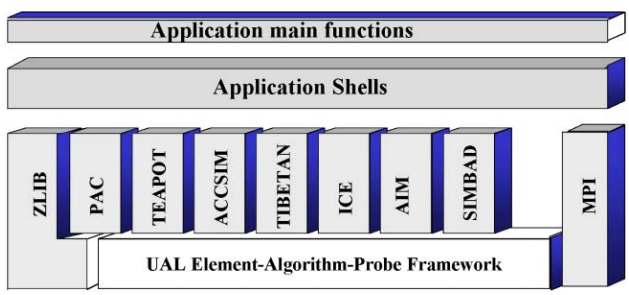

Figure 1: Schematic of the Unified Accelerator Libraries showing the various modules.

Space charge forces are calculated in *SIMBAD* [3] via the solution of the Poisson equation in integral form

$$\Phi(P) = \frac{1}{4\pi\epsilon_0\gamma^2}\int\frac{\rho(Q)}{r}dQ$$

where Φ is the electric potential, P is a field point and Q a source point. The macroparticles are binned on a mesh giving the charge distribution. If beam bunches are long, as is in the case of synchrotrons, the approximation that the beam current is locally parallel to the walls may be made. In this case we can integrate the Poisson equation and represent the partial compensation between space charge repulsion and current attraction with a factor, γ^2.

3D SPACE CHARGE CALCULATIONS AND LOAD BALANCING

An accurate 3-D PIC simulation [4] requires on the order of 10^6 macroparticles to provide meaningful statistics on the meshes commonly used. At each space charge tracker the beam is divided along the longitudinal axis into E_T segments which are, in turn, divided among the processes so that a given process has E_i segments. The global number of macroparticles is N_T and the number of macroparticles in a given process, which is determined by the number of macroparticles in E_i local segments, is denoted by N_i. This technique of dividing the beam into many segments is computationally expensive as it requires E_T solves of the Poisson equation at each space charge element in the ring. Parallel computing is utilized for practical simulations. In allocating the number of segments to be given to each process, a naiive approach may be used that simply divides the number of segments, E_T, by the number of processes, P. While this configuration is acceptable for coasting beams, resulting in roughly equal N_i's, this is not the case for bunched beams which have non-uniform longitudinal densities. For bunched beams the N_i's may be very different. This results in a very uneven allocation of load among the processes. To even the load it is necessary to find a configuration of E_i's and corresponding N_i's such that the work done by each process is as equal as possible. This can be effectively done using a genetic algorithm to determine the optimal configuration.

GENETIC LOAD BALANCING ALGORITHM

Since communication between processes is limited to particle exchange across the process boundaries the 3-D parallelization is reduced to a problem of optimal load balance with the objective being to distribute the computational burden as evenly among the processes as possible. The computational requirements are dependent on two parameters. The local number of space charge segments over which the Poisson equation must be solved and the total number of macroparticles in the local bunch. *SIMBAD* dynamically calculates an optimal decomposition of space charge segments to be given to each process based on both parameters. The genetic algorithm utilizes two parents, F and M, and two offspring, S and D, each of which represents a different distribution of elements among the processes. They can be implemented as arrays where each element of the array maps to a process and the value of the element contains the number of space charge segments for that process.

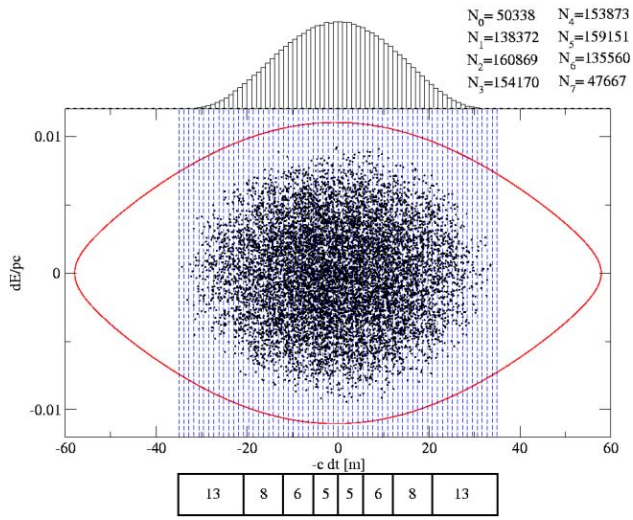

Figure 2: A bunch of macroparticles confined in an RF bucket divided into sixty-four segments over eight processes. The array of numbers represents an optimal configuration of segments where each element in the array denotes a process and the value of the element is the number of segments given to the process. The columns of numbers in the upper right corner lists the number of macroparticles in each process.

There are three phases to the algorithm, mating, natural selection, and mutation. The mating phase combines the parents using an alternating element scheme to create the two offspring. This simply means that an offspring receives its first element value starting with one parent, then gets its second element value from the other parent, repeating this pattern until the last element, which is just $E_T - \sum_{i=1}^{P-1} S_i$, where P is the number of processes and S is the offspring array. The other offspring repeats this procedure but begins with the opposite parent. Natural Selection uses a comparison function $h = f + w * g$ to determine which configuration is optimal where $f = \sum_{i=1}^{P} \left(1 - \frac{P \cdot E_i}{E_T}\right)$ and $g = \sum_{i=1}^{P} \left(1 - \frac{P \cdot N_i}{N_T}\right)$. w is a weighting factor for the two parameters and is a function of both. The mutation component of the algorithm introduces a random variation in one member of the naturally selected pair. The process then repeats and iteration continues until an optimized solution is found.

PERFORMANCE ANALYSIS

Parallel performance is measured by speedup and efficiency, $S(n,P) = \frac{T_s(n)}{T_p(n,P)}$ and $E(n,P) = \frac{S(n,P)}{P}$ respectively. $T_s(n)$ is the runtime of a serial solution with problem size n, $T_p(n,P)$ the runtime of a parallel solution with P processes. For a fixed value of P, typically

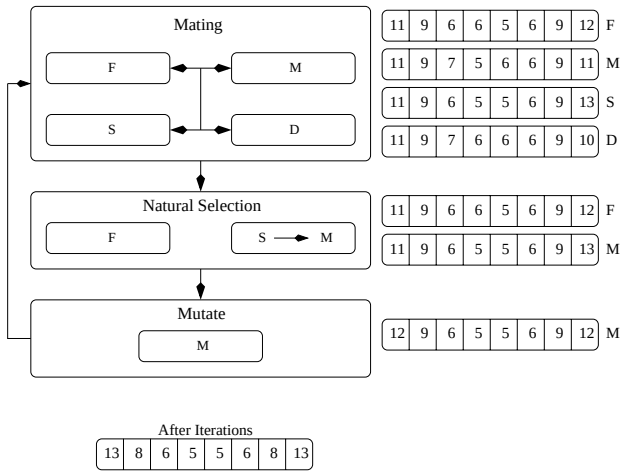

Figure 3: One iteration of the genetic load balancing algorithm together with the optimal result. Mating occurs between F and M to produce S and D. Each offspring takes alternating values from both parents with S beginning with F and D beginning with M. The last element in each offspring array contains $E_T - \sum_{i=1}^{P-1} S_i$ with P the number of processes and S_i the array elements. Natural selection chooses F and S as the optimal pair and so the values of M are replaced with those of S. The mutation is introduced in the new M. The optimal solution shown was reached within one hundred iterations. For comparison, a linear search for a solution would require tens of thousands of iterations.

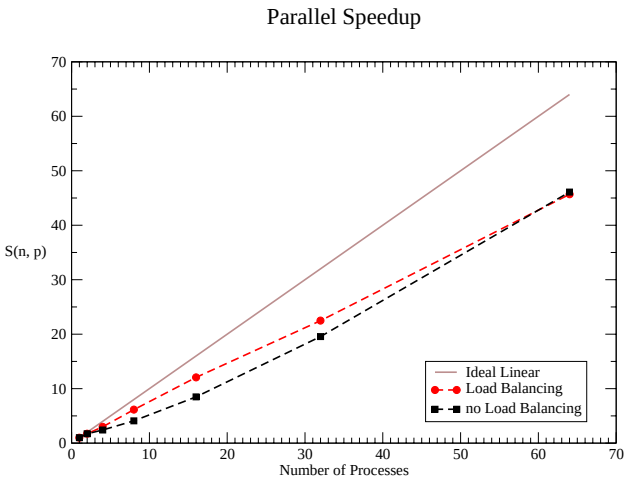

Figure 4: Speedup, while not linear, is acceptable up to sixty-four processes.

$0 < S(n,P) \leq P$. If $S(n,P) = P$ the program has linear speedup. Ideally all parallel programs should exhibit linear speedup, but this is seldom the case. Primarily, the cause is due to communication that is considerably slower than computation in parallel computers. Efficiency measures the process utilization in a parallel program and in most cases $E(n,P) \leq 1$. The program was tested by modeling the AGS on a beowulf cluster using up to 64 proces-

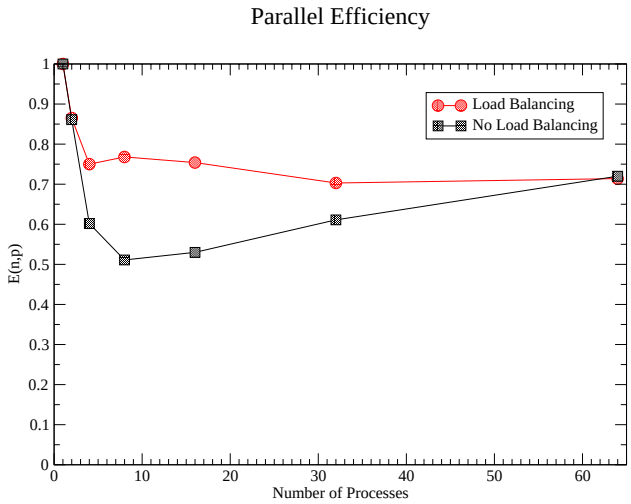

Figure 5: Efficiency improvements of up to 50% are realized with load balancing.

sors. Parallel performance does not scale linearly but this was expected since the communication is Gigabit Ethernet and is not of comparable performance to the computational capabilities of the 2.4 GHz Intel P4 processors. Parallel efficiency shows the improvements acheived using load balancing which, in the best case, was 50 percent. The results for four, eight, and sixteen processes all showed good improvement in efficiency. The corresponding results for one, two and sixty four processes were far less dramatic. For these cases the number of possible configurations is either quite limited or non-existent.

CONCLUSION

A Parallel 3-D space charge module, *SIMBAD*, was implemented for *UAL* utilizing genetic load balancing to improve runtime performance. The AGS was successfully used as a test simulation and efficiency improvements of up to 50% were realized.

REFERENCES

[1] N.Malitsky and R.Talman, 'Unified Accelerator Libraries', Technical Report AIP 391, American Institute of Physics, Melville, New York, 1996

[2] A.U.Luccio and N.L.D'Imperio, 'Simbad User's Manual, Version 1.36', Technical Report C-A/AP/222, Brookhaven National Laboratory, Upton, NY, 2005

[3] J.Beebe-Wang et al., 'Space charge simulation methods incorporated in some multi-particle tracking codes and their results comparison'. Proc. Eighth European Particle Accelerator Conference, La Villete-Paris, France, http://jacow.web.cern.ch/JACOW, June 2002.

[4] A.U.Luccio et al., 'Numerical Methods for the Simulation of High Intensity Hadron Synchrotrons', Proceedings of Coulomb'05, Sept 12-16, Senigallia, Italy, 2005.