Commun Nonlinear Sci Numer Simulat 37 (2016) 143–162

Contents lists available at ScienceDirect

Commun Nonlinear Sci Numer Simulat

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

Review

Numerical characterization of nonlinear dynamical systems using parallel computing: The role of GPUs approach

Filipe I. Fazanaro[a,b,*], Diogo C. Soriano[a], Ricardo Suyama[a], Marconi K. Madrid[c], José Raimundo de Oliveira[b], Ignacio Bravo Muñoz[d], Romis Attux[b]

[a] Centro de Engenharia, Modelagem e Ciências Sociais Aplicadas, Universidade Federal do ABC (UFABC), Avenida dos Estados 5001, 09210-580 Santo André, São Paulo, Brazil
[b] Department of Computer Engineering and Industrial Automation (DCA), School of Electrical and Computer Engineering (FEEC), UNICAMP, Av. Albert Einstein 400, Cidade Universitária Zeferino Vaz, 13083-852 Campinas, São Paulo, Brazil
[c] Department of Systems and Energy (DSE), School of Electrical and Computer Engineering (FEEC), UNICAMP, Av. Albert Einstein 400, Cidade Universitária Zeferino Vaz, 13083-852 Campinas, São Paulo, Brazil
[d] Department of Electronic (DEPECA), University of Alcalá (UAH), Carretera Madrid–Barcelona Km 33.600, 28871 Alcalá de Henares, Madrid, Spain

ARTICLE INFO

Article history:
Received 8 October 2014
Revised 8 October 2015
Accepted 29 December 2015
Available online 26 January 2016

Keywords:
Chaos
CUDA
Duffing oscillator
GPU computing
Lagrangian Coherent Structures
Lyapunov bifurcation diagram
Lyapunov exponents
Parallel computing
Parameter space

ABSTRACT

The characterization of nonlinear dynamical systems and their attractors in terms of invariant measures, basins of attractions and the structure of their vector fields usually outlines a task strongly related to the underlying computational cost. In this work, the practical aspects related to the use of parallel computing – specially the use of Graphics Processing Units (GPUs) and of the Compute Unified Device Architecture (CUDA) – are reviewed and discussed in the context of nonlinear dynamical systems characterization. In this work such characterization is performed by obtaining both local and global Lyapunov exponents for the classical forced Duffing oscillator. The local divergence measure was employed by the computation of the Lagrangian Coherent Structures (LCSs), revealing the general organization of the flow according to the obtained separatrices, while the global Lyapunov exponents were used to characterize the attractors obtained under one or more bifurcation parameters. These simulation sets also illustrate the required computation time and speedup gains provided by different parallel computing strategies, justifying the employment and the relevance of GPUs and CUDA in such extensive numerical approach. Finally, more than simply providing an overview supported by a representative set of simulations, this work also aims to be a unified introduction to the use of the mentioned parallel computing tools in the context of nonlinear dynamical systems, providing codes and examples to be executed in MATLAB and using the CUDA environment, something that is usually fragmented in different scientific communities and restricted to specialists on parallel computing strategies.

* Corresponding author at: Centro de Engenharia, Modelagem e Ciências Sociais Aplicadas, Universidade Federal do ABC (UFABC), Avenida dos Estados 5001, 09210-580, Santo André, SP, Brazil. Tel.: +55 1149960128.

E-mail addresses: filipe.fazanaro@gmail.com, filipe.fazanaro@ufabc.edu.br (F.I. Fazanaro), diogo.soriano@ufabc.edu.br (D.C. Soriano), ricardo.suyama@ufabc.edu.br (R. Suyama), madrid@dsce.fee.unicamp.br (M.K. Madrid), jro@dca.fee.unicamp.br (J.R.d. Oliveira), ibravo@depeca.uah.es (I.B. Muñoz), attux@dca.fee.unicamp.br (R. Attux).

http://dx.doi.org/10.1016/j.cnsns.2015.12.021
1007-5704/

1. Introduction

Since the 1960's, video graphic accelerator cards have experienced a significant development, evolving from a dedicated device primarily related to the acceleration of graphics processing operations – based on fixed-function graphics pipelines – to massively parallel programmable processors [1–5]. Historically, the graphics processors and the video cards have been used to accelerate a wide variety of common tasks on personal computers, such as drawing text characters or even simple graphics, to the synthesis of sophisticated and high quality three-dimensional images in computer games [6]. Particularly, the continuously increasing demand for higher quality graphics on games has been encouraging programmers and developers to refine and propose new implementation techniques and, moreover, the companies to invest in the production of increasingly robust and powerful hardware.

As the amount of transistors on the Graphics Processing Units (GPUs) – typically doubling every 12 to 18 months [3,5] – has overcome the amount found in the Central Processing Units (CPUs), a more natural widespread use out of specific applications started to take place and draw more attention in industry and society, such as business workstations, medical devices, personal computers and entertainment consoles [6]. In particular, the current number of cores (easily reaching hundreds and even thousands) and memory resources available have made the GPUs an interesting alternative for scientists and researchers interested on the parallelization of computationally intensive problems [7,8].

In fact, if the first computing projects employing GPUs could be considered academic experiments and exceptional applications [9,10], since the mid of 2000's, the use of GPUs as a general purpose computing tool began to gain prominence with the introduction of development environments specialized for working with them. In this context, in the end of the year 2006, NVIDIA introduced the GeForce 8-series family and the Compute Unified Device Architecture (CUDA) – based on standard C, C++, Fortran, OpenCL and DirectCompute languages –, allowing a significant advance in the development and debug routines [11] of complex and high performance computational problems [1–3,5,9,10,12].

According to Blythe [6], the migration of analytical methodologies in order to take advantage of the GPUs processing power had initially shown gains of order up to 20 and 30 times when compared to the respective approaches implemented in CPU, which justifies the growth and widespread use of such strategies. Briefly, among the main applications of GPUs for extensive numerical analysis, it is possible to mention:

- Molecular dynamics studies, in which protein folding can play a key role in understanding several diseases, including cancer [13–15];
- Monte Carlo simulations performed in the context of fluid dynamics [16], and also in statistical physics studies, such as those using the Ising model [17–19];
- Artificial neural networks [20,21] – applied through different perspectives, such as in the context of speech pattern recognition [22,23] and faces classification [24] – have benefited from the use of GPUs. In particular, the work of Lopes and Ribeiro [25] has presented an approach based on GPU and CUDA for the Back-Propagation and Multiple Back-Propagation algorithms, which allowed a considerable reduction of the training time associated with this signal processing structure;
- Medical Image Analysis, specially for implementing segmentation and denoising algorithms for large data sets acquired by Computed Tomography (CT) and Magnetic Resonance Imaging (MRI) [26–32];
- Brain function and simulations of realistic firing patterns observed in real biological (Hodgkin–Huxley-type) neurons [33,34]. Particularly, the use of general computing on GPUs allowed the simulation of large networks considering many synapses connections, and, at the same time, maintaining a suitable computational cost [35–39];
- Fluid dynamics, in the context of Lattice-Boltzmann-based methodology [40–43] and the solution of the three-dimensional Navier–Stokes equations [44]. It can be also mentioned the employment of Lagrangian Coherent Structures commonly used for studying the advection phenomenon as, for instance, in (oceanic) fluid flows [45–47], and its general implementation using GPUs for classical models [48–50] and graphical visualization [51,52].

In particular, the last two issues are closely connected to nonlinear dynamics theory, a field that has been extensively developed with the advent of greater computational resources. Currently, computational simulations for characterizing nonlinear dynamical systems are commonly used in the scientific literature. However, it is our point of view that a more extensive analysis based on parallel computing is still restricted to specialized groups, waiting for a user friendly general purpose framework.

With the aim of popularizing these tools, this work presents an overview of parallel computing strategies for the characterization of nonlinear dynamics supported by a representative set of simulations using the forced Duffing oscillator, chosen due to the amount of studies available in the literature [53–56]. The characterization performed here is conducted in terms of the Finite-Time Lyapunov Exponents (FTLE) used to obtain the Lagrangian Coherent Structures (LCSs) and also in terms of the global Lyapunov exponents.

Briefly, LCSs can be defined as geometric objects of maximal repulsion (or attraction) in the phase space [57–60]. Such objects are usually detected using a measure of hyperbolicity (as, for instance, the FTLE field) and can outline the classical stable and unstable manifolds of fixed points for two-dimensional autonomous systems. As the notion of stable and unstable manifolds is not well-defined in the context of non-autonomous systems, the term LCS was introduced in a wider sense [60]. These structures are commonly associated to separatrices of different oscillatory behaviors or even to the complex mixing behavior underlying to chaotic motion, a case that is illustrated here for the forced Duffing dynamical model.

If on the one hand the local FTLE field provides a better insight on the crucial geometric objects that organize the flows in the phase space, the global Lyapunov exponents – classically defined as the mean divergence (or convergence) rate of close initial conditions – establish an important invariant measuring, i.e., a robust metric independent of a specific trajectory or initial condition in a given basin of attraction, and close connected to the topological characteristics of the attractor and either to the process of information generation underlying the dynamics. In the simulations performed here, codimension-one and -two Lyapunov bifurcation diagrams were obtained taking, for instance, the amplitude and the frequency of the external forcing as control parameters for the Duffing model. In this case, a particular complex bifurcation structure is obtained, as commonly observed in nonlinear dynamics.

This work is structured as follows: Section 2 contains the main practical aspects regarding to the use of the parallel computing framework as well as the CUDA environment for the characterization of nonlinear dynamical systems. Section 3 contains a brief overview concerning the Lyapunov exponents and their evaluation. In Section 4, a brief introduction to the LCSs and its calculation in terms of the FTLE field is presented. Section 5 describes the computational gains obtained when the CUDA environment is used to characterize the Duffing oscillator model under periodic forcing. Section 6 presents the main discussions and conclusions of this work.

2. An overview about the parallel computing framework

2.1. The parallel computing toolbox of MATLAB

As a common characteristic of most MATLAB implementations, its parallel computing toolbox presents several attractive facilities, such as a simple programing language, powerful tools and optimized functions in a huge library, which are available for fast use, and, specially, allowing easy access to debugging procedures [61,62]. These features naturally outline a framework of popular use in parallel computing problems, which justifies its use here as one of the main benchmarks for nonlinear dynamical systems characterization.

To accomplish the parallel computing task, the `parfor` function assumes a crucial role, being the main responsible for the loop control – such as the classical `for-while` controllers –, executing, in parallel, pieces of codes and functions that are inside the main loop. According to Sharma and Martin [62], one of the most interesting feature of the `parfor` function consists in its facility to implement a code originally designed to repeat sequentially a given instruction (or several mathematical operations) many times in parallel. Furthermore, one of the great advantages of using the `parfor` is its capability to control the distribution of the computation between all available processors (cores) and the coherently organization (at runtime) of the results obtained, being transparent to the user how these tasks are performed.

Fig. 1 illustrates a simplified source code implemented to generate the Lyapunov bifurcation diagrams (shown in Figs. 13 and 14) for the Duffing dynamical system (described by Eq. (4)). Basically, the script must set the number of processors that will be employed to the execution of the loop (line 7). Herein, the term “processors” can be interpreted as being the cores in the computer CPU. In this case, the connection between the processors is defined as “local”, which means that the `parfor` function should be executed through the cores of the CPU (line 8). After that, for each value of the control parameter (define initially in line 2), the `parfor` function will execute in each core the code necessary to calculate the Lyapunov exponents. This calculation was performed using the Cloned Dynamics (CLDYN) approach [63] – which is briefly reviewed in Section 3. In simple terms, this methodology consists in monitoring difference state vectors defined in terms of the trajectories generated by the original dynamical system and by its copies (clones), being these “clones” slightly modified in linearly independent directions.

It is important to note that the `parfor` function requires attention to certain aspects, as, for instance, those concerning the independence between the loop iterations. In another words, the results obtained at each iteration should be independent of the previous ones. This is exactly the case in the code shown in Fig. 1, as the Lyapunov exponents can be computed in parallel for each value of the control parameter stored in the vector “vEpsilon”. The global Lyapunov exponents are then stored in vector “vBifurcLyap”, and, finally, the code is closed deleting the parfor object (line 55).

According to the results shown in Section 5, the use of the `parfor` function considerably reduced the total time required for computing the LCSs when compared to implementations designed to be executed in a single CPU core. However, it should be noted that this function restricts the maximum number of processors that could be used to 12 [61,62]. In cases for which more computational power is necessary, it is interesting to use other tools based on parallel computing, such as the Compute Unified Device Architecture (CUDA), which is discussed below.

2.2. A brief overview about the CUDA environment

The CUDA environment – or simply CUDA – represents not only a hardware architecture of the NVIDIA GPUs, but also the complete set of software components, such as the NVIDIA CUDA compiler (`nvcc`, that uses the GNU Compiler `gcc` on Linux platforms), system drivers and libraries necessary for the graphics adapter. This parallel computing architecture allows general purpose computation on GPUs based on standard programing languages as, for instance, the C language [5,64].

Considering only the hardware, the NVIDIA GPUs architecture compatible with the CUDA environment follows the so-called Kepler architecture (GK110-300-A1), which, for the device used here, is composed by an array of 12 scalable Streaming Multiprocessors (SMXs). Each one of these multiprocessors is composed of, among others features, 192 Scalar Processors

```
dim = 3; % dynamical system dimension
vEpsilon = 0.0:0.01:0.8; % control parameter
Gamma = 0.3; Omega = 1.0; % dynamical system parameters
nMaxItera = round( (t_final-t_init)/t_gsr );
delta = 1e-4; % Cloned Dynamics perturbation

PoolSize = 4;
poolobj = parpool(PoolSize);

parfor ij = 1:numel(vEpsilon)

    epsilon = vEpsilon(ij);

    y_init_orig = [-0.7, -0.5, 0]; % initial conditions arbitrarily chosen
    y_init_clon = (ones(dim,1)*y_init_orig)' + delta*eye(dim);
    y_init = [y_init_orig, reshape(y_init_clon',1,[])];

    tstep = 0.01; time = 0;

    LyapSum = zeros(dim, nMaxItera); Lyap = zeros(dim, nMaxItera);

    for ii = 1:nMaxItera

        tspan = time:tstep:(time+t_gsr);

        [T,Y] = ode45(@(t,y) Duffing1989_CIDyn (t,y,Gamma,Epsilon,Omega), tspan, y_init,
      Options);

        time = T(end);

        y_orig = Y(end,1:dim)';
        y_clon = Y(end,(dim+1):(dim*(dim+1)));
        y_clon = reshape(y_clon,dim,[])';

        deltax = y_orig*ones(1,dim) - y_clon; % Difference state vector

        [vk, uk, Normk] = GSR2(deltax, dim); % Gram-Schimidt Reorthonormalization

        % Lyapunov exponents calculation
        for jj = 1:dim
            LyapSum(jj,ii+1) = LyapSum(jj,ii) + log(Normk(:,jj)/delta);
            Lyap(jj,ii+1) = (1/(time-t_init))*LyapSum(jj,ii+1);
        end

        % Prepare for the next iteration
        y_init_orig = y_orig';
        y_init_clon = (ones(dim,1)*y_init_orig)' + delta*uk;
        y_init = [y_init_orig, reshape(y_init_clon',1,[])];

    end

    vBifurcLyap(ij,:) = [epsilon, Lyap(:,end)'];

end

delete(poolobj);
```

Fig. 1. An example based on the `parfor` implementation. This piece of code was used to calculate the Lyapunov bifurcation diagrams (cf. Section 5). The complete code is provided as a supplementary material.

(SPs) – also known by CUDA Cores –, 64 double-precision units, 32 Load/Store (LD/ST) Units, 32 Special Function Units (SFU), L1 cache memory shared between the SPs, and instruction cache [65]. Table 1 contains some complementary characteristics related to the GPU employed here.

The CUDA programing model can be classified as Single Program Multiple Data (SPMD) since a single special function – known as *kernel* – is executed multiple times through the (parallel) data [2]. It is interesting to note that several other

Table 1
Summary of the main characteristics related to the GPU employed through the experimental procedures. Note that the GTX 780 is composed by GDDR5 memories.

	Units	GTX 780
Compute capability		3.5
Number of SMXs		12
Number of SPs/CUDA cores per SMX		192
Warp size		32
Maximum number of threads per SMX		2048
Maximum number of threads per block		1024
Max dimension size of a thread block (x, y, z)		(1024, 1024, 64)
Device memory	MB	3072
Memory bandwidth	GB/s	288.4
Bus width	bits	384
GPU maximum clock rate	MHz	902
Memory maximum clock rate	MHz	3004
TDP	Watts	250

authors [3,5,6,18,25,44,66] prefer to adopt the definitions Single Instruction Multiple Thread (SIMT) or Single Instruction Multiple Data (SIMD), depending on the context of their work. Regardless of the programing model classification, the kernel functions are responsible for defining the work sequence that will be developed by different threads.

According to Kirk and Hwu [2], the threads can be understood as a simplification of how the processor unit sequentially executes a program. In GPUs, threads are organized into one-, two- or three-dimensional blocks, which are usually arranged in one- or two-dimensional grids. For more details, Refs. [3,5,25,42], Ref. [12, Chapter 2] and Ref. [2, Chapters 3 and 4] are recommended, which contain diagrams and more detailed information and discussions. Therefore, the kernel is executed by a particular thread which is identified by a single coordinated – the *thread index* – computed based on predefined (built-in) variables – e.g., "`threadIdx`", "`blockIdx`" and "`blockDim`" – which are available to the programmer by the CUDA environment. These variables allow the correct identification of the portion of the data structure that will be processed. Fig. 2 illustrates an example of the function called by the kernel which was used to identify the threads correctly [67].

It is interesting to note that thread blocks are completely independent of each other, allowing them to be executed in any order. However, it should be noted that the threads presented in the same block are executed concurrently by the available multiprocessors, which are the responsible for mapping each thread to a single SP - or CUDA core. For instance, considering the GPU GTX 780, each SMX can allocate no more than 16 blocks of threads [65]. This number is usually limited by the amount of resources available in each SMX (e.g., consider that a percentage of the amount of L1 cache during the kernel execution is compromised by other (graphical) functions) [2]. Therefore, as the threads start their execution independently of each other, in certain applications, it is interesting to define a synchronization barrier between the threads – by the use of the "`__syncthreads()`" function –, which ensures the coordination of parallel activities and improves the process related to the information sharing between threads – e.g., using on-chip memory of the SMX [2,18,25,64].

In Fig. 3, it is shown an example of the structure of one kernel implemented to construct the codimension-one Lyapunov bifurcations diagrams. For the sake of simplicity, only the main aspects are considered. The complete source code is provided with this work as a supplementary material.

```
__device__ size_t calculateGlobalIndex() {

    // Which block are we?
    size_t const globalBlockIndex = blockIdx.x + blockIdx.y * gridDim.x;

    // Which thread are we within the block?
    size_t const localThreadIdx = threadIdx.x + blockDim.x * threadIdx.y;

    // How big is each block?
    size_t const threadsPerBlock = blockDim.x * blockDim.y;

    // Which thread are we overall?
    return localThreadIdx + globalBlockIndex * threadsPerBlock;

}
```

Fig. 2. The thread global index is calculated based on built-in variables. The index value is used during the kernel execution.

```
#include <math.h>
#include <cuda.h>
#include <cuda_runtime.h>     // For the CUDA runtime routines (prefixed with "cuda_")

__device__ size_t calculateGlobalIndex() {/* It was defined before */}

/* Kernel definition */
__global__ void processDuffing1989RK4_Bifurc(
        double * outLyap1,  double * outLyap2, double * outLyap3,
        const double * Y0_var1, const double * Y0_var2, const double * Y0_var3,
        const double * Y0_var4, const double * Y0_var5, const double * Y0_var6,
        const double * Y0_var7, const double * Y0_var8, const double * Y0_var9,
        const double * Y0_var10, const double * Y0_var11, const double * Y0_var12,
        const double * H,
        const double * Gamma, const double * Epsilon, const double * Omega,
        const unsigned int numel) {

    /* Work out which thread we are */
    size_t const globalThreadIdx = calculateGlobalIndex();
    /* If we are off the end, return now */
    if (globalThreadIdx >= numel) {return;}

    /* Parameters initialization */
    double epsilon = Epsilon[globalThreadIdx];
    double gamma = Gamma[globalThreadIdx];
    double omega = Omega[globalThreadIdx];
    const double h = H[globalThreadIdx];
    . . .
    const int nMaxItera = ((t_final - t_init) / t_gsr);
    const int nIntegra = t_gsr / h;
    . . .

    /* Main loop */
    while (idxItera < nMaxItera) {
        while (idxIntegra < nIntegra) {
            // Integration of the dynamical system using the fourth order Runge-Kutta
            . . .
        }
        // Gram-Schimidt Reorthonormalization Procedure
        . . .

        // Implementation of the Cloned Dynamics approach
        //   - Norm1, Norm2, Norm3 are obtained after the application of the GSR procedure
        LyapSoma[0][0] = LyapSoma[0][0] + log(Norm1 / delta);
        LyapSoma[1][0] = LyapSoma[1][0] + log(Norm2 / delta);
        LyapSoma[2][0] = LyapSoma[2][0] + log(Norm3 / delta);
    }

    __syncthreads();

    /* Return the global Lyapunov exponent */
    outLyap1[globalThreadIdx] = LyapSoma[0][0] / (double)(t_final - t_init);
    outLyap2[globalThreadIdx] = LyapSoma[1][0] / (double)(t_final - t_init);
    outLyap3[globalThreadIdx] = LyapSoma[2][0] / (double)(t_final - t_init);

}
```

Fig. 3. A briefly description related to how is the kernel developed to construct the Lyapunov bifurcation diagram is structured.

3. Characterizing nonlinear dynamical systems through the Lyapunov exponents

Nonlinear dynamical systems can be characterized according to distinct manners, with different aspects and measures. A particular interesting one is provided by the Lyapunov spectrum, i.e., the set of all Lyapunov exponents, since they define not only the topological structure of the attractor for continuous-time dynamical systems (i.e., if there is convergence to a fixed point, limit cycle, torus or even to a strange attractor), but also the information generated by the dynamics (or lost by an external observer), its fractal dimension or even a lower bound estimator for the Kolmogorov–Sinai entropy. This explain why the Lyapunov spectrum is commonly taken as a general measure of complexity [63,68–71].

Classically, Lyapunov exponents can be defined in terms of the mean divergence (or convergence) rate of initially close initial conditions in the phase space, given the successive application of the motion equations. Their origins can be traced

to Lyapunov's seminal work and the definition of the first Lyapunov method or Lyapunov indirect method [72]. In this approach, the stability of the dynamics is studied by means of a "step-by-step" linearization of the system, being the long term temporal average of the real part of the eigenvalues of the Jacobian matrix (calculated for every step in time) the key attributes for defining the system stability [73]. Its numerical evaluation is subject to more recent works [74,75], and can be reviewed in Refs. [63,70,71].

Briefly, the calculation of the exponents for an $\mathbf{F}(\mathbf{x}, t)$ n-dimensional dynamical system can be performed with the tangent map approach, in which an n-dimensional identity matrix ($\mathbf{I}_n$) anchored in the fiducial trajectory (the solution of the dynamical system) is successively transformed by the tangent map applications, and the resulting stretching (or contracting) effect is measured and average for a long-term evolution. The principal axes of the tangent map are determined by the variational equations:

$$\dot{\mathbf{\Phi}}(\mathbf{x}, t) = \mathbf{J}(\mathbf{x}, t)\mathbf{\Phi}(\mathbf{x}, t) \tag{1}$$

where $\mathbf{J}(\mathbf{x}, t)$ is the Jacobian of $\mathbf{F}(\mathbf{x}, t)$. Details concerning the method can be found in Refs. [63,70,71], and a MATLAB code for the Duffing oscillator (Eqs. (3) and (4)) can be found in the supplementary material of the present work.

Given the relevance of the Lyapunov spectrum, an alternative way to compute it – called the Cloned Dynamics (CLDYN) approach – was developed, based on perturbation theory [63]. This approach is particularly convenient for non-smooth dynamical systems or even for state equations whose Jacobian would require a laborious process [76,77], since it does not require the construction of the tangent space and the solution of the variational equations as required by the classical approach [70,71].

In essence, the CLDYN approach consists in estimating the Lyapunov exponents monitoring the difference state vector between the fiducial (referential) trajectory – defined by the original dynamical system – and copies (clones) of these state equations departing from initial conditions very close to the fiducial solution, as illustrated in Fig. 4 (modified from Ref. [63]).

In other words, the CLDYN approach starts by obtaining the fiducial trajectory and defining close initial conditions in linearly independent directions for the clones, and, consequently, defining the initial perturbation vectors. Each clone will be responsible for the time evolution of a perturbation applied to a specific and orthogonal direction in the phase space, corresponding to a Lyapunov exponent to be estimated, i.e., the mean divergence (or convergence rate) to the fiducial trajectory in a given direction.

For the sake of simplicity, and without loss of generality, consider a two-dimensional system, which would require only two clones for evaluating the time evolution of the small orthogonal perturbations, denoted by $\boldsymbol{\delta}_{1x}^{(0)}$ and $\boldsymbol{\delta}_{2x}^{(0)}$ in panel A of Fig. 4. These perturbations are propagated by the motion equations for a time interval T and the respective difference state vectors are updated taking the difference from the final point of the fiducial trajectory $\mathbf{x}(t)$ and each clone $\mathbf{x}_{c1}(t)$ and $\mathbf{x}_{c2}(t)$, giving rise to the vectors $\boldsymbol{\delta}_{1x}^{(1)}$ and $\boldsymbol{\delta}_{2x}^{(1)}$, in which the superscript index denoted the current iteration of the algorithm.

The second stage of the CLDYN approach, as illustrated in panel B of Fig. 4, consists in applying the Gram–Schmidt Reorthonormalization (GSR) procedure [70,71] in order to correct the tendency of alignment of the difference state vectors in the most expansive direction (if there is any), and, moreover, maintaining the orthogonality between these vectors. Therefore, a new set of numerically corrected vectors $\mathbf{v}_1^{(1)}$ and $\mathbf{v}_2^{(1)}$ are obtained. Finally, the clones are anchored in the same directions of the numerically corrected vectors (denoted by $\mathbf{u}_1^{(1)}$ and $\mathbf{u}_2^{(1)}$) but close to the fiducial trajectory again, in order to provide the initial condition for the next iteration of the algorithm, as illustrated in the panel C of Fig. 4.

In this case, if a sufficient small perturbation with magnitude δ_{x0} is considered, after the Kth iteration and for K sufficiently large, the ith Lyapunov exponent (with $i = 1, 2, \ldots, n$) can be computed by Eq. (2),

$$\lambda_i = \frac{1}{KT}\sum_{k=1}^{K} \ln \left\| \frac{\mathbf{v}_i^{(k)}}{\delta_{x0}} \right\|. \tag{2}$$

For a complete mathematical description and more details regarding the estimation of the algorithm parameters, see Ref. [63]. In the following, such metric is employed for determining both the local behavior of the vector field and the global (mean) characteristic of the attractor. Local and global behavior are obtained by setting the number of algorithm iterations (K parameter). For a practical introduction, a MATLAB code concerning the CLDYN approach applied to the Duffing dynamical system (Eqs. (3) and (4)) can also be found in the supplementary material.

4. Computing the Lagrangian Coherent Structures

Lagrangian Coherent Structures (LCSs) can be defined in terms of critical repulsion (or contraction) regions of the phase space over a finite-time interval [57–60,78–84]. These structures commonly act as main organizers of the flow, establishing barriers or separatrices between different oscillatory behaviors and illustrate the intricate geometry of the process of decorrelation of close initial conditions in a compact space (the so called mixing process) when turbulence or chaotic phenomena take place. More than that, these "finite-time hyperbolic manifolds" [59] have a wider meaning, since they generalize the concept of stable and unstable manifolds of time-independent (autonomous) systems to non-autonomous dynamics, which implies in time-variant objects that are useful to characterize the vector field behavior.

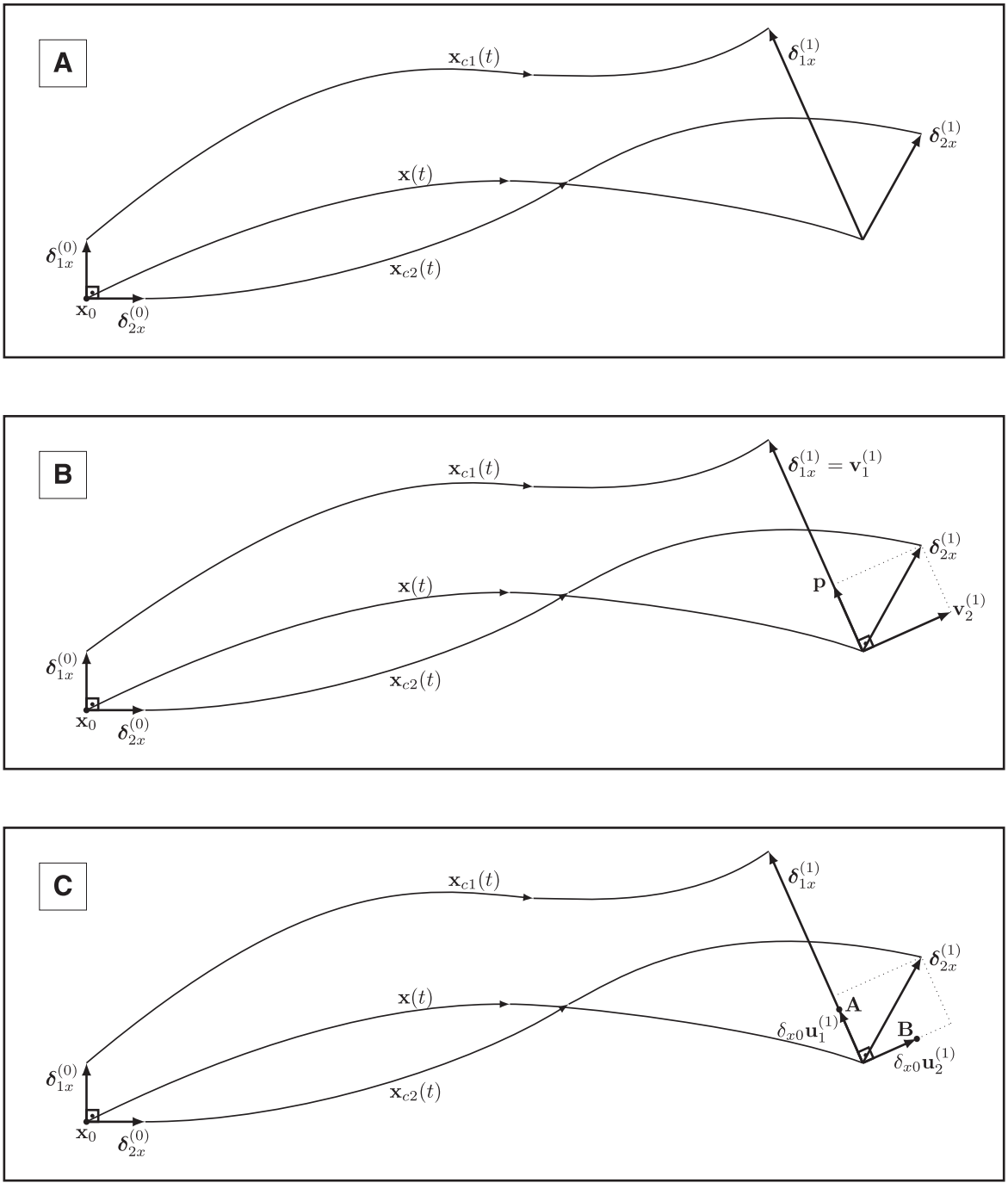

Fig. 4. Illustration of the CLDYN approach. For simplicity, the vectors $\mathbf{P}$ and $\mathbf{v}_2^{(1)}$ are omitted in panel C.

From a practical standpoint, there are different hyperbolic measures that could be used in order to capture the LCSs, being, despite the caveats raised by Haller [78,81,82] and Haller and Yuan [84], the FTLE field a common approach. The structures are usually identified by means of the ridges in the FTLE field, i.e. based on the maximization of the local average Lyapunov exponents obtained in a finite-time interval, as formally introduced by Haller [79–81] and Lekien et al. [57,60].

As mentioned before, the calculation of the LCSs has been relevant to many studies and application dynamics (cf. Ref. [58,85]). In the following, the LCSs are computed for the forced Duffing oscillator model, in view of its extensive numerical and analytical characterization in the literature [53–55,70]. Briefly, the Duffing model can be mathematically

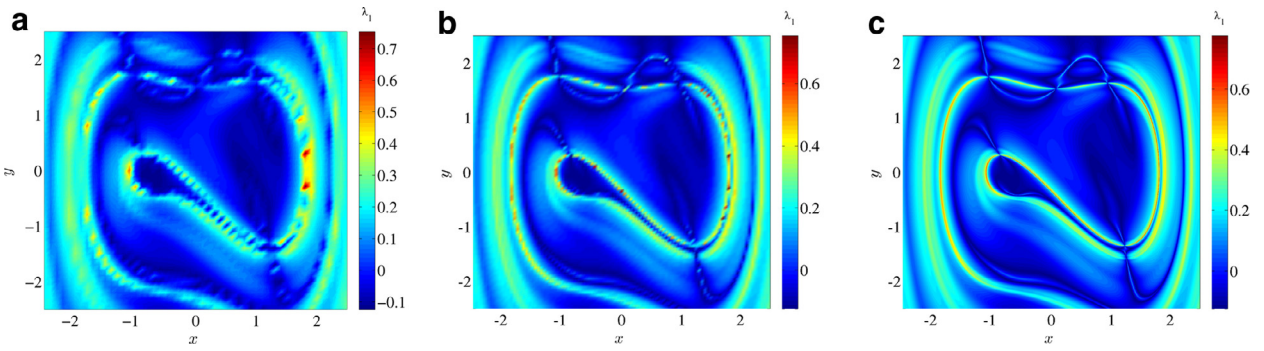

Fig. 5. The Lagrangian Coherent Structures for the forced Duffing oscillator model (Eq. (3)). The parameters were chosen equal to $\epsilon = 0.25$, $\gamma = 0.3$, $\omega = 1$, $t_S = 0$ and $t_{LCS} = 10$. It is interesting to observe that the amount of initial conditions can influence the LCSs resolution. The color scale represents the maximum finite-time Lyapunov exponents (λ_1).

described as:

$$\dot{x} = y$$
$$\dot{y} = x - x^3 - \epsilon y + \gamma \cos(\omega t) \tag{3}$$

where x and y are the state variables and represent the position (displacement) and the velocity of an elastic beam tip under the influence of a sinusoidal exciting force. γ and ω define, respectively, the amplitude and the frequency of such external force. The parameter ϵ represents the damping constant (cf. Ref. [54, Chapter 2] for further details).

In practice, the procedure for constructing the LCSs can be described as follows: after the parameters of the model have been chosen (for instance, taking $\epsilon = 0.25,\ \gamma = 0.3,\ \omega = 1.0$), a finite set of initial conditions is defined as a two-dimensional grid with $x_0 \in [-2.5, 2.5]$ and $y_0 \in [-2.5, 2.5]$. In addition to that, a specific time instant has to be considered in Eq. (3), which is introduced here by the parameterization $t = t_S$. In this case, the choice of a specific value of t_S in the interval $[0,\ 2\pi]$ provides an "instantaneous picture" – i.e., a "snapshot" – of the vector field structure and the evolution of the separatrices can be captured by varying t_S. For each instantaneous picture, a relatively small time interval $t_{LCS} = KT$ has to be defined in order to compute the local divergence or convergence rate. This last parameter can be estimated after some numerical trials and depends on the dynamical system under study. More details concerning the setting of the algorithm for the LCSs evaluation can be found in Ref. [77].

As a numerical example, Fig. 5 shows the simulation obtained for a grid of initial conditions with different resolutions (Δx_0 and Δy_0 representing the distance between consecutive initial conditions), defining different amounts of points in the closed interval. In this case, Fig. 5(a) exhibits a resolution of $\Delta x_0 = \Delta y_0 = 0.1$ and provides a grid of 51×51 points, while Fig. 5(b) establishes a grid of 101×101 points considering $(\Delta x_0, \Delta y_0) = (0.05, 0.05)$. Finally, Fig. 5(c) exhibits the highest resolution concerning $(\Delta x_0, \Delta y_0) = (0.005, 0.005)$ and a grid of 1001×1001 points. Clearly, the resolution defines a key role in the LCSs visualization – i.e. the ridges of the FTLE field – something that naturally requires significant computational resources. In fact, the lack of resolution can suppress small features that may be relevant in different problems [86], and, in contrast, high resolution can make the analysis impractical given the underlying computational cost.

As a second step, once a suitable resolution of the grid of initial conditions has been defined, different values for t_S can be adjusted in order to track the behavior of the LCSs over the time. To accomplish this task, it was chosen $t_S = \pi/4$ (Fig. 6(a)), $t_S = 5\pi/4$ (Fig. 6(b)), $t_S = 5\pi/3$, (Fig. 6(c)) and $t_S = 2\pi$ (Fig. 6(d)). In this particular case, the periodic forcing induces a periodic motion of the separatrices that organize the flow and define the complex oscillatory behavior attained. In fact, the amplitude of the forcing plays a key role in the LCSs displacements, with drastic consequences for the system stability, an issue that has been elegantly addressed with qualitative and quantitative techniques by Guckenheimer and Holmes [54, p. 96–103]. It is not the main objective here to probe into the dynamical structure of the Duffing oscillator under periodic forcing (e.g., by analyzing the global bifurcations and manifolds intersections), despite the fact that LCSs computation with the aid of parallel computing is as a practical approach for doing that.

It is also important to mention that the motion of such separatrices have been extensively studied in the literature in different contexts, specially in oceanic flows [45–47], turbulence [78–84] and hemodynamics, for a better understanding of their general transport and dynamical behavior [58], requiring a computational effort that cannot be disregarded [87–89], more properly treated in parallel computing environments [58,89], as will be discussed in Section 5.

5. Numerical results

Having in mind the parallel computing framework exposed here in the context of general purpose computing with GPUs, when the kernel developed in CUDA C is executed from the MATLAB environment, it is interesting to adopt some stages of data processing before and after the kernel execution. In accordance with the discussion presented by Dziekonski and

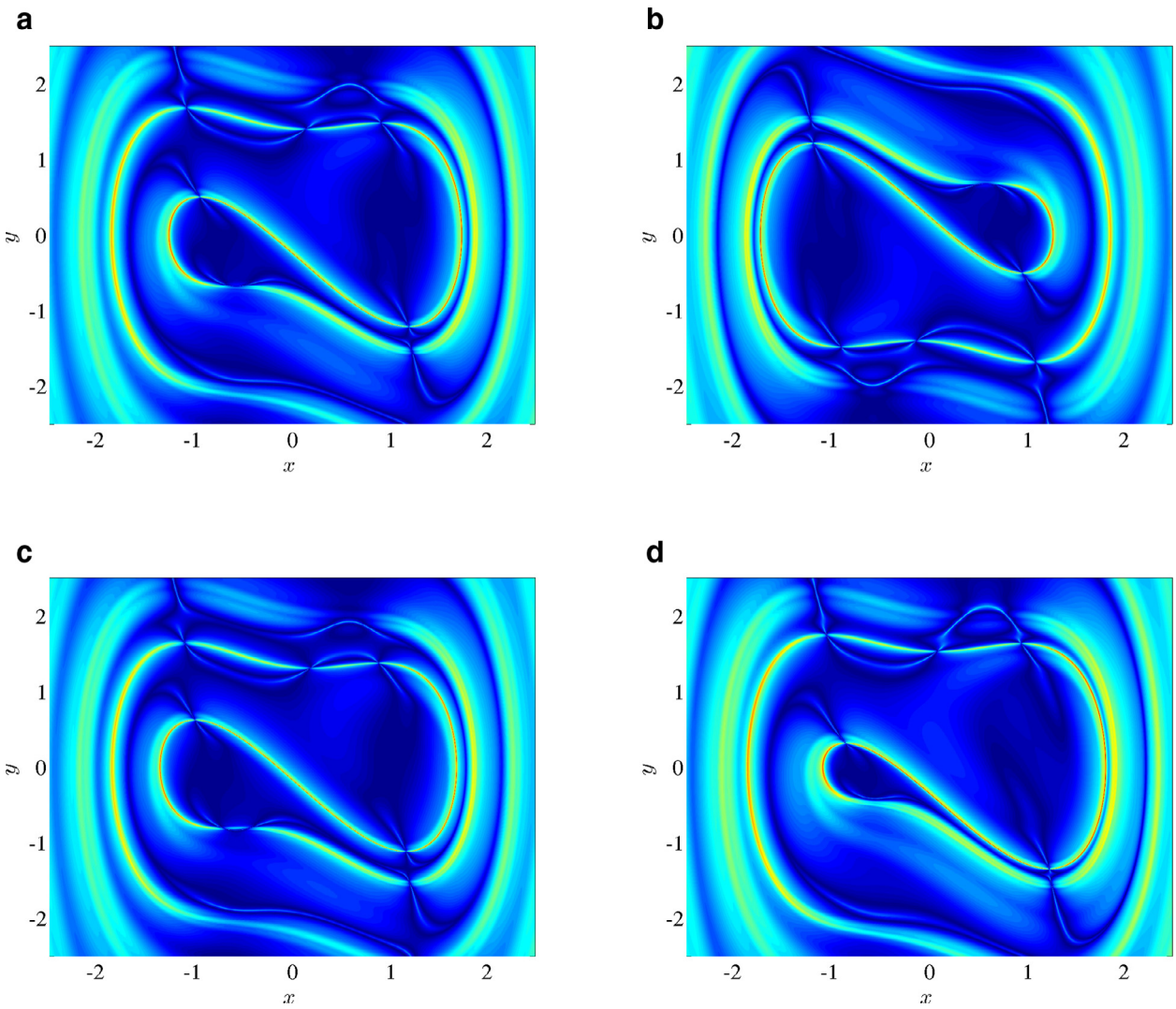

Fig. 6. (a) $t_S = \pi/4$. (b) $t_S = 5\pi/4$. (c) $t_S = 5\pi/3$. (d) $t_S = 2\pi$. The color scale represents the maximum finite-time Lyapunov exponent (λ_1) and follows the same pattern employed in Fig. 5.

collaborators [90], pre-processing the data before sending them to the kernel tends to improve both the utilization of the available resources of the GPU as well as the overall performance. Moreover, post-processing the data, i.e., processing the results obtained at the end of the kernel execution – after transferring them from device (GPU) memory to the host (CPU, global) memory – supports the result analysis and the conclusions about them.

For a better understanding of the pre-processing and post-processing stages used in the experiments based on the kernel in CUDA C [90], consider, for instance, the LCSs computing process. Note that the following discussion can be extended to the codimension-one and -two Lyapunov bifurcation diagrams. Therefore, to accomplish this task, it is necessary to set in the pre-processing stage – defined as the set of instruction establish before the kernel been executed – the vectors containing all possible combinations of the initial conditions associated to the respective FTLE to be calculated and perform the allocation in the host (CPU) memory (Fig. 7). The pre-processing stage is then completed after transferring the data from the host memory to the device (GPU) memory, which is done by the function `gpuArray(.)` (Fig. 8). It is interesting to note that, in applications based on "pure" CUDA C language, allocating data in the host memory and its respective transfer to the device memory are performed, respectively, by the `cudaMalloc(.)` and `cudaMemcpy(.)` functions [2,12]. More details can be found in the source codes provided as supplementary material.

After the transfer of the data vectors to the GPU memory, the kernel must be compiled, which can be done from the MATLAB environment, as can be seen in line 2 of Fig. 9 and, then, the kernel object must be created (line 5). The kernel initialization (lines 7 and 8) can be considered one of the most important procedure to be done during this phase. In this case, the thread block size is defined as a one-dimensional block with no more than `MaxThreadsPerBlock` threads

```
dim = 3; % dimension of the original dynamical system

vEpsilon = 0:0.01:0.8; % control parameter

numElements = numel( vEpsilon );

Gamma = 0.3; Omega = 1.0; % parameters
vGamma = Gamma*ones( numElements, 1 );
vOmega = Omega*ones( numElements, 1 );

[vX0, vY0, vZ0] = deal( 1, 1, 0 ); % initial conditions
vCols_mGridX0 = vX0*ones( numElements, 1 );
vCols_mGridY0 = vY0*ones( numElements, 1 );
vCols_mGridZ0 = vZ0*ones( numElements, 1 );

delta = 1e-4; % Cloned Dynamics perturbation

y_init_orig = [ vCols_mGridX0, vCols_mGridY0, vCols_mGridZ0 ];

% Prepare the perturbations for the clones
mAux1 = [ repmat(vCols_mGridX0,1,dim), repmat(vCols_mGridY0,1,dim), repmat(
      vCols_mGridZ0,1,dim) ];
vPerturbClonesAux = reshape( delta*eye(dim), 1, [] );
vPerturbClones = repmat( vPerturbClonesAux, numElements, 1 );

y_init_clon = mAux1 + vPerturbClones;

y_init = [ y_init_orig, y_init_clon ];  % Initial conditions: fiducial and clones systems

h = 0.01; H = h*ones( numElements, 1 ); % Integration step

nMaxIteracoes = round( (t_final-t_init)/t_gsr );

nIntegra = t_gsr/h;
```

Fig. 7. This figure illustrates the procedure to allocate necessary memory for all the combinations that defines the FTLE field and, moreover, the perturbation procedure employed for the Lyapunov exponents calculation using the CLDYN approach.

```
Lyap1 = gpuArray.zeros( numElements, 1 );    % output vectors
Lyap2 = gpuArray.zeros( numElements, 1 );
Lyap3 = gpuArray.zeros( numElements, 1 );

y010 = gpuArray( y_init( 1:numElements, 1 ) );    % kernel inputs
y020 = gpuArray( y_init( 1:numElements, 2 ) );
y030 = gpuArray( y_init( 1:numElements, 3 ) );
y040 = gpuArray( y_init( 1:numElements, 4 ) );
y050 = gpuArray( y_init( 1:numElements, 5 ) );
y060 = gpuArray( y_init( 1:numElements, 6 ) );
y070 = gpuArray( y_init( 1:numElements, 7 ) );
y080 = gpuArray( y_init( 1:numElements, 8 ) );
y090 = gpuArray( y_init( 1:numElements, 9 ) );
y100 = gpuArray( y_init( 1:numElements, 10 ) );
y110 = gpuArray( y_init( 1:numElements, 11 ) );
y120 = gpuArray( y_init( 1:numElements, 12 ) );
```

Fig. 8. Transferring the data to the device memory. Lines 4–9 are related to the data to be allocated to the clones defined by the CLDYN approach.

(line 7) and the grid size that contains the thread blocks (line 8) must be sufficient to process all the data [2]. For instance, this number is equal to 1024 threads for the GPU GTX 780 [61,65].

In the following, the kernel can be executed (Fig. 9, line 11). The kernels were projected aiming to calculate the Lyapunov exponents using the CLDYN approach. Consequently, they integrate the set of differential ordinary equations composed by the fiducial and the disturbed clones associated to system using the fourth-order Runge–Kutta algorithm [70, Chapter 4], a

```
%% Used to compile the kernel from the MATLAB terminal
eval(['!/usr/local/cuda-7.0/bin/nvcc -ptx cudaDuffing1989RK4_Bifurc_v01.cu']);

%% Configure the kernel object
kernel = parallel.gpu.CUDAKernel( 'cudaDuffing1989RK4_Bifurc_v01.ptx', '
      cudaDuffing1989RK4_Bifurc_v01.cu' );

kernel.ThreadBlockSize = [ kernel.MaxThreadsPerBlock, 1, 1 ];
kernel.GridSize        = [ ceil(numElementos/kernel.MaxThreadsPerBlock), 1 ];

%% Kernel execution
[ Lyap1, Lyap2, Lyap3 ] = feval( kernel, ...
    Lyap1, Lyap2, Lyap3, ...
    y010, y020, y030, y040, y050, y060, y070, y080, y090, y100, y110, y120, ...
    H, vGamma, vEpsilon, vOmega, ...
    numElementos);
```

Fig. 9. Creation, configuration, initialization and execution of the kernel object in MATLAB.

```
%% Move the results
Lyap1 = gather( Lyap1 ); Lyap2 = gather( Lyap2 ); Lyap3 = gather( Lyap3 );

%% Store the results
vBifurcLyap  = [ vEpsilon, vGamma, vOmega, Lyap1, Lyap2, Lyap3 ];

%% Free the GPU memory
gpuDevice([]);
```

Fig. 10. Illustration of the post-processing stage.

classical option in the context of nonlinear dynamical systems, which has been shown to perform very well when employing CUDA C [66].

After the integration process, the Gram–Schimidt Reorthonormalization procedure was applied to the state difference vectors – cf. Section 3 – and the Lyapunov exponents were calculated in accordance with Eq. (2). Then, the kernel returns the vectors containing the obtained values for exponents (cf. Fig. 3), which were transferred to the host memory employing the `gather(.)` function [61], initializing the post-processing stage. In this stage, the data vectors are manipulated so that the FTLE field (or the Lyapunov bifurcation diagrams) can be correctly constructed.

In the following, the analysis and the main practical aspects regarding the use of parallel computing tools (`parfor`) and GPUs for the characterization of nonlinear dynamical systems are presented.

5.1. Improving parallel computing efficiency on LCSs identification

In the context of general purpose computing on GPU, a good programing practice is to transfer the data to the GPU memory and to keep them there as long as possible, i.e. minimizing the data transfer over the PCIe bus, which defines the main "bottleneck" of the system and contributes to decrease the latency time associated to such operation (cf. Ref. [2, Chapter 3] and Refs. [5,9,10]).

To illustrate that, Fig. 11 shows the influence of the total time involved during the execution of the kernel for computing the FTLE field shown in Fig. 5. Initially, this experiment has employed different resolutions of the grid of initial conditions, Δx_0 and Δy_0, within the set [0.1, 0.05, 0.02, 0.01, 0.005] over the close interval $[-2.5, 2.5]$ for both the x and y axes. All possible combinations of resolutions were tested within this grid of initial conditions (e.g. $(\Delta x_0^{(1)}, \Delta y_0^{(1)})$, $(\Delta x_0^{(1)}, \Delta y_0^{(2)})$ and so forth) giving rise to 25 combinations. For each combination, the total time of each task was measured, being them: (1) to allocate the data on the host memory (t_{T1}); (2) to transfer this data to the device memory (t_{T2}); (3) to initialize (t_{T3}) and (4) to execute (t_{T4}) the kernel; (5) to move the data from the device memory to the host memory (t_{T5}); (6) post-processing the data (t_{T6}). Finally, these times were compared to the total time consumed by the script (t_{T7}). Table 2 summarizes these results.

The curves presented in Fig. 11 show that the time required to transfer the data from the device memory to the host memory (t_{T5}) – that is, the gather time – represents the greater part of the total time related to the execution of the script implemented in MATLAB for computing the LCSs. This is in accordance with what is presented in the literature [18] and equally indicates the requirement of avoiding transferring data by the PCIe bus (see also Fig. 11(b)). Moreover, it is interesting to note that the ratio (t_{T5}/t_{T7}) is practically indifferent to the number of points used for the definition of the FTLE field, which emphasizes the power of such framework.

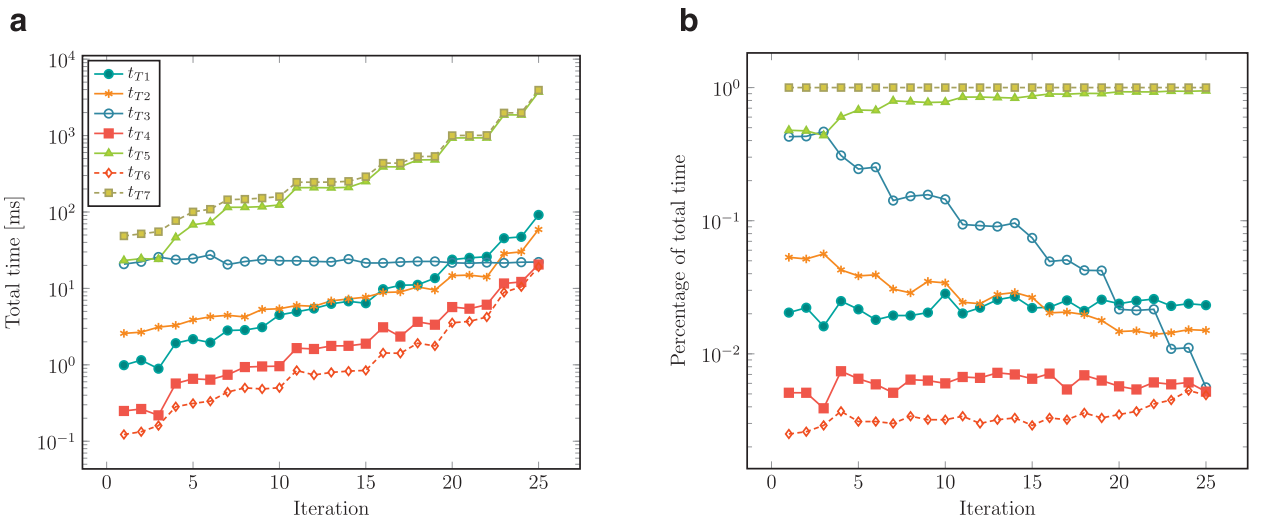

Fig. 11. In (a) and (b) it is possible to verify the total times related to the script execution for computing the LCSs (Fig. 5) on the GPU GTX 780. For simplicity, the legend in (b) is omitted.

Table 2
Total time – in milliseconds – consumed during the kernel execution.

Iteration	t_{T1}	t_{T2}	t_{T3}	t_{T4}	t_{T5}	t_{T6}	t_{T7}
1	0.9889	2.5783	20.7666	0.2489	23.1932	0.1226	48.5445
2	1.1502	2.6749	22.2910	0.2642	24.4982	0.1331	51.8071
3	0.8893	3.1124	25.7497	0.2182	24.1372	0.1602	55.2490
4	1.9191	3.2885	23.7746	0.5693	46.3610	0.2828	76.9637
5	2.1682	3.8675	24.5528	0.6561	68.1218	0.3125	100.4492
6	1.9544	4.2581	27.4548	0.6394	73.2433	0.3342	108.7626
7	2.8158	4.4508	20.5284	0.7410	115.1637	0.4403	144.9079
8	2.8528	4.2285	22.4621	0.9347	115.2965	0.4992	147.1324
9	3.1063	5.3116	23.8118	0.9521	117.4433	0.4831	151.9174
10	4.4987	5.4154	23.0025	0.9598	123.6500	0.5011	158.9443
11	4.9396	5.9786	22.9767	1.6536	207.9540	0.8432	245.2329
12	5.4490	5.8366	22.5411	1.6113	208.3803	0.7365	245.4582
13	6.2766	6.8751	22.2404	1.7671	207.2589	0.7955	246.1190
14	6.7707	7.2799	24.1792	1.7726	210.1023	0.8267	251.8483
15	6.3934	7.6914	21.5111	1.8911	250.6887	0.8460	289.9262
16	9.7620	8.8500	21.5117	3.0975	390.2525	1.4301	435.8857
17	10.9891	8.9848	22.1297	2.3437	389.0763	1.4150	436.0114
18	11.1379	10.4555	22.5261	3.6524	480.3509	1.9163	531.0326
19	13.6232	9.5190	22.5328	3.3476	482.3753	1.7687	534.2269
20	23.8021	14.6940	21.6397	5.6977	930.1681	3.5488	1000.7301
21	25.1163	14.9380	21.3341	5.4234	932.8825	3.7002	1004.5716
22	25.9347	14.0551	21.7772	6.1159	932.8404	4.2246	1006.1919
23	45.3521	28.5578	21.4924	11.5974	1862.9808	8.8671	1980.2752
24	47.2544	30.1168	22.0194	12.1504	1863.7892	10.5040	1987.2945
25	91.5387	59.0446	22.1263	20.5421	3723.5491	19.0986	3937.6340

In addition to that, it is worth noting that the influence of other modules of the algorithm – e.g. by allocating the data in the host memory, transferring data from global (host) memory to the device memory and output data manipulation – causes little impact on the computing overall performance. Finally, note that the most relevant fluctuation in the behavior of these computational times is associated with the total time required for loading the kernel (t_{T3}). Not only during this experiment, but in all other performed in here, MATLAB required an amount of time practically constant to execute this process. Consequently, the equivalent percentage time (compared to the total execution time) of the script tends to decrease as the FTLE field is being incremented, i.e., decreasing the spacing between the initial conditions (Δx_0 and Δy_0).

In a different experiment, the total time required to identify the LCSs related to the Duffing oscillator is measured considering that 10 cores of the CPU are allocated for the execution of the `parfor` function. In this case, the total time consumed by the script execution is compared to the total time required when only a single CPU core is available to perform such task (by means of the initialization option "`-singleCompThread`" as recommended in the Mathworks documentation).

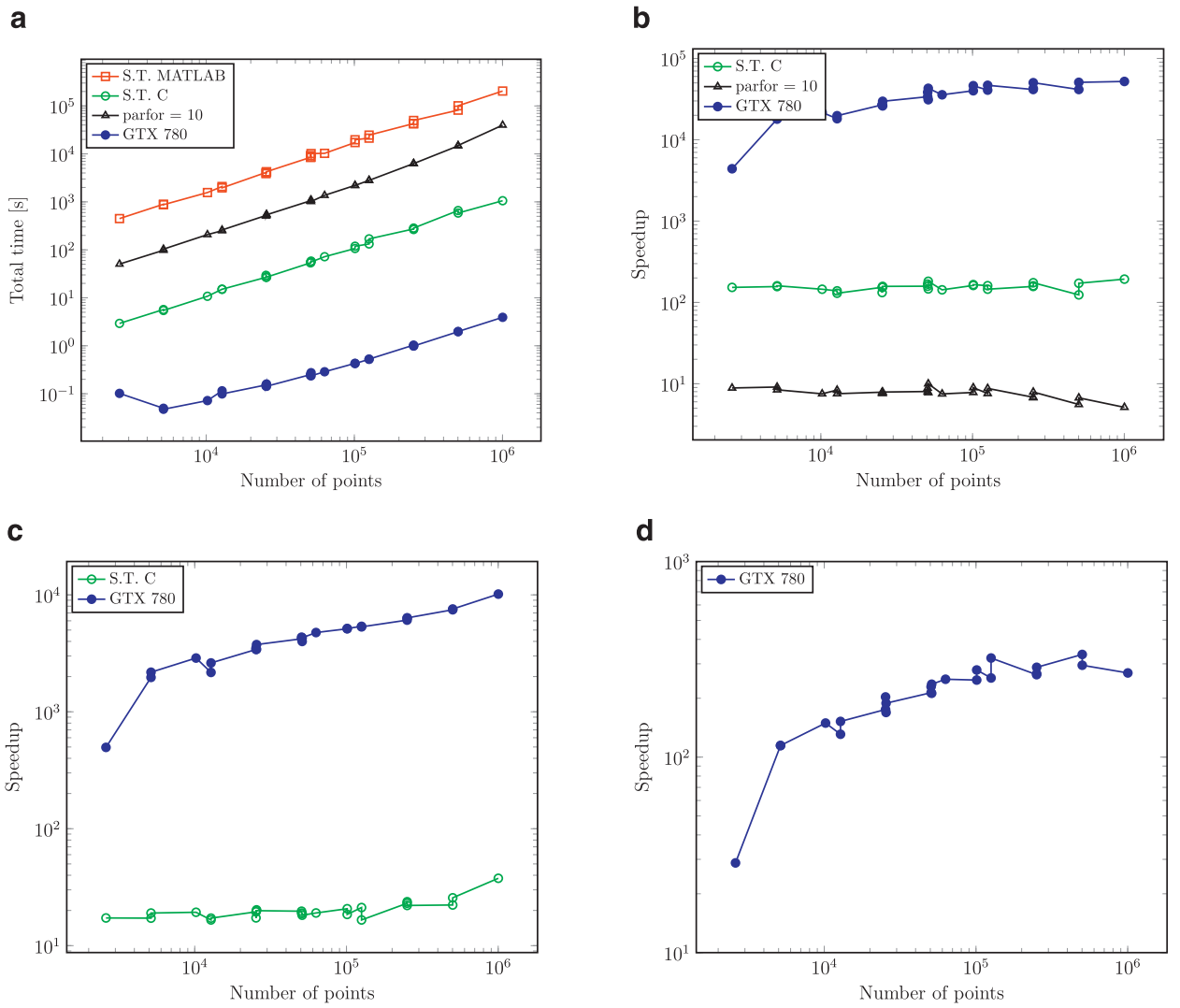

Fig. 12. Total time consumed during the identification of the LCSs related to the non-autonomous Duffing oscillator dynamical system (Eq. (3)). The curves illustrate the speedup as a function of the size of the FTLE field. In (c), the speedup obtained by the use of the "pure C" and the GPU implementations are compared to the use of the `parfor` function. In (d), it is shown the speedup of the GPU related to the "pure C" implementation. In all cases, "S.T." is the abbreviation of "Single Thread".

The comparative analysis can be seen in Fig. 12. It can be observed that the employment of the `parfor` function (see Appendix A for more details about the computer platform) resulted in gains of approximately one order of magnitude.

Although MATLAB consists in an important and powerful development computational platform, providing simple debugging tools – e.g., it is straightforward to follow the changes in the content of vectors and arrays/matrices positions during the execution of any sort of implementation –, in the context of the non-linear dynamical systems characterization, particularly those that can operate by means of chaotic behavior – such as the Duffing oscillator system – the performance of the algorithm executed by the MATLAB cannot achieve satisfactory results. This is evidenced by the curves shown in Fig. 12(a) when the script developed for the identification of LCSs was realigned employing the standard C language. It can be noticed that the use of an algorithm based in the "pure C" language may results in computational gains of, approximately, one to two orders of magnitude when compared both to the implementation using the `parfor` function (with 10 CPU cores) and a single thread, respectively. Consequently, there is a balance achieved between two desirable but incompatible features such as an increase of the application development speed and the possibility that the overall performance could be harmed.

Still considering Fig. 12(a), it is possible to observe the capacity of such methodology of nonlinear dynamical systems characterization based on the general purpose computing on GPUs and the CUDA framework. Such as shown in Fig. 12(c), the employment of the GPU resulted in speedups of up to hundreds times when compared to the implementation based on the `parfor` function (10 CPU cores). This gives support to the premise that the performance achieved by the MATLAB

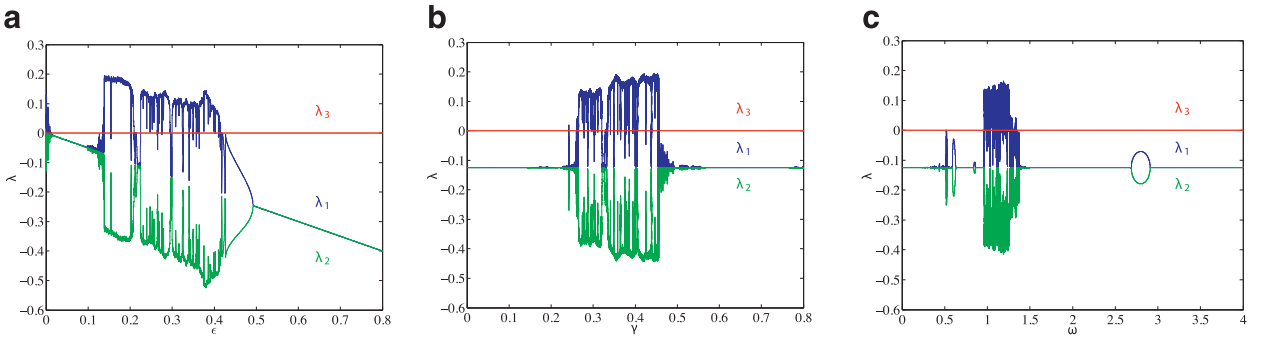

Fig. 13. Lyapunov bifurcation diagrams of codimension-one obtained for the Duffing model (Eq. (4)). In all cases, the step is defined equal to 10^{-5}.

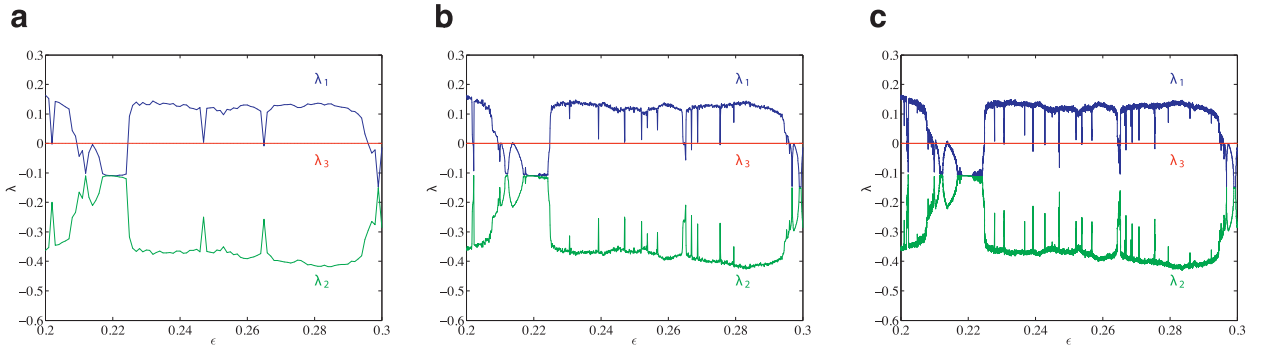

Fig. 14. Comparative between different steps sizes applied during the construction of the Lyapunov spectrum.

application may be impaired when one compares to the performance obtained by algorithms developed using compiled languages (such as the C and the CUDA C). Finally, it is important to observe that the amount of computational time can be continuously reduced as the GPU resources are increasingly required, proving its efficiency in situations of highly intensive computational problems, which is in accordance with important results previously reported in the context of general purpose computing on GPUs [5,9,10,18,90].

5.2. The Lyapunov bifurcation diagrams

In general, the Lyapunov bifurcation diagrams represent the Lyapunov spectrum, or even the largest global Lyapunov exponent in the parameter space, which allows the characterization of the different obtained attractors, as the structural stability is changed [69,91–93]. To match this purpose, the source codes for parallel computing using GPU and CUDA C have been modified for performing such important characterization. As an introduction, codimension-one Lyapunov diagrams are considered followed by the codimension-two (as analyzed in [69]). For these analysis, one consider the modified model of the forced Duffing oscillator [53–55]:

$$\begin{aligned} \dot{x} &= y \\ \dot{y} &= x - x^3 - \epsilon y + \gamma \cos(\omega z) \\ \dot{z} &= 1 \end{aligned} \tag{4}$$

The simulations in Fig. 13 illustrate the case of codimension-one Lyapunov bifurcation diagram. At first, Fig. 13(a) is built setting $\gamma = 0.3$ and $\omega = 1.0$ and varying ϵ in the closed interval [0.0, 0.8] with steps of 10^{-5}, performing an amount of 80,001 Lyapunov spectrum evaluations. In Fig. 13(b), a different "slice" on the space parameter is done, setting $\epsilon = 0.25$, $\omega = 1.0$ and varying $\gamma \in [0.0, 0.8]$ using the same resolution and an equal amount of spectrum evaluations. Finally, in Fig. 13(c), the following configuration was adopted: $\epsilon = 0.25$, $\gamma = 0.3$ and ω varied from 0.0 to 4.0, with steps of 10^{-5}, resulting in 400,001 evaluations.

One of the main reasons to analyze the evolution of the Lyapunov exponents during the characterization of chaotic dynamical systems is the ability to verify for which values of a given parameter the system tends to modify its oscillatory behavior. Therefore, if the resolution of the control parameter is not enough, some particular transitions in the oscillatory behavior can be suppressed. To illustrate this, Fig. 14 shows the Lyapunov spectrum for the Duffing dynamical system, obtained varying the parameter ϵ, such as done in Fig. 13, but limiting the range of variation of the control parameter between 0.2 and 0.3. In Fig. 14(a), ϵ was changed adopting steps of 10^{-3}. In Fig. 14(b), the step was defined as 10^{-4} and,

Table 3
Total times and speedups for the construction of the bifurcation diagrams illustrated in Figs. 13 and 14 using the `parfor` function, the C implementation and the GPU. From Eq. (2), consider the total time equal to $KT = 10,000$ s. "SpeedUp (1)" and "SpeedUp (2)" are obtained, respectively, comparing the `parfor` and the C implementation and the C and the GPU implementations.

	Total time (h)			Total time (min)		
	Fig. 13(a)	Fig. 13(b)	Fig. 13(c)	Fig. 14(a)	Fig. 14(b)	Fig. 14(c)
Parfor = 10	532.3814	529.6966	2751.1758	44.9881	408.4472	4167.9561
C	43.1479	44.0531	223.6568	3.3065	33.0973	338.1388
GTX 780	0.1252	0.1252	0.6344	0.2734	0.4147	1.2417
SpeedUp (1)	12.3385	12.0241	12.3009	13.6059	12.3408	12.3262
SpeedUp (2)	344.5409	351.7293	352.5449	12.0930	79.8152	272.3173

in Fig. 14(c), as 10^{-5}. Such procedure clearly illustrates the requirements for a parameter resolution compatible with the system and application under study in order to meet a suitable perspective of the system behavior. When an extensive number of spectrum evaluations are required, parallel computing strategies, specially that outlined by the GPU, become a naturally attractive approach.

Table 3 contains the total time – in hours – to obtain the bifurcation diagrams illustrated in Fig. 13 and in Fig. 14. It can be seen that the use of GPUs in the process of constructing such diagrams results in a considerable reduction of the computational cost, i.e., the total time involved in the algorithm execution. For simplicity, the gains shown in this table are related only to the implementation using the `parfor` function with a pool of CPU cores equal to 10 and to the "pure C" strategy. In accordance with had been obtained during the LCSs identification, the use of the implementation based on the C language for the construction of the bifurcation diagrams shown in the Fig. 13 resulted in speedups approximately up to 12 times when compared to the MATLAB script execution by the `parfor` function with 10 CPU cores. When one consider the GPU usage and compare it to that of the ANSI C algorithm, the computational gains are, approximately, up to 350 times. Moreover, Table 3 also contains the speedups related to the results shown in Fig. 14. As in the previous case, using the C language resulted in speedups up to 12 to 13 times related to the use of the `parfor` function and, moreover, employing GPUs results in speedups of, approximately, 12 to 272 times when the implementation based on the C language is considered. Finally, it is interesting to note that, as observed during the LCSs identification, the speedups increases as the number of variations of the control parameter increases, i.e., when the GPU is requested as long as possible.

5.3. The parameter space

As previously mentioned, the Lyapunov bifurcation diagrams of codimension-two – i.e., the parameter spaces – are obtained by the simultaneous variation of two parameters that define the dynamics. Therefore, the resulting map will show regions with different degrees of unpredictability, which are closely related to the magnitude of the Lyapunov exponent. In practice, such as observed by Monti and his collaborators [69], the parameter space allows the operation thresholds related to the experimental physical systems to be modified, maintaining the intended oscillatory behavior as one can increase the overall performance while minimizing the operational costs.

In this context, it is interesting to note that the algorithm responsible for identifying the LCSs can be modified for the construction of the parameter space, since, instead of building a field of initial conditions, one defines a space composed by the variation of the control parameters. Therefore, adopting the Duffing oscillator model (Eq. (4)), the Lyapunov exponents are calculated using the Cloned Dynamics approach, setting a total evaluation time equal to $KT = 10000$ s. During the experiments, this time proved to be sufficient so that the exponents reached a "steady state value". In Fig. 15(a), the parameter space was obtained defining $\omega = 1.0$ and varying ϵ in the range [0.0, 1.0] and γ in the range [0.2, 1.0]. In both cases, the steps were defined as equal to 10^{-3}, resulting in a field with 1001×801 points. For the parameter space represented in Fig. 15(b), $\gamma = 0.3$, $\epsilon \in [0.0, 0.5]$ and $\omega \in [0.9, 1.4]$, considering the variation step equal to 10^{-3}, defining a space with 501×501 points. Finally, in Fig. 15(c), $\epsilon = 0.25$, $\gamma \in [0.1, 1.0]$ (in steps equal to 10^{-3}) and $\omega \in [0.8, 1.6]$ (steps equal to 10^{-3}), defining a space with 901×801 points.

Table 4 contains the total times consumed in the construction of the parameter spaces shown in Fig. 15. As in the case of the bifurcation diagrams, it is possible to observe that the use of the C language results in speedups of the order of, approximately, 12 times when comparing the execution of the MATLAB script employing the `parfor` function with 10 cores of the CPU. Moreover, it is possible to observe once more that the gains obtained related to the use of the CUDA C tend to be even larger as the number of combinations of the control parameters increases, i.e., increasing the definition of the parameter space. This reinforces the idea that GPUs tend to have better computational performance – i.e., optimizing the computational costs – as their resources (e.g., memory, SPs, SMs, threads) are kept busy as long as possible. Particularly, it is interesting to note that the refinement of the variation step of the control parameters is very important in the sense that, when nonlinear dynamical systems that can exhibit chaotic behavior are considered, small variations related to its parameters can lead to different operational modes, which, in general, can be verified by the observation of bifurcations of the oscillatory period [53,54,69].

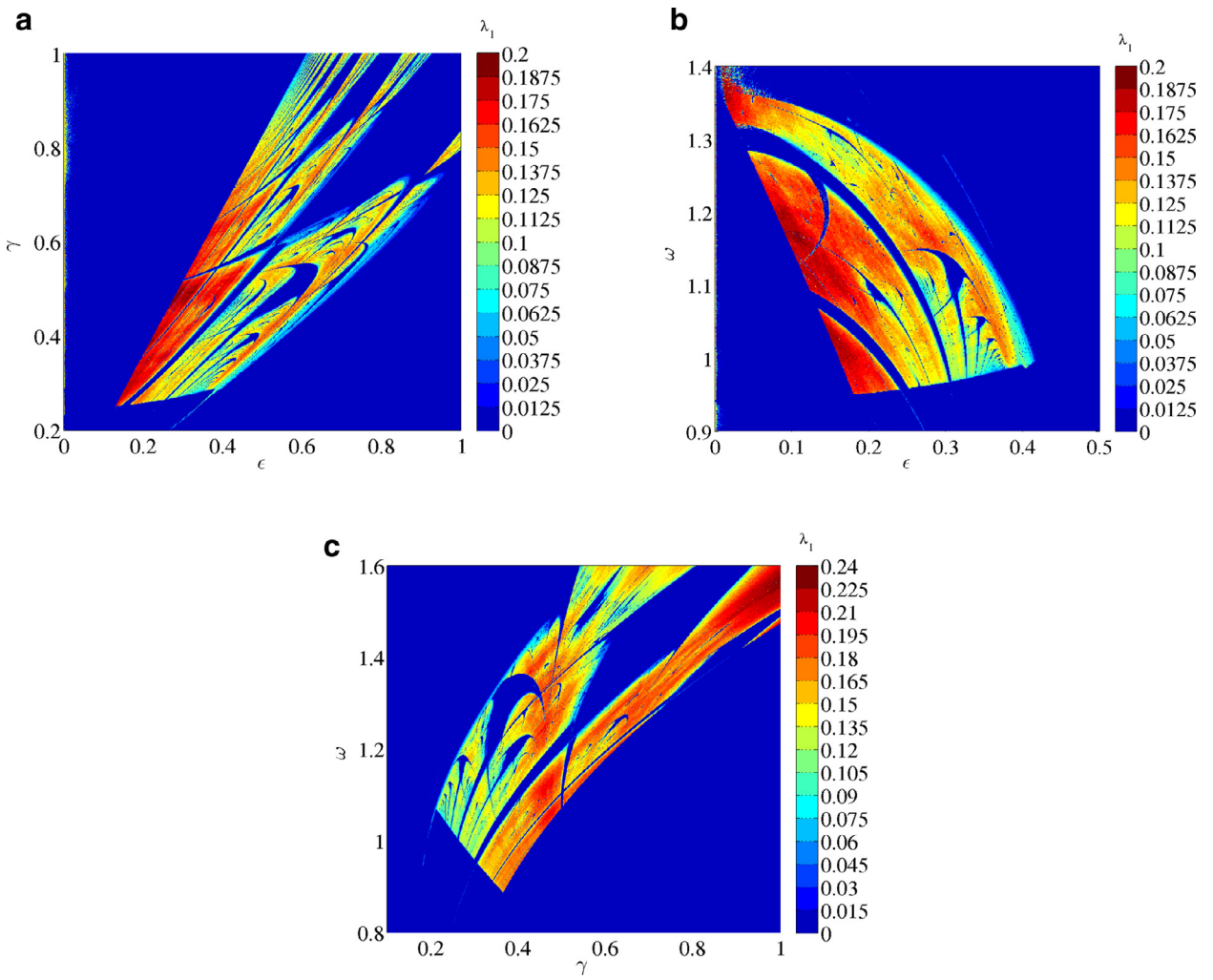

Fig. 15. Parameter spaces obtained for the forced Duffing oscillator (Eq. (4)). The color scale represents the maximum global Lyapunov exponent (λ_1).

Table 4
Overall computational costs and speedups related to the employment of the GPU, the C implementation and the `parfor` function with respect to the parameters spaces shown in Fig. 15. As defined in Table 3, "SpeedUp (1)" is obtained comparing the `parfor` and the C implementation and "SpeedUp (2)" represents the C and the GPU implementations.

	Total time (h)		
	Fig. 15(a)	Fig. 15(b)	Fig. 15(c)
Parfor = 10	5642.2851	1742.2146	5068.0712
C	438.2325	137.1871	394.4530
GTX 780	1.2150	0.4119	1.1765
SpeedUp (1)	12.8751	12.6995	12.8484
SpeedUp (2)	360.6979	333.0650	335.2866

6. Conclusions and discussions

This work addressed the main practical aspects related to the characterization of nonlinear dynamical systems by means of parallel computing tools. In this context, the main current strategies involving the MATLAB parallel computing toolbox [61] and the GPU–CUDA approach [2,12] were reviewed by a representative set of simulations and key-codes for evaluating the local and global Lyapunov spectrum in different scenarios. Despite of the facilities of MATLAB environment, the

implementations based on the C language have shown speed have shown speedup gains up to two orders of magnitude when compared to applications designed for sequential executions in CPU, and up to 10 to 12 times when compared to the use of the `parfor` function of the MATLAB. When comparing the implementation based on the "pure C" language and the CUDA C framework, the obtained results show significant computational gains – approximately, up to 300 times –, which gives support to the principle that, when extensive data parallelism with intensive floating point arithmetics are required, and the data throughput is more important than the data latency [3,9,10] justifies the employment of GPUs.

In general, the numerical characterization performed here was carried out by means of FTLE field for the LCSs computation, the Lyapunov bifurcation diagrams and the parameter spaces, which defined an important framework for understanding the structure of the vector field – revealing the displacements of stable and unstable manifolds under periodic forcing with crucial consequences for the system stability – and also the topological changes in the attractor evoked by the variation of one- and two-dimensional control parameters. For both LCSs and Lyapunov diagrams, the simulations performed here have shown the influence of the initial condition grid size and control parameter step size in the quality and accuracy of the analysis of the dynamical system behavior, as also the amount of computational cost to be deal with. For sure, such analysis is more suitably treated concerning extensive numerical simulations, and, naturally, took the advantage of the parallel computing tools reviewed here, aiming an introduction for "dynamicists" interested in starting to use such powerful framework.

Acknowledgments

F. I. Fazanaro is grateful for the financial support of FAPESP (under Grants 2012/09624-4 and 2014/09954-0) and CNPq (Grant number 449699/2014-5). D. C. Soriano (CNPq, Grant number 449467/2014-7) and R. Suyama also thank FAPESP (Grant number 2012/50799-2) and UFABC for the financial support. R. Attux thanks CNPq (Grant number 302890/2012-2) for financial support. This work was also supported by CAPES and EINTA - Entornos Inteligentes en las Tecnologías de la Asistencia (PHB2006-0077-PC) project.

Appendix A. The computational platform characteristics

All the numerical simulations were performed using a computer with 2 × CPU Intel Xeon E5-2670 v2 (2.5 GHz, 10 cores/20 threads each), 8 × 16 GB DDR3 RAM (1600 MHz), a motherboard with chipset Intel X79. The operational system is the Ubuntu 14.04.3 LTS x64 (Kernel Linux 3.13.0-53-generic). The gcc version is the 4.8.2. The MATLAB version is the release R2014a (v8.3.0.532). The CUDA environment is based on the version 7.0 and the NVIDIA driver version is the release 346.82.

Supplementary material

Supplementary material associated with this article can be found, in the online version, at 10.1016/j.cnsns.2015.12.021.

References

[1] Garland M, Le Grand S, Nickolls J, Anderson J, Hardwick J, Morton S, et al. Parallel computing experiences with CUDA. IEEE Micro 2008;28(4):13–27. doi:10.1109/MM.2008.57.
[2] Kirk DB, Hwu WW. Programming massively parallel processors: A hands-on approach. 2nd ed. Elsevier - Morgan Kaufmann; 2013.
[3] Lindholm E, Nickolls J, Oberman S, Montrym J. NVIDIA Tesla: A unified graphics and computing architecture. IEEE Micro 2008;28(2):39–55. doi:10.1109/MM.2008.31.
[4] Montrym J, Moreton H. The GeForce 6800. IEEE Micro 2005;25(2):41–51. doi:10.1109/MM.2005.37.
[5] Nickolls J, Dally WJ. The GPU computing era. IEEE Micro 2010;30(2):56–69. doi:10.1109/MM.2010.41.
[6] Blythe D. Rise of the graphics processor. Proc IEEE 2008;96(5):761–78. doi:10.1109/JPROC.2008.917718.
[7] Hager G, Wellein G. Introduction to high performance computing for scientists and engineers. 1st ed. CRC Press; 2011.
[8] Rauber T, Runger G. Parallel programming for multicore and cluster systems. 1st ed. Springer; 2010.
[9] Owens JD, Houston M, Luebke D, Green S, Stone JE, Phillips JC. GPU computing. Proc IEEE 2008;96(5):879–99. doi:10.1109/JPROC.2008.917757.
[10] Owens JD, Luebke D, Govindaraju N, Harris M, Krüger J, Lefohn AE, et al. A survey of general-purpose computation on graphics hardware. Comput Graph Forum 2007;26(1):80–113. doi:10.1111/j.1467-8659.2007.01012.x.
[11] Langdon WB. Debugging CUDA. In: Proceedings of the thirteenth annual conference companion on genetic and evolutionary computation GECCO '11. New York, NY, USA: ACM; 2011. p. 415–22. ISBN 978-1-4503-0690-4. doi:10.1145/2001858.2002028.
[12] NVIDIA. CUDA C programming guide - design guide; 2012. Version 5.0. Manual revision PG-02829-001-v5.0.
[13] Liu W, Schmidt B, Voss G, Müller-Wittig W. Molecular dynamics simulations on commodity GPUs with CUDA. In: Aluru S, Parashar M, Badrinath R, Prasanna V, editors. High performance computing - HiPC 2007. Lecture Notes in Computer Science, vol. 4873. Springer Berlin Heidelberg; 2007. p. 185–96. ISBN 978-3-540-77219-4. doi:10.1007/978-3-540-77220-0_20.
[14] Luttmann E, Ensign DL, Vaidyanathan V, Houston M, Rimon N, Oland J, et al. Accelerating molecular dynamic simulation on the cell processor and playstation 3. J Comput Chem 2009;30(2):268–74. doi:10.1002/jcc.21054.
[15] Wirawan A, Schimidt B, Zhang H, Kowoh CK. High performance protein sequence database scanning on the cell broadband engine. Sci Progr 2009;17(1-2):97–111. doi:10.3233/SPR-2009-0274.
[16] Liu W, Schmidt B, Voss G, Müller-Wittig W. Accelerating molecular dynamics simulations using graphics processing units with CUDA. Comput Phys Commun 2008;179(9):634–41. doi:10.1016/j.cpc.2008.05.008.
[17] Dickson NG, Karimi K, Hamze F. Importance of explicit vectorization for CPU and GPU software performance. J Comput Phys 2011;230(13):5383–98. doi:10.1016/j.jcp.2011.03.041.
[18] Preis T, Virnau P, Paul W, Schneider JJ. GPU accelerated Monte Carlo simulation of the 2D and 3D Ising model. J Comput Phys 2009;228(12):4468–77. doi:10.1016/j.jcp.2009.03.018.
[19] Tomov S, McGuigan M, Bennett R, Smith G, Spiletic J. Benchmarking and implementation of probability-based simulations on programmable graphics cards. Comput Graph 2005;29(1):71–80. doi:10.1016/j.cag.2004.11.008.

[20] Ho T-Y, Lam P-M, Leung C-S. Parallelization of cellular neural networks on GPU. Pattern Recognit 2008;41(8):2684–92. doi:10.1016/j.patcog.2008.01.018.
[21] Oh K-S, Jung K. GPU implementation of neural networks. Pattern Recognit 2004;37(6):1311–14. doi:10.1016/j.patcog.2004.01.013.
[22] Dixon PR, Oonishi T, Furui S. Harnessing graphics processors for the fast computation of acoustic likelihoods in speech recognition. Comput Speech Lang 2009;23(4):510–26. doi:10.1016/j.csl.2009.03.005.
[23] Scanzio S, Cumani S, Gemello R, Mana F, Laface P. Parallel implementation of artificial neural network training for speech recognition. Pattern Recognit Lett 2010;31(11):1302–9. doi:10.1016/j.patrec.2010.02.003.
[24] Nasse F, Thurau C, Fink GA. Face detection using GPU-based convolutional neural networks. In: Jiang X, Petkov N, editors. Computer analysis of images and patterns. Lecture Notes in Computer Science, vol. 5702. Springer Berlin Heidelberg; 2009. p. 83–90. ISBN 978-3-642-03766-5. doi:10.1007/978-3-642-03767-2_10.
[25] Lopes N, Ribeiro B. An evaluation of multiple feed-forward networks on GPUs. Int J Neural Syst 2011;21(01):31–47. doi:10.1142/S0129065711002638.
[26] Eklund A, Andersson M, Knutsson H. fMRI analysis on the GPU - possibilities and challenges. Comput Methods Progr Biomed 2012;105(2):145–61. doi:10.1016/j.cmpb.2011.07.007.
[27] Eklund A, Dufort P, Forsberg D, LaConte SM. Medical image processing on the GPU - past, present and future. Med Image Anal 2013;17(8):1073–94. doi:10.1016/j.media.2013.05.008.
[28] Fluck O, Vetter C, Wein W, Kamen A, Preim B, Westermann R. A survey of medical image registration on graphics hardware. Comput Methods Progr Biomed 2011;104(3):e45–57. doi:10.1016/j.cmpb.2010.10.009.
[29] Hlawatsch M, Vollrath JE, Sadlo F, Weiskopf D. Coherent structures of characteristic curves in symmetric second order tensor fields. IEEE Trans Vis Comput Graph 2011;17(6):781–94. doi:10.1109/TVCG.2010.107.
[30] Kutter O, Shams R, Navab N. Visualization and GPU-accelerated simulation of medical ultrasound from CT images. Comput Methods Progr Biomed 2009;94(3):250–66. doi:10.1016/j.cmpb.2008.12.011.
[31] Shams R, Sadeghi P, Kennedy R, Hartley R. Parallel computation of mutual information on the GPU with application to real-time registration of 3D medical images. Comput Methods Progr Biomed 2010;99(2):133–46. doi:10.1016/j.cmpb.2009.11.004.
[32] Stone SS, Haldar JP, Tsao SC, Hwu W-m W, Sutton BP, Liang ZP. Accelerating advanced MRI reconstructions on GPUs. J Parallel Distrib Comput 2008;68(10):1307–18. doi:10.1016/j.jpdc.2008.05.013. General-purpose processing using graphics processing units.
[33] Izhikevich EM. Simple model of spiking neurons. IEEE Trans Neural Netw 2003;14(6):1569–72. doi:10.1109/TNN.2003.820440.
[34] Izhikevich EM. Dynamical systems in neuroscience - The geometry of excitability and bursting. The MIT Press; 2007.
[35] Bernhard F, Keriven R. Spiking neurons on GPUs. In: Alexandrov VN, van Albada GD, Sloot PMA, Dongarra J, editors. Computational science - ICCS 2006. Lecture Notes in Computer Science, vol. 3994. Reading, UK: Springer Berlin Heidelberg; 2006. p. 236–43. ISBN 978-3-540-34385-1. doi:10.1007/11758549_36.
[36] Fidjeland AK, Roesch EB, Shanahan MP, Luk W. NeMo: A platform for neural modelling of spiking neurons using GPUs. In: Proceedings of the twentieth IEEE international conference on application-specific Systems, architectures and processors, 2009, ASAP 2009, Boston, MA; 2009. p. 137–44. doi:10.1109/ASAP.2009.24.
[37] Fidjeland AK, Shanahan MP. Accelerated simulation of spiking neural networks using GPUs. In: Proceedings of the 2010 international joint conference on neural networks (IJCNN); 2010. p. 1–8. doi:10.1109/IJCNN.2010.5596678.
[38] Igarashi J, Shouno O, Fukai T, Tsujino H. Real-time simulation of a spiking neural network model of the basal ganglia circuitry using general purpose computing on graphics processing units. Neural Netw 2011;24(9):950–60. doi:10.1016/j.neunet.2011.06.008. Multi-Scale, Multi-Modal Neural Modeling and Simulation.
[39] Nageswaran JM, Dutt N, Krichmar JL, Nicolau A, Veidenbaum A. Efficient simulation of large-scale spiking neural networks using CUDA graphics processors. In: Proceedings of the international joint conference on neural networks, 2009 (IJCNN 2009).; 2009. p. 2145–52. doi:10.1109/IJCNN.2009.5179043.
[40] Frezzotti A, Ghiroldi GP, Gibelli L. Solving the Boltzmann equation on GPUs. Comput Phys Commun 2011;182(12):2445–53. doi:10.1016/j.cpc.2011.07.002.
[41] Kloss YY, Shuvalov PV, Tcheremissine FG. Solving Boltzmann equation on GPU. Procedia Comput Sci 2010;1(1):1083–91. doi:10.1016/j.procs.2010.04.120. International conference on computational science (ICCS) 2010.
[42] Kuznik F, Obrecht C, Rusaouen G, Roux JJ. LBM based flow simulation using GPU computing processor. Comput Math Appl 2010;59(7):2380–92. doi:10.1016/j.camwa.2009.08.052. International conferences on mesoscopic methods in engineering and science.
[43] Nandapalan N, Brent RP, Murray LM, Rendell AP. High-performance pseudo-random number generation on graphics processing units. In: Wyrzykowski R, Dongarra J, Karczewski K, Waśniewski J, editors. Parallel processing and applied mathematics. Lecture Notes in Computer Science, vol. 7203. Springer Berlin Heidelberg; 2012. p. 609–18. ISBN 978-3-642-31463-6. doi:10.1007/978-3-642-31464-3_62.
[44] Rinaldi PR, Dari EA, Vénere MJ, Clausse A. Uso de GPUs para La Simulación de Fluidos en 3D con El Método de Lattice Boltzmann. Mec Comput 2010;XXIX(72):7095–108. High Performance Computing on Graphics Hardware (GPGPU) (B). (In Spanish).
[45] Mancho AM, Small D, Wiggins S. A tutorial on dynamical systems concepts applied to Lagrangian transport in oceanic flows defined as finite time data sets: Theoretical and computational issues. Phys Rep 2006;437(3-4):55–124. doi:10.1016/j.physrep.2006.09.005.
[46] Mancho AM, Wiggins S, Curbelo J, Mendoza C. Lagrangian descriptors: A method for revealing phase space structures of general time dependent dynamical systems. Commun Nonlinear Sci Numer Simul 2013;18(12):3530–57. doi:10.1016/j.cnsns.2013.05.002.
[47] Wiggins S. The dynamical systems approach to Lagrangian transport in oceanic flows. Ann Rev Fluid Mech 2005;37(1):295–328. doi:10.1146/annurev.fluid.37.061903.175815.
[48] Domínguez JM, Crespo AJC, Valdez-Balderas D, Rogers BD, Gómez-Gesteira M. New multi-GPU implementation for smoothed particle hydrodynamics on heterogeneous clusters. Comput Phys Commun 2013;184(8):1848–60. doi:10.1016/j.cpc.2013.03.008.
[49] Molnár Jr F, Szakály T, Mészáros R, Lagzi I. Air pollution modelling using a graphics processing unit with CUDA. Comput Phys Commun 2010;181(1):105–12. doi:10.1016/j.cpc.2009.09.008.
[50] Singh B, Pardyjak ER, Norgren A, Willemsen P. Accelerating urban fast response Lagrangian dispersion simulations using inexpensive graphics processor parallelism. Environ Model Softw 2011;26(6):739–50. doi:10.1016/j.envsoft.2010.12.011.
[51] Garth C, Li G-S, Tricoche X, Hansen CD, Hagen H. Visualization of coherent structures in transient 2D flows. In: Hege HC, Polthier K, Scheuermann G, editors. Topology-based methodsin visualization II (mathematics and visualization). Springer Berlin Heidelberg; 2009. p. 1–13. ISBN 978-3-540-88605-1. doi:10.1007/978-3-540-88606-8_1.
[52] Sadlo F, Peikert R. Visualizing Lagrangian coherent structures and comparison to vector field topology. In: Hege HC, Polthier K, Scheuermann G, editors. Topology-based methods in visualization II (mathematics and visualization). Springer Berlin Heidelberg; 2009. p. 15–29. doi:10.1007/978-3-540-88606-8_2.
[53] Gilmore R, McCallum JWL. Structure in the bifurcation diagram of the Duffing oscillator. Phys Rev E 1995;51(2):935–56. doi:10.1103/PhysRevE.51.935.
[54] Guckenheimer J, Holmes P. Nonlinear oscillations, dynamical systems, and bifurcations of vector fields. Applied Mathematical Sciences, vol. 42. Springer; 2000.
[55] Kanamaru T. Duffing oscillator. Scholarpedia 2008;3(3):6327. doi:10.4249/scholarpedia.6327.
[56] Mira C, Touzani-Qriouet M, Kawakami H. Bifurcation structures generated by the nonautonomous Duffing equation. Int J Bifurc Chaos 1999;9(7):1363–79. doi:10.1142/S0218127499000948.
[57] Lekien F, Shadden SC, Marsden JE. Lagrangian coherent structures in n-dimensional systems. J Math Phys 2007;48(6):065404. doi:10.1063/1.2740025.
[58] Shadden SC, Astorino M, Gerbeau JF. Computational analysis of an aortic valve jet with Lagrangian coherent structures. Chaos 2010;20(1):017512. doi:10.1063/1.3272780.

[59] Shadden SC, Dabiri JO, Marsden JE. Lagrangian analysis of fluid transport in empirical vortex ring flows. Phys Fluids 2006;18(4):047105. doi:10.1063/1.2189885.
[60] Shadden SC, Lekien F, Marsden JE. Definition and properties of Lagrangian coherent structures from finite-time Lyapunov exponents in two-dimensional aperiodic flows. Physica D 2005;212(3-4):271–304. doi:10.1016/j.physd.2005.10.007.
[61] MATLAB. Parallel computing toolbox; 2014. Version 8.3.0.532 (R2014a).
[62] Sharma G, Martin J. MATLAB: A language for parallel computing. Int J Parallel Progr 2009;37(1):3–36. doi:10.1007/s10766-008-0082-5.
[63] Soriano DC, Fazanaro FI, Suyama R, Oliveira JR, Attux R, Madrid MK. A method for Lyapunov spectrum estimation using cloned dynamics and its application to the discontinuously-excited FitzHugh-Nagumo model. Nonlinear Dyn 2012;67(1):413–24. doi:10.1007/s11071-011-9989-2.
[64] Januszewski M, Kostur M. Accelerating numerical solution of stochastic differential equations with CUDA. Comput Phys Commun 2010;181(1):183–8. doi:10.1016/j.cpc.2009.09.009.
[65] NVIDIA's Next Generation CUDA Compute Architecture: Kepler GK 110/GK210 (whitepaper). NVIDIA Corporation; 2014. Available at: http://international.download.nvidia.com/pdf/kepler/NVIDIA-Kepler-GK110-GK210-Architecture-Whitepaper.pdf [accessed 01.10.14].
[66] Murray L. GPU acceleration of Runge-Kutta integrators. IEEE Trans Parallel Distrib Syst 2012;23(1):94–101. doi:10.1109/TPDS.2011.61.
[67] Shure L. A Mandelbrot set on MATLAB. The Mathworks, Inc.; 2011. Available at http://blogs.mathworks.com/loren/2011/07/18/a-mandelbrot-set-on-the-gpu/?s_tid=Blog_Loren_Category [accessed 01.10.14].
[68] Boffetta G, Cencini M, Falcioni M, Vulpiani A. Predictability: a way to characterize complexity. Phys Rep 2002;356(6):367–474. doi:10.1016/S0370-1573(01)00025-4.
[69] Monti M, Pardo WB, Walkenstein JA, Rosa Jr E, Grebogi C. Color map of Lyapunov exponents of invariant sets. Int J Bifurc Chaos 1999;9(7):1459–63. doi:10.1142/S0218127499001012.
[70] Parker TS, Chua LO. Practical numerical algorithms for chaotic systems. Springer-Verlag; 1989.
[71] Wolf A, Swift JB, Swinney HL, Vastano JA. Determining Lyapunov exponents from a time series. Physica D 1985;16(3):285–317. doi:10.1016/0167-2789(85)90011-9.
[72] Leine RI. The historical development of classical stability concepts: Lagrange, Poisson and Lyapunov stability. Nonlinear Dyn 2010;59(1-2):173–82. doi:10.1007/s11071-009-9530-z.
[73] Anishchenko VS, Astakhov V, Neiman A, Vadivasova T, Schimansky-Geier L. Nonlinear dynamics of chaotic and stochastic systems: Tutorial and modern developments. Springer; 2007.
[74] Shimada I, Nagashima T. A numerical approach to ergodic problem of dissipative dynamical systems. Prog Theor Phys 1979;61(6):1605–16. doi:10.1143/PTP.61.1605.
[75] Benettin G, Galgani L, Giorgilli A, Strelcyn J-M. Lyapunov characteristic exponents for smooth dynamical systems and for Hamiltonian systems; A method for computing all of them. Part 2: Numerical application. Meccanica 1980;15(1):21–30. doi:10.1007/BF02128237.
[76] Fazanaro FI, Soriano DC, Suyama R, Madrid MK, Attux R, Oliveira JR. Information generation and Lagrangian coherent structures in multiscroll attractors. In: Proceedings of the third IFAC conference on analysis and control of chaotic systems 2012, Cancún, México, vol. 3; 2012. p. 47–52. doi:10.3182/20120620-3-mx-3012.00010.
[77] Fazanaro FI, Soriano DC, Suyama R, Madrid MK, Attux R, Oliveira JR. Characterization of multiscroll attractors using Lyapunov exponents and Lagrangian coherent structures. Chaos 2013;23(2):023105. doi:10.1063/1.4802428.
[78] Haller G. Finding finite-time invariant manifolds in two-dimensional velocity fields. Chaos 2000;10(1):99–108. doi:10.1063/1.166479.
[79] Haller G. Distinguished material surfaces and coherent structures in three-dimensional fluid flows. Physica D 2001;149(4):248–77. doi:10.1016/S0167-2789(00)00199-8.
[80] Haller G. Lagrangian structures and the rate of strain in a partition of two-dimensional turbulence. Phys Fluids 2001;13(11):3365–86. doi:10.1063/1.1403336.
[81] Haller G. Lagrangian coherent structures from approximate velocity data. Phys Fluids 2002;14(6):1851–61. doi:10.1063/1.1477449.
[82] Haller G. A variational theory of hyperbolic Lagrangian coherent structures. Physica D 2011;240(7):574–98. doi:10.1016/j.physd.2010.11.010.
[83] Haller G, Sapsis T. Lagrangian coherent structures and the smallest finite-time Lyapunov exponent. Chaos 2011;21(2):023115. doi:10.1063/1.3579597.
[84] Haller G, Yuan G. Lagrangian coherent structures and mixing in two-dimensional turbulence. Physica D 2000;147(3-4):352–70. doi:10.1016/S0167-2789(00)00142-1.
[85] Peacock T, Dabiri J. Introduction to focus issue: Lagrangian coherent structures. Chaos 2010;20(1):017501. doi:10.1063/1.3278173.
[86] Olcay AB, Pottebaum TS, Krueger PS. Sensitivity of Lagrangian coherent structure identification to flow field resolution and random errors. Chaos 2010;20(1):017506. doi:10.1063/1.3276062.
[87] Brunton SL, Rowley CW. Fast computation of finite-time Lyapunov exponent fields for unsteady flows. Chaos 2010;20(1):017503. doi:10.1063/1.3270044.
[88] Conti C, Rossinelli D, Koumoutsakos P. GPU and APU computations of finite time Lyapunov exponent fields. J Comput Phys 2012;231(5):2229–44. doi:10.1016/j.jcp.2011.10.032.
[89] Lipinski D, Mohseni K. A ridge tracking algorithm and error estimate for efficient computation of Lagrangian coherent structures. Chaos 2010;20(1):017504. doi:10.1063/1.3270049.
[90] Dziekonski A, Sypek P, Lamecki A, Mrozowski M. Generation of large finite-element matrices on multiple graphics processors. Int J Numer Methods Eng 2013;94(2):204–20. doi:10.1002/nme.4452.
[91] Albuquerque HA, Rech PC. A Parameter-Space of a Chua System with a smooth nonlinearity. Int J Bifurc Chaos 2009;19(4):1351–5. doi:10.1142/S0218127409023676.
[92] Albuquerque HA, Rubinger RM, Rech PC. Self-similar structures in a 2D parameter-space of an inductorless Chua's circuit. Phys Lett A 2008;372(27-28):4793–8. doi:10.1016/j.physleta.2008.05.036.
[93] Albuquerque HA, Rubinger RM, Rech PC. Theoretical and experimental time series analysis of an inductorless Chua's circuit. Physica D 2007;233(1):66–72. doi:10.1016/j.physd.2007.06.018.