



UNIVERSITÀ POLITECNICA DELLE MARCHE
Repository ISTITUZIONALE

A first parallel programming approach in basins of attraction computation

This is a pre print version of the following article:

Original

A first parallel programming approach in basins of attraction computation / Belardinelli, Pierpaolo; Lenci, Stefano. - In: INTERNATIONAL JOURNAL OF NON-LINEAR MECHANICS. - ISSN 0020-7462. - STAMPA. - 80:(2016), pp. 76-81. [10.1016/j.ijnonlinmec.2015.10.016]

Availability:

This version is available at: 11566/236476 since: 2022-05-25T14:34:35Z

Publisher:

Published

DOI:10.1016/j.ijnonlinmec.2015.10.016

Terms of use:

The terms and conditions for the reuse of this version of the manuscript are specified in the publishing policy. The use of copyrighted works requires the consent of the rights' holder (author or publisher). Works made available under a Creative Commons license or a Publisher's custom-made license can be used according to the terms and conditions contained therein. See editor's website for further information and terms and conditions.

This item was downloaded from IRIS Università Politecnica delle Marche (<https://iris.univpm.it>). When citing, please refer to the published version.

(Article begins on next page)

A first parallel programming approach in basins of attraction computation

P. Belardinelli^{a,*}, S. Lenci^a

^a*DICEA, Polytechnic University of Marche, 60131 Ancona, Italy*

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 message passing interface (MPI) standard are presented; the performance analysis presented encourages 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 the 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.

Keywords: Parallel programming, computing performance, Duffing's equation, basins of attraction, attractors, MPI

PACS: 07.05.Bx, 02.30.Hq, 02.60.Cb

Acknowledgements

This work was partially supported by the Italian Ministry of Education, University and Research (MIUR) by the PRIN funded program 2010/11 N.2010MBJK5B "Dynamics, Stability and Control of Flexible Structures".

*Corresponding author

Email addresses: p.belardinelli@univpm.it (P. Belardinelli), lenci@univpm.it (S. Lenci)

1. Introduction and motivations

The analysis of nonlinear 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 behaviour in several conditions [2, 3, 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 a long-time 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 in [5, 6] have been proposed in order to avoid as much as possible long-time integrations. Recently, Dick [7] has proposed a parallelized version of the multi-degrees-of-freedom cell mapping able to exploit the parallel threads in multicores modern computers. An application of this method has been tested for a dynamical integrity analysis in a coupled linear oscillator and nonlinear 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). Unfortunately, the step from an “old style” programming to a parallel writing of a code, represents a big deal and it means often to rewrite completely existing software. A deep motivation to drastically switch the programs constructs, especially in large scale problems, is represented by the evolution of the calculators architecture. From the early 1970s, we have assisted to an ever increasing performance improvement of computers and computer graphics with a growing feasibility to have access at these resources. The empirical law that summarize the trend (directly correlated to the number of transistors in microprocessors) was elaborated in 1965 by Moore [9]. Recently, due to both technological and economic limit, the trend is not respected any more, thus the core frequency and performance will not grow following the Moore’s law any longer. The roadmap of chip manufactures is now to reinterpret the Moore’s law and to increase the number of cores in order to maintain the architectures evolution. In past years a comfortable way to increase the performance of software was simply to wait the newest and powerful processor in the market: no lines in the

code were modified and the code was speeding-up [10]. Today the hardware evolution imposes a different approach to get better performance and the programming becomes the key [11]. Since the parallel programming is strictly correlated to the hardware, it is a little bit involved: message passing, threads, use of accelerators and GPU are examples of techniques that must be implemented to take advantage of the newer machines. In this work we 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 nonlinear systems, and it can provide an approximate description of many others [13, 14, 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:

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

This is a system of ordinary differential equations in time t with $\mathbf{f} : \Psi \subset \mathbb{R} \times \mathbb{R}^n \rightarrow \mathbb{R}^n$, 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

$$\Phi : [t_0, t_{max}] \times \Psi \rightarrow \mathbb{R}^n, \quad (2)$$

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):

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

The flux allows to define the attracting set: A , a closed subset of \mathbb{R}^n , is an attracting set if \exists a neighbourhood V of A such that $\forall \mathbf{y} \in V \Rightarrow \Phi(t, t_0, \mathbf{y}) \in V$ and $\bigcap_{t \geq t_0} \Phi(t, t_0, \mathbf{y}) = A$. The domain of attraction (basin of attraction) of an attracting set A is defined as the $\bigcup_{t \geq t_0} \Phi(t, t_0, \mathbf{y})$, $\forall \mathbf{y} \in V$ [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.

2.2. The discretized system

In previous subsection we provided mathematical definitions, that involved sets or subsets of \mathbb{R}^n . To permits practical implementation, it is obvious that we have to simplify, namely to discretize both in time and in space, the problem.

For a generic non-periodic system, the number of the elaborations is determinated by the spatial dimension of the initial conditions. This set is discretized by means of a number of cells of fixed size, and the whole (hyper)volume of each cell refers to an unique initial condition commonly that in its centroid. Clearly the density of the cells determines the resolution with which the problem is approximated, and 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.

The discretization process introduces implicitly errors. From the point of view of the starting conditions, 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 really do all the points within the cell belong at the same basins? In the compact internal part of a basin generally this does not entail any kind of problems, but the boundaries and the fractal parts are largely affected by the size of the domain discretization. By reducing the cell size, the error in the membership on a wrong basin can be reduced, but never eliminated. Furthermore, also the identification of the attractors depends on the grid, although to a minor extent. Since it is nothing else that a set of points with specific properties, directly follows, from the size of the cells related to these points, a sort on uncertain on the attractor determinations. From the one side, a very filled domain, helps to

obtain better results, but, from the other side, this can overcharge the available resources, especially for a large sets of initial conditions and for high dimension systems.

Finally, 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]. The previous facts, even more, address the problem approach towards the parallel computing.

2.3. Code architecture

Computing tens, hundreds, thousands or millions of times almost the same evolving system, it may seem for its own nature a parallelizable process. This is only in part true since the attractors belong to the system itself and, in some way, must be shared between all the several computations. The concept in which we have organized the code is quite simple and sketched in Figure 1. There are three main characters involved in the process: 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 a sort of coherence operation. Ideally being the number of computing task equal to the number of initial states to be investigated, we could collect all the needed information almost at the same time. Several operating problems inhibits a pure parallelization. First of all we remark that up to here we have spoken in terms of processes and tasks but not of cores or nodes; the software should, somehow, make the most with the hardware capabilities, by exploiting all its characteristics. This permits, by establishing a concurrency of processes, to avoid long queues of tasks. One could think that the management of the computational tasks is redundant since we could assign in blind mode to the “workers” a slice of the initial conditions set. But several factors lead to a different time in each computation, e.g. the discover 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.

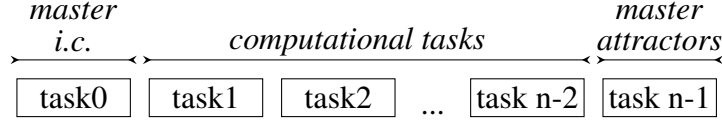


Figure 1: Organization of the tasks for a multi-cores environments.

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 nonblocking 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 to do not use any broadcasting and gathering collective functions to avoid implicit barriers and interruptions due to hardware problem. The only synchronous constrain is thus a wait for the actual receipt by the master of the result.

As soon as an initial conditions is received by a computation tasks (P1, P2, ... Pn-2), a routine for ordinary differential equations based on a C library [21], integrates ahead in time the system. Within the process the method used to compute the trajectory is based on [22], but since the memory of each process cannot be shared, the processes are completely independent. The classical optimization for loops, the hierarchical use of the memory and the common compiling optimization flags have been adopted, as usual in an high performance computing.

As previously discussed, the MPI is not deterministic in time and however the discovery of attractors cannot be *a priori* assigned or scheduled. We have to deal with the redundant discovery of attractors in similar or dissimilar instants of time. Diverging orbits can be considered converging towards the same infinite attractor and the information can be sent directly to the master without problems.

Sharing the information about the attractors represent one of key points of the software. Two variants of the code have been tested to handle with this. In both codes the master of attractors (P_{n-1}) is the chief in charge of to collect and to manage the information. The two codes differ in the way they gather and assembly the data in order to finally obtain one single coherent basins. A functional scheme for the code “A” is shown in Figure 2. Process P_{n-1} collects the attractors by means of I/O operations on file avoiding from the one side the bottleneck in the case of synchronous discovery, and from the other a barrier for the computing tasks. The weaknesses are represented by: i) The synchronization is done only at the end of the whole process introducing a further operation. ii) There are I/O operations commonly slowly in a standard network interconnecting nodes.

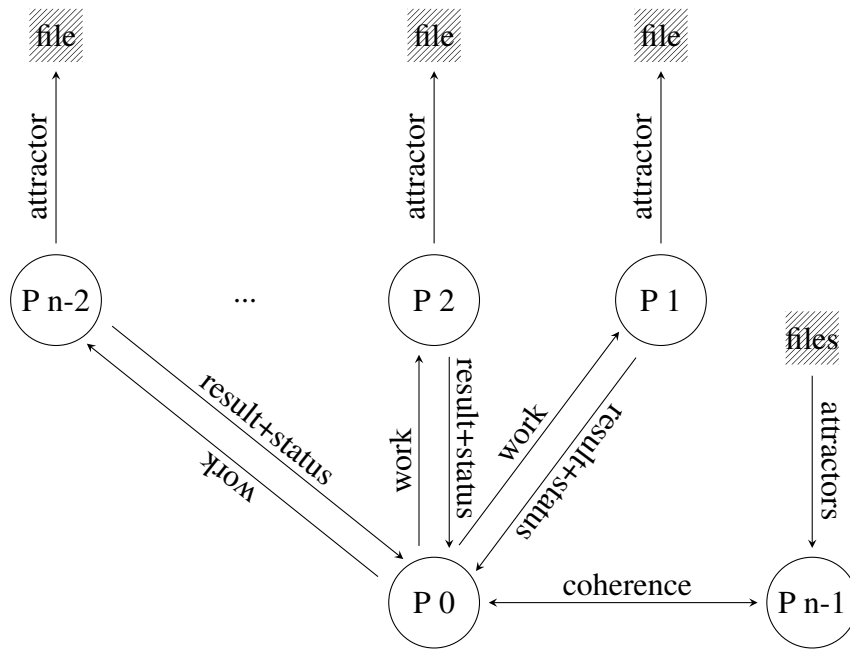


Figure 2: Schematic functional drawing for the code “A”. Each circle represents a process, while hatched zones are I/O operations.

The code “B” follows the scheme of Figure 3. The synchronization happens at each new discovery of an attractor. The process P_{n-1} merges in a database all the information and further performing an update of all the computing nodes. Drawbacks of this approach are: i) The database update is quite slow and a pure serial operation. ii) The update operations break the normal computing work performed by the slaves. iii) A coherence between P_0 and P_{n-1} is still mandatory to

fix the results received prior a full update. iv) The traffic is larger than that of the code “A”. v) The computational time depends on the time instants in which an attractor is discovered. The idea of the code “B” is to instruct all computing nodes as soon as possible and avoid further communication with the attractors database. By doing this all the remaining computations are coherent and can be sent directly as a final result.

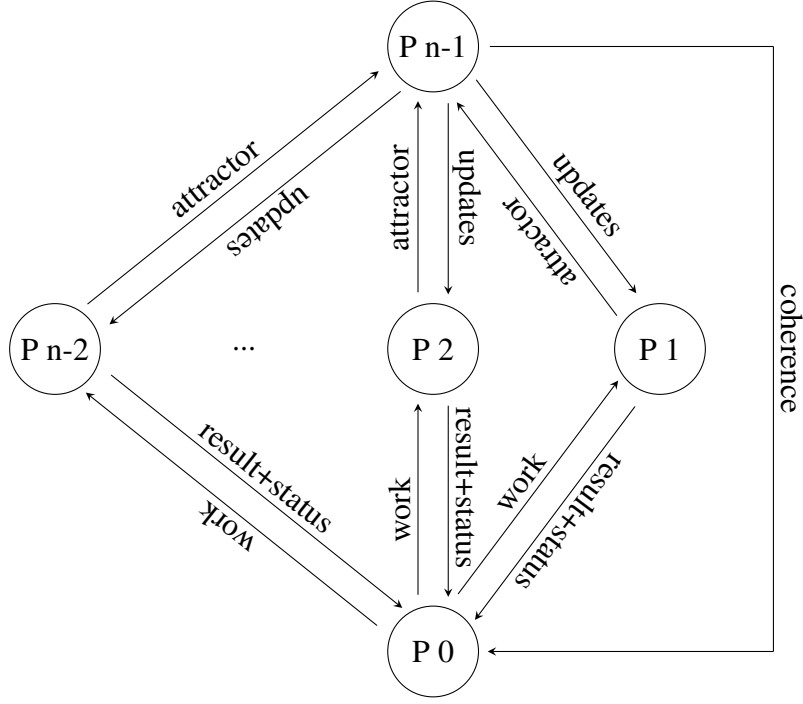


Figure 3: Schematic functional drawing for the code “B”. Each circle represents a process.

3. Results

3.1. Benchmark model

We test the developed software using the Duffing equation [13] as a benchmark:

$$\ddot{y} + 2\zeta\dot{y} - y + \gamma y^3 = F \cos \Omega t. \quad (4)$$

Since the aim of the paper is not a thorough investigation of the eq. (4), we fix all the parameters (see Tab. 1).

parameter	value	description
ζ	0.025	Damping ratio
γ	1	Nonlinear stiffness
F	0.065	Excitation amplitude
Ω	1.15	Excitation (circular) frequency

Table 1: Parameters for the Duffing's equation

Equation (4) is not in the form of eq. (1)₁, but it can be easily written as a system of two first order equations ($n = 2$ and $\mathbf{y} = \{y_1, y_2\}^T$):

$$\begin{cases} \dot{y}_1 = y_2 \\ \dot{y}_2 = -2\zeta y_2 + y_1 - \gamma y_1^3 + F \cos \Omega t \end{cases} \quad (5)$$

where $y_{1,2}$ are respectively the displacement and the velocity of the system. In Fig. 4 we show the basins of attractions for the system (4) obtained with our parallel code “A”. The figure, by means of a sequence of images, illustrates the variation of the basins with respect to the starting time t_0 . Each color represents a basins of attraction of a specific attractor. Since the periodicity of the benchmark problem, the basins, after a period 2π , are coincident.

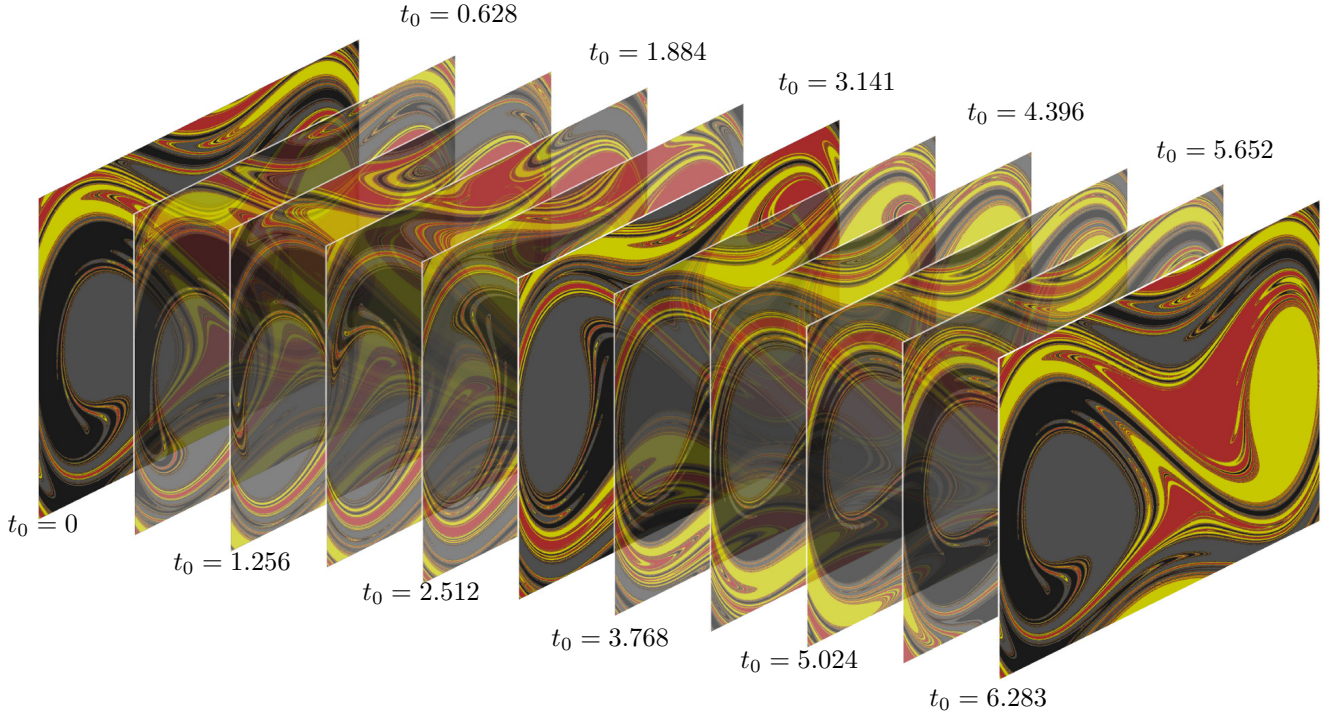


Figure 4: Basins of attraction for the system (4) with $y_1 \in [-1.4, 1.4]$, $y_2 \in [-1, 1]$. Initial time varying with steps of $2\pi/10$. The spatial grid is composed by 2000×2000 points.

3.2. Code scalability and performances

In this part of the paper we report the results obtained running both the codes on the cluster Eurora of CINECA. As described in the introduction, the performances of a parallel code are strictly correlated with its hardware implementation. For this reason we briefly describe the machine architectures to better understand and interpret the results. The cluster is not an homogeneous machine composed by 64 compute nodes (32 slower nodes with a 2 eight-core Intel(R) Xeon(R) CPU E5-2658 @ 2.10GHz and 32 faster nodes with 2 eight-core Intel(R) Xeon(R) CPU E5-2687W @ 3.10GHz) and 128 accelerators (64 nVIDIA Tesla K20 (Kepler) + 64 Intel Xeon Phi (MIC)) not used by the software. Finally the network interface is an high-performance network Qlogic QDR (40Gb/s) Infiniband.

Since the architecture presents faster and less performing CPU, the results must be interpreted knowing which cores are actual involved in the computation. In spite of the prevision the execution on a mixed architecture does not mean a great lose of performance since we have to take into account also the network and the communication time. However, to carried out relevant statistical

data, we impose the constrain of a maximum of 15% of slow cores and mediate the results on three executions. Fig. 5 shows the execution time of the benchmark model elaborated with the code “A” using three discretization grids. Increasing the number of cores the computational time decreases. We observe that the slope is reducing progressively moving towards higher number of cores, primarily due to the rise of time spent on communications. The behaviour is similar for all the grid dimensions but we get a greater advantage with a larger grid dimensions. A comparisons between the two kinds of codes described in section 2 is illustrated in Fig. 6 with the same arrangement of Fig. 5. The simulations lead to a different behaviour without an ever decrescent trend as happens with code “A”. The curves present an U-shape with a minimum. Increasing the number of cores we have first an increasing of the performance but then we assist to worst performances. This trend is due to the updating of all the nodes with the attractors informations that, for large amount of computing nodes, became too heavy. The position of the minimum, as expected, is related to the problem dimension and indicates the saturation of the communication with respect to the work load. For a few-cores application the performance of code “B” are better than code “A”, and increasing the dimension of the grid the advantage became more visible, e.g. up to about 30 cores for a dimensions of 3000^2 the code “B” performs the computation of the basin in less time.

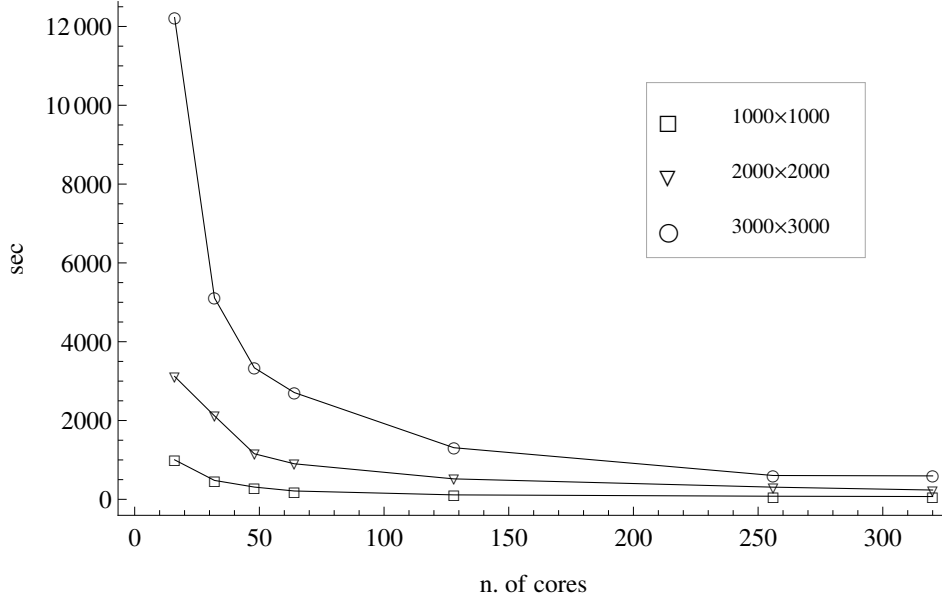


Figure 5: Computational time as function of the number of cores for our parallel code “A” using three different discretization grids.

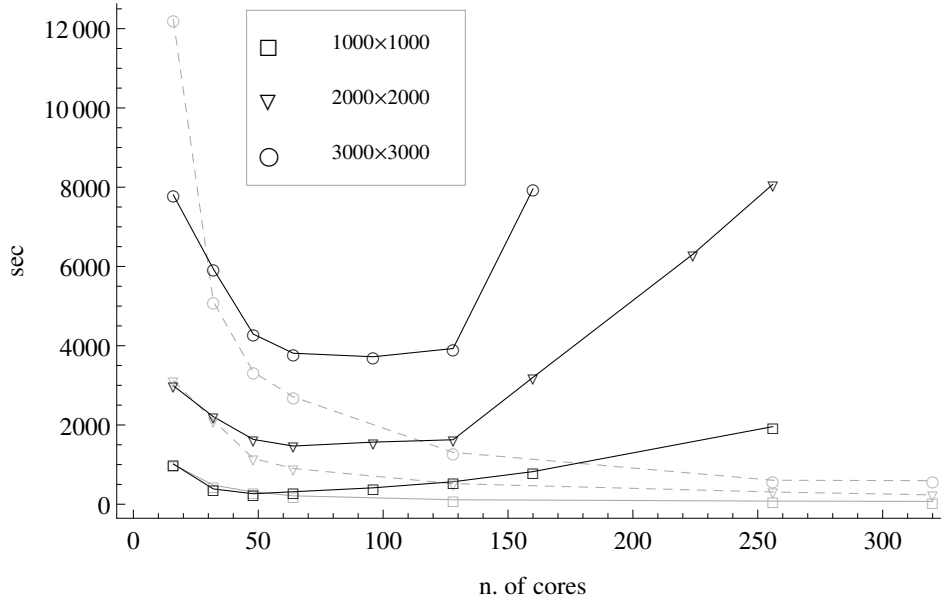


Figure 6: Computational time as function of the number of cores for our parallel code “B” (black lines) using three different discretization grids. To facilitate the comparison, the results of Fig. 5 are also reported with light gray lines.

The heterogeneity of Eurora machine is an excellent test to verify the time execution balance on the nodes. Ideally, an equal distribution of the computing work (for a 2D spatial dimension) is given by $(m \times n)/(c - 2)$ where m, n are the grid dimensions and c the total cores number. As previously described in section 2, a fixed load distribution is not a good choice due to the several unknowns within the time execution and the hardware characteristics and performances. Even by using a perfect homogeneous machine an equal division lead to large delay and random execution times. Figs. 7-8 plot load balance curves as function of the number of computational core utilized. Performing the running of the code only on the slow nodes type, we report the degree of dispersion of the load balance in Fig. 7. The upper and lower curves represents the minimum and maximum values of the initial conditions elaborated, while the dotted central lines is the ideal balancing. With a mixed structure of fast and slow computing nodes the execution is unbalanced: the fast cores performs much more elaboration introducing skewness in the data (see Fig. 8). We underline that also in this configuration the time balance is optimal, mostly thanks to the larger number of job done by the faster cores. It justifies the choice of a master scheduler, and the resulting time difference between the nodes is only of few second and only caused by the MPI initialization and finalization. A very good time balancing is obtained, and this means exploiting at the maximum all the cores. The results showing the timing in two different architecture settings are reported in

Tab. 2. Even with a large discrepancy in the number of data elaborated ($\Delta i.c.$) the gap time Δt difference is less than one second. We can also note that the use of fast computational cores gives a little improvement of the performances, and we obtain a reduction of the total elaboration time (the sum of the time occurred for all the tasks), as well as a decrease of the the maximum time.

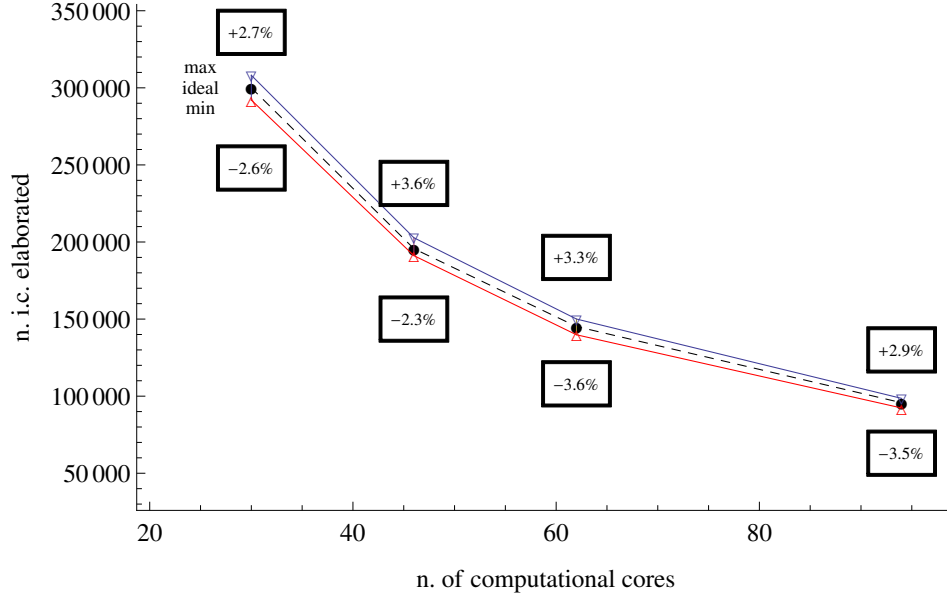


Figure 7: A diagram showing the distribution of loads for a 2000×2000 grid for several numbers of computational cores in an homogeneous architecture. The minimum and maximum number of jobs executed by the cores are respectively the lower and upper curves reported with solid lines (points Δ and ∇). The ideal balancing is also sketched in dotted line (points \bullet).

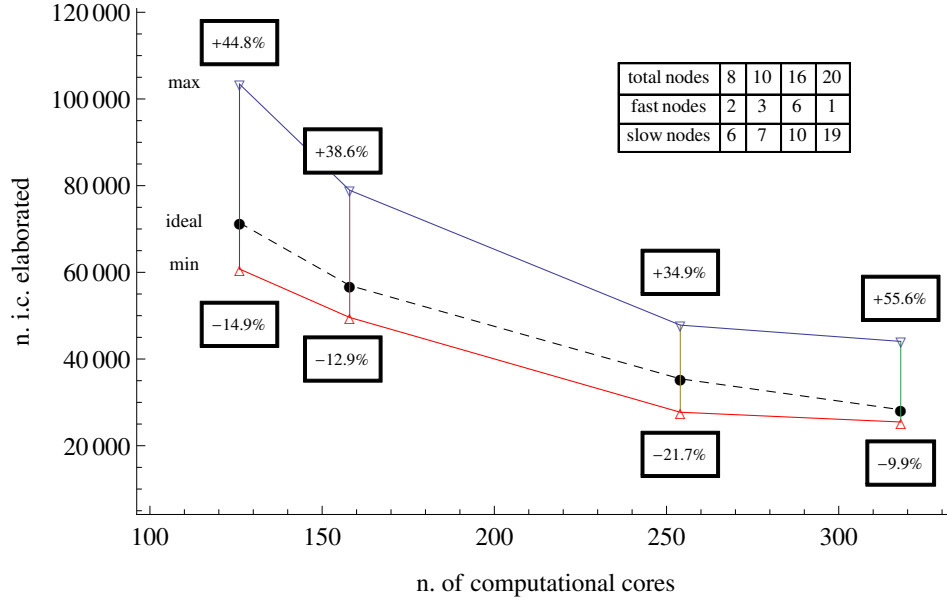


Figure 8: A diagram showing the distribution of loads for a 2000×2000 grid for several numbers of computational cores in an heterogeneous architecture (table within the figure). Same graphical arrangement of Fig. 7.

cores number	n. fast computational cores	n. slow computational cores	elaboration time [sec]	time max core [sec]	time min core [sec]	Δt	ideal balance	n. i.c. max	n. i.c. min	$\Delta i.c.$
128	0	126	61977	500.54	498.86	2.28	31746	32563	30527	2036
128	32	94	61432	496.67	493.6	3.07	31746	45373	26742	18631

Table 2: Report of the elaboration statistics for two hardware configurations. Discretized grid of 2000×2000 .

It is worth to note that in parallel code the mixing of communication operations and pure elaboration makes not so easy to establish the real advantage in the heterogeneous machine. In view of above we report a bar chart (Fig. 9) showing how the computing time is distributed. The data are obtained instrumenting the code by means of the software *scalasca* [23]. Since the procedure introduces numerous instructions, it slows the software. For this reason the data have been collected in different executions with respect to the simulation shown above and they were primarily finalized to a code optimization.

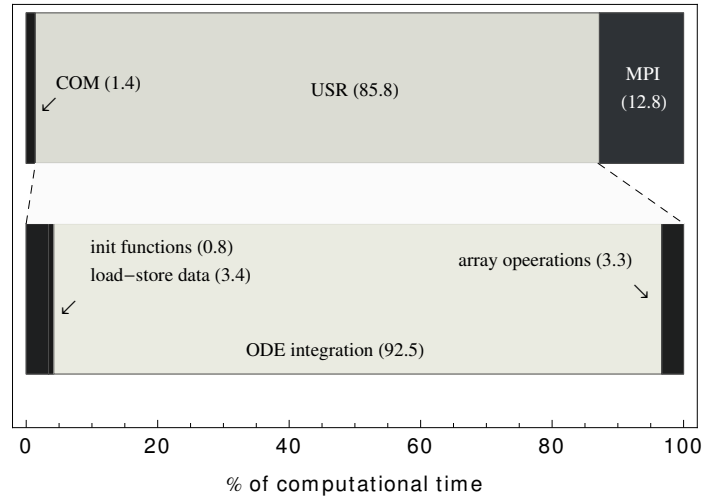


Figure 9: Bar chart showing the distribution of computing time for the code A. The inspected grid is 2000×2000 . The lower bar is a zoom of the USR part in the upper bar.

The MPI slice involves the messaging operations between nodes, the COM operations are instructions that interacts with others containing parallel constructs while the USR part of time contains pure local operations. The figure refers to the code “A” for a grid of 2000×2000 . The code “B” reports almost the same results with a slight increasing of time percentage involving the MPI operations: COM (1.1%), USR (84.3%) MPI (14.6%). A further zoom on the elaboration slice shows that the major part of the time is involved in the step-by-step integrations demonstrating a good efficiency of the software for the specific purpose.

4. Conclusions and further developments

We have investigated the application of parallel programming to the basins of attraction computation. Several issues related to the inner seriality of the problem have been addressed. Two parallel codes and their proprieties have been presented. The results show that for large scale problems, only for a low number of cores an instantaneous synchronism between the nodes is preferred to increase the performance.

The scalability and portability of the code has been tested performing an optimal time balance load on the computing nodes. We believe that our approach can be consider only the first step in the application of parallel programming to the study of the dynamical system. The paper wants to give some skill, rules and results to better deal with large scale problem characterized by a deep seriality.

The computational time as function of the number of cores shows that the problem takes advantage by the use of the parallel programming and good performance can be obtained also in small clusters.

The major drawback is represented by the wasting of memory due to the MPI copies of the software and the use of sparse matrix is under evaluation.

Further implementation will regard the use of an hybrid MPI-openMP infrastructure; moreover taking advantage of GPU and accelerators in the critical sections of the software the performance can be improved. However with a so heterogeneous architecture, not always available, we will introduce a complex and not full stable programming that requires the user an high knowledge of the machine, thus we have to find a compromise between complexity and performance.

References

- [1] J.M.T. Thompson and H.B. Stewart. *Nonlinear Dynamics and Chaos*. Wiley, 2002.
- [2] J.M.T. Thompson. Chaotic phenomena triggering the escape from a potential well. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, 421(1861):195–225, 1989.
- [3] G. Rega and S. Lenci. Identifying, evaluating, and controlling dynamical integrity measures in non-linear mechanical oscillators. *Nonlinear Analysis: Theory, Methods & Applications*, 63(5-7):902–914, 2005.
- [4] S. Lenci and G. Rega. Load carrying capacity of systems within a global safety perspective. part ii. attractor/basin integrity under dynamic excitations. *International Journal of Non-Linear Mechanics*, 46(9):1240–1251, 2011.
- [5] C.S. Hsu. A theory of cell-to-cell mapping dynamical systems. *Journal of Applied Mechanics*, 47(4):931–939, 1980.
- [6] C.S. Hsu and R.S. Guttalu. An unravelling algorithm for global analysis of dynamical systems: An application of cell-to-cell mappings. *Journal of Applied Mechanics*, 47(4):940–948, 1980.

- [7] R.P. Eason and A.J. Dick. A parallelized multi-degrees-of-freedom cell mapping method. *Nonlinear Dynamics*, pages 1–13, 2014.
- [8] R.P. Eason, A.J. Dick, and S. Nagarajaiah. Numerical investigation of coexisting high and low amplitude responses and safe basin erosion for a coupled linear oscillator and nonlinear absorber system. *Journal of Sound and Vibration*, 333(15):3490–3504, 2014.
- [9] G. Moore. Cramming more components onto integrated circuits. *Electronics Magazine*, 38(8), 1965.
- [10] H. Sutter. The free lunch is over: a fundamental turn toward concurrency in software. *Dr. Dobbs's Journal*, 30(3), 2005.
- [11] H. Sutter and J. Larus. Software and the concurrency revolution. *Queue*, 3(7):54–62, September 2005.
- [12] Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard, Version 3.0*. High Performance Computing Center Stuttgart, 2012.
- [13] I. Kovacic and M.J. Brennan. *The Duffing Equation: Nonlinear Oscillators and their Behaviour*. Wiley, 2011.
- [14] E.H. Dowell. Chaotic oscillations in mechanical systems. *Computational Mechanics*, 3(3):199–216, 1988.
- [15] R. Rusinek, A. Weremczuk, K. Kecik, and J. Warminski. Dynamics of a time delayed duffing oscillator. *International Journal of Non-Linear Mechanics*, 65(0):98 – 106, 2014.
- [16] S. Lenci and G. Rega. Optimal control of nonregular dynamics in a Duffing oscillator. *Nonlinear Dynamics*, 33(1):71–86, 2003.
- [17] J. Guckenheimer and P. Holmes. *Nonlinear Oscillations, Dynamical Systems, and Bifurcations of Vector Fields*. Springer, 1993.
- [18] S. Wiggins. *Introduction to Applied Nonlinear Dynamical Systems and Chaos*. Texts in Applied Mathematics. Springer, 2003.
- [19] A. Medio and M. Lines. *Nonlinear Dynamics: A Primer*. Cambridge University Press, 2001.

- [20] Marc Snir, Steve Otto, Steven Huss-Lederman, David Walker, and Jack Dongarra. *MPI: The Complete Reference*. MIT Press, 1996.
- [21] L. Shampine and M. Gordon. *Computer Solution of Ordinary Differential Equations: The Initial Value Problem*. Freeman, 1975.
- [22] H.E. Nusse and J.A. Yorke. *Dynamics: Numerical Explorations*. Springer, 1998.
- [23] M. Geimer, F. Wolf, B.J.N. Wylie, E. Ábrahám, D. Becker, and B. Mohr. The scalasca performance toolset architecture. *Concurr. Comput. : Pract. Exper.*, 22(6):702–719, April 2010.