SLAC PARALLEL TRACKING CODE DEVELOPMENT AND APPLICATIONS

Brian McCandless, Tor Raubenheimer, Ross Richardson, Kwok Ko
SLAC, Stanford University, Stanford, CA94309, USA

Abstract

The increase in single processor speed based on Moore's law alone will not be able to deliver the dramatic speedup needed in many beam tracking simulations to uncover very slowly evolving effects in a reasonable time. SLAC has embarked on an effort to bring the power of parallel computing to bear on such computations with the goal to reduce the turnaround time by orders of magnitude so that the results may impact present facilities and future machine designs. This poster will describe the approaches adopted for parallelizing the LIAR code and the ION_MAD code. The scalability of these tracking codes and their further improvement will be discussed.

1 PIPELINE MODEL FOR PARALLELISM

Beam tracking applications model a beam (composed of many bunches of particles) travelling through an accelerator (defined as a lattice of optical elements). Each bunch travels past each optical element in order, and after all the preceding bunches. The nature of these applications imposes an order on the calculations. This limits the possible approaches to parallelism, which seeks to exploit independent calculations that can occur simultaneously. One successful approach to parallelism for these codes is the pipeline model.

The key observation is that bunch b_i can be processed by element e_i, and at the same time bunch b_{i+1} can be processed by element e_{i-1}. The strategy for pipeline parallelism can be summarised as follows:

- Bunches are grouped into **P** groups of equal size, where P is the number of processes.
- Each process stores and computes with one bunch group and one or more optical elements at a time.
- When process ***p*** is done working on its bunch group, it sends it to process ***p+1***, and simultaneously receives a bunch group from process ***p-1***.
- When a process is done working on the last bunch group in the train, it shifts to the next unvisited optical element

This strategy can also be modified to communicate the optical elements instead of the beam data. It makes no difference to the pipeline model. The decision is made for the approach that reduces the overall communication cost. For simplicity, we will assume for the rest of this paper that that the beam data is communicated instead of the optical element data.

As a small example, assume there are three processors and nine lattice elements. The beam is evenly divided into three bunch groups. Bunches within a grouping are communicated together. The lattice elements are distributed cyclically among the three processors. Figure 1 graphically illustrates the pipeline process.

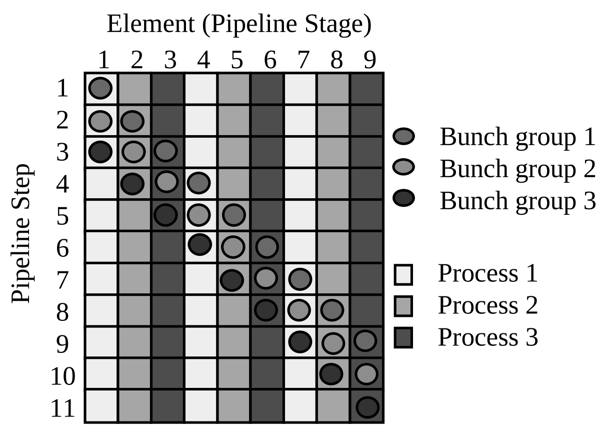

Figure 1: Pipeline example

During the first pipeline step, process 1 computes the effect of optical element 1 on bunch group 1. The other processors are idle. During the second pipeline step, bunch group 2 enters the simulation at process 1, and bunch group 1 is sent to process 2. Two processors are now computing. The third bunch group enters the simulation at step 3. Now all the processors are working, and the pipeline is full. The pipeline remains full until the first bunch group reaches the last optical element. The bunch groups then begin to leave the simulation, and processors become idle again.

The pipeline model can not provide perfect linear speedup due in part to the filling and emptying steps. However, the pipeline speedup can approach the number of processors when the number of stages is large. In the example, it takes 11 pipeline steps to push three bunch groups through nine lattice elements. In serial, this would have taken 27 steps. The speedup due to the pipeline is therefore 27/11 = 2.455. If there were one thousand optical elements instead of nine, the pipeline speedup would be 3000/1002 = 2.994.

2 OBJECT-ORIENTED FRAMEWORK

Beam tracking codes share the same basic computational patterns. The parallelism of these codes using the pipeline model follows the same basic principles. This observation motivated the development of an object-oriented framework to aid in the development of parallel beam tracking applications. FALCO, the Framework for Assembly Line Code Optimisation, provides the pipeline functionality through a general, application independent, object-oriented interface.

FALCO is based on a factory assembly line abstraction. There are five main abstractions: Components, Trays, Tools, Toolboxes, and AssemblyLine. Groups of Components travel along the assembly line in Trays, encountering Tools along the way. Tools are grouped together into Toolboxes. The AssemblyLine co-ordinates the activities of the tools and components. It is also responsible for inter-process communication.

The beam tracking application is parallelized by writing or re-writing portions in C++ to make use of the abstraction in FALCO. A bunch derives from the FALCO Component base class, and an optical element derives from the FALCO Tool base class. The framework handles the grouping of Tools and Components, the distribution of the Trays and Toolboxes, inter-process communication, and the state of the pipeline. The parallelism is hidden from most of the application.

3 BEAM TRACKING APPLICATIONS

In this paper, we discuss two beam tracking applications that have been parallelized: ION_MAD [1], which is used to simulate the fast-beam ion instability and LIAR [2], which is used to model the operation and performance of high energy linear accelerators.

3.1 Ion-Mad

Ions are recognized as a potential limitation in electron storage rings where ions generated by beam-gas collisions can become trapped in the negative potential of the beam. Future storage rings typically have high beam currents and small beam emittances, increasing the deleterious effects of the ions. To avoid ion trapping, most future electron storage rings are designed to include a “gap" in the bunch train. The ions, which are strongly focused by the closely spaced bunches, are over-focused in the gap.

With a sufficiently large gap, ions are not usually thought to be a limitation. However, many of the modern accelerators operate or will operate in a new regime with high current, long bunch trains, and very small transverse beam emittances. In this case, ions generated and trapped within a single bunch train, or, in some cases, within a single bunch, can have significant effects. This is true in transport lines and linacs, where typical vacuum pressures are relatively high, as well as storage rings.

The ions oscillate within the potential of the beam and can modulate the transverse beam position at the ion bounce frequency. This modulation then resonantly drives the trapped ions and quasi-exponential growth results; the instability, referred to as the fast beam-ion instability (FBII), is illustrated schematically in Figure 2. The nature and analytic treatment of the instability closely resemble the beam break-up instability due to transverse wakefields and is described in Refs. [1] and [3].

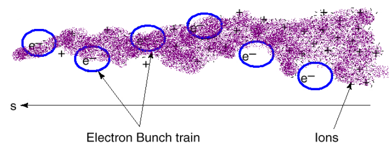

Figure 2: Schematic illustration of FBII

The instability growth rate is sensitive to the spread in the ion bounce frequency due to variation of lattice parameters around the ring and the transverse variation of the beam distribution [1,3,4]. In addition, the growth rate depends on the variation of the ion distribution as the ions are trapped along the beam [1] and the tune shift induced by the ions along the bunch train [5]. Finally, at low vacuum pressures, the growth rate will be sensitive to the ion statistics. Thus, to make accurate predictions of the FBII in storage rings and to compare with experimental measurements a detailed simulation is required which includes the relevant effects.

The ION_MAD [1] program represents the beam as macro-particles and tracks them through a lattice described as a MAD deck. As each beam slice passes through each lattice point, ion macro-particles are generated and tracked in transverse phase space with all the transverse fields treated self-consistently. To accurately model the instability a typical run will track 100 turns in a ring consisting of a 1000 elements with 50,000 beam macro-particles and 50,000 ion macro-particles.

3.2 LIAR

Future linear colliders must accelerate beams with very small transverse emittances to achieve the luminosity goals. The LIAR project [2] was started to create a tool to model these high-energy linear accelerators and provide an open programming platform for the accelerator physics community. The code includes many features that are necessary for simulation of the beam dynamics in future linear colliders including the effect of the short- and long-range wakefields, accelerator imperfections, ground motion, and beam steering and feedback systems.

A major goal for the Next Linear Collider design [6] is to be able to simulate one full hour of operation of the real linac. This time scale is chosen to verify and devise new alignment algorithms to counter ground motion effects and understand the interaction of the various nested feedback loops. This proposed simulation would include about 360,000 pulses and require about three years of computer time with the serial code. Parallel processing is therefore required to finish in a reasonable amount of time.

4 PERFORMANCE

The performance of parallel beam tracking depends on a number of parameters such as the number of processors, the number and size of the bunches, the number of lattice elements, and the number of optical elements grouped together into a pipeline stage. These factors contribute to or detract from either the pipeline efficiency or the computation / communication ratio.

Performance will ideally increase linearly with the number of processors. However, adding more processors decreases the pipeline efficiency and can hurt the computation/communication ratio. These effect could reduce the linear speedup, and lead to diminishing returns.

An example of this can be seen in Figure 3. ION-MAD is used to model the Advanced Photon Source (APS) with a varying number of bunches. An increase in the number of bunches increases both the computation and communication cost, but not equally. As the bunches increase, the ratio of computation to communication increases, which improves the parallel speedup.

In the 24 bunch case, the drop in performance on the last data point is made more sever due to a load imbalance. It it not possible to divide 24 bunches evenly among the 16 processors. Performance is also low in this case due to the reduced computation/communication ratio.

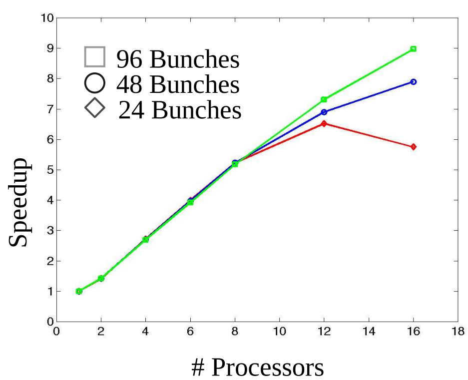

Figure 3: Parallel Speedup of ION-MAD

Grouping lattice elements together has the effect of lengthening each pipeline stage, which increases the computation/communication ratio. The block size is number of lattice elements grouped together in this way. Increases to the block size may also decreases the load imbalances caused if some lattice elements are more computationally expensige than others. However, the decrease in the number of pipeline stages hurts the pipeline efficiency. The following equations illustrate the tradeoff between communucation cost and pipeline speedup.

$$\text{L, Pipeline Stages} = \left\lceil \frac{E}{S} \right\rceil$$

$$\text{T, Pipeline Speedup} = \frac{L \times P}{L + P - 1}$$

$$\text{C, Communication Cost (\# Bunches sent per process)} = \left\lceil \frac{L \times B}{P^2} \right\rceil$$

E = # elements
S = block size
B = # bunches
P = processors

An optimal block size can be found through experimentation. In one experiment, the LIAR code was used to simulate the SLC on 16 processors of the Cray T3E at NERSC. The effect of block size on performance is illustrated in Figure 4.

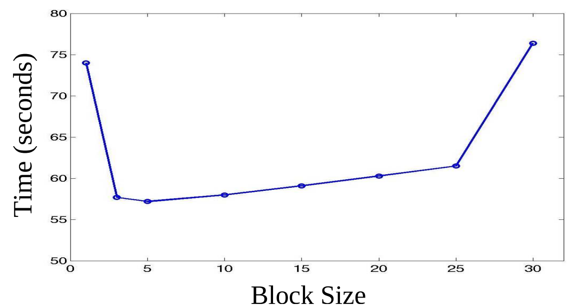

Figure 4 : The effect of block size on performance

In this example, the lattice contained about 2400 optical elements and 96 bunches. The pipeline speedup is greatest (15.9) when the block size is one. However, this is also when the communication costs are greatest (900 bunches per processes). When the block size is 30, the pipeline speedup is 13.5 and the total number of bunches sent per process is 30. When using the optimal block size of five, the speedup is 15.5 and the total bunches communicated is 180.

5 FUTURE WORK

The parallel ION-MAD code is being used in production mode on the SLAC PC Linux cluster to study the Fast Beam-Ion Instability. Longer runs to uncover very slowly evolving effects in the APS are planned for the near future. The parallel LIAR code is not yet ready for production runs. When it is, it will be used in the study of ground motion effects and alignment algorithm of the NLC.

REFERENCES

[1] T.O. Raubenheimer and F. Zimmermann, *Phys. Rev. E*, **52:**5487 (1995).

[2] R. Assmann, et al., Linac 96, Geneva, Switzerland, (1996) p 464.

[3] G. Stupakov, et al., *Phys. Rev. E*, **52:**5499 (1995).

[4] R. Bosch, *Phys. Rev. STAB*, **3:**034402 (2000).

[5] D. Pestrikov, *Phys. Rev. STAB*, **2:**044403 (1999).

[6] For example T.O. Raubenheimer, these proceedings.