On the Efficient Parallel Computing of Long Term Reliable Trajectories for the Lorenz System

Ivan Hristov[1], Radoslava Hristova[1], Stefka Dimova[1], Peter Armyanov[1], Nikolay Shegunov[1], Igor Puzynin[2], Taisia Puzynina[2], Zarif Sharipov[2], Zafar Tukhliev[2]

[1] Sofia University, Faculty of Mathematics and Informatics, Bulgaria
[2] JINR, Laboratory of Information Technologies, Dubna, Russia

ivanh@fmi.uni-sofia.bg, zarif@jinr.ru

Abstract. In this work, we propose an efficient parallelization of multiple-precision Taylor series method with variable stepsize and fixed order. For given level of accuracy the optimal variable stepsize determines higher order of the method than in the case of optimal fixed stepsize. Although the used order of the method is greater than that in the case of fixed stepsize, and hence the computational work per step is greater, the reduced number of steps gives less overall work. In addition, the greater order of the method is beneficial in the sense that it increases the parallel efficiency. As a model problem, we use the paradigmatic Lorenz system. With 256 CPU cores in Nestum cluster, Sofia, Bulgaria, we succeed to obtain a correct reference solution in the rather long time interval – [0,11000]. To get this solution we perform two large computations: one computation with 4566 decimal digits of precision and 5240-th order method, and second computation for verification – with 4778 decimal digits of precision and 5490-th order method.

Keywords: Parallel Computing, Multiple Precision, Variable Stepsize Taylor Series Method, Lorenz System.

1 Introduction

Multiple precision Taylor series method is an affordable and very efficient numerical method for integration of some classes of low dimensional dynamical systems in the case of high precision demands [1], [2]. The method gives a new powerful tool for theoretical investigation of such systems.

A numerical procedure for computing reliable trajectories of chaotic systems, called Clean Numerical Simulation (CNS), is proposed by Shijun Liao in [3] and applied for different systems [4], [5], [6]. The procedure is based on multiple precision Taylor series method. The main concept for CNS is the critical predictable time T_c, which is a kind of practical Lyapunov time. T_c is defined as the time for decoupling of two trajectories computed by two different numerical schemes. The CNS works as follows. An optimal fixed stepsize is chosen. Then estimates of the required order of the method N and the required precision (the number of exact decimal digits K of the floating-point numbers) are obtained. The optimal order N is estimated by computing the $T_c - N$ dependence by means of the numerical solutions for fixed large enough K. The estimate of K is obtained by computing the $T_c - K$ dependence by means of the numerical solutions for fixed large enough N. This estimate of K is in fact an estimate for the Lyapunov exponent [7]. For given T_c the solution is then computed with the estimated N and K and after that one more computation with higher N and K is performed for verification. The choice of N and K ensures that the round-off error and the truncation error are of the same order.

When very high precision and very long integration interval are needed, the computational problem can become large. In this case, the parallelization of the Taylor series method is an important task and needs to be carefully developed. The first parallelization of CNS is reported in [8] and later improved in [9]. A pretty long reference solution for the paradigmatic Lorenz system, namely in the time interval [0,10000], obtained in about 9 days and 5 hours by using the computational resource of 1200 CPU cores, is given in [10]. However, no details of the parallelization process are given in [8], [9], [10]. In our recent work [11] we reported in details a simple and efficient hybrid MPI + OpenMP parallelization of CNS for the Lorenz system and tested it for the same parameters as those in [10]. The results show very good efficiency and very good parallel performance scalability of our program.

This work can be regarded as a continuation of our previous work [11], where fixed stepsize is used. Here we make a modification of CNS with a variable stepsize and fixed order following the simple approach given in [12]. Although the used order of the method is greater than that in the case of fixed stepsize, and hence the computational work per step is greater, the reduced number of steps gives less overall work. In addition, the greater order of the method is beneficial in the sense that it increases the parallel efficiency. With 256 CPU cores in Nestum cluster, Sofia, Bulgaria, we succeed to obtain a correct reference solution in [0,11000] and in this way we improve the results from [10]. To obtain this solution we performed two large computations: one computation with 4566 decimal digits of precision and 5240-th order method, and second computation for verification – with 4778 decimal digits of precision and 5490-th order method for verification. The computations lasted $\approx$ 9 days and 18 hours and $\approx$ 11 days and 7 hours, respectively. Let us note that the improvement of the numerical algorithm does not change the parallelization strategy from our previous work [11], where the parallelization process is explained in more details. The difference from the previous parallel program is one additional OpenMP single section with negligible computational work, which computes the optimal step.

It is important to mention that although our test model is the classical Lorenz system, the proposed parallelization strategy is rather general – it could be applied as well to a large class of chaotic dynamical systems.

2 Taylor series method and CNS for the Lorenz system

We consider as a model problem the classical Lorenz system [13]:

$$\begin{aligned} \frac{dx}{dt} &= \sigma\,(y - x) \\ \frac{dy}{dt} &= Rx - y - xz \\ \frac{dz}{dt} &= xy - bz \end{aligned} \tag{1}$$

where $R = 28$, $\sigma = 10$, $b = 8/3$ are the standard Salztman's parameters. For these parameters, the system is chaotic. Let us denote with x_i, y_i, z_i, $i = 0, ..., N$ the normalized derivatives (the derivatives divided by **i!**) of the approximate solution at the current time t. Then the N-th order Taylor series method for (1) with stepsize τ is:

$$\begin{aligned} x(t+\tau) &\approx x_0 + \textstyle\sum_{i=1}^{N} x_i \tau^i \\ y(t+\tau) &\approx y_0 + \textstyle\sum_{i=1}^{N} y_i \tau^i \\ z(t+\tau) &\approx z_0 + \textstyle\sum_{i=1}^{N} z_i \tau^i \end{aligned} \tag{2}$$

The i-th Taylor coefficients (the normalized derivatives) are computed as follows. From system (1) we have

$$x_1 = \sigma(y_0 - x_0)$$

$$y_1 = Rx_0 - y_0 - x_0 z_0$$

$$z_1 = x_0 y_0 - b z_0$$

By applying the Leibniz rule for the derivatives of the product of two functions, we have the following recursive procedure for computing x_{i+1}, y_{i+1}, z_{i+1} for $i = 0, ..., N\text{-}1$:

$$\begin{aligned} x_{i+1} &= \frac{1}{i+1}\sigma(y_i - x_i) \\ y_{i+1} &= \tfrac{1}{i+1}\left(Rx_i - y_i - \textstyle\sum_{j=0}^{i} x_{i-j} z_j\right) \\ z_{i+1} &= \tfrac{1}{i+1}\left(\textstyle\sum_{j=0}^{i} x_{i-j} y_j - b z_i\right) \end{aligned} \tag{3}$$

To compute the *i+1*-st coefficient in the Taylor series we need all previous coefficients from 0 to *i*. In fact, this algorithm for computing the coefficients of the Taylor series is called automatic differentiation, or also algorithmic differentiation [14]. It is obvious that we need $O(N^2)$ floating point operations for computing all coefficients. The subsequent evaluation of Taylor series with Horner's rule needs only $O(N)$ operations.

Let us now explain how we choose the stepsize τ. We use a variable stepsize strategy, which makes the method much more robust then in the fixed stepsize case. We use a simple strategy taken from [12], which ensures both the convergence of the Taylor series and the minimization of the computational work per unit time. If we denote the vector of the normalized derivatives of the solution with $X_i = (x_i, y_i, z_i)$ and take a safety factor 0.993, then the stepsize τ is determined by the last two terms of the Taylor expansions [12]:

$$\tau = \frac{0.993}{e^2} \min\left\{ \left(\frac{1}{||X_{N-1}||_\infty} \right)^{\frac{1}{N-1}}, \left(\frac{1}{||X_N||_\infty} \right)^{\frac{1}{N}} \right\} \tag{4}$$

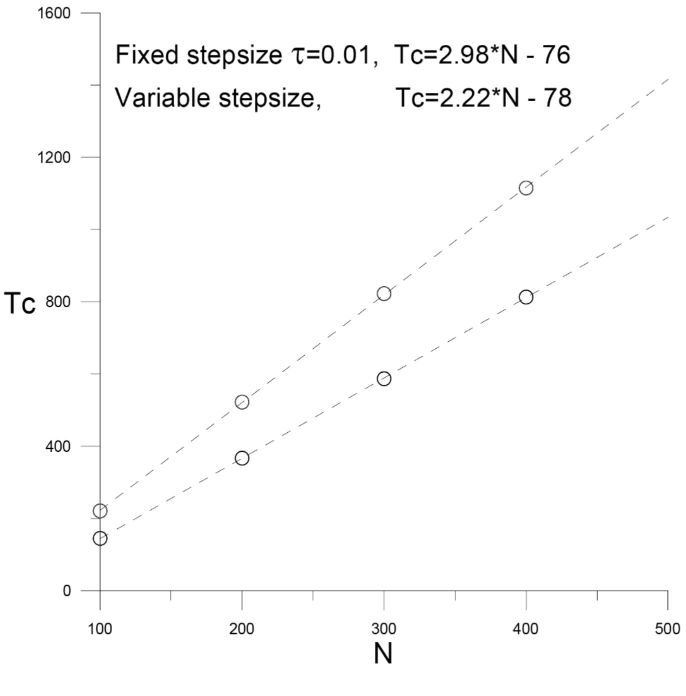

Fig. 1. $T_c - N$ dependencies for fixed and variable stepsize.

In [12] the order of the method is determined by the local error tolerance. However, we do not work explicitly with some local error tolerance and, we do not use any explicit dependence between the local and the global error. Instead of this, as in [3], we compute an a priori estimate of the needed order of the method for a reliable solution. As said before, the critical predictable time T_c is defined as the time for decoupling of two trajectories computed by two different numerical schemes (in this case – by different N). The solutions are computed with large enough precision to ensure that the truncation error is the leading one. As a criterion for decoupling time, we choose the time for establishing only 30 correct digits. The obtained $T_c - N$ dependencies for fixed stepsize $\tau = 0.01$ and variable stepsize are shown in Figure 1. As seen from this figure, the computational work for one-step in the case of variable stepsize is $\approx 80\%$ greater than in the case of fixed stepsize – $(2.98/2.22)^2 \approx 1.80$. However, the reduced number of steps gives less overall work. In addition, the greater order of the method is beneficial in the sense that it increases the parallel efficiency. The reason is that with increasing the order N of the method, the parallelizable part of the work becomes relatively even larger than the serial part and the parallel overhead part.

Similarly, we compute an a priori estimate of the needed precision by means of computing the $T_c - K$ dependence. In this case, we compare the solutions for different K and large enough N. We obtain the dependence $T_c = 2.55K - 81$, which is the same, as expected, for fixed and for variable stepsize.

3 Parallelization of the algorithm

The improvement of the numerical algorithm does not change the parallelization strategy from our previous work [11], where the parallelization process is explained in more details. However, as we will see, the variable stepsize not only decreases the computational work for a given accuracy, but also gives a higher parallel efficiency.

Let us store the Taylor coefficients in the arrays **x**, **y**, **z** of lengths N+1. The values of x_i are stored in **x[i]**, those of y_i in **y[i]** and those of z_i in **z[i]**. As explained in [8], [9], the crucial decision for parallelization is to make a parallel reduction for the two sums in (3). However, in order to reduce the remaining serial part of the code and hence to improve the parallel speedup from the Amdal's law, we should utilize some limited, but important parallelism. We compute **x[i+1]**, **y[i+1]**, **z[i+1]** in parallel. Moreover, we compute **x[i+1]** in advance, before computing the sums in (3), when during the reduction process some of the computational resource is free. In the same way we compute in advance **Rx[i] – y[i]** from the formula for **y[i + 1]** and **bz[i]** from the formula for **z[i + 1]**. These computations are taken in advance; because multiplication is much more expensive than the other used operations, such as division by an integer number is not so expensive. The three evaluations by Horner's rule for the new **x[0], y[0], z[0]** are also done in parallel.

In this work we consider a hybrid MPI + OpenMP strategy [15], [16], i.e., every MPI process creates a team of OpenMP threads. For multiple precision floating-point arithmetic, we use GMP library (GNU Multiple Precision library) [17]. The main reason to consider a hybrid strategy, rather than a pure MPI one, is that OpenMP performs slightly better than MPI on one computational node. For packing and unpacking of the GMP multiple precision types for the MPI messages, we rely on the tiny MPIGMP library of Tomonori Kouya [18], [19], [20], [21].

It is important to note that for our problem the pure OpenMP parallelization has its own importance. First, the programming with OpenMP is easier, because it avoids the usage of libraries like MPIGMP. Second, since the algorithm does not allow domain decomposition, the memory needed for one computational node is multiplied by the number of the MPI processes per that node, while OpenMP needs only one copy of the computational domain and thus some memory is saved.

The sketch of our parallel program is given in Figure 2. Every thread gets its **id** and stores it in **tid** and then the loop with index **i** is performed. Every MPI process takes its portion – the first and the last index controlled by the process. After that the directive **#pragma omp for** shares the work for the loop between threads.

Although OpenMP has a build-in reduction clause, we cannot use it, because we use user-defined types for multiple precisions number and user-defined operations. A manual reduction by applying a standard tree based parallel reduction is done. We use containers for the partial sums of every thread and these containers are shared. The containers are stored in the array **sum**. We have in addition an array of temporary variables **tempv** for storing the intermediate results of the multiplications. To avoid false sharing, a padding strategy is applied [16]. At the point where each process has computed its partial sums, we perform MPI_ALLREDUCE between the master threads [15]. It is useful to regard MPI_ALLREDUCE as a continuation of the tree based reduction process, which starts with the OpenMP reduction. Communications between master threads are overlapped with some computations for **x[i+1]**, **y[i+1]**, **z[i+1]** that can be taken before the computation of the sums in (3) is finished. When the MPI_ALLREDUCE is finished, we compute in parallel the remaining operations for **x[i+1]**, **y[i+1]**, **z[i+1]**.

In between the block which computes the Taylor coefficients and the block which computes the new values of **x[0]**, **y[0]**, **z[0]** in parallel, we compute the new optimal stepsize within an **omp single** section. While the block for computing the Taylor coefficients is $O(N^2)$ and the block for evaluations of the polynomials is $O(N)$, this block is only $O(1)$ and hence the work is negligible. Let us note that the GMP library does not offer a **power** function for the computations from formula (4). The good thing is that we do not need to compute the stepsize with multiple precision and double precision is enough. Therefore, we use the C standard library function **pow** in double precision. We do a normalization of the large GMP floating point numbers in order to work in the range of the standard double precision numbers. The C-code in terms of GMP library of our hybrid MPI + OpenMP program can be downloaded from [22].

Let us mention that if one half of the OpenMP threads computes one of the sums in (3) and the other half computes the other sum, one could also expect some small performance benefit, because for the small indexes **i** the unused threads will be less and the difference from the perfect load balance between threads will be less. However, the last approach is not general because it strongly depends on the number of sums for reduction (two in the particular case of the Lorenz system) and the number of available threads.

```
#pragma omp parallel private(i,j,tid)
{
   tid = omp_get_thread_num();
   for (i = 0; i<N; i++)
   {
      // Every process takes its portion of indexes
      #pragma omp single
      {
        istart=(rank*(i+1))/size;
        ifinal=((rank+1)*(i+1))/size-1;
      }
      # pragma omp for
      for (j=istart; j<=ifinal; j++)
      {
        mpf_mul(tempv[pad*tid],x[i-j],z[j]);
        mpf_add(sum[pad*tid],sum[pad*tid],tempv[pad*tid]);
        mpf_mul(tempv[pad*tid],x[i-j],y[j]);
        mpf_add(sum[pad*tid+1],sum[pad*tid+1],tempv[pad*tid]);
      }
      //Explicit OpenMP Parallel Reduction for log(p) additions
      // The result of reduction is in sum[0] and sum[1]
      ...................................................
      #pragma omp barrier
      ...................................................
      //MPI_ALLREDUCE for sum[0] and sum[1]
      //Communication is only between master threads
      //Communication is overlapped with some computations
      //for x[i+1],y[i+1],z[i+1] that can be taken in advance
      ....................................................
      #pragma omp barrier
      ....................................................
      // The rest computations
      // for y[i+1],z[i+1] independently in parallel
      ..................................................
      #pragma omp barrier
      // Setting sum[pad*tid] and sum[pad*tid+1] to zero
      ..................................................
   }
   #pragma omp single
   {
      // Computing the optimal stepsize
      // from x[N-1],y[N-1],z[N-1],x[N],y[N],z[N]
   }
   // One step forward with Horner's rule
   #pragma omp sections
   {
      // Computing the new x[0],y[0],z[0]
      // independently in three parallel sections
   }
}
```

Fig. 2. The sketch of hybrid MPI+OpenMP code in terms of GMP library.

4 Computational resources. Performance and numerical results

The preparation of the parallel program and the many tests are performed in the **Nestum** Cluster, Sofia, Bulgaria [23] and in the **HybriLIT** Heterogeneous Platform at the Laboratory of IT of JINR, Dubna, Russia [24]. The large computations for the reference solution in the time interval [0,11000] and the presented results for the performance are from **Nestum** Cluster. **Nestum** is a homogeneous HPC cluster based on two socket nodes. Each node consists of 2 x Intel(R) Xeon(R) Processor E5-2698v3 (Haswell-based processors) with 32 cores at 2.3 GHz. We have used Intel C++ compiler version 17.0, GMP library version 6.2.0, OpenMPI version 3.1.2 and compiler optimization options -O3 -xhost.

We use the same initial conditions as those in [10], namely $x(0) = -15.8$, $y(0) = -17.48$, $z(0) = 35.64$, in order to compare with the benchmark table in [10]. We computed a reference solution in the rather long time interval [0,11000] and repeated the benchmark table up to time 10000. Computing this table by two different stepsize strategies is a good demonstration that Clean Numerical Simulation (CNS) is a correct and valuable approach for computing reliable trajectories of chaotic systems.

We performed two large computations with 256 CPU cores (8 nodes in Nestum). The first computation is with 4566 decimal digits of precision and 5240-th order method (5% reserve from the a priori estimates). The second computation is for verification – with 4778 decimal digits of precision and 5490-th order method (10% reserve from the a priori estimates). The first computation lasted $\approx$ 9 days and 18 hours and the second $\approx$ 11 days and 7 hours. The overall speedup with 256 cores for the first computation is 162.8, for the second – 164.6.

By estimating the time needed for the same accuracy and with fixed stepsize 0.01, we conclude that by applying variable stepsize strategy we have **2.1x** speedup. There are two reasons for this speedup – less overall work and increased parallel efficiency. Although the work per step in the case of variable stepsize increases by $\approx$ 80%, the average stepsize is $\approx$ 0.034 and thus the overall work is $\approx$ 53% from the work in the case of fixed stepsize 0.01. In addition, the parallel efficiency increases from 55.5% up to 63.6% for the first computation and from 56.2% up to 64.3% for the second. This is because by increasing the order of the method N, we increase the amount of the parallel work, which mitigates the impact of the serial work and the parallel overhead work.

As we computed the reference solution with some reserve of the estimated N and K, we actually obtain the solution with some more correct digits. The reference solution with 60 correct digits at every 100-time units can be seen in [22]. The reference solution at $t = 11000$ is:

x = 6.1062926905568997191778200309537005526718588505397086273550‌8

y =-3.3379535092871242817397497814455236081421054269851246264074‌8

z = 34.160347153258364886745033471071226184091330735824261000528‌5

5 Conclusions

Parallelized version of multiple precision Taylor series method and particularly the Clean Numerical Simulation should be used with a variable stepsize strategy as a better alternative of the fixed stepsize one. An important observation is that variable stepsize not only decreases the computational work for a given accuracy, but also gives a higher parallel efficiency.

6 Acknowledgement

We thank for the opportunity to use the computational resources of the Nestum cluster, Sofia, Bulgaria. We would like to give our special thanks to Dr. Stoyan Pisov for his great help in using the Nestum cluster and Prof. Emanouil Atanassov from IICT, BAS for valuable discussions and important remarks on the parallelization process. We also thank the Laboratory of Information Technologies of JINR, Dubna, Russia for the opportunity to use the computational resources of the HybriLIT Heterogeneous Platform. The work is supported by a grant of the Plenipotentiary Representative of the Republic of Bulgaria at JINR, Dubna, Russia.

References

1. Barrio, R.: Performance of the Taylor series method for ODEs/DAEs. Applied Mathematics and Computation 163.2, 525-545 (2005)
2. Barrio, R., et al.: Breaking the limits: the Taylor series method. Applied mathematics and computation 217.20, 7940-7954 (2011)
3. Liao, S.: On the reliability of computed chaotic solutions of non-linear differential equations. Tellus A: Dynamic Meteorology and Oceanography 61.4, 550-564 (2008)
4. Liao, S.: On the numerical simulation of propagation of micro-level inherent uncertainty for chaotic dynamic systems. Chaos, Solitons & Fractals 47, 1-12 (2013)
5. Liao, S.: On the clean numerical simulation (CNS) of chaotic dynamic systems. Journal of Hydrodynamics, Ser. B 29.5, 729-747 (2017)
6. Li, X., Jing, Y., Liao, S.: Over a thousand new periodic orbits of a planar three-body system with unequal masses. Publications of the Astronomical Society of Japan 70.4, 64 (2018)
7. Wang, P., Li, J.: On the relation between reliable computation time, float-point precision and the Lyapunov exponent in chaotic systems. arXiv preprint arXiv:1410.4919 (2014)
8. Wang, P., Li, J., Li, Q.: Computational uncertainty and the application of a high-performance multiple precision scheme to obtaining the correct reference solution of Lorenz equations. Nu-

merical Algorithms 59.1, 147-159 (2012)
9. Wang, P., Liu, Y., Li, J.: Clean numerical simulation for some chaotic systems using the parallel multiple-precision Taylor scheme. Chinese science bulletin 59.33, 4465-4472 (2014)
10. Liao, S., Wang, P.: On the mathematically reliable long-term simulation of chaotic solutions of Lorenz equation in the interval [0, 10000]. Science China Physics, Mechanics and Astronomy 57.2, 330-335 (2014)
11. Hristov, I., et al.: Parallelizing multiple precision Taylor series method for integrating the Lorenz system. arXiv preprint arXiv:2010.14993 (2020).
12. Jorba, A., Zou, M.: A software package for the numerical integration of ODEs by means of high-order Taylor methods. Experimental Mathematics 14.1, 99-117 (2005)
13. Lorenz, E.: Deterministic nonperiodic flow. Journal of the atmospheric sciences 20.2, 130-141 (1963)
14. Moore, R.: Methods and applications of interval analysis. Society for Industrial and Applied Mathematics (1979)
15. Gropp, W., et al.: Using MPI: portable parallel programming with the message-passing interface (Vol. 1). MIT press (1999)
16. Chapman, B., Jost, G., Van Der Pas, R.: Using OpenMP: portable shared memory parallel programming (Vol. 10). MIT press (2008)
17. GNU GMP library, https://gmplib.org/, last accessed 2021/06/24
18. Kouya, T.: BNCpack, http://na-inet.jp/na/bnc/, last accessed 2021/06/24
19. Kouya, T.: A Brief Introduction to MPIGMP & MPIBNCpack
20. Nikolaevskaya, E., et al.: MPIBNCpack library. Studies in Computational Intelligence 397, pp. 123-134 (2012)
21. Kouya, T.: Performance Evaluation of Multiple Precision Numerical Computation using x86 64 Dualcore CPUs. FCS2005 Poster Session (2005).
22. Article source code, https://github.com/rgoranova/hpcvss, last accessed 2021/06/24
23. Nestum Home Page, http://hpc-lab.sofiatech.bg/, last accessed 2021/06/24
24. HybriLIT Home Page, http://hlit.jinr.ru/, last accessed 2021/06/24