University of Castilla-La Mancha

A publication of the

Computing Systems Department

AN HYBRID MPI+PTHREAD IMPLEMENTATION FOR COMPUTING THE FALSE NEAREST NEIGHBORS METHOD ON CLUSTER OF SMP ARCHITECTURES

by

I. Marín, E. Arias, M. M. Artigao and J. J. Miralles

Technical Report **#DIAB-09-07-1** July, 2009

This work has been funded by the Spanish CICYT projects CGL2004-06099-C03-03/CLI and CGL2007-66440-C04-03

DEPARTAMENTO DE SISTEMAS INFORMÁTICOS
ESCUELA SUPERIOR DE INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE CASTILLA-LA MANCHA
Campus Universitario s/n
Albacete - 02071 - Spain
Phone +34.967.599200, Fax +34.967.599224

An Hybrid MPI+Pthread implementation for computing the False Nearest Neighbors method on cluster of SMP architectures *

I. Marín[1], E. Arias[2], M. M. Artigao [1], J. J. Miralles [1]
[1] Applied Physics. Escuela Superior de Ingeniería Informática
[2] Computing Systems Department. Escuela Superior de Ingeniería Informática
Universidad de Castilla-La Mancha. 02071 - Albacete, SPAIN
{ismael.marin,enrique.arias,mariamar.artigao,juan.miralles}@uclm.es

Resumen

In different fields of science and engineering (medicine, economy, oceanography, biological systems, etc) the False Nearest Neighbors (FNN) method is particularly relevant. In some of these applications, it is important to give results within a reasonable time scale, thus the execution time of the FNN method has to be reduced. This paper describes two parallel implementations of the FNN method for hybrid (shared and distributed) memory architectures. As far as the authors know the hybrid implementation presented in this paper represents to be the first parallel implementation. Moreover, by considering a hybrid implementation it is possible to take advantage of different parallel architectures due to the fact that knowing the number of nodes and the number of cores by node, the software is autotunned in order to exploit the maximum degree of parallelism existing on the target machine. A "Single-Program, Multiple Data" (SPMD) paradigm is employed using a simple data decomposition approach where each processor runs the same program but acts on a different subset of the data. The

*This work has been partially supported by the Spanish CICYT projects CGL2004-06099-C03-03/CLI and CGL2007-66440-C04-03.

computationally intensive part of the method is mainly within the neighbors search and therefore this task is parallelized and executed using from 2 to 64 processors. The accuracy and performance of the two parallel approaches are then assessed and compared to the best sequential implementation of the FNN method which appears in the TISEAN project [1]. The results indicate that the two parallel approaches, when the method is run using 64 processors, is between 75 and 95 times faster than the sequential one. The efficiency is also discussed. In the context of time saving with 64 processors compared to the best sequential implementation, the time saving on the MareNostrum supercomputer are around 115 % to 145 %.

Keywords: Parallel Computing, Message Passing Interface, POSIX threads, Physics, Nonlinear Time Series Analysis, Method of False Nearest Neighbors

1. Introduction

Dynamical systems are studied from two different view points. One is from a previously known model which explains its behavior and the other is from a time series carried out by means of successive data acquisition per constant time periods $\{s_i\}_{i=1}^{i=N}$. This becomes the basis of nonlinear time series analysis. This methodology is based on the reconstruction of state space in a dynamical system from theorem of Takens [2]. The basic idea is the dynamical states space M of a dynamical system with dimension m is able to be characterized uniquely by m independent quantities. One of these systems of independent quantities are the own coordinates of the phases space (more precisely, the coordinates related to the basis that causes this phases space M).

$$y^{\rho}(t) = (y_1(t), y_2(t), \ldots, y_i(t), \ldots, y_m(t))^{\tau} \tag{1}$$

The most important phase space reconstruction technique is the method of delay. This technique takes m consecutive elements from the time series as coordinates in the

phase space in order to find a vectorial space that contains the same information than the original states space [3]. This implies transforming a set of scalar data of dimension 1 (the time series) into vectorial data of dimension m (the reconstructed space phase). This phase space is the natural basis to formulate nonlinear time series algorithms from the chaos theory, rather than the time or the frequency domain. Vectors in this new space, the embedding space, are formed from time delayed values of the scalar measurements:

$$\overrightarrow{s}_i = [s_{(i-(\mathbf{m}-1)\tau)}, ..., s_{i-\tau}, s_i]^T \quad i : 1, 2, 3, ..., N - (m-1)\tau \tag{2}$$

All local information will be gained from neighborhood relations of various kinds from time series elements. Thus a recurrent computational issue will be that of defining local neighborhoods in the phase space [1].

In Eq. (2), the number m of elements is called the embedding dimension, the time τ is generally referred to as the delay.

In practice, the natural questions are what time delay and what embedding dimension is the most appropriate for the reconstruction in the phase space. There are several methods to find out the minimum embedding dimension, m, i.e. global embedding dimension. A remarkable method is the method of False Nearest Neighbors (FNN) proposed by Kennel et al. [4]. This method identifies the number of "false nearest neighbors", points that appear to be nearest neighbors because the embedding space is too small, of every point on the attractor associated with the orbit $y(n), n = 1, 2, ..., N$. When the number of false nearest neighbors drops to zero, we have unfolded or embedded the attractor in $\mathbb{R}^{\mathbf{d}}$, a $d-$dimensional Euclidean space.

One of the important features of an attractor is that it is often a compact object in phase space. Hence, points of an orbit on the attractor acquire neighbors in this phase space. The utility of these neighbors is, for example, to allow the information on how

phase space neighborhoods evolve to be used to generate equations for the prediction of the time evolution of new points. They could also allow accurate computations of the Lyapunov exponents of the system.

The basis of FNN method is the supposition that the minimal embedding dimension for a given time series x_i is m_0. This means that in a m_0-dimensional delay space the reconstructed system 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. But let us consider now we embed in an $m-$dimensional space with $m < m_0$. With this projection the topological structure is no longer preserved. Points are projected into neighborhoods of other points to which wouldn't belong in higher dimensions. These points are called false neighbors. If we now apply the dynamics, these false neighbors are not typically mapped into the image of the neighborhood, but somewhere else, thus the average "diameter" becomes rather large.

The idea of the false nearest algorithm is the following. For each point x_i in the time series look for its nearest neighbor x_j in a $m-$dimensional space. Calculate the distance $\parallel \overrightarrow{x_i} - \overrightarrow{x_j} \parallel$. Iterate both points and compute:

$$R_i = \frac{\mid x_{i+1} - x_{j+1} \mid}{\parallel \overrightarrow{x_i} - \overrightarrow{x_j} \parallel} \tag{3}$$

If R_i exceeds a given heuristic threshold R_t, this point is marked as a false nearest neighbor. The criterion that the embedding dimension is sufficiently high is that the fraction of points for which $R_i > R_t$ is zero, or at least sufficiently small. Two examples of well-known dynamical systems (described in Sec. 5.2) are shown in Fig. 1: Lorenz system, Hénon system, and Hénon time series corrupted by 10 % of Gaussian white noise. A conclusion is that, as expected, $m = 2$ is sufficient for the Hénon model, since this system is determined by two coordinates, whereas the signature is less clear in the noisy case. Also, the Lorenz system is described by three differential equations, thus $m = 3$ is the minimal dimension. In other cases, the minimal sufficient embedding

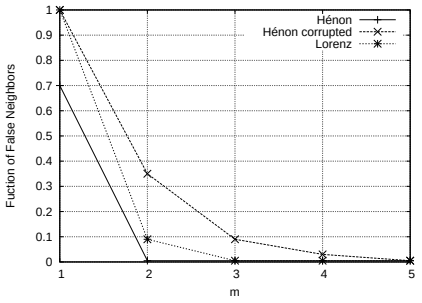

Figura 1: The fraction of false nearest neighbors as a function of the embedding dimension for noise free Lorenz and Hénon time series, as well as a Hénon time series corrupted by 10 % of noise.

dimension could become substantially greater.

The main contributions of this work consist of the development of two different parallel implementations for hybrid memory architectures of the FNN method which appears in the TISEAN package [1]. This work is specially important due to the fact that the FNN method represents the starting point for the estimation of minimal sufficient embedding dimension and Lyapunov exponent. Thus, a good implementation of FNN method represents the successful of whole the future implementations.

Some references of software [5, 6, 1] for computing the FNN method are found in the literature. For this work the parallelization of the FNN implementation developed by TISEAN has been studied.

This paper is organized as follows. First of all, Section 2 summarizes methods for the neighbors search with particular emphasis in the box-assisted method. The sequential implementation of the FNN method given by TISEAN project is described in Section 3. In Section 4, the parallel approaches are introduced. The cases of study and the experimental results are presented in Section 5. Finally, the conclusions and future work are outlined in Section 6.

2. Methodology

Finding neighbors in $k-$dimensional space is a task encountered in many data processing problems. In the context of time-series analysis e.g. it occurs if one is interested in local properties for a reconstructed phase space. Examples are predictions, noise reduction or Lyapunov exponent estimates based on local fits to the dynamics, or the calculation of dimensional estimates.

As long as only small sets are evaluated, neighbors can be found in a straightforward way by computing the $\frac{n^2}{2}$ distances between all pairs of points. However, numerical simulations and to an increasing degree experiments are able to provide much larger amounts of data. With increasing data sets efficient handling becomes an important question.

Neighbor searching and related problems of computational geometry have been extensively studied in computing science, with many publications covering both theoretical and practical issues [7, 8, 9, 10, 11].

Multidimensional tree structures [12, 13, 14, 15, 16] are widely used and have attractive theoretical properties. Finding all neighbors in a set of n vectors takes $O(log\ n)$ operations, thus the total operation count is $O(n\ log\ n)$. A fast alternative that is particularly efficient for relatively low dimensional structures embedded in multidimensional spaces is given by box-assisted neighbor search methods [17, 18, 19] which can push the operation count down to $O(n)$ under certain assumptions. Both approaches are reviewed in [20] with particular emphasis on time series applications. In the TISEAN project, fast neighbor search is done using a box-assisted approach, as described in [21].

No matter what space dimension we are working in, it is possible to define candidates for nearest neighbors in two dimensions using a grid of evenly spaced boxes.

With a grid of spacing ϵ, all neighbors of a vector x closer than ϵ must be located in the adjacent boxes. But not all points in the adjacent boxes are neighbors, they may be up to away in two dimensions and arbitrarily far in higher dimensions. Neighbors search is thus a two stage process. First the box-assisted data base has to be filled and then for each point a list of neighbors can be requested. There are a few instances where it would be advisable to abandon this neighbor search strategy.

2.1. Box-assisted algorithm

The box-assisted algorithm given here has been heuristically developed in the context of time series analysis [22, 23]. Next, a very simple version of a box-assisted algorithm for finding all points closer than a given distance ϵ is described.

Consider a set of n vectors x_i in k dimensions, for simplicity rescaled to fall into the unit cube. For each x_i, it can determine all neighbors closer than ϵ, or, strictly speaking, determine the set of indices:

$$u_i(c) = \{j :\| x_j - x_i \| < \epsilon\} \tag{4}$$

The idea of box-assisted methods is the following. Divide the phase space into a grid of boxes of side length ϵ. Then each point falls into one of these boxes. All its neighbors closer than ϵ have to lie in either the same box or one of the adjacent boxes. Thus in two dimensions only 9 boxes have to be searched for possible neighbors, i.e. for $\epsilon < \frac{1}{3}$ we have to compute less distances than in the simple approach.

The technical problem is how to put the points into the boxes without wasting memory. In [22], the main idea of the algorithm presented is to associate to each box a linked list, and write all these linked lists into one large array named `LIST`. In addition to this large array, which needs exactly n elements of n data points, an array (called

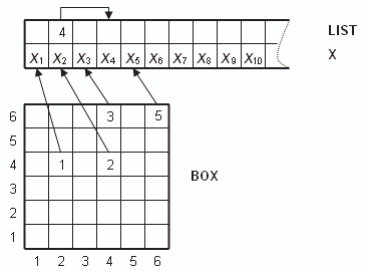

Figura 2: In a mesh of 6x6 boxes, the first data point $x_i = (x_1, x_2)$ has fallen into $box(2, 4)$, points x_2 and x_4 have fallen into $box(4, 4)$, x_3 into $box(4, 6)$, and x_5 into $box(6, 6)$. The figure shows the contents of the arrays LIST(k), X(k), and BOX(i,j) after reading in the first five points (all empty elements are zero).

`BOX`) of the size of the mesh is defined, which provides the information where the individual lists start in `LIST`. More precisely, each element of `BOX` contains a pointer to the head of the list for the box. It is empty if no such list exists yet. Schematically, this is illustrated in Fig. 2.

Initially, all elements of `BOX` and `LIST` are set to zero. Then the data points are read in, one after the other. Let us assume that the k-th data point falls into `BOX(n,m)`. If `BOX(n,m)` is still zero at this time, then we set `BOX(n,m) = k`, and read in the next data point. If however `BOX(n,m)` is equal to $l_1 \neq 0$, then we go to the l_1th element of `LIST`. If this zero, then we set `LIST(l1) = k`. Otherwise, if `LIST(l1)` is $l_2 \neq 0$, we go to `LIST(l2)`. We continue with the list until we reach the first empty place, set this equal to k, and read in the next data point.

In order to find all near neighbors of the kth data point, we go through the nine neighboring boxes of `BOX(n,m)` (including this box itself). Whenever the corresponding element of BOX is $l_1 \neq 0$, we first of all know that the l_1th data point is a candidate for a near neighbor. In addition, if `LIST(l1)` is $l_2 \neq 0$, then $x(l_2)$ is also a candidate, and so on. Thus, by walking through the linked lists of the neighboring boxes, we collect all candidates for near neighbors.

We have to take great care in speaking about "candidates" for near neighbors. One reason is that not all points in neighboring boxes have distance $< \epsilon$, if the box index $(n, n^{'})$ of a point (x_k, x_{k+1}) is computed by a simple coarse graining. Another reason is that, in order to reduce the size of the array `BOX`, we do not use exactly this coarse graining. Instead of this, we give to the mesh the topology of a torus, and "wrap" phase space several times around this torus. This "wrapping" is needed whenever the range of values in the time sequence is not known a priori. In other circumstances it reduces memory space substantially for small values of the cut-off distance ϵ, although it must have a serious impact on the performances.

3. Sequential implementation of the FNN method

The sequential implementation of the FNN method developed by TISEAN project will be described in this section. The implementation presented here is based on a box-assisted algorithm.

In Fig. 3, the TISEAN implementation of the FNN method is represented. This program looks for the nearest neighbors of all data points in m dimensions and iterates these neighbors one step into the future. If the ratio of the distance of the iteration and that of the nearest neighbor exceeds a given threshold, the point is marked as a wrong neighbor. The output is the fraction of false neighbors for the specified embedding dimensions [4]. In addition, TISEAN project has implemented a new second criterion: If the distance to the nearest neighbor becomes smaller than the standard deviation of the data divided by the threshold, the point is omitted [1].

More in detail, this program evaluates the specified embedding dimensions, and also takes as inputs the time series, the time delay τ and the threshold. One or more iterations are performed in every estimated dimension. A new and greater ϵ distance

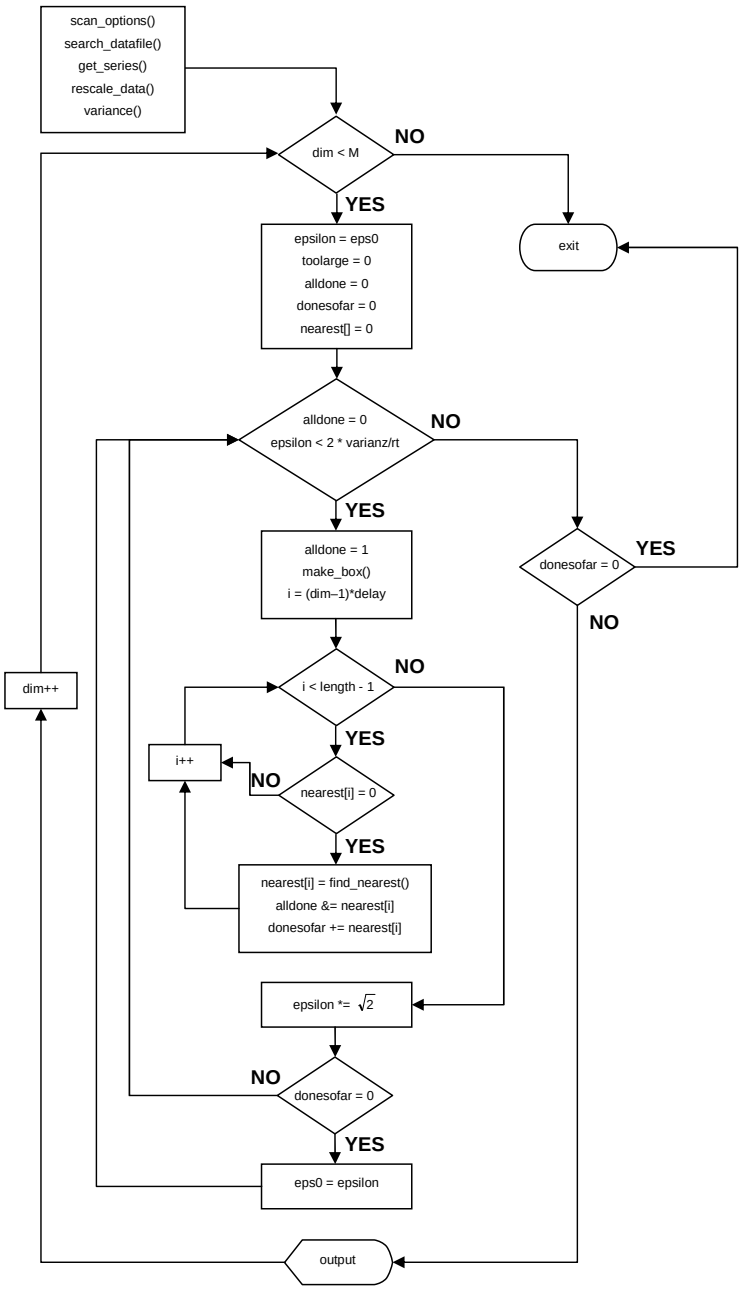

Figura 3: Flowchart of the FNN implemented in TISEAN package

is computed for every one of the iterations. An iteration is formed by two main tasks: the phase space reconstruction and the nearest neighbors search. In the first task, the `BOX` and `LIST` arrays are filled, i.e. a grid of boxes of side length ϵ is made. Time delay τ is needed for computing this task. This task is implemented by the `make_box()` function. In the second one, the box and adjacent boxes of each point are searched for finding their nearest neighbor. These neighbors must be closer than the given distance ϵ. Moreover, it is checked whether these points are false neighbors. This task is supported by the `find_nearest()` function.

Two conditions are necessary in order to break this loop. The first one concerns the value of the distance ϵ: If ϵ is greater than a given threshold, the dimension is completed. The second one concerns the nearest neighbors found: If all points have found their nearest neighbor, the dimension is completed. In fact, every time a point has found their nearest neighbor, it doesn't make sense to compute it in next iterations again, i.e. using a greater distance ϵ, while the dimension is not over. In this way, the `NEAREST` array determines whether each point has found their nearest neighbor.

If after an iteration is finished, the total number of nearest neighbors found is 0 up to this ϵ, the next embedding dimensions iterations will begin using a distance ϵ greater than this one.

Finally, the embedding dimension value, the fraction of false nearest neighbors, the average size of the neighborhood, and the average of the squared size of the neighborhood are shown for each specified embedding dimension.

4. An hybrid memory implementations of the FNN method

In this Section, the two hybrid memory implementations of the FNN method based on the sequential implementation previously introduced are described. For communication and synchronization purposes, a combination of the Message Passing Interface (MPI) and POSIX threads parallel programming standards [24, 25, 26] have been used.

As far as the authors know these hybrids implementations presented in this paper represent the first parallel implementation in this scientific issue. Moreover, by considering an hybrid implementation it is possible to take advantage of different parallel architectures due to the fact that, knowing the number of nodes and the number of cores by node, the software is autotunned in order to exploit the maximum degree of parallelism existing on the target machine.

MPI basically provides interfaces to send/receive data and synchronize operations between the multiple tasks of a parallel application, while Pthreads provides a library specifically to exploit shared memory parallelism on each node.

An MPI process per node must be executed and as many Pthreads as available CPUs per node must be dispatched. Node threads resolves parts of problem using a shared memory system, while processes manage threads and resolves the problem using a distributed memory system.

The main effort in the parallelization of the problem concerns the nearest neighbors search. Although, the parallelization of the phase space reconstruction is treated. Nevertheless, time consumption in this task is only really important for problems with many iterations, and for very large mesh sizes.

The two implementations describe the same flow structure than the sequential

implementation. These proposals are formed by these four basic tasks:

- **Domain Decomposition**: Points are distributed to the threads dispatched by the processes. Two ways of distribution have been developed: TS (Time Series) and M (Mesh). This task is supported by the `localdim()` function.
- **Phase Space Reconstruction**: `BOX` and `LIST` arrays are filled, i.e. a grid of boxes of side length ϵ is made. Only a way of phase space reconstruction has been developed: P (Parallel). This task is implemented by the `make_box()` function.
- **Nearest Neighbors Search**: In this task, each thread/process solves their subproblem given the domain decomposition way. This task is implemented by the `evaluate()` and `find_nearest()` functions.
- **Communication of results**: This task is divided in two subtasks: per iteration and per dimension.

With reference to what has been previously explained, in the two parallel implementations the following nomenclature can be established:

- **HM-P-TS** meaning a **H**ybrid **M**emory implementation considering that the phase space reconstruction has been carried in **P**arallel and the **T**ime **S**eries has been distributed.
- **HM-P-M** meaning a **H**ybrid **M**emory implementation considering that the phase space reconstruction has been carried in **P**arallel and instead of distributing the Time Series, the **M**esh has been distributed.

4.1. Implementation issues

Following are given some considerations for parallel implementations.

4.1.1. Decomposition Domain

In a TS data distribution, each thread obtains a set of data points from the time series, independently of their location on the grid of boxes. Time series array is divided into p (number of available threads) uniform parts, excluding first $(dim - 1) * \tau$ data points for each considered dimension dim.

In a M data distribution, each thread obtains the group of boxes (points on them) located over a set of assigned mesh rows. Since a process only works with a set of delimited rows (by their first and last thread), this only needs to allocate a part of the mesh on their local memory. Moreover, they must have their adjacent rows in order to evaluate appropriately the boxes located in border rows. Consequently, row indices must be mapped (global-to-local). This characteristic enable the mesh size be flexible (i.e. the size can be greater when more available resources).

Both data distributions do not take into account the data points which must not be evaluated in successive iterations, i.e. points that have found their nearest neighbors in previous iterations.

4.1.2. Phase Space Reconstruction

In a P approach, these arrays are filled in parallel among all threads dispatched by each process (HM-P-TS) or among all threads dispatched by the whole of the processes (HM-P-M). Therefore, each thread fills a part of the grid of boxes. They fill the group of boxes in particular located over a set of assigned mesh rows.

4.1.3. Nearest Neighbors Search

The `evaluate()` function iterates over assigned data points and calls the `find_nearest()` function for every one of them. Consequently, each thread must hold their partial results.

In case of M data distribution, each thread can use a different group of points in every new iteration, while in TS data distribution, each thread uses the same group of points for all iterations of a same dimension. Consequently, each process must always have the `NEAREST` array updated for M approach, and only needs to have the corresponding part of `NEAREST` array updated for TS approach. Nevertheless, threads work independently over a part of the `NEAREST` array.

4.1.4. Communication of results

Initially mutexes are set in order to avoid incoherencies of global variables when these are updated. Mutual exclusion locks (mutexes) are a common method of serializing thread execution. Mutual exclusion locks synchronize threads, usually by ensuring that only one thread at a time executes a critical section of the code.

A barrier is implemented in order to synchronize the threads. A barrier makes all threads wait until all the threads reach the barrier. This is supported by one condition variable and one thread counter.

After an iteration is finished, the local variables (thread and process level variables) are collected and updated, which are necessary in order to decide whether another iteration must be performed. This task is implemented by the `collect_all()` function. This function uses a barrier and a mutex combined with the `MPI_Allreduce()` function for this purpose. In this way, each process updates the local variables required. Please

note that, in HM-P-M proposal, the local `NEAREST` arrays must be communicated and updated.

After a dimension is finished, the local results (thread and process level results) are collected per thread and process in order to show the corresponding global results to the embedding dimension. This task is implemented by the `collect()` function. This function uses another barrier and mutex combined with the `MPI_Reduce()` function for their purpose. In this way, only one process (#0) updates the global results.

Pseudocode 1 describes the parallel algorithm shown in this section.

5. Experimental Results

The experimental results presented here have been carried out in Barcelona Supercomputing Center-Centro Nacional de Supercomputación (Spain) under the "Performance analysis of parallel algorithms for nonlinear time series analysis" project.

5.1. Platform: Description of MareNostrum and related issues

BSC-CNS hosts MareNostrum, the most powerful supercomputer in Europe and the number 5 in the world, according to the Top500 list (November 2006). MareNostrum is a supercomputer based on processors PowerPC, the architecture BladeCenter, a Linux system and a Myrinet interconnection. These four technologies configure the base of an architecture and design that will have a big impact in the future of supercomputing.

Please see below a summary of the system:

- Peak Performance of 94,21 Teraflops

Algorithm 1 Computes minimal emdedding dimension by means of parallel FNN method

Program $FNN : (mindim, maxdim, \tau, timeseries)$

Inputs: $mindim$ and $maxdim$ are minimal and maximal dimension of the delay vectors respectively; τ is delay of the vectors; and $timeseries(n)$ is a vector of n data points

Outputs: the embedding dimension; the fraction of false nearest neighbors; the average size of the neighborhood; the average of the squared size of the neighborhood

1: Distributing n data to p process /*TS and M types*/
2: **for** $mindim$ to $maxdim$ **do**
3: **while** n data haven't a nearest neighbor AND $\epsilon >$ threshold **do**
4: Making a grid of boxes of side length ϵ given a τ
5: **for** subset of data points assigned (depending of TS or M type) **do**
6: Finding the nearest neighbors closer than ϵ and determining if they are false neighbors
7: **end for**
8: Communication of results in order to decide whether another iteration must be performed
9: Update ϵ
10: **end while**
11: Communication of results for this dimension
12: Showing results per dimension
13: **end for**

- 10240 IBM Power PC 970MP processors at 2.3 GHz (2560 JS21 blades)
- 20 TB of main memory
- 280 + 90 TB of disk storage
- Interconnection networks: Myrinet and Gigabit Ethernet
- Linux: SuSe Distribution

The nodes (Server Blade JS21) have two dual-core processors PowerPC 970MP at 2.3 GHz and 8GB of shared memory (really, not all memory is available).

5.2. Case studies

The proposed algorithm is applied to three benchmark problems of chaotic time series, known as the Lorenz, Hénon and Rössler time series, respectively. The benchmark problems are mainly concerned with chaotic dynamics which is difficult to predict.

These benchmark problems allow us analyze the behavior of the parallel implementations developed in this work.

- **Lorenz**: The Lorenz system shows how the state of a dynamical system (the three variables of a three-dimensional system) evolves over time in a complex, non-repeating pattern, often described as beautiful. The equations that describe the system were introduced by E. Lorenz in 1963, which derived it from the simplified equations of convection rolls arising in the equations of the atmosphere. These equations are the following

$$\frac{dx}{dt} = \alpha(y - x),$$

$$\begin{aligned} \frac{dy}{dt} &= x(\beta - z) - y, \\ \frac{dz}{dt} &= xy - \gamma z, \end{aligned} \tag{5}$$

where α, β and γ are the parameters of Lorenz system. α is called the Prandtl number and β is called the Rayleigh number.

- **Hénon**: The Hénon map is a discrete-time dynamical system. It is one of the most widely studied examples of dynamical systems that exhibit chaotic behavior. The Hénon map takes a point (x,y) in the plane and maps it to a new point. It is shown by

$$\begin{aligned} x_{n+1} &= y_n + 1 - \alpha x_n^2, \\ y_{n+1} &= \beta x_n, \end{aligned} \tag{6}$$

where α and β are the parameters of Hénon map.

- **Rössler**: This case study consists of a system of three non-linear ordinary differential equations. These differential equations define a continuous-time dynamical system that exhibits chaotic dynamics.

$$\begin{aligned} \frac{dx}{dt} &= -y - z, \\ \frac{dy}{dt} &= x + \alpha y, \\ \frac{dz}{dt} &= \beta + z(x - \gamma), \end{aligned} \tag{7}$$

where α, β and γ are the parameters of Rössler system.

5.3. Experimental results

The performance obtained in the parallel implementations are evaluated in terms of:

- Execution time: Time spent in order to solve the problem.
- Speed-up: The ratio of the time taken to solve a problem on a processor to the time required to solve the same problem on a parallel computer with p identical processors.
- Efficiency: A measure of the fraction of time for which a processor is usefully employed; it is defined as the ratio of the speed-up to the number of processors.

The used time series are formed by 10 millions of data points. The results presented here have been performed for the first embedding dimension. Note that this embedding dimension is not the minimal sufficient dimension in these cases. Nevertheless, this is usually more time-consuming than the upper dimensions in these three dynamical systems. Moreover, 4 threads per process/node are dispatched.

Fig. 4 shows the results obtained for the Lorenz case study in terms of execution time, speed-up and efficiency.

In Fig. 5 are shown the results obtained for the Hénon case study in terms of execution time, speed-up and efficiency.

Finally, Fig. 6 shows the results for Rössler case study in terms of execution time, speed-up and efficiency.

Both parallel implementations provides a similar results for theses three systems. The results indicate that the two parallel approaches, when the method is run using 64

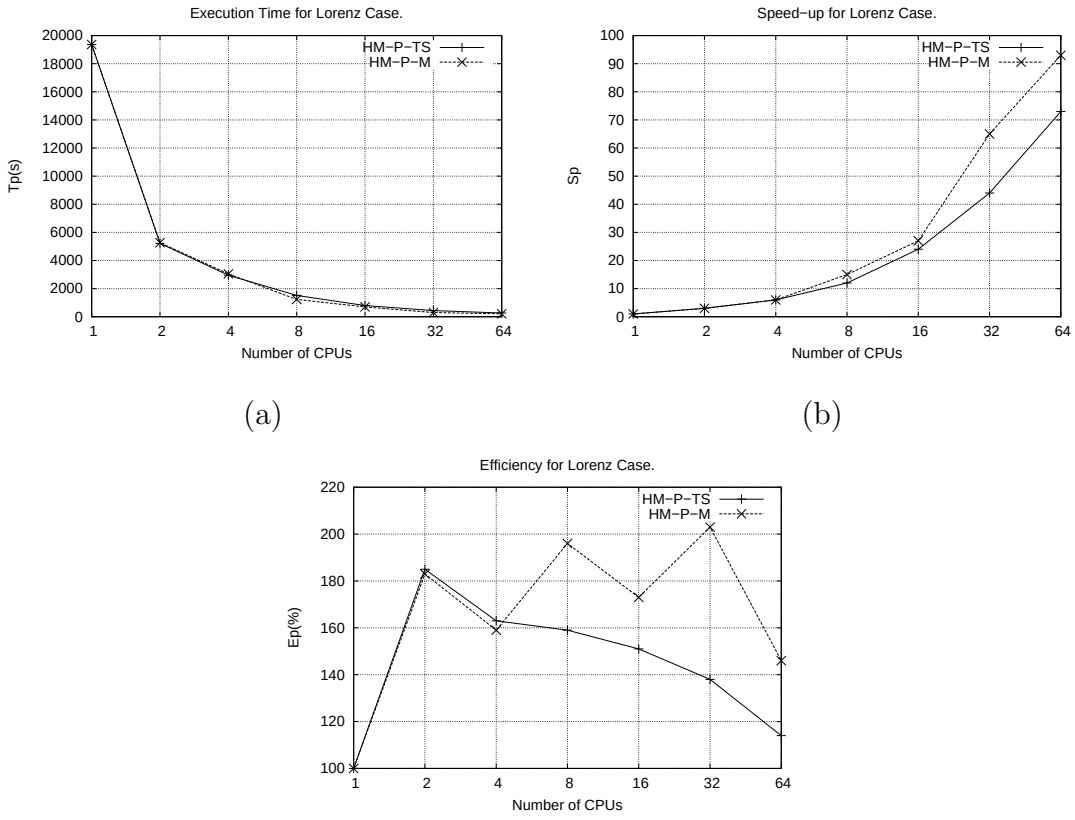

Figura 4: Lorenz case study. (a) Execution time. (b) Speed-up. (c) Efficiency

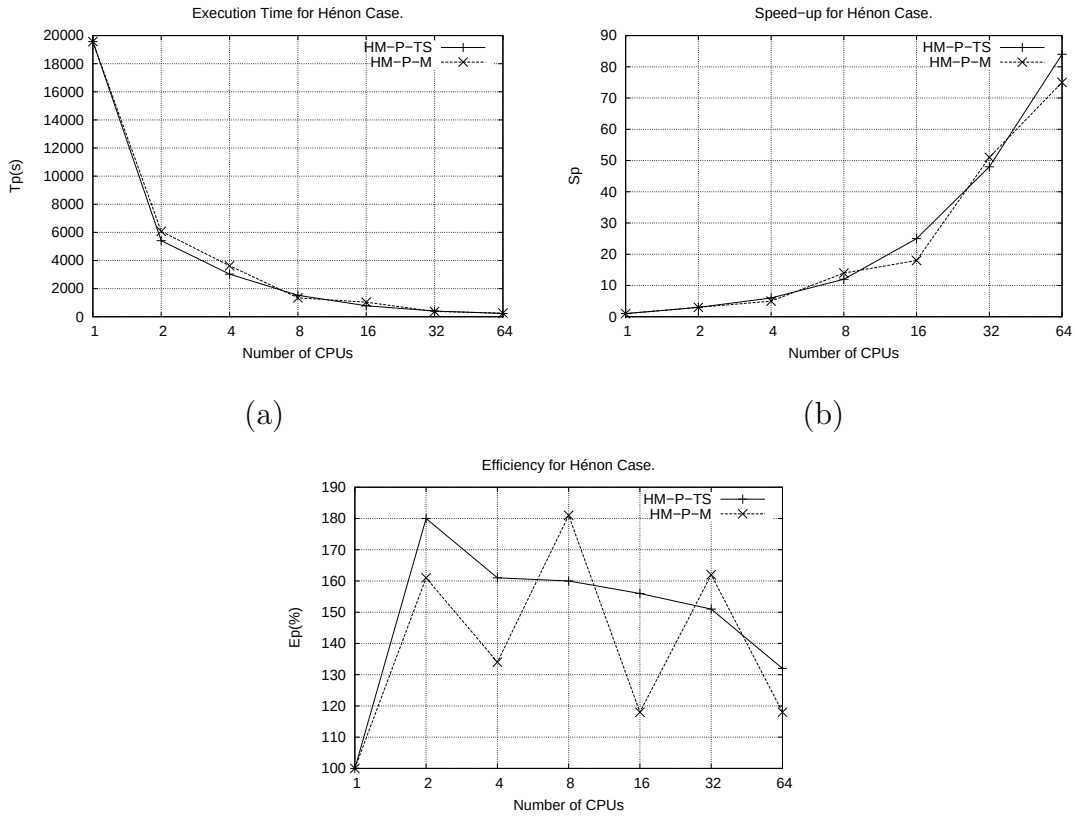

Figura 5: Hénon case study. (a) Execution time. (b) Speed-up. (c) Efficiency

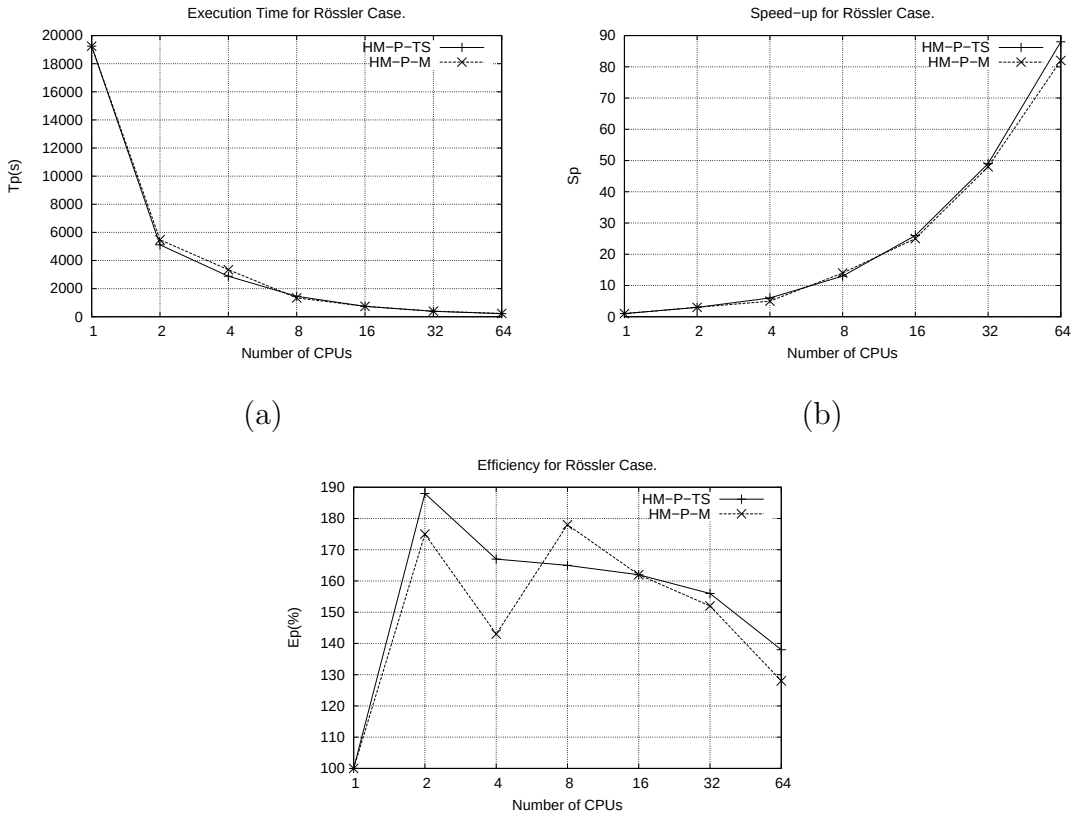

Figura 6: Rössler case study. (a) Execution time. (b) Speed-up. (c) Efficiency

processors, is between 75 and 95 times faster than the sequential one. In the context of time saving with 64 processors compared to the best sequential implementation implemented in TISEAN, the time saving on the MareNostrum supercomputer are around 115 % to 145 %.

As figures show, the performance is better when more CPUs are used. In fact, due to a efficient use of system architecture, all proposals provide superspeed-up in some cases. However, this is limited in HM-P-TS.

In addition, the mesh sizes used are the best performance give for those cases.

In the next section, the main conclusions of this work are outlined.

6. Conclusions and future work

This paper presents two parallel implementations of the FNN method for hybrid memory architectures (HM-P-TS and HM-P-M). To the authors knowledge, these parallel implementations are the first to be carried out in this area.

The experimental results previously presented show that the main goal of this work has been accomplished in full. That is, the execution time for applying the FNN method in order to search the minimal sufficient embedding dimension has been dramatically reduced by the use of parallelism.

According to the results presented in the previous section, both parallel implementations provides a similar results for these three systems.

In general, the type M approach provides a worse data distribution than type TS. This is influenced by the mesh size used.

The type P approach allows the performance to be improved, mainly when more

and more iterations are performed, greater mesh sizes are used, and more threads are launched by MPI-process (i.e. more CPUs per node are available). Please note that the same data distribution must be assumed.

In some cases, due to a efficient use of system architecture, all proposals provide superspeed-up thanks to an adaptable mesh size (see Sec. **??**)., although this is limited in HM-P-TS.

Regarding related works, parallel implementations for distributed and shared platforms have been developed. Also, a parallelization focusing on embedding dimensions have been studied.

Certainly, more experiments need to be conducted in order to extensively asses the performance of our parallel implementations, since only three of the most relevant theoretical cases have been considered. These implementations will be tested for different case studies such as ECG, Internet traffic, weather data or ozone measurements.

Acknowledgments

The simulations have been done in the supercomputer MareNostrum at Barcelona Supercomputing Center - Centro Nacional de Supercomputación (The Spanish National Supercomputing Center).

Referencias

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

[2] F. Takens, Detecting strange attractors in turbulence. In *Lecture Notes in Math.* 898. Springer, New York (1981).

[3] N.H. Packard, J.P. Crutchfield, J.D. Farmer, and R.S. Shaw, Geometry from a time series. *Phys. Review Lett.* **45** 712-716 (1980).

[4] M.B. Kennel, R. Brown and H.D.I. Abarbanel, Determining embedding dimension for phase-space reconstruction using a geometrical construction. *Phys. Rev. A* **45** 3403 (1992).

[5] http://hpux.cs.utah.edu/hppd/hpux/Physics/embedding-26.May.93

[6] M.B. Kennel, KDTREE Fortran 95 and C++ software to efficiently search for near neighbors in a multi-dimensional Euclidean space. *Arxiv preprint physics/0408067* (2004).

[7] R. Sedgewick, *Algorithms in C.* Addison-Wesley, Reading (1990).

[8] F.P. Preparata and M.I. Shamos, *Computational Geometry: An Introduccion.* Springer, New York (1985).

[9] G.H. Gonnet and R. Baeza-Yates, *Handbook of algorithms and data structures in Pascal and C.* Addison-Wesley, Wokingham (1991).

[10] K. Mehlhorn, *Data Structures and Algorithms 3: Multimdensional Searching and Computational Geometry.* Springer (1984).

[11] S. Arya, D.M. Mount, N.S. Netanyahu, R. Silverman and A.Y. Wu, An optimal algorithm for approximate nearest neighbor searching. *J. of the ACM* **45** 891-923 (1998).

[12] S.M. Omohumdro, Efficient algorithms with neural network behavior. *Complex Syst.* **1** 273 (1987).

[13] J.L. Bentley, Multidimensional divide-and-conquer. *Comm. of the ACM* **23** 214 (1980).

[14] J.L. Bentley, K-d trees for semidynamic point sets. *Sixth Annual ACM Symposium on Computational Geometry* 91. San Francisco (1990).

[15] I. Al-furaih, S. Aluru, S. Goil and S. Ranka, Parallel construction of multidimensional binary search trees. *IEEE Transactions on Parallel and Distributed Syst.* **11** 136 (2000).

[16] A.W. Moore, *An introductory tutorial on kd-trees*. Extract from PhD. Thesis, Tech. Report No. 209. Computer Laboratory, University of Cambridge (1991).

[17] M.T. Noga and D.C.S. Allison, Sorting in linear expected time. *Bit* **25** 451-465 (1985).

[18] L. Devroye, Lecture notes on bucket algorithms. In *Progress in Comp. Sci.* 6. Birkhäuser, Boston (1986)

[19] T. Asano, M. Edahiro, H. Imai, M. Iri and K. Murota, Practical use of bucketing techniques in computational geometry. In *Computational Geometry*, (Edited by G.T. Toussaint), pp. 153-195. Elsevier (1985).

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

[21] H. Kantz and T. Schreiber, *Nonlinear Time Series Analysis*. Cambridge University Press, Cambridge (1997).

[22] P. Grassberger, An optimized box-assisted algorithm for fractal dimensions. *Phys. Lett. A* **148** 63 (1990).

[23] P. Grassberger, T. Schreiber and C. Schaffrath, Nonlinear time sequence analysis. *Int. J. Bifurc. and Chaos* **1** 521 (1991).

[24] W. Gropp E. Lusk and A. Skjellum, *Using MPI: Portable Parallel Programming with the Message-Passing Interface*. MIT Press (1994).

[25] F. Mueller, *Pthreads Library Interface*. Institut für Informatik Tech Report, Humboldt-Universität zu Berlin (1999).

[26] A. Grama, A. Gupta, G. Karypis and V. Kumar, *Introduction to Parallel Computing*. Pearson Education (2003).