Universidad de Investigación de Tecnología
Experimental Yachay Tech
School of Mathematics and Computer sciences

Program : Undergraduate ———— Semester : VII
Subject : HPC Professor : Saravana Prakash T
Date : 18/04/2019 Time:
Semester: Jan-May-2019
Student's Surname, Name: Osiris Anael Román Eras
Student Reg.No: 1724771645

Duration: 10 minutes *INFORMATICS LAB* Maximum Marks: 100

Elaboration Date 09/05/2019
Submission Date : /05/2019

Final HPC Project: A Car Simulator - A Comparison Between MPI, OPENMP and Serial

1. Introduction

This project we discuss parallel implementations of numerical methods for solving ordinary differential equations (ODE) of initial-value problems (IVP). The project implements a Car Simulator taken from the book "Physics for Game Programmers". In this work you'll see a comparison between MPI, OPENMP and the serial implementation of this simulator. The ODE solvers used were 4th ordered Runge Kutta and Euler's Algorithm.

The Car Simulator model was code in C programming language and simulate the forward motion and the speed of a car. The simulation is very complete because have a system that determines whether the car is accelerating, cruising at a constant velocity, or slowly down by braikng. The simulation includes the current velocity and engine turnover rate of the car as well as what gear the car is currently in. Also includes the distance that the car has traveled and the total elapsed time of the simulation. If the engine turn-over rate exceeds the redline rpm value the engine and the simulation will stop, it happens when the engine turnover rate exceeds 8000 rpm. The car is assumed to be driving in a stright line on flat ground. When the car is braking, the acceleration due to the braking is assumed to be a constant $-5{,}0m/s^2$

Since the ODE solver will be used to update the position and velocity of the car, the car simulator has to declare a function carRightHandSide to define the right-hand sides of the equations to be solved. The intermediate values of location and velocity for the car are computed. In this simulation, we only are cooncerned with the x-components of location and velocity. The final part of the carRightHandSide method defines the right-hand side of the ODEs that describe the motion of the car. If the car is accelerating, the acceleration of the

car is computed from the Runge Kutta equations or in the other implementations, from the Euler's approximation equation. The action of move performs every 0.05 seconds. The first thing the code does when it runs is to determine whether the car is accelerating, cruising at a constant speed, or braking and sets the value of the mode field accordingly. The rpm value of the engine is computed using the formulas provided in the book, the Equation 8.13 to be precise.

2. Methodology

As we saw in the introduction the IVP solvers for ODE used in this work were the 4th ordered Runge Kutta and the Euler's approximation method.

Numerical methods for solving ODE equations of initial-value problems of the form:

$$\begin{cases} y^{'} = f(x,y), x \in [a,b] \\ y(x_0) = y_0 \end{cases} \tag{1}$$

where y is a function of x, f is a function of y and x, x_0 is called the initial point, and y_0 the initial value. The numerical values of $y(x)$ on an interval containing x_0 are to be determined.

4th Ordered Runge Kutta

The general form of second-order Runge-Kutta method:

$$y(x+y) = y + w_1 hf(x,y) + w_2 hf(x + \alpha h, y + \beta hf) + O(h^3) \tag{2}$$

By letting $w_1 = 0$, $w_2 = 1$, $\beta = 1/2$ leads to the modified Euler's method.

Fourth-order Runge-Kutta method

$$y(x+y) = y + \frac{1}{6}(F_1 + 2F_2 + 2F_3 + F_4) + O(h^5) \tag{3}$$

where:

$$\begin{cases} F_1 = hf(x,y) \\ F_2 = hf(x + \frac{1}{2}h, y + \frac{1}{2}F_1) \\ F_3 = hf(x + \frac{1}{2}h, y + \frac{1}{2}F_2) \\ F_4 = hf(x + h, y + F_3) \end{cases} \tag{4}$$

The fourth Runge Kutta method involves a local truncation error of $O(h^5)$

Euler's Approximation method

One of the simplest methods for solving IPVs. The basic idea behind the Euler's method is subdivide $[a,b]$ into n subintervals of equal length h with mesh points $x_0, x_1, x_2, ..., x_n$, where

$$x_i = a + ih, \forall i = 0, 1, 2, ..., n. \tag{5}$$

We have the formulation of the Euler's method as

$$\begin{aligned} x_{k+1} &= x_k + h = x_0 + (k+1)h \\ y_{k+1} &= y_k + hf(x_k, y_k) \end{aligned} \tag{6}$$

It implies that Euler's method is a first-order ($O(h)$)method.

3. Results

The characteristics of the computer used for the comparison are:

- **Processor:** Intel(R) Core(TM) i5-7300hq CPU 2.5 GHz
- **RAM:** 8Gb
- **Number of Cores:** 4

In order to compare the results 5 principal comparisons where taken into account

- **4th Order Runge Kutta - OMP vs MPI vs Serial:** As we can see on Fig 1 the MPI implementation is better than the OMP, but the Serial implementation of the ODE solver was faster even when the acceleration of the car goes up to 320 seconds.
- **Euler's Method - OMP vs MPI vs Serial:** For the Euler's Method the trend looks similar than for the Runge Kutta. The serial implementations is better than OMP and MPI parallelization.
- **Serie - Euler's Method vs 4th order Runge Kutta:** We can see from Fig 3 that the Euler's method is not so fast as Runge Kutta when we review the serial implementation.
- **MPI - Euler's Method vs 4th order Runge Kutta:** In Fig 4 we can see that the MPI parallel version of the Euler's method have a peak when it reaches 160 seconds of acceleration after that calculate faster for 320 seconds of acceleration. Maybe it is caused because several solutions were calculated before and then it didn't need some extra time to made final computations.
- **OMP - Euler's Method vs 4th order Runge Kutta:** Finally the OMP implementation for both 4th order Runge Kutta and for Euler's Method was very similar. It is necessary to mention that the way to calculate the approximation for both solvers is very similar and that the OMP implementation was the worst in both cases. It is interesting that both presents a increasing trend according the acceleration time is increased.

Kutta.png

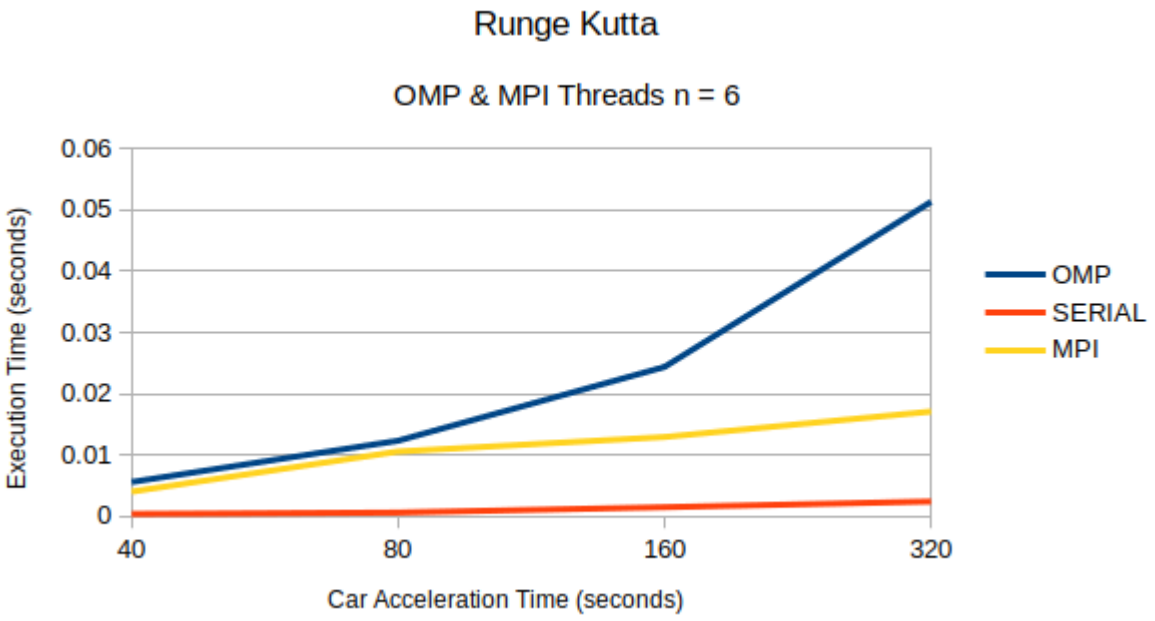

Figura 1: 4th Order Runge Kutta - OMP vs MPI vs Serial

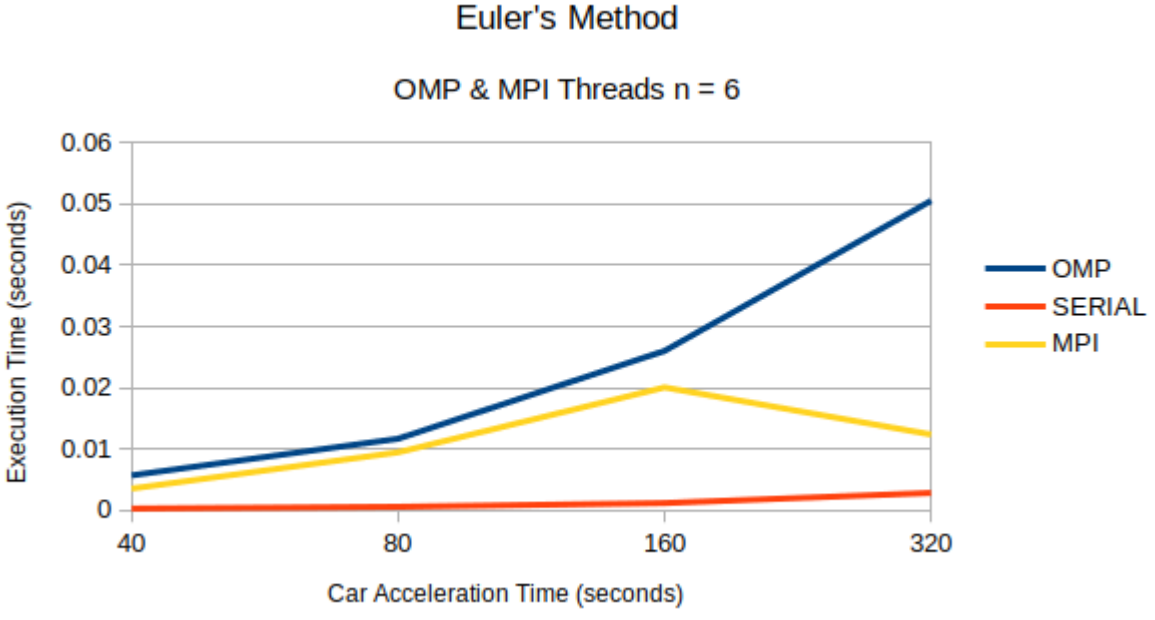

Figura 2: Euler's Method - OMP vs MPI vs Serial

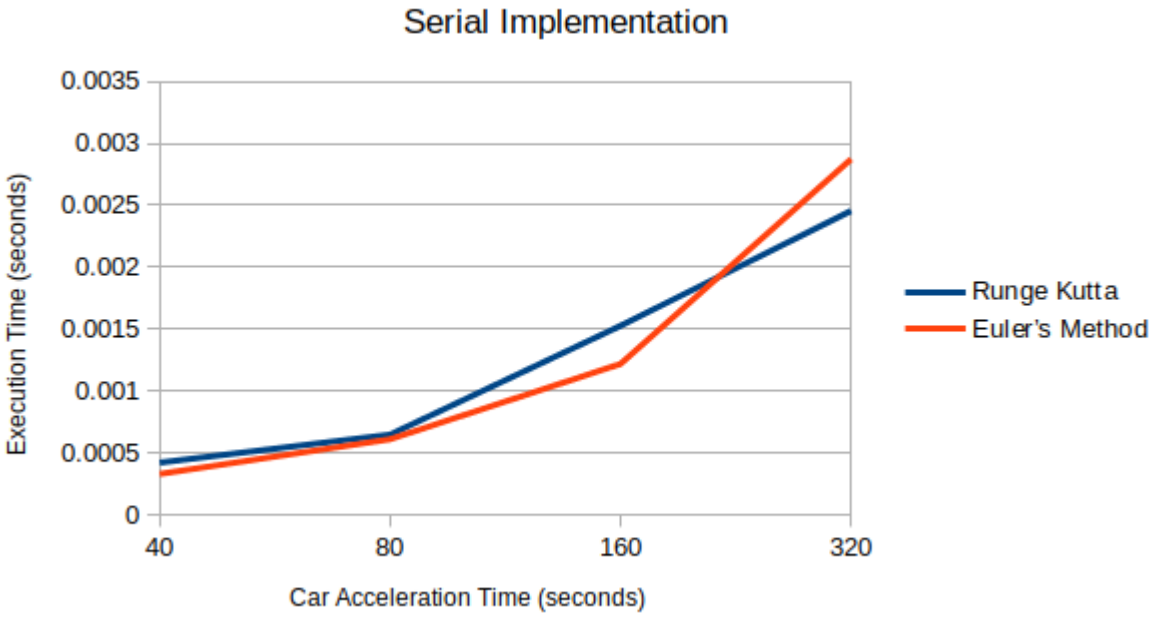

Figura 3: Serie - Euler's Method vs 4th order Runge Kutta

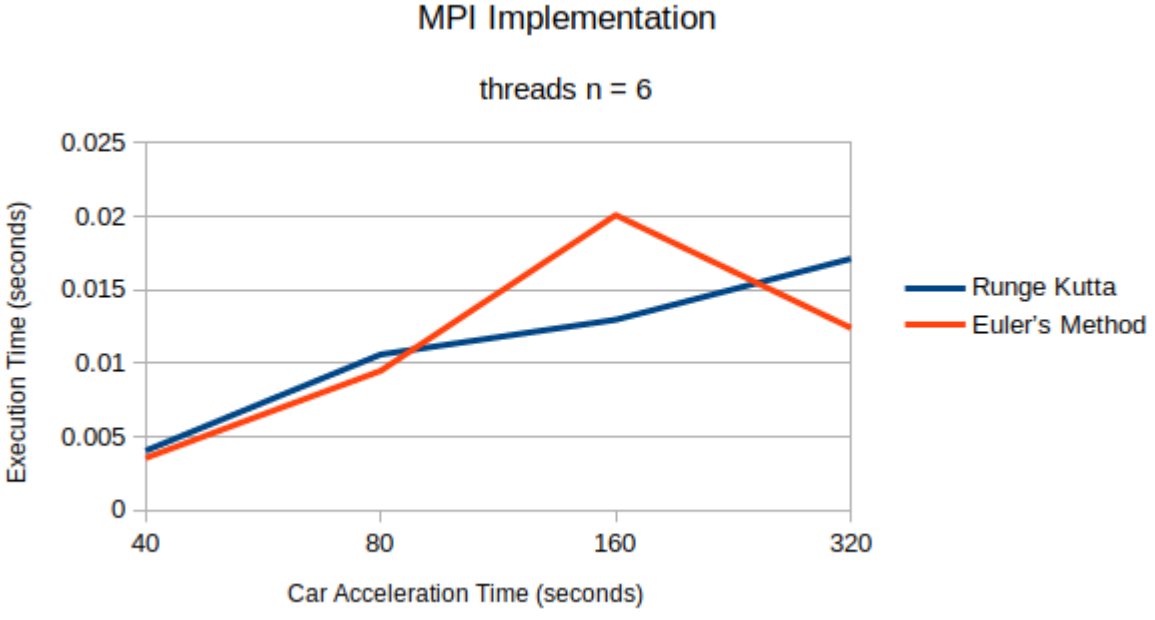

Figura 4: MPI - Euler's Method vs 4th order Runge Kutta

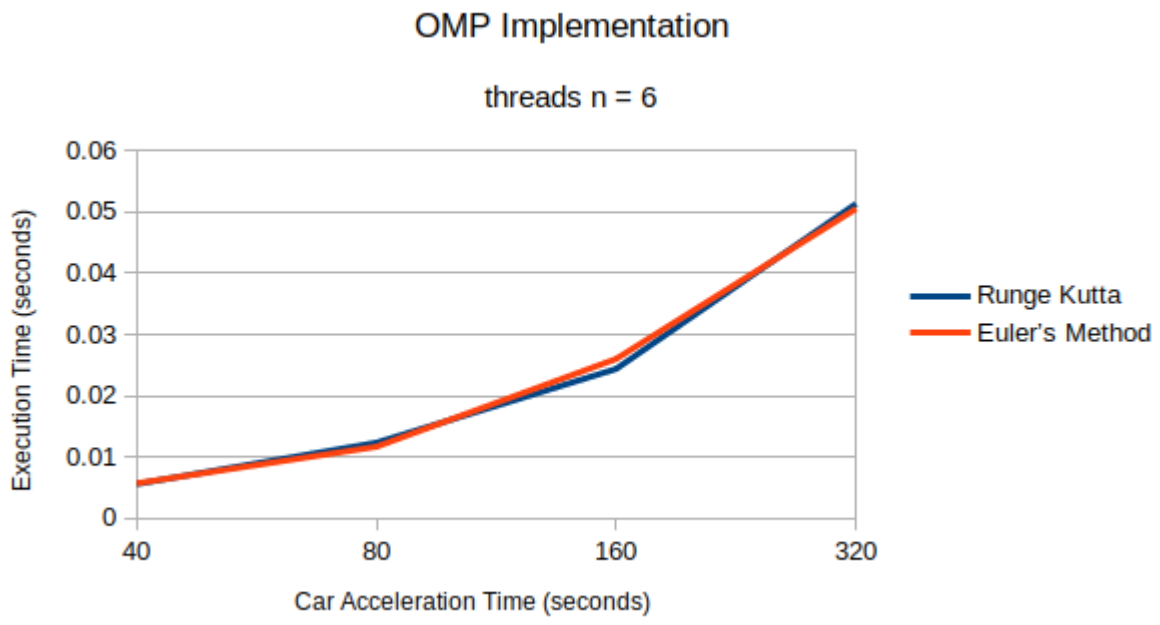

Figura 5: OMP - Euler's Method vs 4th order Runge Kutta

4. Conclusions

In conclusion thanks to this project we can see that not always the parallelization of a code is faster than its serial computation. Maybe it was caused due to the lack of difficulty of the problem. The car simulation here exposed just implement parallelization for the ODE solver. and inside the code there is no too much to parallelize. Nevertheless it is not the common case, most times the parallelization of a code is faster when the percentage of parallelism represent a great part of the code. It is also important to mention that the Runge Kutta showed an important execution time, being faster than Euler's method in its parallel form. It is important to mention that the implementation of the Euler's method shows a little bit of errors more than the 4th order Runge Kutta implementation. For future works we recommend to use the famous 4th order Runge Kutta solver instead Euler's method thanks to its accuracy in its approximation and also because to its execution time.

Note: The code and all the data generated is uploaded on osiris.roman folder on Quinde supercomputer.