Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2015, Article ID 687313, 11 pages
http://dx.doi.org/10.1155/2015/687313

Research Article

Parallel kd-Tree Based Approach for Computing the Prediction Horizon Using Wolf's Method

J. J. Águila,[1] E. Arias,[2] M. M. Artigao,[2] and J. J. Miralles[2]

[1]*Universidad de Magallanes, Avenida Bulnes 01855, Casilla Postal 113-D, Punta Arenas, Chile*
[2]*Universidad de Castilla-La Mancha, Campus Universitario, s/n, 02071 Albacete, Spain*

Correspondence should be addressed to E. Arias; enrique.arias@uclm.es

Received 20 July 2015; Revised 2 October 2015; Accepted 11 October 2015

Academic Editor: Meng Du

In different fields of science and engineering, a model of a given underlying dynamical system can be obtained by means of measurement data records called time series. This model becomes very important to understand the original system behaviour and to predict the future values of that system. From the model, parameters such as the prediction horizon can be computed to obtain the point where the prediction becomes useless. In this work, a new parallel kd-tree based approach for computing the prediction horizon is presented. The parallel approach uses the maximal Lyapunov exponent, which is computed by Wolf's method, as an estimator of the prediction horizon.

1. Introduction

In large amount of applications in science and engineering, the main purpose to study a given dynamical system is to be able to obtain a prediction in order to carry out preventive actions. For example, in the case of the analysis of electrocardiogram signals, the information obtained through that study could be used to prevent heart related diseases; in air temperature regulation, the prediction information could be used to manage energy in an efficient manner either into a building or in a crop; moreover, breakdown in wind turbine generators caused by a broken wing could be prevented applying the same principles over the study of the wind speed. However, in all these cases and in the most part of the real-world applications based on dynamical systems, there is not a mathematical model or description of the underlying dynamical system. It is precisely the context in which the nonlinear time series analysis plays an important role. According to this kind of analysis, from time series of one single physic variable, for example, voltage, air temperature, or wind speed, it is possible to obtain parameters which describe the behaviour of the underlying dynamical systems represented for these time series.

This work deals with the problem of computing an estimation for the prediction horizon, that is, the point where the prediction becomes useless. To calculate this value, we used the parameter known as the maximal Lyapunov exponent. From nonlinear time series analysis, there are several methods in the literature to obtain this parameter: Wolf et al. [1], Rosenstein et al. [2], and Kantz [3]. The advantage of Wolf's method is the conceptually simpler and direct approach which converges to the maximal Lyapunov exponent value without subsequent computing. However, the implementation presented in [1] uses a brute force approach which is enough for shorter time series but becomes very expensive in terms of the execution time for larger data records. As far as the authors know, there is not any implementation of the method considering other approaches.

Therefore, the main goal of this work is to obtain a runtime reduction of Wolf's method, bearing in mind the time requirements of the real-world applications for the subsequent prediction stage. Thus, to improve the runtime of the method, we developed a parallel approach introducing kd-tree data structure into the implemented code. In order to assert the results, four time series have been considered as case studies. These data records of one single physic variable

were obtained from the Lorenz system, the Hénon map, the Rössler system, and electrocardiogram signal.

The parallel approach was developed as part of a multidisciplinary framework. Into the framework, a group formed by computer scientists and physicists is working applying high performance programming onto nonlinear time series analysis algorithms. To present this particular work, we have structured the paper as follows. Wolf's method is introduced in Section 2. The parallel approach is introduced in Section 3. The experimental results are shown in Section 4 and the conclusions and future works are presented in Section 5.

2. Wolf's Method

Into the context of nonlinear time series analysis, we mainly focused on time series from deterministic dynamical systems that exhibit sensitive dependence on initial conditions. For the purpose of quantifying this sensitivity, the maximal Lyapunov exponent λ_1 can be used. This exponent describes the average rate at which the predictability is lost [4]. Thus, a crucial fact is that the prediction horizon ρ can be defined by

$$\rho = \frac{1}{\lambda_1}. \tag{1}$$

Before computing λ_1, let us introduce you to the basic concepts related with nonlinear time series analysis. Let $V = \{v_t \in \mathbb{R} \mid t = 0, 1, \ldots, \eta - 1\}$ be a time series, a set of sequence scalar values of one single physic variable which are typically measured at successive times and spaced at uniform time intervals. From these measurement values, points in delay coordinates, also known as delay vectors, can be constructed according to

$$\vec{v}_t = \left(v_t, v_{t+\tau}, \ldots, v_{t+(d-1)\tau}\right), \tag{2}$$

where τ is the embedding delay and d is the embedding dimension (Fraser and Swinney [5]). Note that the number of delay vectors constructed in this manner is $n = \eta - d\tau$.

Takens' embedding theorem [6] states that, for a large enough embedding dimension greater than or equal to $d \in \mathbb{N}$, where $d > 0$, the delay vectors yield a phase space that has exactly the same properties as the one formed by the original variables of the system. We can use the mutual information method [5] to obtain the proper value of τ and the false nearest neighbors method [7] for determining the minimal embedding dimension d.

For the reconstructed d-dimensional phase space obtained from a single variable, we can compute d Lyapunov exponents $\Lambda = \{\lambda_i \in \mathbb{R} \mid \lambda_1 > \lambda_2 > \cdots > \lambda_d\}$. These exponents determine the rate of convergence or divergence for two initially nearby trajectories in that phase space. The fact that if one or more of these exponents are larger than zero we are in presence of a system that exhibits sensitive dependence on initial conditions is a well-established fact [8, 9]. In this case, two arbitrary close trajectories of the system will diverge apart exponentially, eventually ending up in completely different phase space areas as time progresses.

For our purpose, as we mentioned at Introduction, only the maximal Lyapunov exponent computation will be sufficient. To calculate this parameter we use the method introduced by Wolf et al. [1], referred to by Wolf's method. The method works as follows. In the embedding dimension d, we selected a point $\vec{v}_j$ as the starter point, also known as fiducial point, which by default is $\vec{v}_0$. A near neighbor searching is performing to obtain a point $\vec{v}_k$ which satisfies the relation $\|\vec{v}_j - \vec{v}_k\| < \epsilon$, where $\|\cdots\|$ is the square norm of a vector and ϵ is a small constant. Then, both points are iterated forward in time for a fixed evolution time δ, which should be a couple of times larger than the embedding delay τ but not much larger than $d\tau$. If the dynamical system has sensitive dependence on initial conditions, the distance ϵ_δ after the evolved time $\|\vec{v}_{j+\delta} - \vec{v}_{k+\delta}\|$ will be larger than the initial ϵ; that is, $\epsilon_\delta \gg \epsilon$. But, if the dynamical system does not have sensitive dependence on initial conditions, then $\epsilon \approx \epsilon_\delta$. After the evolution, a near neighbor searching is performing to find a new point $\vec{v}_l$, the replacement point, whose distance to the evolved point $\vec{v}_{j+\delta}$ is less than ϵ. The angular separation θ between the vectors constituted by the points $\vec{v}_{j+\delta}$ and $\vec{v}_{k+\delta}$ and $\vec{v}_{j+\delta}$ and $\vec{v}_l$ should be small. If we found a point $\vec{v}_l$ that satisfies the constraint, the point $\vec{v}_{k+\delta}$ is replaced by that point. This procedure is repeated until the fiducial point of the trajectory reaches the last one which is called the fiducial trajectory. Finally, the maximal Lyapunov exponent λ_1 can be calculated according to

$$\lambda_1 = \frac{1}{m\delta}\sum_{t=1}^{m} \ln \frac{\epsilon_\delta}{\epsilon}, \tag{3}$$

where m is the total number of replacement steps and ln denotes the natural logarithm. Figure 1 shows a partial snapshot of the trajectories followed by the computing, and Algorithm 1 depicts the algorithmic notation by Wolf's method. The algorithm is called `fet1` by fixed evolution time for λ_1 such as in the open source code presented by Wolf et al. [1].

3. Parallel Approach Using kd-Tree

The most time-consuming part in the algorithm `fet1` is the neighbor searching (lines 10 and 17 in Algorithm 1). This task searches for the point using the brute force approach. This approach will be enough when working with shorter time series but not for working with larger data records. For this task, a review of neighbor search algorithms, which are especially useful for the study of time series data, can be found in Schreiber [10].

In our experience into the context of nonlinear time series analysis, we have worked with a kd-tree data structure to implement the false nearest neighbor method [7] onto a distributed memory platform [11]. In that work, we showed that the implementation based on kd-tree provides better results in terms of the execution time. Thus, we decide to keep this data structure in the present work.

3.1. The kd-Tree Data Structure. From the computational theory point of view, the kd-tree based algorithms [12, 13] have the advantage of providing an asymptotic number of operations proportional to $O(\log n)$ for a set of n points, which

Program `fet1`(V, τ, d, δ)
Input:
V: data record of η scalar quantities;
τ: embedding delay;
d: minimal embedding dimension;
δ: fixed evolution time;
Output:
For each replacement step $t = 1, 2, \ldots, m$: (1) λ_1: maximal Lyapunov exponent estimation; (2) $t\delta$: evolving step; (3) ϵ: initial separation between points; (4) ϵ_δ: final separation between points.

(1) **begin**
/* `Initialization`. */
(2) Set $n = \eta - d\tau - \delta$ as the useful size of V;
(3) Set $m = n/\delta$ as the number of replacement steps;
(4) Build $\vec{V}$ using (2) as the set of delay vectors;
(5) Compute σ as the standard deviation of V;
(6) Set scalmn = 0.0125σ as the noise scale;
(7) Set scalmx = 0.05σ as an estimation of the useful length scale;
(8) Set $\lambda_1 = 0$;
/* `Computing maximal Lyapunox exponent`. */
(9) Select $\vec{v}_{j=0}$ as the fiducial point;
(10) Search $\vec{v}_k$ such as $\|\vec{v}_j - \vec{v}_k\| >$ scalmn;
(11) **for** $t = 1$ **to** m **do**
(12) Set $\epsilon = \|\vec{v}_j - \vec{v}_k\|$ as the initial separation;
(13) Set $\epsilon_\delta = \|\vec{v}_{j+\delta} - \vec{v}_{k+\delta}\|$ as the final separation;
(14) Set $\lambda_1 = \lambda_1 + \ln(\epsilon_\delta/\epsilon)/t\delta$;
(15) Print $\lambda_1, t\delta, \epsilon, \epsilon_\delta$;
(16) **repeat**
(17) Search $\vec{v}_l$ such as scalmn $\le \|\vec{v}_{j+\delta} - \vec{v}_l\| \le$ scalmx;
(18) **until** θ *was a minimum*;
(19) **if** *a replacement point* $\vec{v}_l$ *was found* **then** $k = l$ **else** $k = k + \delta$;
(20) Set $j = j + \delta$;
(21) **end**

Algorithm 1: Wolf's method by computing λ_1.

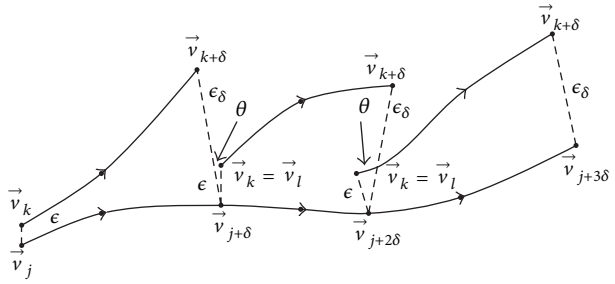

Figure 1: A partial snapshot of Wolf's method. The fiducial point is signed by $\vec{v}_j$. The fiducial trajectory is formed by the points $\vec{v}_j$, $\vec{v}_{j+\delta}$, $\vec{v}_{j+2\delta}$, and $\vec{v}_{j+3\delta}$. For each evolved step, a new replacement point $\vec{v}_l$ for $\vec{v}_k$ is computed. The value of λ_1 converges according to (3) using the values of ϵ and ϵ_δ obtained for each evolved step.

is the best possible performance for arbitrary distribution of elements. The disadvantage of the brute force search approach is the fact that it involves a lot of comparisons with all the array elements one by one. Then, the algorithm complexity of the searching is equivalent to $O(nk)$ for a set of n points with k coordinates. Therefore, it is expected that the runtime will be reduced replacing the brute force algorithm by a kd-tree based algorithm.

In general, a kd-tree data structure considers a set of n points $\vec{v}_i \in \mathbb{R}^k$, where $i = 0, 1, \ldots, n-1$. This data structure is a k-dimensional binary search tree that represents a set of points in a k-dimensional space. The variant described in Freidman et al. [13] distinguishes between two kinds of nodes: internal nodes partition the space by a cut plane defined by a value from the k dimensions; external nodes, also known as buckets, store the points in the resulting hyperrectangles of the partition. For instance, Figure 2 depicts a 2-dimensional space partitioned using cut planes defined by the dimension containing the maximum spread, and Figure 3 shows the kd-tree that represents these partitions.

Although many different versions of kd-trees have been developed, their purpose is always to hierarchically decompose the k-dimensional space into a relatively small number of buckets such that no bucket contains too many points. This decomposition provides a fast way to access any point by position. We traverse down the hierarchy until we find the bucket containing the point and then scan through the few points in the bucket to identify the right one.

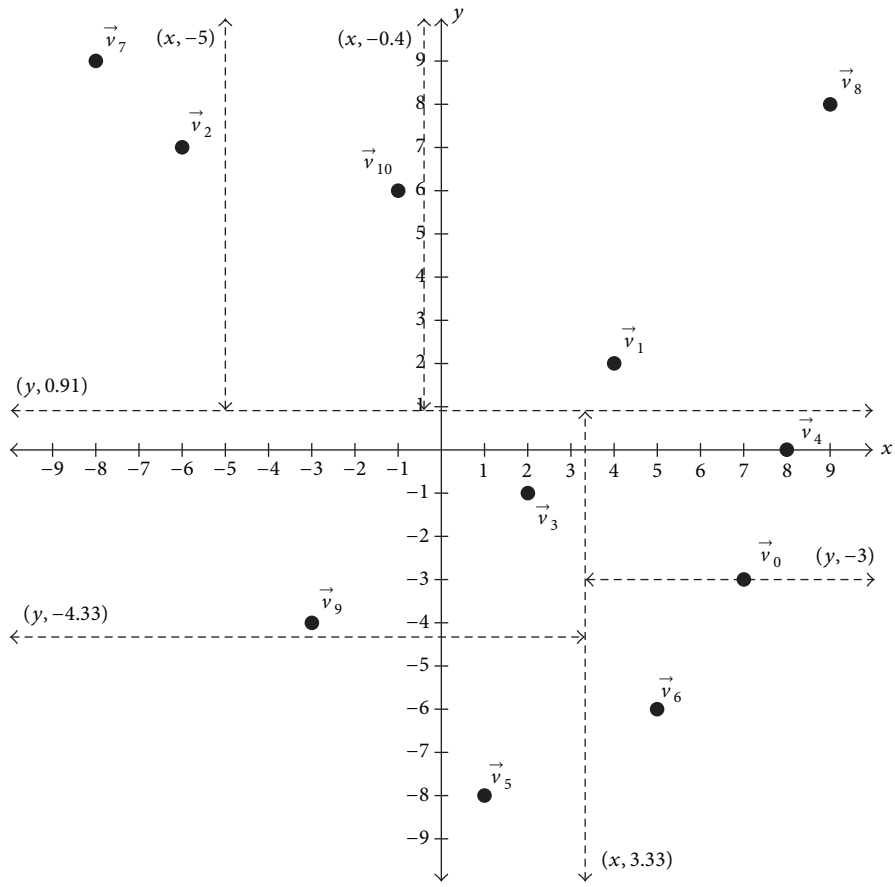

Figure 2: A 2-dimensional space partitioned using cut planes defined by the dimension containing the maximum spread. In the first stage, the whole space is split using the cut dimension y with the cut value 0.91. In the second stage, the bottom partition is split using the cut dimension x with the cut value 3.33, and the top partition is split using the cut dimension x with the cut value −0.4 and so on. Figure 3 shows the kd-tree that represents these partitions.

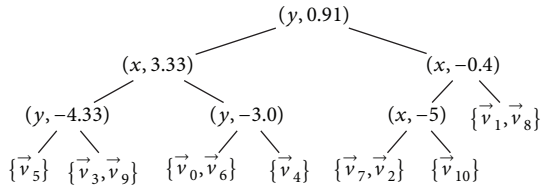

Figure 3: The kd-tree that represents the partitions in Figure 2. The internal nodes have the cut dimension and the cut value for each partition. For each level, all the points contained in the left subtree have values less than or equal to the cut value in the cut dimension; all the points contained in the right subtree have values greater than the cut value in the cut dimension. The external nodes, or buckets, store the points in the resulting hyperrectangles of the partitions.

To perform this work, we have implemented the kd-tree data structure and the kd-tree based functions using the outlines described in [13, 14]. The most important functions are the following:

(i) `newkdtree:` building a kd-tree using as input a set of n points with d dimensions. By definition, a kd-tree data structure has no repeated points, that is, points with similar coordinates. Due to the fact that the index of each point is very important in the course of Wolf's method, we have updated the algorithms in [14] to implement a kd-tree that supports these feature.

(ii) `frnn:` performing a fixed-radius-near-neighbor query. This query reports all points within a fixed radius of the given target point under a specified metric. In this case, the metric used is the square norm of a vector. We also have updated the algorithms in [14] to use both the scalmn and the scalmx parameters (lines 10 and 17 in Algorithm 1). Thus, the resulting query is a set of points which have a distance between scalmn and scalmx to the fiducial point.

3.2. Tasks Decomposition. We use data decomposition to implement the parallel approach. Conceptually, the set of n delay vectors $\vec{V}$ is splitted into p consecutive subsets of length n/p, where p is the number of processes. This data decomposition technique allow us to build a local kd-tree which represents a d-dimensional subspace formed for all the

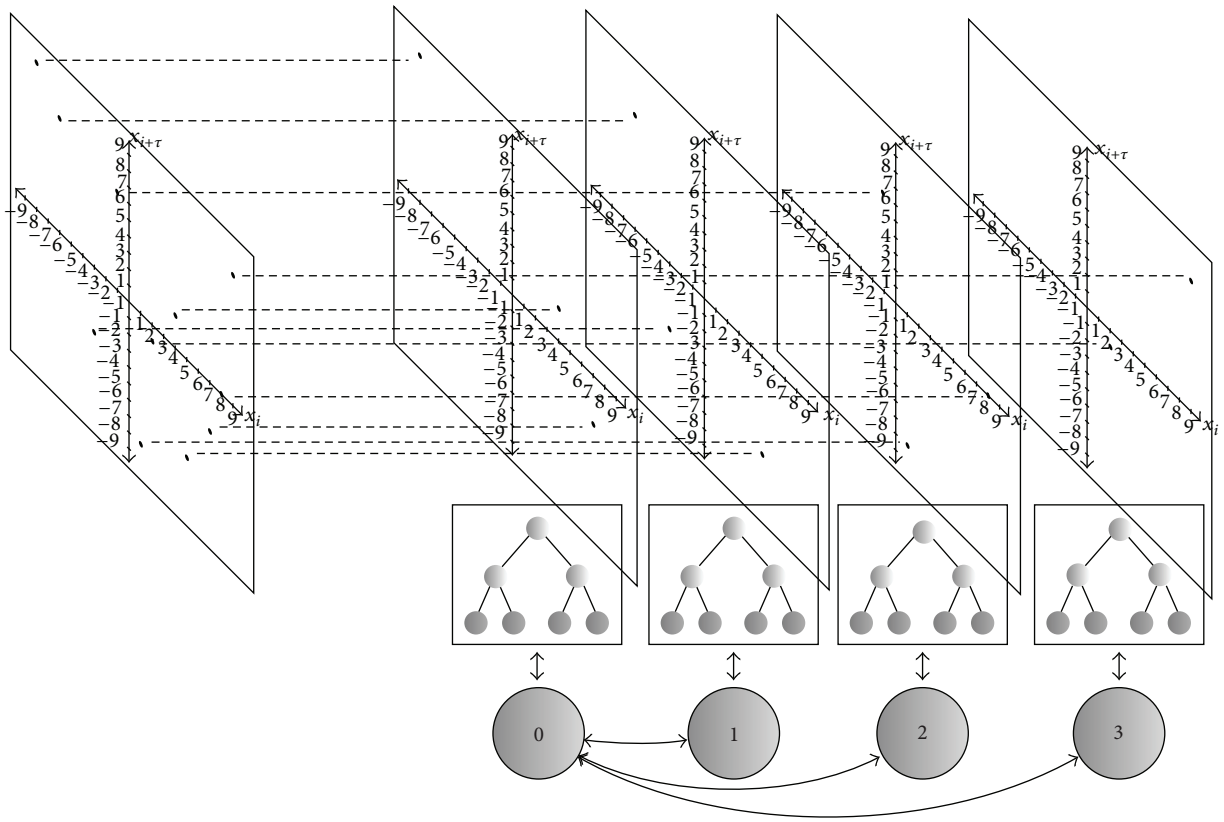

Figure 4: The circles in the figure are the processes with a unique identifier. Each process has a local kd-tree which represents a 2-dimensional subspace formed by the data decomposition technique. The transposition of all the subspaces represents the whole space, such as the space shown in the example of Figure 2. The lines between processes represent the bidirectional synchronization.

points in the subset $\vec{V}_q$, where $q = 0, 1, \ldots, p-1$. Hence, the transposition of all the local subspaces represents the whole space as shown in Figure 4 as an example of 2-dimensional space and 4 processes.

We try to obtain a well-balanced task workload with this coarse-grained decomposition. For this goal, we use balanced trees; that is, to split each subspace in the resulting hyperrectangles, the median of the cut dimension is selected as the cut value.

3.3. Tasks Synchronization. A master-slave model is implemented as the tasks synchronization. We assume a parallel platform model which allows us to run copies of a single program in different processes with unique identifiers for each. Hence, each process uses its process-identifier q to know if it is the MASTER or a slave, where $q = 0, 1, \ldots, p-1$.

For each near neighbors searching, each process calls the function `frnn` to obtain a set of points N from the local kd-tree which are between scalmn and scalmx of the fiducial point $\vec{v}_j$. From N, each process computes which is the local nearest neighbor which has the minimum distance and excludes the others. A synchronization between processes is performed to gather the local results in the MASTER. This process computes which is the global nearest neighbor from p candidates and a new synchronization is performed to broadcast the result from the MASTER to the slaves.

Table 1: Runtime for the sequential brute force approach, considering the four case studies, using time series of one million data records.

Case study	T_s (seconds)
Lorenz [15]	764.103
Hénon [16]	4878.367
Rössler [17]	90.6
ECG	1094.323

Algorithm 2 depicts the algorithmic notation by the parallel Wolf's method approach. We call this approach `pkdfet1` by parallel kd-tree-fixed evolution time for λ_1. Note that the MASTER process has the task to print λ_1 and ρ (see (1)) at the end of the entire processing.

4. Experimental Results

As we mentioned previously, the sequential brute force approach of Wolf's method is enough for shorter time series but becomes very expensive in terms of the execution time for larger data records. This is put in evidence in Table 1, which depicts the sequential runtime T_s of the method presented in Algorithm 1, using data records of one million pieces of data.

Program `pkdfet1`(V, τ, d, δ)
Input:
V: data record of η scalar quantities;
τ: embedding delay;
d: minimal embedding dimension;
δ: fixed evolution time;
Output:
λ_1: maximal Lyapunov exponent estimation;
ρ: prediction horizon;
(1) **in parallel do**:
`/* A process is treated as MASTER and other processes are treated as slaves. The number of processes is` p. `*/`
(2) begin
`/* Initialization. */`
(3) Set $n = \eta - d\tau - \delta$ as the useful size of V;
(4) Set $m = n/\delta$ as the number of replacement steps;
(5) Compute ini and fin as local bounds, where fin − ini + 1 ≈ n/p;
(6) Build $\vec{V}_q = \{\vec{v}_i \in \mathbb{R}^d \mid i = \text{ini}, \ldots, \text{fin}\}$ using (2);
(7) Call `newkdtree`($\vec{V}_q$, fin − ini + 1, d);
(8) Compute σ as the standard deviation of V;
(9) Set scalmn = 0.0125σ as the noise scale;
(10) Set scalmx = 0.05σ as an estimation of the useful length scale;
(11) Set $\lambda_1 = 0$;
`/* Computing maximal Lyapunox exponent. */`
(12) Call `frnn`($\vec{v}_{j=0}$, scalmn, scalmx);
(13) Compute the local nearest neighbor $\vec{v}_k$ from N_q;
(14) **synchronization** Gather all the local nearest neighbor $\vec{v}_k$;
(15) **if** q = MASTER **then** Compute the global nearest neighbor $\vec{v}_k$ from the p candidates;
(16) **synchronization** Broadcast the global nearest neighbor $\vec{v}_k$;
(17) **for** $t = 1$ **to** m **do**
(18) **if** q = MASTER **then**
(19) Set $\epsilon = \|\vec{v}_j - \vec{v}_k\|$ as the initial separation;
(20) Set $\epsilon_\delta = \|\vec{v}_{j+\delta} - \vec{v}_{k+\delta}\|$ as the final separation;
(21) Set $\lambda_1 = \lambda_1 + \ln(\epsilon_\delta/\epsilon)/t\delta$;
(22) **repeat**
(23) Call `frnn`($\vec{v}_{j+\delta}$, scalmn, scalmx);
(24) Compute the local replacement point $\vec{v}_l$ from N_q;
(25) **synchronization** Gather all the local replacement point $\vec{v}_l$;
(26) **if** q = MASTER **then** Compute the global replacement point $\vec{v}_l$ from the p candidates;
(27) **synchronization** Broadcast the global replacement point $\vec{v}_l$;
(28) **until** θ *was a minimum*;
(29) **if** *a replacement point* $\vec{v}_l$ *was found* **then** $k = l$ **else** $k = k + \delta$;
(30) Set $j = j + \delta$;
(31) **if** q = MASTER **then** Print λ_1, $\rho = 1/\lambda_1$;
(32) **end**

Algorithm 2: Parallel kd-tree based approach for computing ρ using Wolf's method.

As far as the authors know, there is not any implementation of that method considering other approaches.

Therefore, the main goal of our parallel implementation is to obtain a runtime reduction of the values shown in Table 1. In general, in order to test the performance and the accuracy of the parallel approach, presented in this work, we have considered four case studies as benchmark. Three time series of one single variable were obtained from the Lorenz system [15], the Hénon map [16], and the Rössler system [17], which correspond with classical examples of discrete and continuous dynamical systems in nonlinear time series analysis. The fourth is a realistic synthetic electrocardiogram (ECG) signal obtained from the reference of McSharry et al. [18]. For each case study, the program computes the prediction horizon using data record of one million pices of data.

We check the performance of the program onto a distributed memory platform called GALGO. (The platform is in the Albacete Research Institute of Informatics, http://www.i3a.uclm.es/.) This parallel computer is a cluster of 64 Intel Xeon E5450 3.0 GHz dual-processor nodes with 32 GB RAM per node and uses an Infiniband interconnection

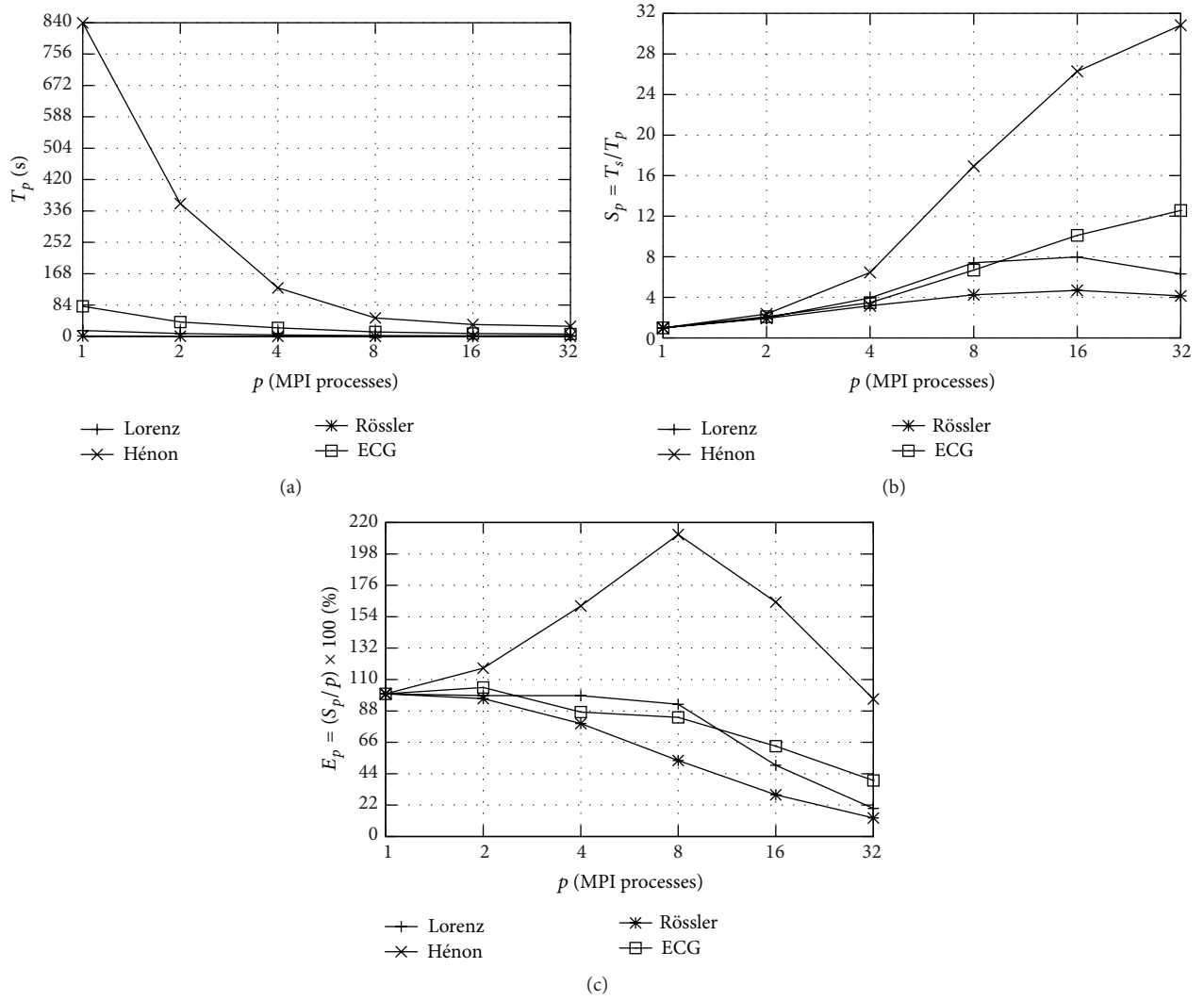

Figure 5: Parallel metrics for the case studies using data records of one million pieces of data. (a) Execution time. (b) Speed-up. (c) Efficiency.

network. Each processor has 4 cores with 6144 KB of cache memory per each. This platform uses Red Hat Enterprise Linux operating system.

The experimental results are presented in terms of execution time, speed-up, and efficiency. These parallel metrics are defined as follows:

Execution time: the sequential runtime of a program is the elapsed time between the beginning and the ending of its execution on a sequential computer. The parallel runtime is the time that elapses from the moment when a parallel computation starts to the moment when the last processing element finishes its execution. We denote the sequential runtime by T_s and the parallel runtime by T_p.

Speed-up: it is a measure that captures the relative benefit of solving a problem in parallel. It is defined as the ratio of the time taken to solve a problem in a single processing element to the time required to solve the same problem on a parallel computer, with p identical processing elements. We denote speed-up by the symbol S_p. Mathematically, it is given by $S_p = T_s/T_p$.

Efficiency: it is a measure of the fraction of time for which a processing element is usefully employed; it is defined as the ratio of speed-up to the number of processing elements. We denote efficiency by the symbol E_p. Mathematically, it is given by $E_p = S_p/p$.

We use the Message Passing Interface MPI [19] to implement the tasks synchronization into the source code of the parallel approach. The implementation of Algorithms 1 and 2 is written in C language (all the codes can be obtained from http://www.dsi.uclm.es/ntsa/). The results obtained are depicted in Figure 5. The runtimes involves the calculus performed from line 12 to line 30 in Algorithm 2.

For clarity, we present the results of the execution time in Table 2. Note that the parallel runtimes for the case $p = 1$ are not the same as those shown in Table 1. This is because at the first stage of this work we implement a sequential approach of

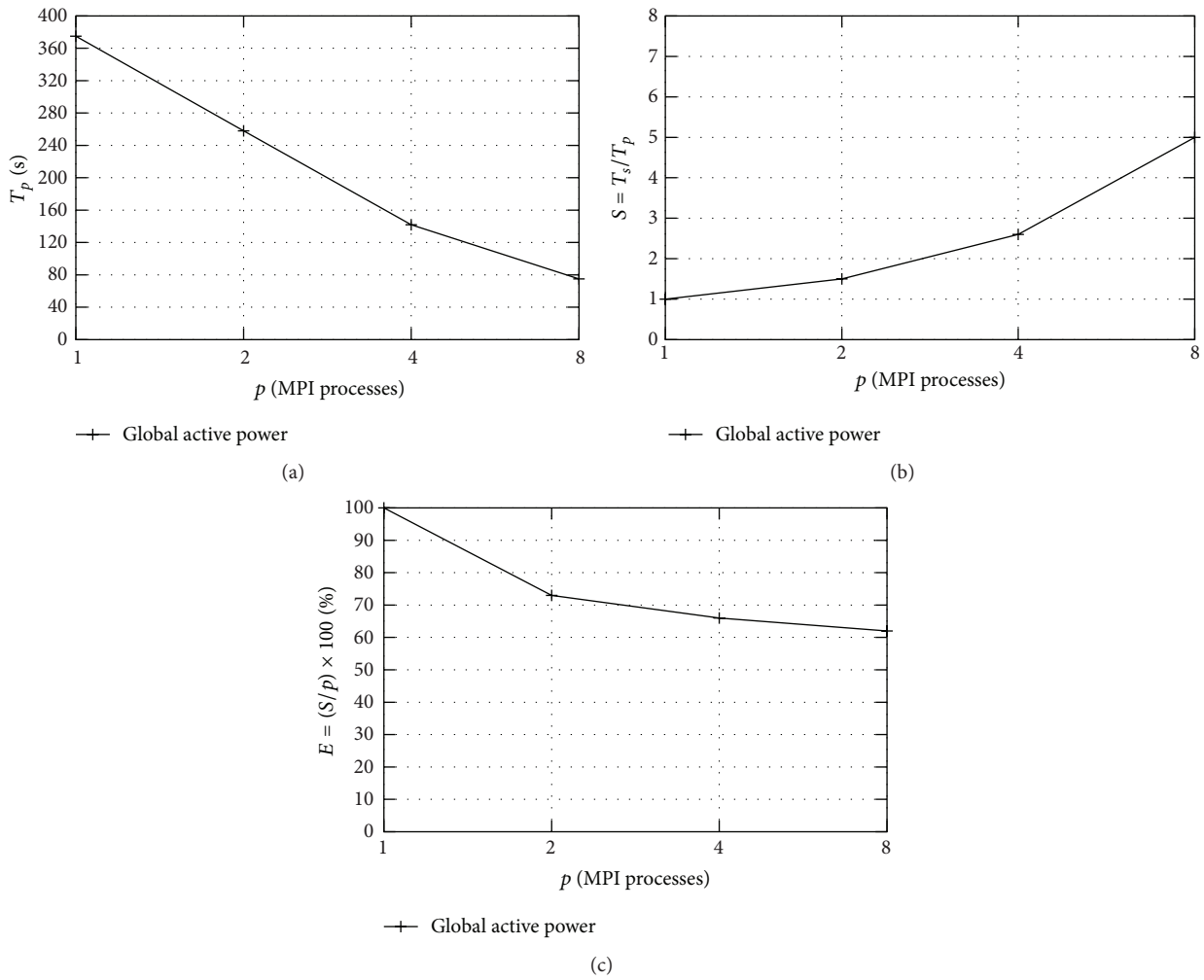

Figure 6: Parallel metrics for a *household global minute-average active power* real case study, using data records of 2 million pieces of data. (a) Execution time. (b) Speed-up. (c) Efficiency.

Table 2: Parallel runtime for the case studies using data records of one million pieces of data.

MPI processes	T_p (seconds) for the case studies			
p	Lorenz [15]	Hénon [16]	Rössler [17]	ECG
1	16.23	839.609	1.647	80.766
2	8.216	355.945	0.853	38.682
4	4.108	129.908	0.52	23.163
8	2.191	49.612	0.387	12.091
16	2.03	31.949	0.352	7.979
32	2.569	27.236	0.397	6.426

Algorithm 1 using a kd-tree data structure, and with this new approach we take times for that case. Hence, as a first result of our work, we have obtained an important reduction of Wolf's method which is from $6x$ (Hénon case) up to $55x$ (Rössler case) times faster than the original approach. In practice, at the second stage of this work, we introduce MPI in this new sequential approach to obtain the final parallel version.

As you can note, for each case, the parallel metrics are quite different between them, despite the fact that all the cases have the same value of n. We can explain this as follows. In first term, we have the fixed evolution time which determines the number of replacement steps that `pkdfet1` must do, for example, we use $\delta = 2$ for Hénon and $\delta = 120$ for Rössler, corresponding with the maximal T_p and the minimal T_p, respectively. In second place, we have the size of the fixed-radius-near-neighbor query which is determined by both scalmn and scalmx variables. At the same time, the query size determines the number of buckets that must be visited. This number of buckets must be increased or decreased or kept constant when we add a major number of processes. If the number of buckets decreases, then the performance gain achieved could be unexpected due to the best behaviour of the kd-tree based algorithms; for example, Hénon depicts a phenomenon known as superlinear speed-up from $p = 2$ to $p = 16$. If the number of buckets increases or remains constant, the speed-up has the expected behaviour as shown

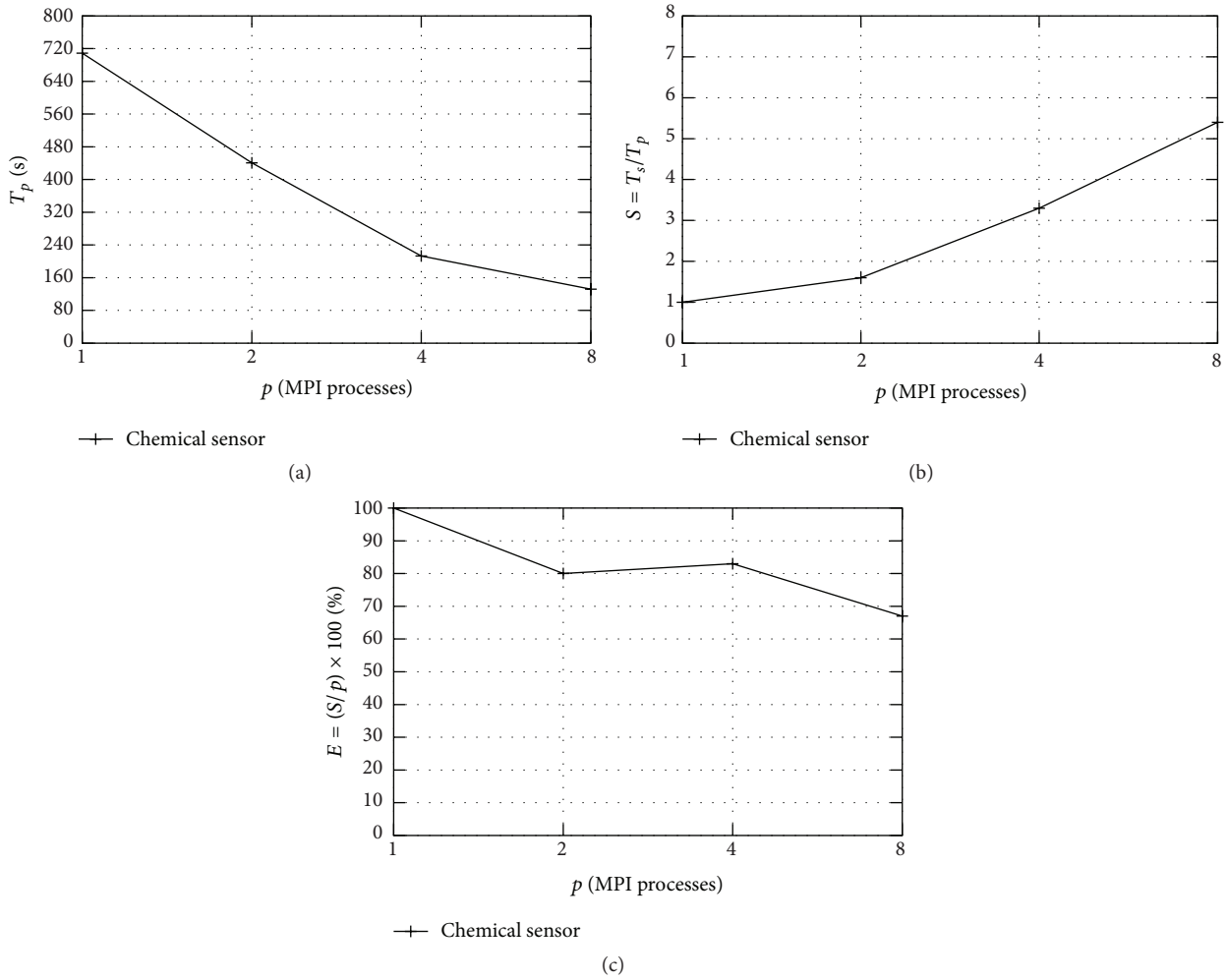

Figure 7: Parallel metrics for a *chemical sensor exposed to gas mixtures* real case study, using data records of 4 million pieces of data. (a) Execution time. (b) Speed-up. (c) Efficiency.

in the other cases, where $S_p < p$. Moreover, the Hénon case does not have a superlinear speed-up in $p = 32$, probably due to the fact that the number of buckets cannot decrease more. In the Rössler case, we cannot achieve better results for parallel metrics due to the fact that the introduction of the kd-tree at the first stage of the work allows us to obtain a very small value for the neighbors searching using $p = 1$. So for $p > 1$ the synchronization has a most important influence on the parallel runtime.

In general, as we expected, the parallel runtime is better when we use a major number of processes, due to the less number of comparisons for each query. However, T_p also involves the processes synchronization. Hence, at the moment when the neighbors searching reaches a very small value, the synchronization has a most important influence on the parallel runtime. This is the situation shown in $p = 32$ for both the Lorenz and Rössler case studies.

Finally, the efficiency showed us that the parallel approach has a better performance when $p \leq 8$ for almost all the cases. In a real-world application, the reasons to use less or more processes will be a decision of the researchers.

At this part, we want to mention that the kd-tree query is performed using a top-down approach over balanced trees. For a bottom-up approach and for unbalanced trees, we have obtained quite similar results.

To check the performance of the parallel approach using real time series, we download two time series, related with engineering system applications, from the UCI Machine Learning Repository [20]. In these cases we use the parallel approach considering the better performance reached when $p \leq 8$, as we commented before. We run the implementation of Algorithm 2 using default values for both the embedding delay and the minimal embedding dimension parameters.

The first case is a time series with measurements of electric power consumption in one household with a one-minute sampling rate over a period of almost 4 years. In particular, the time series is a household global minute-average active power (in kilowatt). The metrics for this case are depicted in Figure 6.

The second case is a time series with measurements of a chemical sensor exposed to gas mixtures at varying concentration levels. This time series was constructed by the

continuous acquisition of the sensor signals for a duration of about 12 hours without interruption. In particular, the sensor was exposed to a gas mixtures of Ethylene and CO in air. The metrics for this case are depicted in Figure 7.

As you can see, both cases have similar performance using default values into the parallel approach, despite the fact that both cases have different value of n: the first case has 2 million pieces of data and the second case has 4 million pieces of data. In general, as we can observe for the theoretical case studies, the parallel runtime is better when using a major number of processes; at the same time both the speed-up and the efficiency decrease with $p \leq 8$, but this one is over 60% until 8 processes.

5. Conclusions and Future Works

A new parallel kd-tree based approach for computing the prediction horizon was presented. The results obtained can be extrapolated below over Wolf's method, which is used as an estimator of the prediction horizon. Moreover, a new sequential kd-tree based approach for that method was developed as part of this work. This sequential approach is from 6 to 55 times faster than the original approach and by itself represents an improvement of the mentioned method.

In the case of the new parallel approach, this is from 2 to 31 times faster than the new sequential approach, considering from 2 to 32 processes onto a distributed memory platform. As was presented, the results depend on each particular case. The better results were obtained when the advantage in the data organization of the kd-tree data structure can be fully exploited.

In terms of the efficiency, for almost all the cases, the better performance is achieved using $p \leq 8$. But, considering the time requirements of the real-world application, if we use 32 processes, the parallel program is from 170 to 297 times faster than the original brute force approach, which represents a time reduction from one hour and 20 minutes to 27 seconds in the best case.

As far as the authors know, to work with a kd-tree data structure onto a distributed memory platform, the literature suggests the same approach by different ways. That is, the local kd-tree building represents a subtree of the original tree formed by a sequential building. Here, a new approach is presented which proposes a decomposition of the whole space in subspaces represented by the local trees; that is, the data decomposition technique is focused on the phase space instead of the kd-tree. For this particular work, this new approach allows us to obtain a well-balanced task workload.

As a future work, we will apply the QR decomposition into Wolf's method to improve the accuracy of the maximal Lyapunov exponent estimation. Through this new parallel implementation, we thought that the whole spectrum of Lyapunov exponents could be computing, despite the fact that we work with time series of one single physic variable instead of working with a model of the dynamical system.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

References

[1] A. Wolf, J. B. Swift, H. L. Swinney, and J. A. Vastano, "Determining Lyapunov exponents from a time series," *Physica D: Nonlinear Phenomena*, vol. 16, no. 3, pp. 285–317, 1985.

[2] M. T. Rosenstein, J. J. Collins, and C. J. De Luca, "A practical method for calculating largest Lyapunov exponents from small data sets," *Physica D: Nonlinear Phenomena*, vol. 65, no. 1-2, pp. 117–134, 1993.

[3] H. Kantz, "A robust method to estimate the maximal Lyapunov exponent of a time series," *Physics Letters A*, vol. 185, no. 1, pp. 77–87, 1994.

[4] J. C. Sprott, *Chaos and Time-Series Analysis*, Oxford University Press, Oxford, UK, 2003.

[5] A. M. Fraser and H. L. Swinney, "Independent coordinates for strange attractors from mutual information," *Physical Review A*, vol. 33, no. 2, pp. 1134–1140, 1986.

[6] F. Takens, "Detecting strange attractors in turbulence," in *Dynamical Systems and Turbulence, Warwick 1980*, vol. 898 of *Lecture Notes in Mathematics*, pp. 366–381, Springer, Berlin, Germany, 1981.

[7] M. B. Kennel, R. Brown, and H. D. I. Abarbanel, "Determining embedding dimension for phase-space reconstruction using a geometrical construction," *Physical Review A*, vol. 45, no. 6, pp. 3403–3411, 1992.

[8] J.-P. Eckmann, S. O. Kamphorst, D. Ruelle, and S. Ciliberto, "Liapunov exponents from time series," *Physical Review A*, vol. 34, no. 6, pp. 4971–4979, 1986.

[9] M. Perc, "Introducing nonlinear time series analysis in undergraduate courses," *Fizika A*, vol. 15, no. 2, pp. 91–112, 2006.

[10] T. Schreiber, "Efficient neighbor searching in nonlinear time series analysis," *International Journal of Bifurcation and Chaos*, vol. 5, no. 2, pp. 349–358, 1995.

[11] J. J. Águila, E. Arias, M. M. Artigao, and J. J. Miralles, "A distributed memory implementation of the False Nearest Neighbors method based on kd-tree applied to electrocardiography," *Procedia Computer Science*, vol. 1, no. 1, pp. 2573–2581, 2010.

[12] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Communications of the ACM*, vol. 18, no. 9, pp. 509–517, 1975.

[13] J. H. Freidman, J. L. Bentley, and R. A. Finkel, "An algorithm for finding best matches in logarithmic expected time," *ACM Transactions on Mathematical Software*, vol. 3, no. 3, pp. 209–226, 1977.

[14] J. L. Bentley, "K-d trees for semidynamic point sets," in *Proceedings of the 6th Annual Symposium on Computational Geometry*, pp. 187–197, June 1990.

[15] E. N. Lorenz, "Deterministic nonperiodic flow," *Journal of the Atmospheric Sciences*, vol. 20, no. 2, pp. 130–141, 1963.

[16] M. Hénon, "Numerical study of quadratic area—preserving mappings," *Quarterly of Applied Mathematics*, vol. 27, no. 3, pp. 291–312, 1969.

[17] O. E. Rössler, "An equation for continuous chaos," *Physics Letters A*, vol. 57, no. 5, pp. 397–398, 1976.

[18] P. E. McSharry, G. D. Clifford, L. Tarassenko, and L. A. Smith, "A dynamical model for generating synthetic electrocardiogram signals," *IEEE Transactions on Biomedical Engineering*, vol. 50, no. 3, pp. 289–294, 2003.

[19] W. Gropp, E. Lusk, and A. Skjellum, *Using MPI: Portable Parallel Programming with the Message-Passing Interface*, MIT Press, Cambridge, Mass, USA, 1994.

[20] K. Bache and M. Lichman, "UCI machine learning repository," vol. 901, 2013, http://archive.ics.uci.edu/ml/.