Computational Efficiency of Three Programming Languages in N-body Simulation

Junnan Pan and Zhongmu Li*

Institute for Astronomy and History of Science and Technology, Dali University, Dali 671003, China

*Corresponding author email: zhongmuli@126.com

Abstract. The computational efficiency of FORTRAN, C and Python languages in N-body simulation is investigated. The potential of these languages to promote the research of N-body simulation is therefore shown by this paper. Our work utilizes Particle-Particle (PP) algorithm, which not only balances the accuracy and efficiency, but also simplifies the traditional numerical calculation. The experimental results show that the computational efficiency of the three is almost the same in the case of a small number of particles, but FORTRAN shows the highest computational efficiency in the case of a large number of particles. The efficiency of Python is the lowest among three languages. The result suggests that FORTRAN is the best choice for N-body simulation, and Python should be used after optimizing the algorithm or working on extreme high-performance computers.

Keywords: N-body simulation; PP algorithm; Numerical calculation; FORTRAN; C; Python.

1. Introduction

N-body simulation is to calculate the interaction and motion of N particles, which is one of the most basic problems in astrophysics, hydrodynamics and molecular dynamics (Chen et al. 2021[1], Li et al. 2021[2], Ballone et al. 2021[3]). The research of N-body simulation helps people well understand the birth of the universe and provides theoretical support for the evolution of binary stars and mathematical physics (Li et al. 2020[4], Tasif et al. 2020[5]). In practical application, it can not only provide theoretical guidance for satellite launch, but also make space rocket orbit (Salazar et al. 2021[6]). Therefore, N-body simulation is of great significance to social and scientific development. However, the traditional numerical algorithm of N-body simulation in astronomy is very complex and needs a lot of astrophysical knowledge (Aarseth et al. 1994[7]). This limits the research population and slows down the speed of N-body simulation research, significantly. How to simplify the N-body program, promote the simulation, improve efficiency and expand the researchers has become an important issue of N-body studies. There are many candidate programming languages for N-body simulation. We want to study three of them in this paper. FORTRAN is a famous one. It is widely used in astronomy researches because of its high numerical calculation efficiency. C is also widely used in different areas. It is not only a flexible and practical language, but also the basis of most programming languages. Python is a newer language that is widely used in recent years. It has the characteristics of portability, scalability and rich code base. It has been applied successfully in the field of, e.g., network programming and artificial intelligence. This paper attempts to study C, Python and FORTRAN, as they have the potential to promote the research of N-body simulation.
In practice, N-body simulation needs high accuracy and efficiency. The common numerical methods in N-body simulation are PP algorithm (Isigaki et al. 2020[8]), Barnes-Hut (BH) algorithm (Rodriguez et al. 2021[9]) and symplectic algorithm (Tamayo et al. 2019[10]). They affect the accuracy and efficiency significantly. Although BH algorithm can quickly calculate the force of each point, the calculation accuracy is not high. Symplectic algorithm can keep the Hamiltonian system strictly, but the energy error can not be controlled within the precision criterion. With the development of high performance computing technology, CPU-GPU technology is used to realize large-scale computing in N-body simulation (Bocharo et al. 2015[11]). But this technology has high cost and can not achieve load balancing, so it is not suitable for N-body promotion. The idea of PP method is simple, and the numerical solution can be obtained by a variety of techniques. Thus it is used in many N-body codes. Besides numerical method, language has obvious effect. However, the difference of efficiency of different languages remains unclear. The innovation of this paper is to study the efficiency of the previously mentioned programming languages in N-body simulation. PP method will be used for the experiments. Based on this, this paper gives the calculation efficiency comparison of FORTRAN, C and Python in N-body simulation.

The paper is organized as follows. In section 2, we introduce the PP algorithm that is used by this work, and the integration requirements of N-body simulation. Next, in section 3, we report the experimental results of three programming languages and make a comparative analysis. Finally, we conclude this work in section 4.

2. Algorithm Introduction

2.1. The Basic Physical Principle of PP Algorithm

In the system of N particles, we assume that the masses of N particles are \( m_1, m_2, \ldots, m_n \), and the initial positions are \((x_1, y_1, z_1), (x_2, y_2, z_2), \ldots, (x_n, y_n, z_n)\). The evolution process of these particles under their interaction forces can be calculated. The external force of each particle is calculated by the direct summation of the forces of the other \( N-1 \) particles. This force leads to changes in the position and velocity of particles. Each calculation needs the mass, velocity and position of each particle. Because this work does need to simulate a real system, we set the initial velocity of each particle as zero. The space distribution of particles on x-y plane can be seen in figure 1.

![Figure 1. View of surrounding particle forces.](image)

In order to accurately determine the positions of particles, the numerical integration of the equation of motion must take a very short time step. We therefore assume that each particle evolution has its own time step. The main requirements of effective integration in N-body simulation are difference scheme, independent time step, neighborhood scheme and close contact. They are all considered in our simulation.

2.2. Difference Scheme

Each particle is described by its mass \( m \), position \( r \), and velocity \( v \). Denoting time derivatives by dots, the equation of motion for particle \( i \) is given by

\[
\ddot{r}_i = -G \sum_j \frac{m_j (r_i - r_j)}{|r_i - r_j|^3}, \ j \neq i
\]
where $G$ is the gravitational constant and the acceleration relates to the sum of other $N-1$ particles.
In the following, we will use the scale unit of $G = 1$ and introduce the force per unit mass $F$. The subscript of the particle is omitted for convenience. According to the $F$ values at the previous four times $t_0, t_1, t_2, t_3$ (where $t_0$ is the latest time), we write out the fourth-order polynomial of time $t$ as

$$F(t) = F(t_0) + D^1(t-t_0) + D^2(t-t_0)(t-t_1) + D^3(t-t_0)(t-t_1)(t-t_2) + D^4(t-t_0)(t-t_1)(t-t_2)(t-t_3)$$

(2)

Considering the subsequent complexity of merging two force polynomials, choosing the fourth-order polynomial is a scheme to balance efficiency and programming workload.

2.3. Independent Time Step
We now design an integral algorithm based on force polynomial. All subsequent methods have their own time steps. In this case, the relative accuracy is maintained in the time range. Thus, the position of each particle is calculated by the largest step of polynomial convergence. A additional requirement of this scheme is to make full coordinate prediction in the intermediate time. The main integral cycle starts with the particle that determines the next integral, i.e., $i = \min \{t_j + \delta t_j\}$. The current time is defined by $i = t_1 + \delta t_1$. In order to prevent full particle search at every step, list $M$ contains all particles satisfying $t_j + \delta t_j < t_m$. Initially $t_m = \delta t_m$, where $\delta t_m$ is a small time interval. A redetermination of the time-step list $M$ is made if $t > t_m$, followed by a second search to include new members. Otherwise, time $t$ might decrease. The independent time step requires two sets of coordinates, which are expressed as primary coordinates and secondary coordinates respectively. They are determined by $r_j(t_j)$ and $r_j(t)$, where the latter is derived from the former by the predictor. All coordinates are predicted to the order $F^{(1)}$ using a fast expression

$$r_j(t) = \left[ \left( \hat{F}^{(1)}_j \delta t_j + \hat{F} \right) \delta t_j + v_j \right] r_j(t_j) + r_j(t), \quad j = 1, \ldots, N$$

(3)

where $F^{(1)} = F^{(1)}/6$, $\hat{F} = F/2$ and $\delta t_j = t - t_j$. The integral cycle is completed by a new time step. The expression used in this paper is as follows

$$\delta t = \left( \frac{\eta F}{\hat{F}} \right)^{1/2}$$

(4)

where $\eta$ is a dimensionless accuracy parameter. The overall accuracy is controlled by parameter $\eta$, which plays the role of relative tolerance. In the case of no close contact, the typical value is set as $\eta = 0.03$. The independent time-step requires a total of 30 variables per particle as follows: $m, r(t_0), r(t), v, v, F, F^{(1)}, D^1, D^2, D^3, t_0, t_1, t_2, t_3$. It is recommended that $r(t_0), r(t), v(t_0), t_0$, as well as the current time $t$, be defined in double precision at little extra cost.

3. Computational Efficiency of Three Languages
This experiment mainly runs in an operating system environment of Ubuntu 18.04.3. System environment configurations are: Intel (R) core (TM) i5-33305 CPU @2.70 GHz, 64 processors, and 7.7 Gigabyte system memory.

In order to compare the computational efficiency of FORTRAN, C and Python in N-body simulation, an initial particle sample is generated by Monte Carlo method. In this work, 4177 particles are produced for testing. The initial mass distribution and position distribution of particles are shown in figure 2 and figure 3 respectively.
Figure 2. Initial mass distribution. 

Figure 3. Initial position distribution of particles.

In figure 2, \(1 \, M_\odot = 2.0 \times 10^{30} \) kg. The mass of the sun is suitable for the mass unit of stars or large objects such as galaxy. In figure 3, \(1 \, \text{PC} = 3.08568 \times 10^{16} \) m. Parsec is a unit of length in astronomy, and it is the most standard method to measure the distance between stars.

Because languages perform differently for various calculation amount, we do N-body simulation with 50 particles and 1000 particles respectively. Figure 4 shows the running time of three programs with 50 particles. 1000 steps are calculated here. The running time of C and FORTRAN programs are 0.46s and 0.83s respectively, but the running time of Python program is 21.03s, which is 45 times and 25 times of the former two respectively.

Figure 4. Relationship between calculation time and total steps, for the case of 50 particles.

The unit of time T is second.

Figure 5 shows the computation time of each step in the simulation of system of 50 particles. The result of Python language is not plotted in the figure because it takes much longer times than C and FORTRAN counterparts.

Figure 5. Calculation time of each step, for the case of 50 particles.

Figure 6 shows the running time of as a function of the step number, for the case of 1000 particles. Similar to the case of 50 particles, the running time of Python program in 1000 particle simulation is
much longer than that of the C and FORTRAN programs.

**Figure 6.** Relationship between calculation time and total steps, for the case of 1000 particles. The unit of time T is second.

Figure 7 shows the computation time of each step in the 1000-particle experiments. As in figure 5, the result of Python code is not plotted as its computation is rather slow. As we see, after 8000 steps, the computation time for a step drops rapidly when FORTRAN program runs, while C language drops slowly. This means that FORTRAN program will run faster than C code after 8000 steps, although C program runs faster in the early steps. At the end of evolution, the computation time for a step is shorter in FORTRAN simulation than C simulation. It indicates that FORTRAN program should be more efficient than C program in simulations of a large number of particles.

**Figure 7.** Calculation time of each step, for the case of 1000 particles.

The CPU usage of three kinds of N-body simulations is also studied in this work. We automatically record the CPU usage of each program through the background setting program. Figure 8 shows the CPU utilization rates of three programs. We observe that Python program always uses 100% of the total CPU, while FORTRAN program and C program use around 50% of the total CPU.

**Figure 8.** CPU utilization of three programs, for the case of 1000 particles.
4. Conclusion
This paper compares the computational efficiency of FORTRAN, C and Python, using N-body code based on PP algorithm. It is shown that C and FORTRAN have more advantages than Python in the field of N-body simulation. FORTRAN is more stable and the computing time is shorter. Thus it is better to choose FORTRAN as the primary programming language of N-body simulations.
We also find that when the number of simulated particles is lower than $10^3$, C can be chosen as language of N-body simulation. Python seems not suitable for N-body simulations of a large number of particles, e.g., the simulations of large globular clusters.
We suggest to take the following points into account in future studies of computational efficiency of N-body simulations:
- For the efficiency of python, more efficient algorithms should be studied.
- In order to improve the accuracy, other influence factors of N-body simulation should be investigated further by future works.

Acknowledgments
This work has been supported by the Yunnan Academician Workstation of Wang Jingxiu (No. 202005AF150025), National Natural Science Foundation of China (No. 11863002), and Sino-German Cooperation Project (No. GZ 1284).

References
[1] Chen Y., Li H. and Vogelsberger M., Effects of initial density profiles on massive star cluster formation in giant molecular clouds [J], Monthly Notices of the Royal Astronomical Society, 2021, 502, (04), 6157-6169
[2] Li Z., Shen J., Gerhard O. and Clarke J. P., Gas Dynamics in the Galaxy: Total Mass Distribution and the Bar Pattern Speed [J], arXiv e-prints, 2021, arXiv: 2103.10342
[3] Ballone A., Torniamenti S. and Mapelli M., From hydrodynamics to N-body simulations of star clusters: mergers and rotation [J], Monthly Notices of the Royal Astronomical Society, 2021, 501, (02), 2920-2933
[4] Li Z., N-body Simulation of a Young Open Cluster [J], Journal of Physics Conference Series, 2020, (1624), 022036
[5] Tasif T. and Elgohary T., An Adaptive Analytic Continuation Method for Computing the Perturbed Two-Body Problem State Transition Matrix [J], Journal of the Astronautical Sciences, 2020, 67, (4), 1412-1444
[6] Salazar F., Alkhaja A., Fantino E. and Alessi E. M., Science orbits in the Saturn-Enceladus circular restricted three-body problem with oblate primaries [J], Acta Astronautica, 2021, 180, 398-416
[7] Aarseth S. J., Direct methods for N-body simulations [J], Lecture Notes in Physics, Berlin Springer Verlag, 1994, 433, 277-312
[8] Isigaki Y., Kominami J. and Makino J., Particle-particle particle-tree code for planetary system formation with individual cut-off method: GPLUM [J], arXiv e-print, 2020, arXiv: 2007.15432
[9] Rodriguez S. N., Lliopoulos A. P. and Carlberg K. T., Projection-tree reduced order modeling for fast N-body computations [J], arXiv e-prints, 2021, arXiv: 2103.01983
[10] Tamayo D., Rein H. and Hernandez D. M., Operator splitting methods for numerical integration of weakly perturbed N-body systems [J], AAS/Division of Dynamical Astronomy Meeting, 2019, 51, 304.03
[11] Bocharov A. N., Balakirev B. A. and Bityurin V. A., Numerical simulation as an important tool in developing novel hypersonic technologies [J], Journal of Physics Conference Series, 2015, 653, 012070