BEAM DYNAMICS SIMULATIONS USING A PARALLEL VERSION OF PARMILA

Robert Ryne
Accelerator Operations and Technology Division
Mail Stop H 817
Los Alamos National Laboratory
Los Alamos, NM 87545

Abstract

The computer code PARMILA has been the primary tool for the design of proton and ion linacs in the United States for nearly three decades. Previously it was sufficient to perform simulations with of order 10000 particles, but recently the need to perform high resolution halo studies for next-generation, high intensity linacs has made it necessary to perform simulations with of order 100 million particles. With the advent of massively parallel computers such simulations are now within reach. Parallel computers already make it possible, for example, to perform beam dynamics calculations with tens of millions of particles, requiring over 10 GByte of core memory, in just a few hours. Also, parallel computers are becoming easier to use thanks to the availability of mature, Fortran-like languages such as Connection Machine Fortran and High Performance Fortran. We will describe our experience developing a parallel version of PARMILA and the performance of the new code.

Introduction

Many countries are now involved in efforts aimed at developing high power linacs for transmutation of radioactive waste, disposal of plutonium, production of tritium, and as drivers for next-generation spallation neutron sources. For these projects, high-resolution modeling far beyond that which has ever been performed in the accelerator community will be required to reduce cost and technological risk, and to improve accelerator efficiency, performance, and reliability. Such accelerators will have to operate with extremely low beam loss (0.1-1 nA/m) in order to prevent unacceptably high levels of radioactivity. High resolution simulations using on the order of 100 million particles will be needed to help ensure that this requirement can be met. Such simulations can only be performed on high performance computing (HPC) platforms. For example, near term massively parallel processors and clusters of shared memory processors will have memories of 100's of GBytes and performance of a few TFLOPs. Compared with high-end workstations (500 MFLOPs) and high-end PCs (100 FLOPs), a 1 TFLOP HPC platform would outperform these by factors of 2000 and 10000, respectively.

The computer code PARMILA is the most widely used code in the United States for the design of proton and ion linacs. We have developed a parallel version of PARMILA that runs on the massively parallel CM5 at the Advanced Computing Laboratory of Los Alamos National Laboratory. This version of the code is written in CM Fortran. In addition to moving the code to the CM5, we also replaced the 2D (r, ϑ) space charge routine of the serial code with a new 3D (x,y,z) routine. The code will be used to model the LANSCE linac and linac designs for the Accelerator Production of Tritium (APT) project. As an example of its performance, a 2 million particle simulation of a 1.7 GeV superconducting linac for APT requires 3 hours on the 512 node partition of the CM5. Simulations of shorter linacs have been performed with up to 30 million particles.

Approaches to Parallelization

There are three main parallel programming paradigms: (1) single-instruction-multiple-data (SIMD), (2) single-program-multiple-data (SPMD), and (3) multiple-instruction-multiple-data (MIMD). SIMD is the easiest to use but is the least flexible; all the processors perform the same operations synchronously on different data. The SPMD approach is slightly more flexible; every processor runs the same program, but the programs may execute differently depending on the data. Finally, the MIMD approach is the most flexible and powerful, but it requires the most effort by the programmer to use it; here every processor can run a different program with different data.

To parallelize PARMILA we adopted the data parallel approach augmented by the use of utility libraries and scientific software libraries. This has the advantage that much of the resulting code looks like the original serial version; it can be easily used and modified by a person with little parallel programming experience. Also, if the serial version changes it is easy to make corresponding changes in the parallel version. The parallel version looks like a Fortran90 code, but in addition all DO loops over large arrays have been replaced with FORALL loops. Also, compiler directives appear after array declarations to specify how data is to be distributed across processors.

Steps in Parallelizing PARMILA

The serial version of PARMILA consists of approximately 5000 lines of Fortran 77 code. To port PARMILA to the CM5 we began by running the serial version on a workstation for a problem with zero current. As the parallel code evolved, the results were checked against the serial results. Eventually all DO loops over large arrays, such as the particle array, were replaced with FORALL loops. (FORALL loops are parallel DO loops and recognized as such by the compiler.) This involved rewriting large sections of the serial code between DO/ENDO statements, frequently making use of temporary arrays. For much of the code this task was tedious but

straightforward. Slight complications such as testing for lost particles inside of loops could be easily dealt with by using masked FORALL statements. A more complicated situation arose when tests inside loops effected the program flow. Consider, for example,the serial code used to generate a 4D waterbag distribution:

```
    do 100 i=1,nptcls <loop over particles>
50  <generate 4 random numbers x1,x2,x3,x4>
    if(x1**2+x2**2+x3**2+x4**2.gt.1)goto 50
    <generate coords/momenta for this particle>
100 continue
```

This had to be replaced with code of the following form:

```
100 <generate four LARGE arrays x1,x2,x3,x4>
    <mask off if x1**2+x2**2+x3**2+x4**2.gt.1>
    <pack good data into final array>
    <if final array is not complete, goto 100>
    <generate coords/momenta for all particles>
```

This exemplifies a situation where utility routines (namely PACK) that are not part of CMF or HPF are essential.

Besides rewriting large sections of code associated with DO loops, some other simple tasks were required to port PARMILA. As mentioned above, it is necessary to insert compiler directives in subroutines to specify the layout of parallel arrays. Another task was related to subroutine calls and data reshaping. In CM Fortran, parallel arrays cannot be reshaped through subroutine calls as they can in Fortran 90. Thus, a 2D array **coord(6,ntot***)* cannot be used as in **call mysub(coord(1))** and treated like a 1D array in subroutine mysub. Also, a 1D array **x(ntot)** cannot be used as in **call mysub(x(ntot/2))** and treated as a 1D array of half the original length in the subroutine. Again, these situations are straightforward to deal with, but it can be tedious to find all such occurrences and they can easily go unnoticed until the program crashes or produces garbage.

The major difficulty in porting serial codes to parallel machines using CM Fortran or High Performance Fortran is dealing with those operations that cannot be easily dealt with in the data parallel paradigm. In the case of PARMILA, the difficulty is associated with the space charge calculation. This is discussed in the next section.

Space Charge Calculation

PARMILA uses a Particle-In-Cell approach to computing the beam space charge. Charge is deposited on a grid, the fields are calculated on the grid, and the resulting fields are interpolated back to the particles. The steps involving charge deposition and field interpolation are not easily parallelizable. Consider, for example, charge deposition on a two-dimensional grid using area weighting. A serial routine would look like the following:

```
      do 100 n=1,np
      i=(x(n)-xmin)/hx
      j=(y(n)-ymin)/hy
      ab=xmin-x+i*hx
      cd=ymin-y+j*hy
      rho(i,j)=rho(i,j) + ab*cd
      rho(i+1,j)=rho(i+1,j)+cd*(hx-ab)
      rho(i,j+1)=rho(i,j+1)+ab*(hy-cd)
100   rho(i+1,j+1)=rho(i+1,j+1)+(hx-ab)*(hy-cd)
```

The equivalent parallel routine is the following:

```
 i=(x-xmin)/hx   ! i,j,x,y,ab,cd = arrays
 j=(y-ymin)/hy   ! hx,hy,xmin,ymin = scalars
 ab=xmin-x+i*hx
 cd=ymin-y+j*hy
 forall(n=1:np)rho(i(n),j(n))=
#       rho(i(n),j(n))+ab(n)*cd(n)
 forall(n=1:np)rho(i(n)+1,j(n))=
#       rho(i(n)+1,j(n))+cd(n)*(hx-ab(n))
 forall(n=1:np)rho(i(n),j(n)+1)=
#       rho(i(n),j(n)+1)+ab(n)*(hy-cd(n))
 forall(n=1:np)rho(i(n)+1,j(n)+1)=
#rho(i(n)+1,j(n)+1)+(hx-ab(n))*(hy-cd(n))
```

The above parallel routine has poor performance. First, the FORALL statements cause significant interprocessor commu-nication. Second, if the density array rho is uniformly spread across processors, then the routine will not be load balanced. For example, if one deposited a Gaussian charge distribution on the grid, then processors associated with the tail of the distribution would finish accumulating charge sooner than processors associated with the core. Performance can be improved in several ways:

- One can use SEND routines. These are optimized utility routines that send data to processors based on index arrays and perform binary operations on the data (e.g. add, overwrite, min, max).

- One can use SCAN routines (also called Parallel Prefix routines). These are optimized routines that perform binary operations cumulatively on a sequence of array elements. For example, a SCAN-ADD operation on an array (1,2,3,4,5) would result in (1,3,6,10,15).

- One can use MIMD-style routines written with message passing libraries. In this approach the programmer explicitly writes the code that includes logic to determine how to partition the data so that the load is balanced.

The approach based on SEND routines is easy to use but the performance improvement is modest. The approach based on SCAN routines is more difficult to implement, but the performance improvement is much better. This approach, using segmented-scan operations and data ordering, was implemented by Ferrell and Bertschinger in an N-body code for astrophysical simulations [1]. Finally, the MIMD-style approach is the most difficult to implement but yields the best performance improvement. This approach has been used as part of the Numerical Tokamak Project, a DOE-funded High Performance Computing and Communications project [2] [3].

Currently, the parallel version of PARMILA uses the method of Ferrell and Bertschinger for charge deposition and field interpolation. The field equations are solved using an

FFT-based technique to convolve the charge density on the grid with a Green function defined on the grid. Using standard techniques it is possible to treat a bunch of charge assuming open boundary conditions [4]. We have also implemented a procedure that uses open boundary conditions transversely and periodic boundary conditions longitudinally.

Performance

We have used the parallel version of PARMILA to perform linac simulations with 1-30 million particles. For example, a 2 million particle simulation of a 1.7 GeV superconducting linac for the APT project required 3 hours on the 512 node partition of the CM5. The job used only 2 GBytes,
well below the 14.3 GByte maximum for the partition, so much larger jobs are possible.

The success of the parallel approach depends on scalability, i.e., the ability to run larger problems in the same amount of time using more processors, or the ability to run problems of
a fixed size in less time using more processors. (Note however that increasing the number of processors while keeping the problem size fixed cannot cause the execution time to decrease indefinitely: If the problem size is too small, the processors will do too little calculation, and the execution time will be dominated by communication.) The parallel version of PARMILA has excellent scalability as shown in Table 1:

Table 1: Scaling Results (3.75M particles, 64x64x64 grid)

Procs	CPU (min)	MEM (GB)
128	15.5	1.8
256	8.1	2.0
512	4.3	2.5

Conclusions/Future Work

We have developed a parallel version of PARMILA that runs on the CM5 at the Los Alamos Advanced Computing Laboratory. This version of the code is written in CM Fortran. In addition to moving the code to the CM5, we also replaced the 2D (r, ϑ) space charge routine of the serial code with a new 3D (x,y,z) routine. Using the present version of the parallel code, simulations with 1-30 million particles are possible depending on the length of the linac being modeled. We plan to move the code to the new Cray T3E at the National Energy Research Scientific Computing Center. We expect a significant improvement in performance through the use of charge deposition and field interpolation routines that use message passing.

Acknowledgements

The author thanks Lawrence Rybarcyk, Frank Merrill, Robert Garnett, and Kenneth Crandall for helpful discussions about the PARMILA code. This research was supported by the U.S. Department of Energy, Office of Energy Research, through the Division of Mathematical, Information, and Computational Sciences, the Division of High Energy and Nuclear Physics, and by the Office of Defense Programs, Accelerator Production of Tritium program. This research was performed in part using the resources located at the Advanced Computing Laboratory of Los Alamos National Laboratory, Los Alamos, NM 87545.

References

[1] R. Ferrell and E. Bertschinger, *Int. J. Mod. Phys. C*, **5**, (1994), 933-956.
[2] V. K. Decyk, *Computer Physics Communications*, **87**, (1995), 87-94.
[3] J. Wang, P. Liewer, and V. Decyk, *Computer Physics Communications*, **87**, (1995), 35-53.
[4] R. W. Hockney and J. W. Eastwood, *Computer Simulation Using Particles*, (Adam Hilger, New York, 1988).