A Comparative Study of Box-assisted and Kd-tree Approaches for the False Nearest Neighbors Method

Julio J. Águila[a,c], Ismael Marín[b], Enrique Arias[a],
María del Mar Artigao[b], Juan J. Miralles[b*]

Abstract—**In different fields of science and engineering (medicine, economics, oceanography, biological systems, etc.) the False Nearest Neighbors (FNN) method has a special relevance. In some of these applications, it is important to provide the results in a reasonable time scale, thus the execution time of the FNN method has to be reduced. To achieve this goal, a multidisciplinary group formed by computer scientists and physicists are collaborative working on developing High Performance Computing implementations of one of the most popular algorithms that implement the FNN method: based on box-assisted algorithm and based on kd-tree data structure. In this paper, a comparative study of the distributed memory architecture implementations carried out in the framework of this collaboration, is presented. As a result, two parallel implementations for box-assisted algorithm and one parallel implementation for the kd-tree structure are compared in terms of execution time, speed-up and efficiency. In terms of execution time, the approaches presented here are from 2 to 16 times faster than the sequential implementation, and the kd-tree approach is from 3 to 7 times faster than the box-assisted approaches.**

Keywords: ***Nonlinear Time Series Analysis, False Nearest Neighbors method, box-assisted algorithm, kd-tree data structure, Message Passing Interface***

1 Introduction

In nonlinear time series analysis the False Nearest Neighbors (FNN) method is crucial to the success of the subsequent analysis. Many fields of science and engineering use the results obtained with this method. But the complexity and size of the time series increases day to day and it is important to provide the results in a reasonable time scale. For example, in the case of electrocardiogram study (ECG), this method have to achieve real-time performance in order to take some prevention actions. With the development of the parallel computing, large amounts of processing power and memory capacity are available to solve the gap between size and time.

We have applied the paradigm of parallel computing to implement three approaches directed towards distributed memory architectures, in order to make a comparative study between the method based on the box-assisted algorithm and the method based on the kd-tree data structure. The results are presented in terms of performance metrics for parallel systems, that is, execution time, speed-up and efficiency. Two case studies have been considered to carried out this comparative study. A theoretical case study which consists on a Lorenz model, and a real case study which consists on a time series belonging to electrocardiography.

The paper is organized as follow. After this introduction, a description of the FNN method is presented in Section 2 and the approaches considered are introduced on Section 3. On section 4, the experimental results are presented. Finally, on Section 5 some conclusions and future work are outlined.

2 The FNN Method

Dynamical systems are studied from two different points of view. One is from a previously known model which explains its behavior while the other is from a time series carried out by means of successive data acquisition per constant time periods. This methodology becomes the basis of nonlinear time series analysis and is based on the reconstruction of the phase space in a dynamical system from the Takens embedding theorem [3].

There are many situations in which only one of the relevant dynamical variables can be measured. For this reason, time series data should be converted into phase space vectors. This procedure is known as phase space reconstruction, which is based on Takens' theorem. Let $X = \{x(i) : 0 \leq i < n\}$ a time series, the phase space reconstruction can be performed using the method of delays as is shown in Eq. (1):

*Albacete Research Institute of Informatics[a] and Applied Physics Dept.[b], University of Castilla-La Mancha, Avda. España s/n, 02071-Albacete, Spain. Depto. Ingeniería en Computación [c], Universidad de Magallanes, Avda. Bulnes 01855, Punta Arenas, Chile. Email for correspondence: juliojose.aguila@alu.uclm.es.

$$y(i) = [x(i), x(i+\tau), \ldots, x(i+(d-1)\tau)] \quad (1)$$

where τ is the embedding delay and d is the embedding dimension (Fraser and Swinney [2]). The number of delay vectors in delay coordinates that we can obtain, given a number n of time series data, is $n-(d-1)\tau$. The Takens' theorem states that for a large enough embedding dimension $d \geq m_0$, the delay vectors yield a phase space that has exactly the same properties as the one formed by the original variables of the system.

The FNN method was introduced by Kennel et al. [1] and supposes that the minimal embedding dimension for a given time series is m_0. In this way, the reconstructed system in a m_0-dimensional delay space is a one-to-one image of the system in the original phase space. Thus, the neighbors of a given point are mapped onto neighbors in the delay space. If a d-dimensional space ($d < m_0$) is considered, then the topological structures are not preserved and the points are projected into neighborhoods of other points to which they would not belong in higher dimensions. In this case, these points are called false neighbors.

The idea of the FNN method is to measure the distances between a point $y(i)$ and its nearest neighbor $y(j)$; as this dimension increases, this distance should not change if the points are really nearest neighbors. If we define the distance between a point and its nearest neighbor using Euclidean distance, we can evaluate the change in distance by adding one more dimension and then we can look at the relative change in the distance as a way to see if our points were not really close together but a projection from a higher phase space. The criterion for falseness is thus:

$$\frac{|x(i+d\tau) - x(j+d\tau)|}{||y(i) - y(j)||} > R_{tr} \quad (2)$$

where R_{tr} is a given threshold. Eq. (2) has to be calculated for the whole time series and for several dimensions $d = \{1, 2, \ldots, m\}$ until the fraction of points, which must be lower than R_{tr}, is zero, or at least sufficiently small (in practice, lower than 1%). Working in any dimension larger than the minimum m_0 leads to excessive computation when investigating any subsequent question (Lyapunov exponents, prediction, etc.).

While greater is the value of n (length of the time series), the task to find the nearest neighbor for each point is more computationally expensive. A review of methods to find nearest neighbors, which are particularly useful for the study of time series data, can be found in [4]. We focused in two approaches: based on the box-assisted algorithm, optimized in the context of time series analysis by Grassberger [5]; and the based in a kd-tree data structure [6, 7] developed in the context of computational geometry.

3 Parallel Approaches

According to Schreiber [4], for time series that have a low dimension of embedding (e.g. up to the 10's), the box-assisted algorithm is particularly efficient. This algorithm can offer a lower complexity of $O(n)$ under certain conditions. By the other hand, accordingly with the literature if the dimension of embedding is moderate an effective method for nearest neighbors searching consists in using a kd-tree data structure [6, 7]. From the computational theory point of view, the kd-tree-based algorithm has the advantage of providing an asymptotic number of operations proportional to $O(n \log n)$ for a set of n points, which is the best possible performance for arbitrary distribution of elements.

We selected two programs to start this work: the `false_nearest` program based on the box-assisted algorithm [8, 9]; and the `fnn` program based on a kd-tree data structure [10].

We employ the paradigm *Single-Program, Multiple Data* (SPMD) [11] to design the three parallel approaches. A coarse-grained decomposition [12] has been considered, i.e. we have a small number of tasks in parallel with a large amount of computations. The approaches are directed towards distributed memory architectures using the Message Passing Interface (MPI [13]) standard for communication purpose. Two approaches are based on the box-assisted algorithm and the another approach is based on the kd-tree data structure.

3.1 Approaches Based on Box-assisted Algorithm

The box-assisted algorithm [5] considers a set of n points $y(i)$ in k dimensions. For each $y(i)$ the algorithm determines all neighbors closer than ϵ, i.e. the set of indices shown in Eq. (3).

$$U_i(\epsilon) = \{j : \| y(i) - y(j) \| < \epsilon\} \quad (3)$$

The idea of the method is as follow. Divide the phase space into a grid of boxes of side length ϵ. Each point $y(i)$ lies into one of these boxes. The nearest neighbors closer than ϵ there are located in either the same box or one of the adjacent boxes.

In practice, the grid of boxes is implemented using an array `BOX` of $s \times s$ elements. Given a time series X, each $x(i)$ is put in `BOX` using Eqs. (4, 5), where & is the bitwise operator. These equations are based on Eq. (1), i.e. these values represents the point $y(i)$.

$$row = \frac{x(i-(d-1)\tau}{\epsilon}\&(s-1) \quad (4)$$

$$column = \frac{x(i)}{\epsilon}\&(s-1). \quad (5)$$

The `false_nearest` program is a sequential implementation of the FNN method based on this algorithm. By profiling the `false_nearest` program in order to carry out the parallel approaches, four tasks were identified. Let p the number of processes, two parallel implementations were formed based on these four tasks:

Grid Construction The `BOX` array is filled. Two ways of grid construction have been developed: S (Sequential) and P (Parallel). In a S construction each process fills the BOX sequentially, thus each one has a copy. In a P construction, a range of $\frac{s}{p}$ rows is assigned to each process, that is, each one considers the $x(i)$ values that has its row on the assigned range according to the Eq. (4).

Domain Decomposition Time series X is distributed to the processes considering two ways of decomposition related with the grid construction: Time Series (TS) and Mesh (M). In a TS data distribution the time series is split into p uniform parts of length $\frac{n}{p}$, being n the length of the time series. In a M data distribution, each process considers the values in X that has its row on the assigned range by the P construction.

Nearest Neighbors Search The nearest neighbors are searched into the ϵ ratio used in the grid construction. Each process computes the fraction of false nearest neighbors as a subproblem given the values $x(i) \in X$ assigned according to the domain decomposition.

Communication of Results When the nearest neighbors were searched, the processes are synchronized using the MPI interface. If the ϵ value was not appropriate to find all the nearest neighbors, then all the processes must be adjust the ϵ value and rebuild the grid. When the calculus of false nearest neighbors fraction, for a given dimension d, has been completed, the processes are synchronized to communicate their results to the master process assigned by programming code.

The approaches were called following the next nomenclature: `DM-S-TS` meaning a **D**istributed **M**emory implementation considering that the grid construction is **S**equential and the **T**ime **S**eries is uniformly distributed to the processes; `DM-P-M` meaning a **D**istributed **M**emory implementation considering that the grid construction is in **P**arallel and the time series is distributed according to the **M**esh.

The algorithmic notation for both the `DM-S-TS` and `DM-P-M` approaches are depicted in Algorithms (1, 2). We have introduced MPI functions into the source codes to obtain the programs that can be run into a distributed memory platform. The most important MPI functions used in these programs are the follows:

- `MPI_Reduce` Combines values provided from a group of MPI processes and returns the combined value in the MASTER process.
- `MPI_Allreduce` Same as `MPI_Reduce` except that the result appears in all the MPI processes.

Algorithm 1: `DM-S-TS` approach: FNN method based on a box-assisted algorithm

```
Program DM-S-TS(m, τ, X)
Input:
m = maximal embedding dimension to compute
τ = embedding delay
X = time series data record of length n
Output:
The fraction of false nearest neighbors fnn for each dimension
d = {1, 2, ..., m}.
in parallel do:
begin
    /* Let p the total number of MPI processes. Each
    process has an identifier q = {0, 1, ..., p − 1}. The
    process q = 0 is treated as MASTER and processes
    with q ≠ 0 are treated as slaves.                    */
    Computing bounds ini and end to get same number of data
    n/p of X;
    for d = 1 to m do
        Setting the initial value of ε;
        Setting the control variable alldone to FALSE;
        while alldone = FALSE and ε < threshold do
            Building BOX using ε, τ, d and X;
            foreach i into X(ini : end) do
                Searching the nearest neighbor of y(i);
                if nearest(i) is found then
                    Computing if nearest(i) is a false nearest;
                end
            end
            synchronization Calling to
            MPI_Allreduce(alldone);
            /* If alldone = TRUE meaning that all
            nearest were founded.                        */
            Updating ε;
        end
        synchronization Calling to MPI_Reduce(fnn);
        if q = MASTER then
            Printing fnn;
        end
    end
end
```

3.2 Approach Based on the kd-tree Data Structure

A kd-tree data structure [6, 7] considers a set of n points $y(i)$ in k dimensions. This tree is a k-dimensional binary search tree that represents a set of points in a

k-dimensional space. The variant described in Friedman et al. [7] distinguishes between two kinds of nodes: internal nodes partition the space by a cut plane defined by a value of the k dimensions (the one containing a maximum spread), and external nodes (or buckets) store the points in the resulting hyperrectangles of the partition. The root of the tree represents the entire k-dimensional space.

Algorithm 2: DM-P-M approach: FNN method based on a box-assisted algorithm

```
Program DM-P-M(m, τ, X)
Input:
m = maximal embedding dimension to compute
τ = embedding delay
X = time series data record of length n
Output:
The fraction of false nearest neighbors fnn for each dimension
d = {1, 2, ..., m}.
in parallel do:
begin
    /* Let p the total number of MPI processes. Each
    process has an identifier q = {0, 1, ..., p − 1}. The
    process q = 0 is treated as MASTER and processes
    with q ≠ 0 are treated as slaves.                  */
    Computing bounds first and last to get same number of
    rows s/p of BOX;
    for d = 1 to m do
        Setting the initial value of ε;
        Setting the control variable alldone to FALSE;
        while alldone = FALSE and ε < threshold do
            Building BOX(first : last) using ε, τ, d and X;
            foreach y(i) into BOX(first : last) do
                Searching the nearest neighbor of y(i);
                if nearest(i) is found then
                    Computing if nearest(i) is a false nearest;
                end
            end
            synchronization Calling to
            MPI_Allreduce(alldone);
            /* If alldone = TRUE meaning that all
            nearest were founded.                      */
            Updating ε;
        end
        synchronization Calling to MPI_Reduce(fnn);
        if q = MASTER then
            Printing fnn;
        end
    end
end
```

From the point of view of computational theory, the number of operations required to build a kd-tree with n points in k dimensions is proportional to $O(kn \log n)$. The expected value to find a nearest neighbor is $O(\log n)$, and the number of operations required to find all the neighbors is proportional to $O(n \log n)$.

The `fnn` program is a sequential implementation of the FNN method based on this structure. `fnn` program has been also analyzed by means of a profile tool before making the parallel implementation, identifying five main tasks. Thus, let X a time series, n the length of the time series, Y a set of points constructed according to Eq. (1), `KDTREE` a data structure that implements the kd-tree, p the number of processes, and $q = \{0, 1, \ldots, p-1\}$ a process identifier. For convenience we assume that p is a power of two. The parallel implementation called `KD-TREE-P` was formed based on these five tasks:

Global kd-tree building The first $\log p$ levels of `KDTREE` are built. All processors perform the same task, thus each one has a copy of the global tree. The restriction $n \geq p^2$ is imposed to ensure that the first $\log p$ levels of the tree correspond to nonterminal nodes instead of buckets.

Local kd-tree building The local `KDTREE` is built. In the level $\log p$ of the global tree are p nonterminal nodes. Each processor q builds a local kd-tree using the $(q+1)$th-node like root. The first $\log p$ levels are destroyed and `KDTREE` is pointed to local tree.

Domain Decomposition Time series X is distributed to the processes. The building strategy imposes a distribution over the time series. Thus, the time series is split according to the kd-tree algorithm and the expected value of items contained in each local tree is approximately $\frac{n}{p}$.

Nearest Neighbors Search Each process solves their subproblems. Each process searches the nearest neighbors for all points in Y that are in the local `KDTREE`.

Communication of Results Processes use MPI to communicate theirs partial results at the end of whole dimensions. The master process collects all partial results and reduces them.

Algorithm 3 depicts the algorithmic notation for the `KD-TREE-P` approach.

4 Experimental results

In order to test the performance of the parallel implementations, we have considered two case studies: the Lorenz time series generated by the equations system described in 1963 by E.Lorenz [14]; the electrocardiogram (ECG) signal generated by a dynamical model introduced in 2003 by McSharry et al. [15]. The Lorenz system is a benchmark problem in nonlinear time series analysis and the ECG model is used for biomedical science and engineering [16].

The parallel implementations have been run in a supercomputer called GALGO, which belongs to the Albacete Research Institute of Informatics [17]. The parallel platform consists in a cluster of 64 machines. Each machine has two processors Intel Xeon E5450 3.0 GHz and 32 GB of RAM memory. Each processor has 4 cores with 6144 KB of cache memory. The machines are running RedHat

Algorithm 3: KD-TREE-P approach: FNN method based on a kd-tree data structure

```
Program KD-TREE-P(m, τ, X)
Input:
m = maximal embedding dimension to compute
τ = embedding delay
X = time series data record of length n
Output:
The fraction of false nearest neighbors fnn for each dimension
d = {1, 2, ..., m}.
in parallel do:
begin
    /* Let p the total number of MPI processes. Each
    process has an identifier q = {0, 1, ..., p − 1}. The
    process q = 0 is treated as MASTER and processes
    with q ≠ 0 are treated as slaves.                    */
    for d = 1 to m do
        Building delay vectors Y using X and τ;
        Building first log(p) levels of KDTREE;
        Building (q + 1)th-node of KDTREE on level log(p);
        /* The (q + 1)th-node is the new root for
        KDTREE.                                          */
        foreach y(i) into KDTREE do
            Searching the nearest neighbor of y(i);
            Computing if nearest(i) is a false nearest;
        end
        synchronization Calling to MPI_Reduce(fnn);
        if q = MASTER then
            Printing fnn;
        end
    end
end
```

Enterprise version 5 and using an Infiniband interconnection network. The cluster is presented as a unique resource which is accessed through a front-end node.

The results are presented in terms of performance metrics for parallel systems described in Grama et al. [12]: execution time T_p, speed-up S and efficiency E. These metrics are defined as follows:

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

- **Speed-up** 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 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.

- **Efficiency** 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. Mathematically, it is given by $E = \frac{S}{p}$.

Let p the number of processors, the execution time of the approaches have been tested for $p = \{1, 2, 4, 8, 16, 32\}$, where $p = 1$ corresponds to the sequential version of the approaches. We used one million records of the time series to calculate the ten first embedding dimensions. We have obtained that the optimal time delay for Lorenz time series is $\tau = 7$ and for ECG signal is $\tau = 5$ using the mutual information method.

In order to obtain the best runtime of the approaches based in a box-assisted algorithm we found the best size of `BOX` for each value of p (tables 1 and 2). The size of `BOX` defines the number of rows and columns for the grid of boxes. The values for $p = 1$ corresponds to the sequential version of the program `false_nearest`.

Table 1: Size of `BOX` for each value of p using a Lorenz time series.

p	DM-P-M	DM-S-TS
1	8192	8192
2	4096	4096
4	2048	4096
8	2048	4096
16	2048	2048
32	2048	2048

Table 2: Size of `BOX` for each value of p using a ECG time series.

p	DM-P-M	DM-S-TS
1	4096	4096
2	4096	4096
4	4096	4096
8	2048	4096
16	2048	2048
32	2048	2048

We have run 10 tests to obtain the median value of the execution time T_p. In total 360 tests were performed. The performance metrics results are shown in Figs.[1-6].

Sequential kd-tree implementation shows a lower execution time than box-assisted approach, since the grid construction stage on box-assisted implementation in TISEAN is very expensive in terms of execution time.

The behavior of the Lorenz case study and the ECG case study is quite similar. Notice that, according to Fig. 2 and Fig. 5, it is possible to appreciate a super-linear speed-up for kd-tree implementation when $p < 8$ and these performance decreases when $p > 8$. The super-linear speed-up is explained due to the fact that the cache memory is better exploited and that when the tree is split

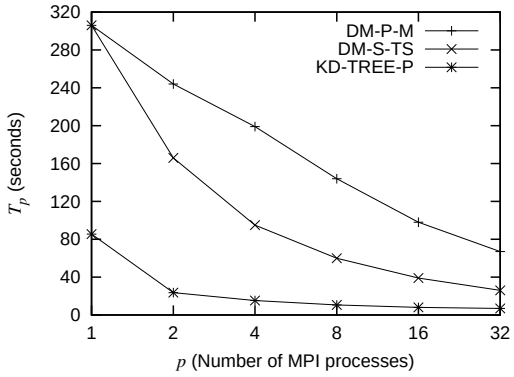

Figure 1: Execution Time for Lorenz case study.

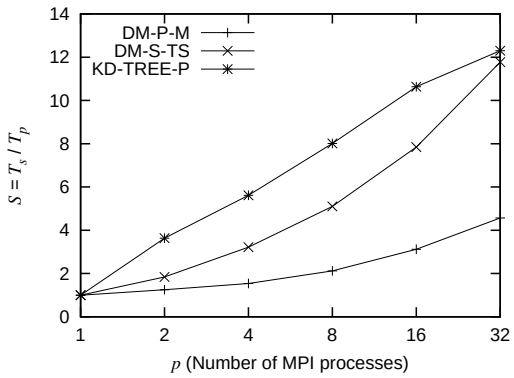

Figure 2: Speed-up for Lorenz case study.

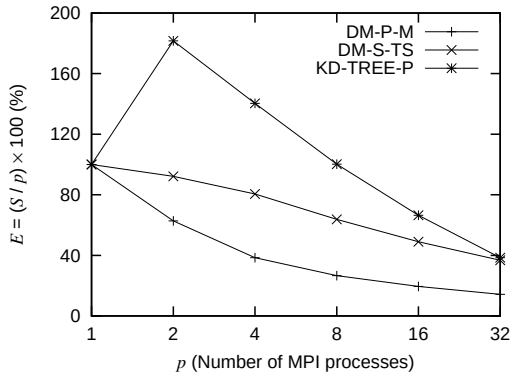

Figure 3: Efficiency for Lorenz case study.

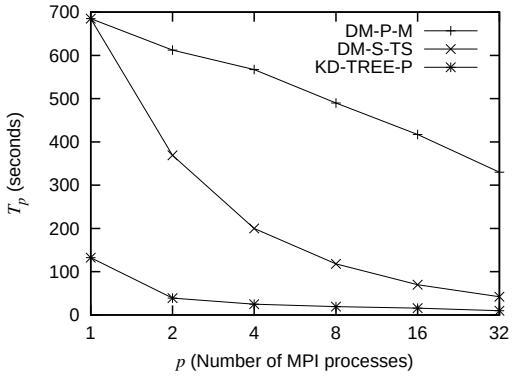

Figure 4: Execution Time for ECG case study.

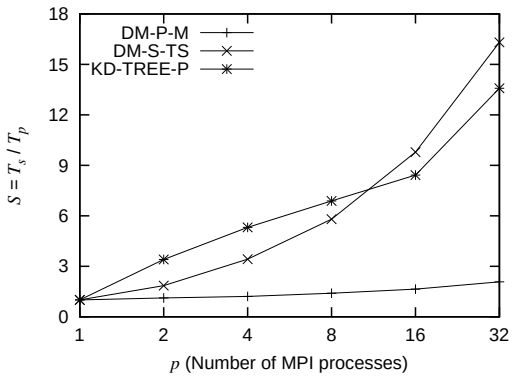

Figure 5: Speed-up for ECG case study.

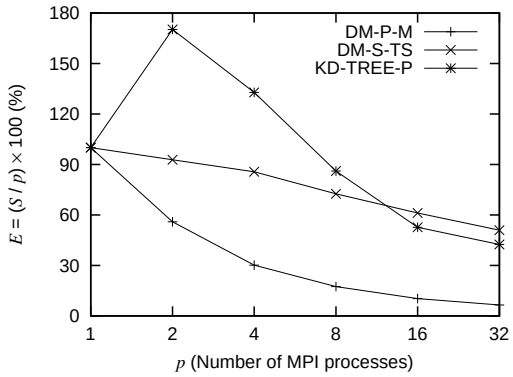

Figure 6: Efficiency for ECG case study.

less searches have to be done at each subtree. With respect to the lost of performance, this situation is produced due to different causes. The first one is that, evidently, the overhead due to communications increases. Also, the most important cause is that the sequential part of our implementation becomes every time more relevant with respect to the parallel one.

Considering only the box-assisted implementations, DM-S-TS is the box-assisted approach that provides the best results for the Lorenz attractor and the ECG signal. The reason is the very best data distribution with regard to DM-P-M. However, the reconstruction of the mesh is not parallelized in DM-S-TS implementation. So, the sequential part makes the reduction of execution time less significant when more CPUs are used. However, as the execution time of find neighbors is increased (e.g. in larger times series data), this circumstance becomes very less important.

For Lorenz attractor, the DM-S-TS implementation is around 1.8 faster than the sequential program when it

uses 2 CPUs, and around 12 when it uses 32 CPUs. This means that the efficiency for 2 CPUs is around 92% and decreases to 37% when using 32 CPUs. For ECG signal, the best box-assisted parallel implementation achieves a speed-up of around 16 when it is run on 32 CPUs of GALGO. Moreover, the time saving is around 93% using 2 CPUs and 51% using 32 CPUs. Unlike previous case, the efficiency of best implementation decreases more slowly.

An optimization of TISEAN has been used. It allows the best mesh size to be tuned for each case. In case of use original TISEAN (fixed mesh size), the reduction of execution time would be more important.

According to the experimental results, kd-tree-based parallel implementation obtains the best performance than the box-assisted-based parallel implementation, almost in terms of execution time, for both case studies. Due to the spectacular execution time reduction provided by the kd-tree-based parallel implementation, the performance in terms of speed-up and efficiency seems to be worst, with respect to the other approaches.

5 Conclusions

In this paper, a comparative study between the distributed memory implementations of two different ways to compute the False Nearest Neighbors method have been presented, that is, the based on the box-assisted algorithm and the based on kd-tree data structure. To make this comparative study three different implementations have been developed: two implementations based on box-assisted algorithm, and one implementation based on kd-tree data structure.

The most important metric to consider is how well the resulting implementations accelerate the compute of the minimal embedding dimension, which is the ultimate goal of the FNN method. In terms of the execution time, the parallel approaches are from 2 to 16 times faster than the sequential implementation, and the kd-tree approach is from 3 to 7 times faster than the box-assisted algorithm.

With respect to the experimental results, the kd-tree-based parallel implementation provides the best performance in terms of execution time, reducing dramatically the execution time. As a consequence, the speed-up an efficiency are far from the ideal. However, it is necessary to deal with more case studies of special interest for the authors: wind speed, ozone, air temperature, etc.

About related works, in the context of parallel implementations to compute FNN method, the work carried out by the authors could be considered as the first one. The authors are working also on considering shared memory implementations (Pthreads [18, 19] or OpenMP [20, 21]) and hybrid (MPI+Pthreads or MPI+OpenMP) parallel implementations. Also, as a future work, the author are considering to develop GPU-based parallel implementation of the algorithms considered in this paper.

To sum-up, we hope that our program will be useful in applications of nonlinear techniques to analyze real time series as well as artificial time series. This work represents the first step of nonlinear time series analysis, that it is becomes meaningful when considering ulterior stages on the analysis as prediction, and when for some applications the time represents a crucial factor. This work is based on the paper that was presented in the WCE 2010 [22].

Acknowledgement

This work has been supported by National Projects CGL2007-66440-C04-03 and CGL2008-05688-C02-01/CLI. A short version was presented in [22]. In this version, we have introduced the algorithmic notation by the parallel implementations and more details about the FNN method and the box-assisted algorithm.

References

[1] Kennel, M.B. and Brown, R. and Abarbanel, H.D.I. (1992). Determining Embedding Dimension for Phase Space Reconstruction Using the Method of False Nearest Neighbors. *Physics Review A*, 45(6):3403–3411.

[2] Fraser, A.M. and Swinney, H.L. (1986). Independent coordinates for strange attractors from mutual information. *Physical Review A*, 33(2):1134–1140.

[3] Takens, F. (1981). Detecting strange attractors in turbulence. *In Rand, D.A. and Young, L.-S. (eds.) Dynamical Systems and Turbulence, Warwick 1980, Springer: New York, pp. 366-381.*

[4] Schreiber, T. (1995). Efficient neighbor searching in nonlinear time series analysis. *Int. J. Bifurcation and Chaos*, 5:349.

[5] Grassberger, P. (1990). An optimized box-assisted algorithm for fractal dimensions. *Physics Letters A*, 148(1-2):63–68.

[6] Bentley, J.L. (1975). Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517.

[7] Friedman, J.H. and Bentley, J.L. and Finkel, R.A. (1977). An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software (TOMS)*, 3(3):209–226.

[8] Hegger, R. and Kantz, H. and Schreiber, T. (1999). Practical implementation of nonlinear time series methods: The TISEAN package. *Chaos*, 9(2):413–435.

[9] Hegger, R. and Kantz, H. and Schreiber, T. (2007). Tisean: Nonlinear time series analysis. *http://www.mpipks-dresden.mpg.de/tisean*.

[10] Kennel, M.B. (1993). Download page of `fnn` program. *ftp://lyapunov.ucsd.edu/pub/nonlinear/fns.tgz*.

[11] Darema, F. (2001). The spmd model: Past, present and future. *LECTURE NOTES IN COMPUTER SCIENCE*, pages 1–1.

[12] Grama, A. and Gupta, A. and Karypis, G. and Kumar, V. (2003). *Introduction to Parallel Computing*. Addison-Wesley New York.

[13] Message Passing Interface. *http://www.mcs.anl.gov/research/projects/mpi/*.

[14] Lorenz, E.N. (1963). Deterministic Nonperiodic Flow. *Journal of the Atmospheric Sciences*, 20(2):130–141.

[15] McSharry, P.E. and Clifford, G.D. and Tarassenko, L. and Smith, L.A. (2003). A dynamical model for generating synthetic electrocardiogram signals. *IEEE Transactions on Biomedical Engineering*, 50(3):289–294.

[16] ECGSYN (2003). Ecgsyn: A realistic ecg waveform generator. *http://www.physionet.org/physiotools/ecgsyn*.

[17] Albacete Research Institute of Informatics. *http://www.i3a.uclm.es*.

[18] Mueller, F. (1999). Pthreads Library Interface. Institut fur Informatik.

[19] Wagner, T. and Towsley, D. (1995). Getting started with POSIX threads. Department of Computer Science, University of Massachusetts.

[20] Dagum, L. (1997). OpenMP: A Proposed Industry Standard API for Shared Memory Programming. OpenMP.org.

[21] Dagum, L. and Menon, R. (1998). OpenMP: An industry-standard API for shared-memory programming. *IEEE Computational Science and Engineering*, 46–55.

[22] Águila, J.J. and Marín, I. and Arias, E. and Artigao, M.M. and Miralles, J.J. (2010). Distributed Memory Implementation of the False Nearest Neighbors Method: Kd-tree approach versus Box-assisted approach. *Lecture Notes in Engineering and Computer Science: Proceedings of The World Congress on Engineering 2010*, WCE 2010, 30 June - 2 July, 2010, London, U.K., pp 493-498.