Universidad de Investigación de Tecnología Experimental Yachay Tech

School of Mathematics and Computer Sciences

Subject : HPC	Professor : Saravana Prakash
Date : 23/05/2019	Semester: Jan-May-2019
Student's Surname, Name : Caraguay Ordoñez Henry	
Semester : IX	

Programming Spin Effects into the Projectile

This problem is taken from the book Physics for Game Programmers, it is important know how to model physical phenomena because these kind of simulations take an crucial role in video games development.

1. Problem statement

Model the spin effects to the projectile trajectory model, taking into account variables as gravity, velocity and Magnus force. The model must include a Runge–Kutta solver method for ODE. The code must be parallelized using MPI and OpenMP.

2. ODE scheme used

The ODE scheme used was the Runge - Kutta ODE solver, this method belongs to a family of implicit and explicit iterative methods, all including the Euler Method, used for discretization. In the code is included a solver of fourth order.

2.1. Mathematical Formula

A Runge - Kutta mehtod of order s can be written as:

$$y_{t+h} = y_t + h \cdot \sum_{i=1}^{s} a_i k_i + \mathcal{O}\left(h^{s+1}\right) \tag{1}$$

where

$$k_i = f\left(y_t + h \cdot \sum_{j=1}^{s} \beta_{ij} k_j, t_n + \alpha_i h\right) \tag{2}$$

are incrementes obtained evaluating the derivatives of y_t at the $i - th$ order.

We develop the derivation for the Runge–Kutta fourth-order method using the general formula with $s = 4$ evaluated, as explained above, at the starting point, the midpoint and the end point of any interval $(t, t + h)$.

3. Results

3.1. Analyzing the code

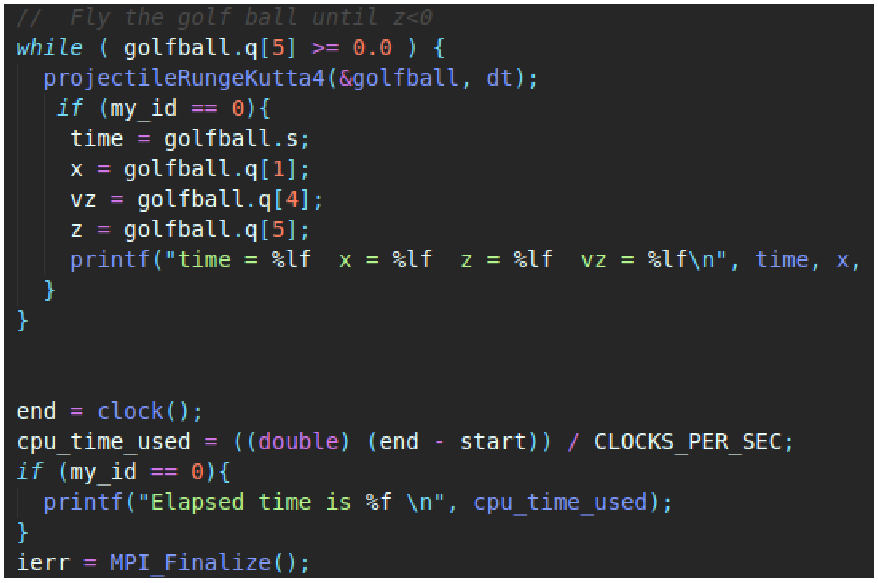
The code has tree functions:

- Function *projectileRungeKutta4*: this function implements the Runge - Kutta solver, using five auxiliar arrays to execute the four steps of the solver.
- Function *projectileRightHandSide*: called by the previous function, it manages to compute the apparent velocities, considering variables like wind velocity and Magnus force.
- Function *main*: It starts a spin projectile and calculates all the variables for the golf ball motion using the Runge-Kutta solver.

Analyzing its structure there are very few regions that can be rewritten to run in parallel, because most of the code must be executed sequentially, since the same variables are used one after another.

3.2. Modifying the code

The code has been modified to run with MPI and OpenMPI. In the case of the MPI version, we are executing the while loop for all processors, but we are getting and displaying the time just for processor 0,as shown in figure 1, this section is in the main function.

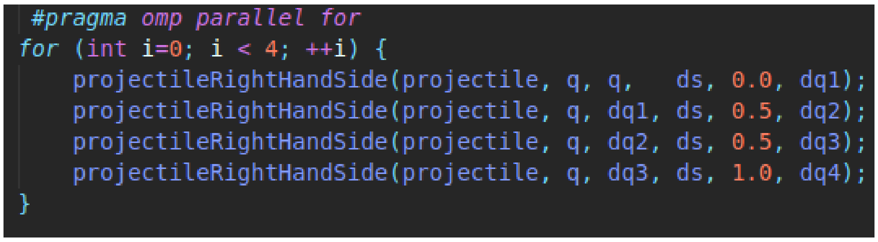


```
//  Fly the golf ball until z<0
while ( golfball.q[5] >= 0.0 ) {
  projectileRungeKutta4(&golfball, dt);
   if (my_id == 0){
    time = golfball.s;
    x = golfball.q[1];
    vz = golfball.q[4];
    z = golfball.q[5];
    printf("time = %lf  x = %lf  z = %lf  vz = %lf\n", time, x,
  }
}


end = clock();
cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;
if (my_id == 0){
  printf("Elapsed time is %f \n", cpu_time_used);
}
ierr = MPI_Finalize();
```

Figura 1: MPI code

In the case of OpenMPI, I used a pragma omp paralell, to call the function *projectileRightHandSide*, in four threads at the same time, as can be seen in figure 2



```
 #pragma omp parallel for
for (int i=0; i < 4; ++i) {
    projectileRightHandSide(projectile, q, q,   ds, 0.0, dq1);
    projectileRightHandSide(projectile, q, dq1, ds, 0.5, dq2);
    projectileRightHandSide(projectile, q, dq2, ds, 0.5, dq3);
    projectileRightHandSide(projectile, q, dq3, ds, 1.0, dq4);
}
```

Figura 2: OpenMPI code

3.3. Time measures

Considering that the modifications done in the case of OpenMPI consider four threads, we have taken times for this number of processors. In the following table can be seen a comparative of execution times between serial, MPI and OpenMPI version. Each time was taken as the average of ten executions for each version.

Number of Processors	1	4
Serial	0.001829	N/A
OpenMPI	N/A	0.01247
MPI	0.001889	0.000682

4. Conclusions

After analyzing the results, we can conclude the following:

- The results were better for the MPI version, it reduced the execution time to one third of the original.
- The OpenMP version was not successful, probably because making threads was more expensive than calling sequentially the functions.
- The code was not very friendly, in terms of parallelism, because it comprises a lot of successive operations that depend in the result of previous lines.

PD: The codes can be found in the Quinde directory: */HENRY-ORDOÑEZ/23-05-2019/*