International Journal of Non-Linear Mechanics 80 (2016) 76–81

Contents lists available at ScienceDirect

International Journal of Non-Linear Mechanics

journal homepage: www.elsevier.com/locate/nlm

A first parallel programming approach in basins of attraction computation

P. Belardinelli *, S. Lenci

DICEA, Polytechnic University of Marche, 60131 Ancona, Italy

ARTICLE INFO

Article history:
Received 12 August 2014
Received in revised form
25 October 2015
Accepted 25 October 2015
Available online 30 November 2015

Keywords:
Parallel programming
Computing performance
Duffing's equation
Basins of attraction
Attractors
MPI

ABSTRACT

The paper focuses on the development of a numerical code for the computation of basins of attraction by using the parallel programming. Two different approaches based on the massage passing interface (MPI) standard are presented; the performance analysis presented encourages us to use a massive communication between nodes only for a few-cores architecture. The critical issues arising from the study of a generic dynamical system are discussed while the computation of basins is performed on a benchmark system described by Duffing's equation. We paid attention at the optimization of the computing time as well as the work time load on each node in order to develop a performing and portable code. For the presented codes, both the scalability with an implementation on a professional cluster and the capabilities of the parallelism in the elaborations of basins with a large set of initial conditions have been tested.

1. Introduction and motivations

The analysis of non-linear systems of differential equations is a common task for scientists and engineers of many disciplines, often with the aim to look more carefully at complicated phenomena in their own fields [1]. In particular the study of dynamic attractors and of their basins of attraction represents a key point to get an overall description of the problem and to predict the behavior in several conditions [2–4].

Due to the large computational costs, several attempts have been done to improve the elaboration techniques for building basins of attraction. The full processing of all trajectories by means of numerical integrations is a widely applied algorithm; beside, other routines have been developed to reduce the computational effort. Numerous modifications of the cell-to-cell mapping method introduced by Hsu [5,6] have been proposed in order to reduce the computational time. Recently, Eason and Dick [7] have proposed a parallelized version of the multi-degrees-of-freedom cell mapping able to exploit the parallel threads in multicore modern computers. An application of this method has been tested for a dynamical integrity analysis in a coupled linear oscillator and non-linear absorber system [8].

Here we undertake the computation of basins of attraction, by addressing first the computation itself, looking to develop an efficient algorithm to perform the whole calculation of the trajectories. The powerful tool we want to use, by taking a conscious look at the applicability and at the performances, is the parallel programming in the framework of the high performance computing (HPC). The parallel writing of a code represents a big deal and it means often to rewrite completely existing software. A motivation to change the software constructs is, especially in large scale problems, represented by the evolution of the calculators' architecture. From the early 1970s, an ever increasing performance improvement of computers happened, beside to the computer graphics and with an easier access to these resources. The empirical law that summarize the hardware trend (directly correlated to the number of transistors in microprocessors) was elaborated in 1965 by Moore [9]. Recently, due to both technological and economic limits, the trend cannot be respected any more, thus the core frequency and performance will not grow following Moore's law any longer. The roadmap of chip manufactures is now to reinterpret Moore's law and to increase the number of cores in order to maintain the architectures evolution. Up to now, in order to speed-up an elaboration of a written software, newest powerful processors could be used avoiding any modification on the code [10]. Today the hardware evolution imposes a different approach and the programming itself becomes the key to get better performances [11]. The parallel programming is strictly correlated to the hardware, an indicative example is represented by the copies of processes generated in the message passing; furthermore, in order to take advantage of the newer machines, multiple threads

* Corresponding author.
E-mail addresses: p.belardinelli@univpm.it (P. Belardinelli), lenci@univpm.it (S. Lenci).

http://dx.doi.org/10.1016/j.ijnonlinmec.2015.10.016
0020-7462/

and the use of accelerators and GPU should be implemented. In this work the aim is to exploit the MPI programming interface [12] to develop a parallel code for the computation of basins of attraction. We present two schemes of implementation based on a real-time synchronization between the computing nodes or with *a posteriori* processing.

As a benchmark of our codes we chose the Duffing equation since it can describe many non-linear systems, and it can provide an approximate description of many others [13–15]. It is also an easy benchmark with large amount of results; for example, a study of this equation focused on the basins of attraction and dynamical integrity can be found in [16].

The paper is organized as follows: in Section 2, after an overview on dynamical systems, we discuss the method to apply the parallel programming to the basins computation. The results carried out on the benchmark model are shown in Section 3, while we state our conclusions in Section 4 that round up the paper.

2. Proposed approaches

2.1. Dynamical systems

In the following we briefly introduce some definitions and concepts in order to clearly present the discussion. We consider a non-autonomous initial value problem (henceforth IVP) given by:

$$\begin{cases} \dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y}) \\ \mathbf{y}(t_0) = \mathbf{y}_0 \end{cases} \tag{1}$$

This is a system of ordinary differential equations in time t with $\mathbf{f} : \Psi \subset \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}^n$, where n is the spatial dimension of the problem and dot stands for the time derivative. The equation must satisfy the $(1)_2$ that represents the initial condition $(t_0, \mathbf{y}_0)$ in the domain of $\mathbf{f}$. It is also convenient to introduce the evolution function (or flux) of the dynamical system

$$\Phi : [t_0, t_{max}] \times \Psi \to \mathbb{R}^n, \tag{2}$$

being t_{max} the maximal time for which the IVP exists. The flux $\Phi(\cdot, t_0, \mathbf{y}_0)$ defines a solution curve (or trajectory, or orbit) [17], and it is a solution for the Cauchy problem (1):

$$\begin{cases} \dot{\Phi} = \mathbf{f}(t, \Phi(t, t_0, \mathbf{y}_0)) \\ \Phi(t_0, t_0, \mathbf{y}_0) = \mathbf{y}_0 \end{cases}. \tag{3}$$

The flux allows us to define the attracting set: A, a closed subset of $\mathbb{R}^n$, is an attracting set if $\exists$ a neighborhood V of A such that $\forall \mathbf{y} \in V \Rightarrow \Phi(t, t_0, \mathbf{y}) \in V$ and $\cap \Phi(t, t_0, \mathbf{y}) = A$, $\forall t \geq t_0$. The domain of attraction (basin of attraction) of an attracting set A is defined as the $\bigcup_{t \leq 0} \Phi(t, t_0, \mathbf{y}))$, $\forall \mathbf{y} \in V$ [18]. The operative way to perform the computation of a basin of attraction is to evolve, forward in time and starting from a specific time t_0, the set of all the initial conditions, by looking for their attracting sets.

To permit a practical implementation, we have to discretize, both in time and in space, the continuous problem: this discretization will determine the number of the elaborations to be performed. The initial conditions become a set of cells of fixed size and the density of the cells determines the resolution of the approximated problem. It has to be noted that for a generic non-periodic system, the number of the elaborations to be performed is determined by the spatial dimension of the initial conditions. It is more complex to estimate the sequence if a cell mapping method is preferred.

We have to take care of the competition between the precision (that requires a large partition sets) and the computational cost, in terms of both time and resources. By reducing the cell size, the error in the membership on a wrong basin can be reduced.

The discretization process introduces implicitly errors since all the points belonging to a specific cell are assimilated to a unique initial condition and consequently the result of its time integration will be only one. But issues arise from the practical implementation process; e.g. we have to deal with the uncertain if all the points within the cell can belong at the same basins. Generally the boundaries and the fractal parts are largely affected by the domain discretization whereas compact internal part of a basin are less influenced.

Also the evolution in time, performed with a time integration, requires a steps subdivision. The timing path from the starting t_0 up to the attractor can require numerous steps especially for chaotic attractor [19].

A very dense domain helps to obtain better results, but can overcharge the available resources, this is especially the case for large dimension systems.

As consequence of that, the problem is addressed toward the parallel computing.

2.2. Code architecture

The calculation of a basin is not a well-posed parallelizable process since the attractors belong to the system and must be shared among several computations; however, since the dynamic system must be analyzed numerous times, the aim is to execute in parallel more elaborations we can. The concept in which we have organized the code is sketched in Fig. 1. The mere computation is done by the *computing tasks*, a master process denominated *master of the initial conditions* distributes the work and collects the results, finally, another master, namely the *master of the attractors*, picks up the information about the attractors and acts as coherence operations. Ideally, by computing all the initial states at same time, it is possible to collect simultaneously all the needed information. A pure parallelization is unfortunately unfeasible, i.e. the software must make the most with the available hardware capabilities, by exploiting all its characteristics. The aim is to avoid long queues of tasks by establishing a concurrency of processes. The computational tasks must be managed in a specific way since several factors lead to a different time in each computation, e.g. the discovery of a new attractor, the match with an old attractor, a diverging orbit, and so on, thus one further issue is the balancing of work. We have also to keep in mind that the MPI implementation generates a spread of processes well defined in the code initialization but there are some others numerous unknown processes defined by the operating system that can be scheduled within the running of our software and that can delay or modify the operation order.

The master of the initial conditions (from here called P0) plays the role of a scheduler rules probing the status of the nodes and distributing the initial conditions to elaborate. However the first dispatch of works does not require a status query since the slave processes are free. The P0 acts preliminary a discretization of the range of initial conditions to be elaborated, knowing both the system and the grid dimensions.

Each available worker, in its free state, is waiting instructions from the master while at the end of a computation declares its status and communicates the result. If all the initial conditions have been sent to the workers, further requests of work are killed in order to permit the message passing finalization. Since the memory of each process is private, and the performance of a remote memory access defined in the standard MPI-2 requires too many barriers and large windows of reading, we have to exchange information trying to optimize the time. In view of the above, the communication is based on non-blocking send (MPI_Isend) and blocking receive (MPI_Recv), including a test function on the slaves prior to reuse variables under update [20]. We look for to scale the software up to a large numbers of cores, thus we have decided not to use any broadcasting and gathering collective functions in order to avoid implicit barriers and hardware