UN MÉTODO NUMÉRICO PARA LA IDENTIFICACIÓN DE SISTEMAS CAÓTICOS

TESIS DE LICENCIATURA

Alumnos: María Gabriela Bonelli L.U. 807/88
Hernán Pablo Codari L.U. 1561/87
María Laura Curone L.U. 1840/87

Directores: Ing. Marisa Bauzá
Dr. Pablo Coll

Noviembre de 2002

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Pabellón 1 - Planta Baja - Ciudad Universitaria
Buenos Aires – Argentina

Índice

Agradecimientos

Gabriela agradece:

- A mis padres y hermanos, que me apoyaron durante toda la carrera y me impulsaron siempre a finalizarla.
- A Sandra y a Bianca, por cedernos su casa y sobre todo el tiempo de Hernán para realizar esta tesis.
- Principalmente a Laura y Hernán, por haber compartido tantos buenos momentos y otros más difíciles durante toda la carrera. Realmente es un placer haber llegado hasta aquí con ellos. No podría haber sido de otra manera.

Hernán agradece:

- A mi esposa Sandra, que siempre me apoyó para poder completar esta tesis.
- A mi hijita Bianca, que desinteresadamente nos prestó su habitación que es donde está la computadora.
- A mis padres, que impulsaron el deseo de finalizar la carrera.
- A Laura y Gabriela, ya que formamos un excelente grupo de trabajo.

Laura agradece:

- A mi familia, que siempre me apoyó.
- A Sandra y a Bianca, por brindarnos cálidamente su hogar para trabajar en esta tesis.
- A Gabriela y Hernán, que supieron comprender en todo momento mi situación de tener que trabajar desde lejos, y con los que fue un verdadero placer compartir esta etapa.
- A Mauricio, que me motivó para completar esta etapa y me apoyó para que las obligaciones laborales afectaran lo menos posible.

Los autores agradecen muy especialmente a:

- Marisa que nos aconsejó, apoyó y guió, interesándose siempre por nuestro progreso.
- Alejandra que nos presentó a nuestra directora de tesis.

o o o

Resumen

Los sistemas caóticos están ganando interés día a día debido a que pueden estudiarse con una mayor profundidad dado el incremento del poder computacional. Estos sistemas se encuentran en muchas ramas de la ciencia, por lo que es importante tener la posibilidad de caracterizarlos a través de parámetros como la dimensión de correlación o la entropía de Kolmogorov, para nombrar sólo algunos.

El presente trabajo presenta un método para la identificación y cuantificación de sistemas caóticos usando series de tiempo del sistema a estudiar. El parámetro elegido es la dimensión de correlación y a partir de allí se calcula una cota inferior a la entropía de Kolmogorov, denominada número K_2.

Se prueba el método propuesto mediante la utilización de sistemas ya conocidos que presentan un comportamiento caótico, como el de Henon y el de Lorentz, y sistemas no caóticos, como el periódico y el aleatorio, cuyos parámetros han sido calculados por otros medios.

Los valores obtenidos en este trabajo concuerdan con los que se pueden encontrar en la bibliografía utilizada.

Se paralelizó parte del algoritmo utilizando PVM a fin de reducir los tiempos de ejecución total.

Palabras clave: Caos – Entropía de Kolmogorov – Dimensión de Correlación – Métodos Numéricos – Paralelismo – Máquina Paralela

o o o

Capítulo 1 – Introducción

Cuando un investigador desea identificar un sistema complejo, puede hacerlo usando datos experimentales (muestras) o a través de un modelo matemático que represente el comportamiento del sistema. Esta identificación permite clasificar un sistema, por ejemplo, en determinístico o no determinístico, en periódico o no periódico. Los sistemas determinísticos, a su vez, pueden ser caóticos o no caóticos.

Un sistema caótico es un sistema determinístico que exhibe un comportamiento aleatorio, también llamado comportamiento extraño, netamente dependiente de sus condiciones iniciales. En otras palabras, es un sistema que aún contando con toda la información anterior acerca de su comportamiento, no podemos saber cómo será éste en el futuro.

Es posible detectar la existencia de estos sistemas en muchas actividades científicas, por lo que es importante tener la posibilidad de caracterizarlos y cuantificar de alguna manera cuán caótico es el sistema. Existen varios métodos que calculan parámetros que identifican a estos sistemas a partir de un conjunto de muestras. Dichos parámetros pueden ser la entropía de Kolmogorov o la dimensión de correlación, por nombrar algunos [1][2].

Los datos del sistema a analizar provienen de señales tomadas a intervalos regulares formando una serie temporal. Se asume que la serie de tiempo ha sido registrada durante un tiempo suficientemente largo, con precisión finita y, si la señal proviene de un experimento, que el nivel de ruido es bajo. La suposición de que el lapso durante el cual se muestreó el sistema es extenso asegura que dicho sistema superó su estado transitorio y se encuentra dentro de su comportamiento natural [3]. La consideración de bajo nivel de ruido es para contar con una señal que proviene realmente del sistema.

El método que se presenta a continuación calcula la dimensión de correlación usando una serie temporal perteneciente al sistema a ser estudiado. Con este resultado se calcula el número K_2. Esta medida, desarrollada por Grassberger y Procaccia, es una estimación del límite inferior asintótico de la entropía de Kolmogorov [4].

Los sistemas que se usarán para ejemplificar el método propuesto son: el sistema Henon, el sistema Lorentz (ambos sistemas ya reconocidos como caóticos), un sistema no determinístico y un sistema periódico. Los resultados aquí obtenidos concuerdan con otros ya publicados.

Una ventaja sustancial de nuestro trabajo sobre los métodos anteriores consiste en reducir el tiempo total de cálculo. Como se indicó anteriormente, para asegurar que el sistema se encuentre dentro de su comportamiento natural, la serie debe ser extensa en el tiempo. Esto implica considerar una serie de datos integrada por un gran número de valores.

El análisis de todas las muestras del sistema insume gran cantidad de tiempo de CPU. El uso del paralelismo reduce en gran medida los tiempos de ejecución al posibilitar el uso de más de un procesador para ejecutar una tarea. Dividiendo la tarea principal en subtareas y asignando cada subtarea a un procesador, se obtiene la reducción de tiempo deseada. El hecho de utilizar la PVM (Máquina Paralela Virtual) facilitó en gran medida la paralelización de algunas rutinas del método propuesto.

o o o

Capítulo 2 – Caos

2.1 – Conceptos básicos de caos

Como ya señalamos en el capítulo anterior, los sistemas complejos pueden ser clasificados en sistemas determinísticos o no determinísticos, periódicos o no periódicos, etc. Los sistemas determinísticos, a su vez, pueden ser caóticos o no, siendo necesario cuantificar, en algunos casos, cuán caóticos son.

Anteriormente se pensaba que todos los sistemas determinísticos eran completamente predecibles: dadas las condiciones iniciales y las ecuaciones que describían el sistema, el comportamiento del sistema podía ser predicho para todo tiempo. De esta manera cualquier imprecisión en los datos iniciales derivaba solamente en un pequeño error de predicción.

El estudio de los sistemas caóticos ha cambiado radicalmente este punto de vista ya que se pueden definir como un sistema determinístico que exhibe un comportamiento aleatorio, también llamado comportamiento extraño, netamente dependiente de sus condiciones iniciales. En otras palabras, es un sistema que aún tomando en cuenta toda la información anterior, no podemos saber cómo se comportará.

Parker y Chua, en su libro "Practical Numerical Algorithms for Chaotic Systems" [18] describen conceptos básicos de caos, referenciando entre otros a Henri Poincaré y Edward Lorentz.

Henri Poincaré (ver [18]) examinó la capacidad de predicción y observó que, por una parte, los sistemas son determinísticos, pero por la otra, se viola el fuerte principio de causalidad. Observó que causas similares no conducen a efectos similares y concluyó que no hay una fórmula que relacione el estado de un sistema en un tiempo determinado con el

estado del mismo en un tiempo futuro. Demostró que existen diferentes tipos de trayectorias, entendiendo por trayectoria la traza de la evolución del sistema en un espacio definido por sus variables. También observó que los tipos de trayectorias pueden ser estables e inestables y que una pequeña perturbación en el sistema puede resultar en un cambio en la naturaleza de las mismas.

Edward Lorentz (ver [18]) investigó la aplicación de sistemas computacionales a pronósticos meteorológicos y reconoció que usando diferentes condiciones iniciales, el programa llegaba a predecir condiciones climáticas totalmente diferentes. Esto era una clara evidencia de la falla del principio de causalidad.

La amplia disponibilidad y el continuo incremento del poder computacional ha facilitado y consecuentemente propiciado en gran medida los avances de la investigación en la mecánica caótica.

Dado que los mapas iterativos (el valor de las variables en un instante discreto dado dependen de su valor en el intervalo anterior) son la manera más simple de visualizar sistemas caóticos, los utilizaremos más adelante para demostrar los algoritmos propuestos en el desarrollo de esta tesis; más precisamente utilizaremos el sistema Henon y la ecuación de Lorentz.

2.2 - El sistema Henon

El sistema Henon es un sistema caótico definido por las siguientes iteraciones [7]:

$$x_{n+1} = 1 - ax_n^2 + y_n$$
$$y_{n+1} = bx_n$$

El siguiente gráfico (ver Fig. 2.1) muestra las primeras 5000 iteraciones con $a = 1.4$ y $b = 0.3$.

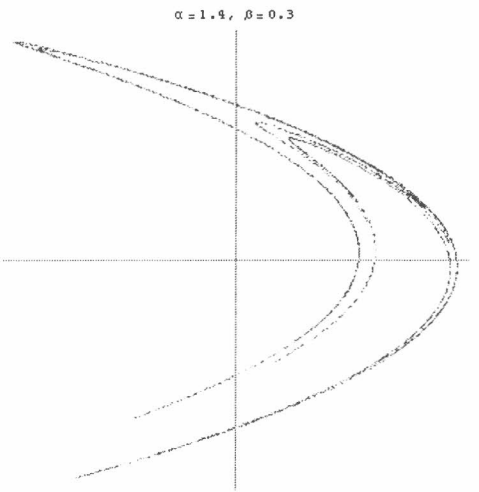


Fig. 2.1 – El Sistema Henon

2.3 - El sistema de Lorentz

Las ecuaciones de Lorentz son 3 ecuaciones diferenciales: una ecuación de primer orden para cada uno de los componentes x, y y z. Se definen de la siguiente manera [19]:

$$dx/dt = -ax + ay$$

$$dy/dt = bx - y - zx$$

$$dz/dt = -cz + xy$$

En los siguientes gráficos observamos ejemplos de las trayectorias definidas por las ecuaciones de Lorentz (ver Fig. 2.2 y Fig. 2.3).

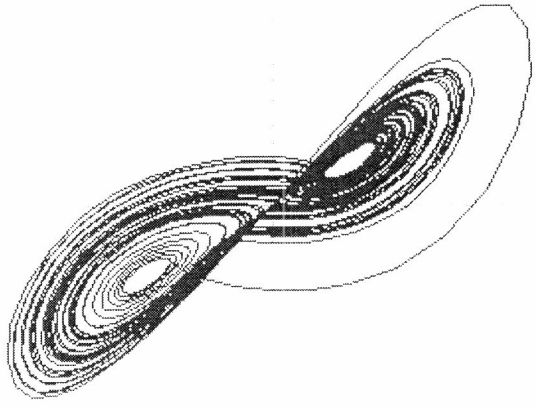


Fig. 2.2 – El Sistema de Lorentz

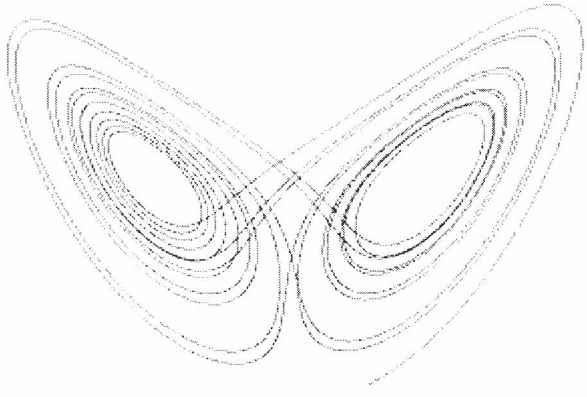


Fig. 2.3 – El sistema de Lorentz (Otra vista)

2.4 - Ejemplo práctico de un sistema caótico

Volviendo a la definición de caos, y tal como lo mencionamos anteriormente, desde un punto de vista práctico un sistema caótico puede definirse como un sistema altamente dependiente de las condiciones iniciales: dadas dos condiciones iniciales diferentes arbitrariamente próximas una de la otra, las trayectorias que emanan de estos dos puntos divergen hasta que no presentan ninguna correlación.

Esta dependencia de las condiciones iniciales tiene una implicancia importante. En la práctica, el estado inicial de un sistema no puede ser especificado exactamente, dada la precisión finita, sino sólo dentro de una tolerancia $\varepsilon > 0$. Si dos condiciones iniciales están cerca una de otra dentro de esa tolerancia ε, no pueden ser distinguidas. Sin embargo, después de una cantidad finita de tiempo las trayectorias $\phi_{1\,t}(x)$ y $\phi_{2\,t}(x)$ divergirán y se volverán "no relacionadas" [18]. Por lo tanto, no importa cuán precisamente se conozca la condición inicial, el comportamiento a largo plazo de los sistemas caóticos no puede ser predicho. Esto es a lo que nos referíamos cuando los sistemas caóticos fueron descriptos como "sistemas determinísticos que exhiben un comportamiento aleatorio".

El comportamiento aleatorio en un sistema determinístico puede resultar sorprendente al principio, pero hay muchos ejemplos simples para demostrarlo. Si consideramos el sistema discreto de 1 dimensión:

$$x_{k+1} = f(x_k) := 10\, x_k \bmod 1, \quad x_0 \in [0,1]$$

Si x_0 es escrito como un número decimal:

$$x_0 = \Sigma\, d_{k+i}\, 10^{-i} = 0.d_{k+1}\, d_{k+2}\, d_{k+3} \ldots$$

Luego de repetidas aplicaciones de la función, inicialmente se pierden los dígitos más significativos, luego los dígitos menos significativos se vuelven más significativos hasta que ellos también se descartan uno a uno.

Para mostrar que este sistema es no-predecible, asumamos que la condición inicial es $x_0 = 0.70710678$, pero que un observador puede tomar sólo 3 dígitos significativos. El observador puede predecir que:

$0.7065 < x_1 < 0.7075$

$0.065 < x_2 < 0.075$

$0.65 < x_3 < 0.75$

Pero no se puede decir nada de x_4 y los sucesivos x_i. Para este sistema, la condición inicial observada pierde poder predictivo luego de repetidas aplicaciones de la función. También notemos que si el observador mide el valor de la función después de cada iteración, la condición inicial puede ser reconstruida seleccionando el primer dígito después de cada medida. En otras palabras, a través de nuevas mediciones, el observador gana información sobre el estado a medida que el sistema evoluciona.

2.5 – Definiciones

Se enumeran a continuación algunas definiciones utilizadas en este trabajo [20]:

2.5.1 – Notación: $f^n(x)$

Se denota la composición de dos funciones como f o $g(x) = f(g(x))$. La composición n-ésima de la función f consigo misma se denota como $f^n(x) = f$ o ... o $f(x)$ (n veces).

2.5.2 – Punto periódico – Período primo

El punto x es un punto fijo para f si $f(x) = x$. El punto x es un punto periódico de período n si $f^n(x) = x$. El menor positivo n para el cual $f^n(x) = x$ es llamado el período primo de x.

2.5.3 – Punto límite

Sea $S \subset \mathrm{R}$. Un punto $x \in \mathrm{R}$ es un punto límite de S si existe una secuencia de puntos $x_n \in S$ que convergen a x. S es un conjunto cerrado si contiene todos sus puntos límite.

2.5.4 – Notación: Clausura de un conjunto

Para cualquier conjunto U, notamos la clausura de U como $\ddot{U}$. $\ddot{U}$ consiste de todos los puntos en U juntamente con todos los puntos límites de U.

2.5.5 – Subconjunto denso

Un subconjunto U de S es denso en S si $\ddot{U} = S$.

2.5.6 – Conjunto abierto

Sea $S \subset \mathrm{R}$. S es un conjunto abierto si para cualquier $x \in S$ existe un $\varepsilon > 0$ tal que todos los puntos t en el intervalo abierto $x - \varepsilon < t < x + \varepsilon$ están contenidos en S.

2.5.7 – Función topológicamente transitiva

f: J $\rightarrow$ J es topológicamente transitiva si para cualquier par de conjuntos abiertos U, V $\subset$ J existe k > 0 tal que $f^{k}(\mathrm{U}) \cap \mathrm{V} \neq 0$.

2.5.8 – Función con dependencia sensitiva en las condiciones iniciales

f: J $\rightarrow$ J tiene una dependencia sensitiva en las condiciones iniciales si existe $\delta > 0$ tal que, para cualquier $x \in$ J y cualquier vecindario N de x, existe $y \in$ N y existe $n \geq 0$ tal que $| f^{n}(x) - f^{n}(y) | > \delta$.

2.5.9 – Caos

Sea V un conjunto. f: V $\rightarrow$ V es caótico en V si:

1. f tiene una dependencia sensitiva en las condiciones iniciales.

2. f es topológicamente transitiva.
3. Los puntos periódicos son densos en V.

2.5.10 – Punto periódico hiperbólico

Sea p un punto periódico de período primo n. El punto p es hiperbólico si $|(f^n)'(p)| \neq 1$. El número $|(f^n)'(p)|$ es llamado el multiplicador del punto periódico.

2.5.11 – Atractor

Sea p un punto periódico hiperbólico de período n con $|(f^n)'(p)| < 1$. El punto p es llamado un punto periódico atrayente o atractor.

2.5.12 – Cuenca de un Atractor

Sea z_0 un punto periódico atrayente de período n (atractor). Existe un vecindario U de z_0 en el cual $F^n(z) \rightarrow z_0$ para todos los $z \in U$.

2.5.13 – Espacio de fase

Es el conjunto de estados posibles de un sistema dinámico.

2.5.14 – τ

Intervalo de tiempo en el que se puede medir el estado de un sistema dinámico.

2.5.15 – Entropía de Kolmogorov

Se considera un sistema dinámico con F grados de libertad. Se supone que el espacio de fase F-dimensional está particionado en cajas de tamaño ε^F. Se supone que existe un

atractor en el espacio de fase y que la trayectoria $\vec{x}(t)$ está en la cuenca del atractor. El estado del sistema es ahora medido a intervalos de tiempo τ. Sea $p(i_1,i_2,\ldots,i_d)$ la probabilidad conjunta de que $\vec{x}(t = \tau)$ está en la caja i_1, $\vec{x}(t = 2\tau)$ está en la caja i_2,..., $\vec{x}(t = d\tau)$ está en la caja i_d. La entropía de Kolmogorov se define entonces como [4]:

$$K = -\lim_{\tau\to 0}\lim_{\varepsilon\to 0}\lim_{d\to\infty}\frac{1}{d\tau}\sum_{i_1,\ldots,i_d} p(i_1,\ldots,i_d) * \ln p(i_1,\ldots,i_d)$$

2.5.16 – Dimensión fractal

La dimensión de un sistema dinámico es el mínimo número de variables de estado requeridas para describir la dinámica del sistema. En casi todos los sistemas caóticos se pueden observar dimensiones fractales, entendiéndose por estas últimas las que no son enteras [21].

2.6 - Caracterización de sistemas caóticos

Uno de los parámetros utilizados para caracterizar sistemas caóticos es la Dimensión de Correlación, la cual se define como [1]:

$$D_C := \lim_{\varepsilon\to 0}\frac{\ln\sum_{i}^{N(\varepsilon)} P_i^2}{\ln\varepsilon}$$

$$C(\varepsilon) := \lim_{N\to\infty}\frac{1}{N^2}\left\{\text{el número de pares de puntos } x_i, x_j \text{ tales que } \|x_i - x_j\| < \varepsilon\right\}$$

Para poder interpretar el numerador de la fórmula anterior, supongamos que N puntos de una trayectoria han sido reunidos ya sea a través de una simulación o de mediciones. Se define la Correlación como [1]:

Entonces:

$$D_C = \lim_{\varepsilon \to 0} \frac{\ln C_{(\varepsilon)}}{\ln \varepsilon}$$

Para mostrar la factibilidad de esta última fórmula, sea n_i el número de puntos situados en el i-ésimo elemento de volumen. Por lo tanto:

$$P_i = \lim_{n \to \infty} \frac{n_i}{N}$$

Y como el elemento de volumen tiene diámetro ε, todos los n_i puntos se sitúan a una distancia entre sí menor o igual a ε formando n_i^2 pares de puntos. Se sigue que:

$$C(\varepsilon) = \lim_{N \to \infty} \frac{1}{N^2} \sum_{i}^{N(\varepsilon)} n_i^2 = \lim_{N \to \infty} \sum_{i}^{N(\varepsilon)} \frac{n_i^2}{N^2} = \sum_{i}^{N(\varepsilon)} \left\{ \lim_{n \to \infty} \frac{n_i^2}{N^2} \right\} = \sum_{i}^{N(\varepsilon)} p_i^2$$

En otras palabras, dado un sistema dinámico de dimensión *n*, que exhibe un atractor caótico, la dimensión de correlación se define como la probabilidad de que un par de puntos elegidos al azar en el atractor estén separados por una distancia menor que ε.

Entonces, si deseamos estimar la dimensión de un atractor A que está embebido en un espacio euclidiano de dimensión *m* a partir de una muestra de N puntos en el atractor, esto es, a partir del conjunto $\{x_1, x_2, ..., x_N\}$ con $x_i \in A \subset R^m$, se mide la distancia entre cada par de puntos y luego se calcula la correlación integral [6].

Se define de la siguiente manera:

$$C_d(\varepsilon) = \lim_{N\to\infty}\left[\frac{1}{N^2}\left|\left\{(n,m) \text{ tales que } \left[\sum_{i=1}^{d}\left|x_{n+i}-x_{m+i}\right|^2\right]^{1/2} < \varepsilon\right\}\right|\right] \quad (1)$$

donde d es el tamaño de las tuplas entre las que se calcula la distancia. Es decir que, en realidad, la distancia es tomada entre pares de vectores de tamaño d.

D_C puede ser fácilmente calculada ya que C(ε) es fácil de estimar, y es la dimensión de correlación que calcularemos para los sistemas de Henon y Lorentz, como veremos más adelante.

Otro de los parámetros que nos permite caracterizar un sistema caótico es la entropía de Kolmogorov (entropía K), que es una medida cuantitativa de la complejidad o grado del comportamiento caótico del sistema dinámico subyacente en la serie de tiempo. En general, valores más bajos de la entropía K indican un cierto grado de orden (o periodicidad) en el comportamiento del sistema dinámico, mientras que valores más altos de la entropía K indican un comportamiento más complejo o caótico.

Los valores de la entropía son interpretados y clasificados de la siguiente manera:

1. Entropía K = 0 implica que el sistema dinámico es constante o periódico.
2. Entropía K >> 0 (tendiendo a ∞) implica que el sistema dinámico es aleatorio.
3. 0 < Entropía K << ∞ implica que el sistema dinámico es caótico.

Para modelos definidos analíticamente, es muy fácil estimar la entropía K a partir de las ecuaciones que describen la evolución de la distancia entre dos puntos. Sin embargo, es muy difícil determinar la entropía K directamente a partir de la medición de una señal.

Grassberger y Procaccia desarrollaron una medida, llamada número K_2, como una estimación del límite inferior asintótico de la entropía de Kolmogorov, que se calcula a partir de los datos de la serie de tiempo [4].

Para ver cómo se calcula el número K_2, consideremos el conjunto de orden q de las Entropías de Renyi que se definen de la siguiente manera[4]:

$$K_q = -\lim_{\tau\to 0}\lim_{\varepsilon\to 0}\lim_{d\to\infty}\frac{1}{d\tau}\frac{1}{q-1}\ln\sum_{i_1,\ldots,i_d} p^q(i_1,\ldots,i_d)$$

También Grassberger y Procaccia[2][4] muestran que $C(\varepsilon)\sim\varepsilon^{v}$, denominando v el exponente de correlación. Ha sido probado[4] que v estima la dimensión fractal D del atractor.

Estas dos últimas ecuaciones llevan a[4]:

$$\tilde{C}_d(\varepsilon) \underset{\substack{d\to\infty\\ \varepsilon\to 0}}{\sim} \varepsilon^{v}\exp(-d\tau K_2)$$

A partir de aquí, se puede ver [4] la implementación práctica: si se realiza un gráfico de $\ln C_d(\varepsilon)$ como una función de $\ln\varepsilon$ para una serie de valores crecientes de d, obtendremos una serie de rectas con una pendiente v, las cuales están desplazadas una de la otra por el factor $\exp(-d\tau K_2)$.

Por lo tanto:

$$K_2 = \lim_{\varepsilon \to 0} \lim_{d \to \infty} \left[\frac{1}{\tau} \ln \frac{C_d(\varepsilon)}{C_{d+1}(\varepsilon)} \right]$$

Este número tiene las siguientes propiedades:

1. $K_2 \geq 0$
2. $K_2 \leq K$
3. K_2 es ∞ para sistemas aleatorios
4. $K_2 \neq 0$ para sistemas caóticos

Para casos típicos, el número K_2 es numéricamente cercano a la entropía K, con la ventaja de que $K_2 > 0$ es una condición suficiente para caos. Sin embargo, la característica más importante de K_2 es que puede ser calculado a partir de señales experimentales.

Calcularemos la dimensión de correlación y el número K_2 para cada uno de los sistemas Henon y Lorentz, definidos por una serie de tiempo correspondiente a un modelo caótico subyacente.

2.7 - Cálculo de la dimensión de correlación y del número K_2

A continuación se describe cómo proceder para calcular la dimensión de correlación y el número K_2.

La muestra perteneciente al sistema a ser estudiado se representa por una secuencia de números $x_1, x_2, \ldots, x_i, \ldots, x_N$, donde el subíndice *i* indica el orden dentro de la secuencia y N es el tamaño de la muestra.

Se toma una dimensión de *embedding* d_E, (entendiendo por dimensión de *embedding* la mínima cantidad de variables de estado requeridas para describir la dinámica del sistema) y se construyen tuplas a partir de la serie original de datos de dimensión d_E, con d_E variando de 2 a infinito. Cada tupla será definida de la siguiente manera:

$$X_i = (x_i, x_{i+1} \ldots, x_{i+dE-1})$$

Con $i = 1, 2, \ldots, N\text{-}d_E\text{-}1$, siendo N el tamaño de la serie.

Luego, con las tuplas se calcula la correlación integral para cada dimensión $d = d_E$, según la ecuación (1), definiendo una hiperesfera de radio ε centrada en $\overline{x_i}$ y contando la cantidad de elementos que se encuentran dentro de dicha hiperesfera.

Una vez que se obtienen los valores correspondientes a la correlación integral $C_d(\varepsilon)$, ésta se grafica en función de ε, en escala logarítmica, para varios valores de la dimensión *d*. Nos referiremos a las curvas en dicho gráfico como "curvas *d*".

Veamos el gráfico correspondiente a la correlación integral para el mapa Henon:

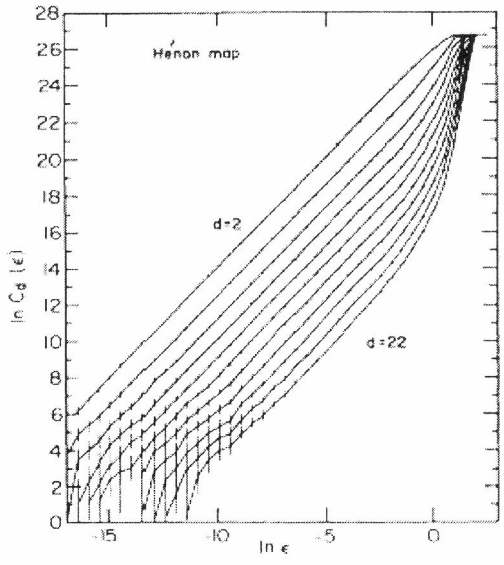


Fig. 2.4 – Correlación integral para Henon

El gráfico de la Figura 2.4 presenta 3 zonas bien diferenciadas. La primera, conocida como *zona de despoblación*, la segunda, con una pendiente constante y, finalmente, una región denominada de *región de saturación*.

La región de despoblación es un área donde no se encuentran elementos dentro de la hiperesfera de radio ε; esto es, donde existen pocos o ningún par de puntos cuya distancia entre ellos en R^d sea menor que el valor de ε.

El área donde todas las curvas *d* tienen una pendiente constante y son paralelas unas a otras se define como la *región de escala.*

La región donde todas las curvas *d* confluyen en una sola línea horizontal es llamada región de saturación. En la región de saturación, las distancias entre todos los posibles pares de puntos están contenidas dentro de una hiperesfera de radio ε.

Para calcular la dimensión de correlación es necesario analizar la pendiente de la curva ln C_d= f(ln(ε)) para cada una de las dimensiones *d*. Si para todas las dimensiones analizadas sucede que: esta pendiente es constante, puede determinarse una *región de escala*, las pendientes son aproximadamente iguales y la distancia entre las curvas en la *región de escala* tiende a permanecer constante, entonces el sistema es caótico. El valor de la pendiente en esta región es la dimensión de correlación.

Con respecto al número K_2, éste se calcula como la distancia asintótica entre sucesivas curvas *d* dentro de la *región de escala* en el límite de d→∞ [5].

El algoritmo propuesto en el desarrollo de esta tesis determina la *región de escala* y calcula a partir de ella la dimensión de correlación y el número K_2.

Antes de entrar de lleno en la implementación de este algoritmo, nos vamos a detener en el que calcula la correlación integral, valor que es usado tanto en el cálculo de la dimensión de correlación como en el cálculo de K_2.

o o o

Capítulo 3 – Descripción del Método

Para calcular la dimensión de correlación y el número K_2 de un sistema dinámico, se ha desarrollado un algoritmo basado en valores de la correlación integral $C_d(\varepsilon)$ del sistema, para distintos valores de ε y distintas dimensiones *d*. Por consiguiente, el método propuesto consta de dos partes:

1. Cálculo de la correlación integral
2. Cálculo de la dimensión de correlación y de K_2

3.1 – Formación de las tuplas – Cálculo de la correlación integral

El cálculo de la correlación integral se ha implementado sobre la base de la ecuación (1) presentada en la sección 2.6. La figura 3.1 muestra la porción del algoritmo serial que realiza dicho cálculo.

```
/* Compute Correlation Integral */
    for ( d = 2; d < D; d++)
    {   for (j=0; j<BUFFL; j++)
              comparevec[j] = 0;
        for ( n = 50; n < (N-2*d+1); n++)
            {     for ( m = n+1 ; m < (N-1); m++)
                  {     distance = 0;
                        for ( i=0; i <=(d-1);i++)
                                    distance = distance+(x[n+i]-
                                    x[m+i])*(x[n+i]-x[m+i]);
                        distance = sqrt(distance);
                        i = 0;
                        for ( eps=epsinc; eps <= epsfinal;
                              eps=eps+epsinc)
                        {if ( distance <= eps)
                              { comparevec[i] = comparevec[i]+2.0;}
                              i = i+1;
                        }
                  }
            }
      }
```

Fig. 3.1 – Cálculo de la Correlación integral

3.2 – Determinación de la dimensión de correlación y del número K_2

Para el cálculo de la dimensión de correlación y del número K_2 se ha desarrollado un algoritmo que se ocupa de hallar las rectas paralelas de la *región de escala,* calcular su pendiente y la distancia entre ellas.

El algoritmo se basa en valores de la correlación integral $C_d(\varepsilon)$ del sistema (en escala logarítmica), para distintos valores de ε y distintas dimensiones *d*. Entonces, los valores obtenidos mediante el cálculo presentado en la sección anterior, se toman como datos de entrada a través de un archivo plano y se almacenan en una matriz.

Luego, para cada curva *d*, el algoritmo realiza el siguiente procedimiento:

1. Define segmentos, a los que denominamos "ventanas"; las mismas se obtienen de la siguiente manera:

    1.1. Se toma el mínimo valor de ln(ε) de la muestra como inicio de la primera ventana.

    1.2. Al valor considerado como inicio de la ventana, se suma el tamaño de las ventanas, que se determinó igual al 10% de la diferencia entre el valor máximo y el valor mínimo de ln(ε) en la muestra.

    1.3. Se toma como extremo final de la ventana el primer valor de ln(ε) de la muestra que sea mayor o igual al obtenido en 1.2.

1.4. El extremo final de la ventana se toma como inicio de la ventana siguiente, y se repite el proceso a partir de 1.2 hasta que el extremo final de una ventana coincida con el valor máximo de ln(ε) en la muestra. La figura 3.2 es un ejemplo de cómo quedarían definidas las ventanas.

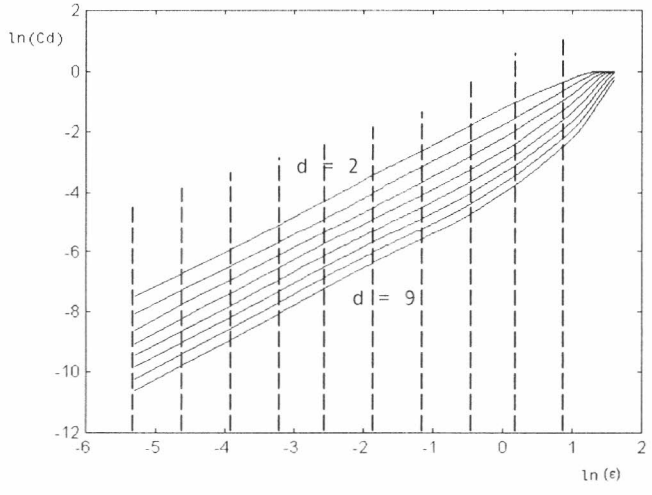


Fig. 3.2 – Determinación de "ventanas"

2. Para cada ventana, evalúa si se encuentra en la zona de despoblación, comparando el valor de la correlación integral ($C_d(\varepsilon)$) en los puntos que son extremos de la ventana con el valor tomado como cota de despoblación. Se define como cota de despoblación a un porcentaje de la cantidad total de puntos de la muestra, y se utiliza para evitar el análisis del sistema en la zona de despoblación. Si el fin de la ventana está dentro de la zona de despoblación, dicha ventana se descarta. Si el inicio de la ventana se encuentra en la zona de despoblación, entonces se toma como inicio el siguiente punto y se verifica otra vez, hasta encontrar un inicio adecuado.

3. Para cada ventana considerada como adecuada en el punto anterior, se calcula el coeficiente de correlación a fin de determinar el error al interpolar una recta con los puntos dentro de la ventana. Si el error es mayor que una determinada cota, la ventana se descarta.

4. En cada una de las ventanas aceptadas en el punto anterior, se interpola una recta, cuya pendiente se compara con la de la recta obtenida en la ventana anterior. Si la diferencia entre ambas pendientes supera cierta cota, se da por concluido el recorrido de las ventanas. De esta forma, se descarta la región donde la curva deja de aproximar una recta y se evita entrar en la zona de saturación.

5. Se interpola una nueva recta que abarque todas las ventanas aceptadas en las condiciones anteriores. De esta manera, se obtiene una única recta para cada curva *d*.

6. La pendiente de las rectas obtenidas es la dimensión de correlación. K_2 se calcula a partir de la relación trigonométrica entre dos de estas rectas para un mismo valor de abscisa (Ver figura 3.3).

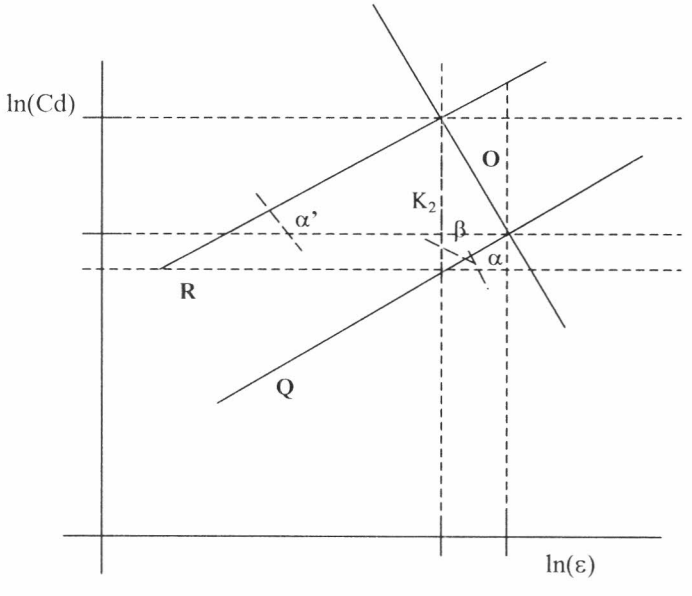


Fig. 3.3 – Determinación de K_2 a partir de $C_d(\varepsilon)$

En la figura 3.3 se observa lo siguiente:

$\alpha' = \alpha \Rightarrow tg(\alpha') = tg(\alpha)$

pendiente de R = $tg(\alpha')$

pendiente de Q = $tg(\alpha)$

} Dimensión de Correlación

O = distancia (*dist*) entre las dos rectas paralelas R y Q tomada sobre la normal a las dos rectas

$sen(\beta) = sen(90 - \alpha) = O / K_2$ por lo que:

$$K_2 = \frac{dist}{sen(90-\alpha)}$$

Los datos de entrada del algoritmo son el incremento de ε, la cantidad de valores distintos de ε y la cantidad de dimensiones o curvas *d*. Asimismo, utiliza los siguientes datos: cota para comparar entre sí las pendientes de las rectas de cada ventana y cota para el coeficiente de correlación.

Estos valores son dependientes del sistema a evaluar y de los datos disponibles para analizar.

Una implementación en Matlab del método descrito se puede encontrar en el Apéndice 1 de este trabajo.

o o o

Capítulo 4 – Paralelización del algoritmo utilizando PVM

4.1 – Breve análisis de la complejidad del algoritmo serial

La complejidad computacional del algoritmo serial para el cálculo de la dimensión de correlación y de K_2 radica principalmente en el cálculo de la correlación integral según la ecuación (1). Cuando se calcula el número K_2 con una dimensión de *embedding* $D << N$ (donde N es la cantidad de puntos de datos de la serie de tiempo), hay aproximadamente $N*(N-1) / 2$ pares de vectores de dimensión d (donde $d \leq D$) a ser considerados, observándose una complejidad computacional de $O(N^2)$ [5].

Por lo tanto, para valores grandes de N, el cálculo serial en sistemas monoprocesadores convencionales se vuelve inmanejable, y el tiempo de respuesta limita la utilidad del algoritmo en aplicaciones críticas.

Como alternativa para mejorar el tiempo de ejecución del algoritmo, más adelante presentaremos una implementación paralela del mismo utilizando el sistema PVM

4.2 – Descripción básica de la herramienta PVM

4.2.1 – Introducción

PVM (Parallel Virtual Machine) es una herramienta de software, ampliamente divulgada en el ambiente académico, que permite simular una máquina paralela virtual sobre una red de computadoras heterogéneas.

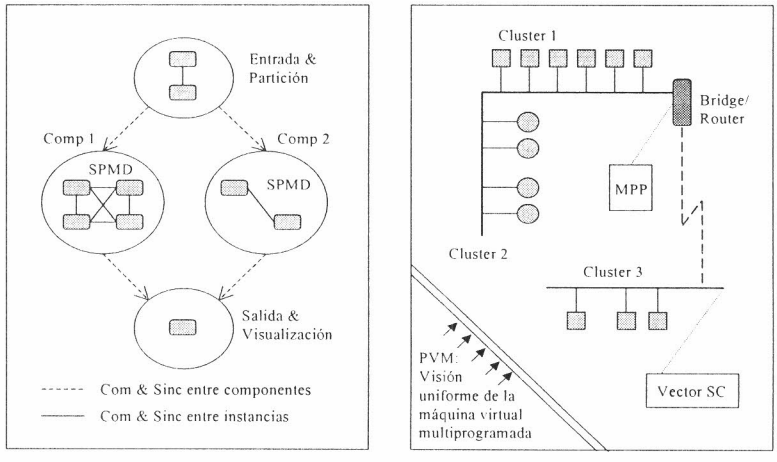
Provee un esquema de trabajo unificado, dentro del cual pueden ser desarrollados programas paralelos de una manera directa y eficiente utilizando el hardware existente.

El sistema PVM usa el modelo de pasaje de mensajes para permitir a los programadores explotar la computación distribuida a través de una amplia variedad de tipos de computadoras. Hasta ahora, PVM está disponible para 40 arquitecturas diferentes, permitiendo combinar estaciones de trabajo Unix, máquinas de memoria compartida y máquinas masivamente paralelas en una única máquina paralela virtual. También se ha extendido hacia el ambiente de computadoras personales, al incorporar la arquitectura Win32. De esta forma, se abre una gran oportunidad de popularización para la programación en paralelo mediante la utilización de este ambiente de desarrollo.

4.2.2 - El modelo computacional

El proyecto PVM comenzó en 1989 y ha evolucionado a través de tres versiones del software, habiéndose distribuido públicamente las dos últimas.

El modelo computacional básico, que ha permanecido semánticamente sin cambios, ve a las aplicaciones como conformadas por componentes, cada uno de los cuales representa un sub-algoritmo. Cada componente es un programa SPMD (Single Process Multiple Data), potencialmente manifestado en múltiples instancias que cooperan tanto entre ellas como con instancias de otros componentes a través de los mecanismos de comunicación y sincronización soportados (ver Fig. 4.1).



(a) Modelo computacional de PVM (b) Visión global de la arquitectura de PVM

Fig. 4.1 – Visión Global del Sistema PVM

La unidad de concurrencia en PVM es el proceso. Las dependencias en el grafo de flujo de procesos son implementadas embebiendo las primitivas de PVM apropiadas para el manejo y sincronización de procesos dentro de las construcciones de flujo de control del lenguaje de programación huésped (C ó Fortran).

El modelo de implementación utiliza la noción de *host pool* (una colección de sistemas de computación interconectados que conforman la máquina virtual), sobre los que determinados *daemons* (procesos de kernel) se ejecutan y cooperan para emular un sistema de computación concurrente. Las aplicaciones solicitan y reciben servicios de estos *daemons*. Las prestaciones entran esencialmente dentro de las categorías de manejo de procesos y configuración de la máquina virtual, pasaje de mensajes, sincronización, y tareas de mantenimiento y de chequeo de estados.

Bajo PVM, un conjunto de computadoras definido por el usuario emula a una gran computadora de memoria distribuida. El término "máquina virtual" se usa para designar a esta computadora lógica de memoria distribuida. Múltiples usuarios pueden configurar máquinas virtuales superpuestas, y cada usuario puede ejecutar varias aplicaciones PVM simultáneamente. PVM provee las funciones para disparar tareas sobre la máquina virtual y permite que las tareas se comuniquen y sincronicen entre sí.

Una tarea se define como una unidad de computación en PVM, análogamente a un proceso en Unix. Las aplicaciones, que pueden ser escritas en Fortran77 o C, pueden ser paralelizadas usando construcciones de pasaje de mensajes, comunes a la mayoría de las computadoras de memoria distribuida. A través del envío y recepción de mensajes, múltiples tareas de una aplicación pueden cooperar para resolver un problema en paralelo. La figura 4.1 muestra el modelo computacional de PVM así como una abstracción de la arquitectura del sistema [10].

El modelo asume que cualquier tarea puede enviar un mensaje a cualquier otra tarea PVM, y que no existe límite para el tamaño o número de mensajes. El modelo de comunicación de PVM provee funciones de *send* (envío) bloqueante asincrónico, *receive* (recepción) bloqueante asincrónico y *receive* no bloqueante. En esta terminología, un *send* bloqueante retorna el control tan pronto como el buffer esté libre para ser reusado, sin importar el estado del receptor. Un *receive* no bloqueante retorna inmediatamente, con los datos o con una indicación de que el buffer está vacío, mientras que un *receive* bloqueante retorna sólo cuando los datos se encuentran en el buffer.

Además de estas funciones de comunicación punto-a-punto, el modelo soporta el *multicast* a un conjunto de tareas y el *broadcast* a un grupo de tareas definido por el usuario. El modelo de PVM garantiza que se preserve el orden de los mensajes entre cualquier par de entidades comunicantes.

4.2.3 - Características principales

PVM provee rutinas que permiten a un proceso usuario registrarse y salir de un grupo de procesos, agregar y eliminar hosts de la máquina virtual, iniciar y terminar tareas PVM, enviar señales y sincronizarse con otras tareas, y también rutinas para obtener información acerca de la configuración de la máquina virtual y de las tareas activas. La sincronización puede lograrse de varias maneras, por ejemplo notificando a un conjunto de tareas de la ocurrencia de un evento enviándoles un mensaje con un identificador (*tag*) especificado por el usuario y que la aplicación puede revisar. Los eventos de notificación incluyen la terminación de una tarea, la eliminación (o falla) de un host y el agregado de un host.

PVM también provee rutinas para empaquetar y enviar mensajes entre tareas. Las principales rutinas de comunicación incluyen el *send* asincrónico a una única tarea y el *multicast* a una lista de tareas. PVM transmite los mensajes sobre la red subyacente usando el mecanismo más rápido disponible; por ejemplo: UDP, TCP sobre redes basadas en los protocolos de Internet, u otras interconexiones de alta velocidad disponibles entre los procesadores comunicantes. Los mensajes pueden recibirse filtrando por el origen o por el

tag (o también sin filtrar), tanto con rutinas de *receive* bloqueantes como con rutinas no bloqueantes. Se puede llamar a una rutina para obtener información acerca de los mensajes recibidos, como ser el origen, *tag* y tamaño de los datos.

Los buffers de mensajes son asignados dinámicamente, permitiendo entonces mensajes limitados en tamaño sólo por parámetros nativos de cada máquina. Existen rutinas para crear y manejar múltiples buffers de *send* y *receive*. Esta característica permite al usuario escribir librerías matemáticas PVM e interfases gráficas que pueden ser llamadas dentro de otras aplicaciones PVM, sin conflictos de comunicación. El usuario puede cambiar de contexto, de un conjunto de buffers (por ejemplo, usados por la aplicación) a otro conjunto de buffers (por ejemplo, usados dentro de la llamada a una librería matemática).

4.2.4 - Características adicionales

Las últimas versiones de PVM están diseñadas para que las llamadas nativas de un multiprocesador puedan ser compiladas en el programa fuente. Gracias a esto, se logra optimizar la performance del pasaje de mensajes, aprovechando las primitivas de comunicación propias de esta arquitectura. Los mensajes entre dos nodos de un multiprocesador usan sus rutinas nativas de pasaje de mensajes, mientras que los mensajes destinados a un host externo son ruteados a través del *daemon* PVM del multiprocesador. En los sistemas de memoria compartida, el movimiento de datos puede ser implementado con un conjunto compartido de buffers y primitivas de bloqueo.

4.2.5 – Implementación

Desde el punto de vista de la implementación, el sistema PVM está compuesto de dos partes. La primera es un *daemon*, llamado *pvmd*, que se ejecuta sobre todas las computadoras que conforman la máquina virtual. *Pvmd* está diseñado para que cualquier usuario válido pueda instalarlo en una máquina.

Un usuario que desee usar PVM primero debe configurar una máquina virtual, especificando una lista de hosts; los *daemons* se levantan en cada uno de estos hosts y

cooperan para emular una máquina virtual. La aplicación PVM puede luego ser iniciada desde la línea de comandos en cualquiera de estas computadoras.

El primer *pvmd* (iniciado a mano desde la consola) se denomina maestro (*master*), mientras que a los otros (iniciados por el maestro) se los denomina esclavos (*slaves*) (ver Fig. 4.2). Durante un normal funcionamiento, todos son considerados iguales. Pero sólo el maestro puede iniciar nuevos esclavos y agregarlos a la configuración. Los pedidos de reconfiguración originados en un host esclavo son reenviados al *master*. Asimismo, sólo éste puede eliminar hosts de la máquina virtual.

La segunda parte del sistema es una librería de rutinas de interfase de PVM llamada *libpvm3.a*. Esta librería contiene rutinas invocables por el usuario para pasaje de mensajes, creación de procesos, coordinación de tareas y modificación de la máquina virtual. Los programas de aplicación deben ser vinculados con esta librería para usar PVM.

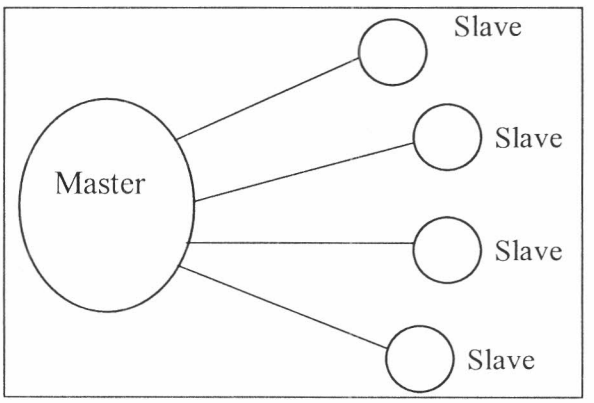


Fig. 4.2 – Paradigma Maestro-Esclavo

Los componentes del sistema PVM han sido compilados y probados sobre una gran variedad de arquitecturas, incluyendo desde computadoras portátiles 386 hasta Cray C90s y computadoras MPP. Además, varios fabricantes están proveyendo y apoyando versiones

optimizadas de PVM para sus sistemas multiprocesadores, incluyendo Cray Research, IBM, Convex, Intel, SGI y DEC.

PVM usa un identificador de tarea (*TID*) para direccionar a *daemons* (*pvmd*), tareas y grupos de tareas dentro de una máquina virtual. Los *pvmds* asignan localmente los TIDs a las tareas que se ejecutan sobre la misma máquina, sin necesidad de comunicación entre hosts.

PVM asigna un nombre de arquitectura a cada tipo de máquina sobre la cual corre para distinguir entre máquinas que corren diferentes ejecutables, debido a diferencias en el hardware o en el sistema operativo. Se han definido muchos nombres estándares, y otros pueden ser agregados. A veces, máquinas con ejecutables incompatibles usan la misma representación binaria de datos. PVM saca ventaja de esto, para evitar la conversión de datos.

Las tareas y *daemons* de PVM pueden componer y enviar mensajes de longitud arbitraria. Para el intercambio entre hosts con formatos incompatibles, los datos se convierten usando el protocolo XDR.

4.2.6 - Interfase de principales funciones

A continuación, enunciaremos las funciones usadas en nuestro trabajo para la interfase con PVM, dando una somera descripción de cada una de ellas. Para una explicación más detallada, se puede consultar la bibliografía indicada en [9].

4.2.6.1 - Información y Control de Procesos

int tid = **pvm_mytid** (void)

Retorna el identificador de tarea del proceso llamador.

int info = **pvm_exit** (void)

Le comunica al *daemon* local que el proceso llamador sale de PVM.

int numt = **pvm_spawn** (char *task, char **argv, int flag, char *where, int ntask, int *tids)

Inicia la ejecución de nuevos procesos PVM.

int tid = **pvm_parent** (void)

Retorna el identificador de tarea del proceso que inició al proceso llamador.

4.2.6.2 - Pasaje de mensajes

int bufid = **pvm_initsend** (int encoding)

Limpia el buffer de envío default y especifica el tipo de codificación del mensaje.

int info = **pvm_pkint** (int *np, int nitem, int stride)

Empaqueta el buffer de mensajes activo con un vector de tipo entero.

int info = **pvm_send** (int tid, int msgtag)

Envía los datos que se encuentran en el buffer de mensajes activo.

int bufid = **pvm_recv** (int tid, int msgtag)

Recibe un mensaje. El proceso llamador queda bloqueado hasta la llegada de algún mensaje.

int info = **pvm_upkint** (int *np, int nitem, int stride)

Desempaqueta el buffer de mensajes activo en un vector de tipo entero.

4.3 - Descripción del algoritmo paralelo del cálculo de la correlación integral

Tal como explicamos en la sección 4.1, el algoritmo serial para el cálculo de la correlación integral demanda mucho tiempo de procesamiento. Por ello, el objetivo de la paralelización del algoritmo es disminuir el tiempo de ejecución del mismo.

Debido a que el algoritmo ejecuta el procedimiento para las distintas dimensiones, se tomó esta variable como la más apropiada para la creación de los procesos paralelos. En el proceso serial, estos cálculos se realizan mediante un loop iterativo. Esta técnica de paralelización se denomina *loop-unfolding* o *loop-transformation* (transformación de bucle).

Las razones generalmente esgrimidas para justificar el uso de esta técnica (y que se aplican en nuestro caso) son:

1. Poca o ninguna sincronización dentro del loop iterativo seleccionado; es decir, que exista poca dependencia entre el conjunto de operaciones contenidas en una iteración dada, respecto de los resultados obtenidos en iteraciones anteriores.

2. Posibilidad de diseñar estructuras de datos paralelos que puedan ser asignadas a la memoria local de cada proceso esclavo.

Por lo tanto, el proceso maestro genera los procesos esclavos y les envía la serie de datos de la función a procesar (ver Fig. 4.3).

```
/* enroll in pvm */
     mytid = pvm_mytid();

/* Lanzar procesos esclavos paralelos que calculan para distintas d */
    numt=pvm_spawn(SLAVENAME, (char**)0, 0, "", nproc, tids);
    printf("NUMT es %d\n", numt);
    if( numt < nproc )
    { printf("\n Trouble spawning slaves. Aborting. Error codes
                  are:\n");
      for ( i=numt ; i<nproc ; i++ )
          printf("TID %d %d\n",i,tids[i]);
      for ( i=0 ; i<numt ; i++ )
          pvm_kill( tids[i] );
      pvm_exit();
      return(1);  }
    printf("SUCCESSFUL\n");


/* Begin User Program */

/* Envio de datos a los procesos esclavos */
        msgtag = 3;
       for (i=0; i<nproc;i++)
      { pvm_initsend(PvmDataDefault);
        pvm_pkint(&nproc, 1, 1);
        pvm_pkint(&i, 1, 1);
        pvm_pkint(&n, 1, 1);
        pvm_pkint(&dimmax, 1, 1);
        pvm_pkint(&h, 1, 1);
        pvm_pkint(&canteps, 1, 1);
        pvm_pkdouble(&epsinc, 1, 1);
        pvm_pkdouble(&epsfinal, 1, 1);
        pvm_pkdouble(&x[0], n, 1);
        info = pvm_send(tids[i], msgtag);
        if( info == 0 )
            printf("\n Spawn exitoso par10k20\n");
              else
            printf("\n Spawn Error\n");
      }
          for (i=0; i<nproc;i++)
            printf(" %x", tids[i]);
```

Fig. 4.3 – Proceso Maestro – Envío de datos

Los procesos esclavos arman las tuplas correspondientes a la dimensión que les fue asignada y realizan el cálculo descrito en la ecuación (1) empleando los datos recibidos (ver Fig. 4.4).

```
/* enroll in pvm */
      mytid = pvm_mytid();

/* Receive data from master */
    msgtype = 3;
    pvm_recv( -1, msgtype);
    pvm_upkint(&nproc, 1, 1);
    pvm_upkint(&proc, 1, 1);
    pvm_upkint(&N, 1, 1);
    pvm_upkint(&D, 1, 1);
    pvm_upkint(&h, 1, 1);
    pvm_upkint(&canteps, 1, 1);
    pvm_upkdouble  (&epsinc, 1, 1);
    pvm_upkdouble  (&epsfinal, 1, 1);
    pvm_upkdouble (&x[0], N, 1);

/* Compute Correlation Integral  */
      for (j=0; j < h; j++)
      {     for (i=0; i< canteps; i++)
                  comparevec[j][i] = 0;
      }
    j=0;
    for ( d =2 ; d < D; d++)
    {
      if ((d % nproc)== proc)
      {
      for ( n = 50; n < (N-2*d+1); n++)
      {
            for ( m = n+1 ; m < (N-1); m++)
            {
                  distance = 0;
                  for ( i=0; i <= (d-1); i++)
                  distance = distance+(x[n+i]-x[m+i])*(x[n+i]-x[m+i]);
                  distance = sqrt(distance);
                  i = 0;
                  for ( eps=epsinc; eps <= epsfinal; eps=eps+epsinc)
                  {
                     if ( distance <= eps)
                         comparevec[j][i] = comparevec[j][i]+2.0;
                     i = i+1;
                  }
            }
      }
      j = j+1;
      }
}
/* Send result to master */
      pvm_initsend( PvmDataDefault );
      pvm_pkint( &comparevec[0][0], (h*canteps), 1 );
      pvm_pkint( &proc, 1, 1 );
      pvm_pkint( &mytid, 1, 1 );
    msgtype = 5;
    master = pvm_parent();
```

```
    pvm_send( master, msgtype );

/* Program finished. Exit PVM before stopping */
    pvm_exit();
}
```

Fig. 4.4 – Proceso Esclavo

El proceso maestro, luego de recibir los resultados de cada proceso esclavo, arma la matriz CompareVec conteniendo los datos requeridos para el cálculo de la correlación integral (ver Fig. 4.5).

```
/* Recibe de slaves */
    msgtype = 5;
    for( i=0 ; i<nproc ; i++ )
    { pvm_recv( -1, msgtype );
      pvm_upkint( &comparevec[0][0], (h*canteps), 1 );
      pvm_upkint( &proc, 1, 1 );
      pvm_upkint( &who, 1, 1 );
      printf("Recibi Proceso%d\n", proc);
      printf("Recibi Who%d\n", who);
      for( j=0 ; j<h ; j++ )
      {     l =  ((proc + 2) % nproc) + (j*nproc);
            for( k=0 ; k<canteps ; k++ )
               comparevecfinal[l][k] = comparevec[j][k];
      }
  }
```

Fig. 4.5 – Proceso Maestro – Recepción de datos

En la figura 4.6 se puede observar la arquitectura utilizada para este algoritmo paralelo.

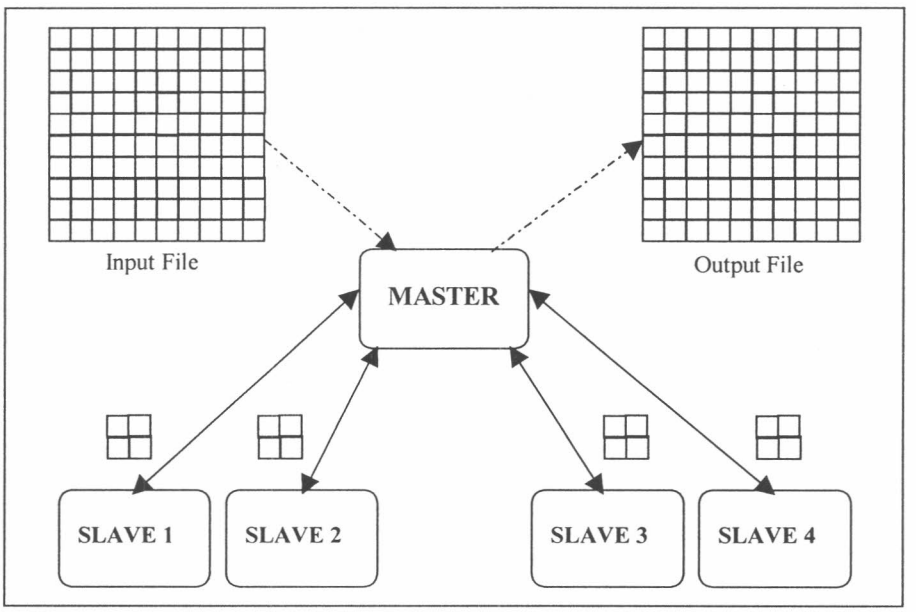


Fig. 4.6 - Arquitectura del Algoritmo Paralelo

Dado que el cálculo requiere más tiempo de CPU cuanto mayor es la dimensión que se está obteniendo, hemos introducido una mejora adicional al algoritmo paralelo, que es asignarle a cada proceso esclavo más de una dimensión, pero considerando la cantidad total de procesos paralelos para asignar dimensiones de similar complejidad a cada algoritmo. Por ejemplo, si estamos calculando 11 dimensiones (de la 2 a la 12) y contamos con 4 procesadores, el esquema de asignación sería el siguiente:

Proceso esclavo 1:	Dimensión 2 – 6 – 10
Proceso esclavo 2:	Dimensión 3 – 7 – 11
Proceso esclavo 3:	Dimensión 4 – 8 - 12
Proceso esclavo 4:	Dimensión 5 – 9

El objetivo de esta mejora es lograr que todos los procesos esclavos terminen su ejecución en un tiempo similar, a fin de poder devolver al control al proceso maestro para poder continuar con el procesamiento.

o o o

Capítulo 5 – Análisis de los resultados obtenidos

5.1 – Método de cálculo

El método anteriormente descrito fue aplicado a conjuntos de datos obtenidos de: un sistema periódico (función seno(x)), un sistema aleatorio (tomando muestras a intervalos regulares) y dos sistemas caóticos (Henon y Lorentz).

Las tuplas fueron construidas para una cantidad significativa de dimensiones que permitieron observar el comportamiento de cada sistema.

La selección del rango del radio ε de la hiperesfera depende del sistema a ser estudiado y, en nuestro caso, seleccionamos un valor de ε basado en el rango de las muestras.

Consideramos los valores mínimo y máximo de la muestra a analizar (por ejemplo, en el caso de Henon, los valores mínimo y máximo son –2 y 2, respectivamente) y consideramos un valor empírico de 0,125% de la diferencia en valor absoluto entre estos dos valores. Pudimos comprobar, sobre la base de los resultados obtenidos, que este valor de ε ha permitido obtener ventanas representativas de los sistemas estudiados.

Otros valores que fue necesario determinar fueron los siguientes:

- Tamaño de las ventanas: Fue establecido en el 10% del rango de valores de ε.

- Cota de variación de pendiente: Es el valor máximo aceptado como diferencia entre las pendientes de las rectas interpoladas en dos ventanas sucesivas. Este valor fue tomado en forma empírica igual a un 5% y nos permitió obtener resultados aceptables para el cálculo efectuado.

- Cota de despoblación: Para asegurarnos de no considerar datos en la zona de despoblación, consideramos una cota de despoblación del 0,005% de las tuplas analizadas.

Los siguientes resultados fueron obtenidos a partir de un conjunto de 10.000 puntos:

Sistema	Dim. Correlación calculada	Dim. Correlación	K_2 calculado	K_2
Henon	1.19	1.22[2]	0.32	0.32[4]
Lorentz	1.78	2.06[2]	0.25	---
Seno(x)	No calculable	---	No calculable	0[4]
Random	En aumento	---	∞	∞[4]

A partir de la tabla anterior es evidente que los valores obtenidos para los sistemas estudiados concuerdan con trabajos anteriores [2][4].

En el caso de los sistemas caóticos analizados, el método propuesto ha podido encontrar una *región de escala* válida, por lo tanto hemos obtenido los valores de la dimensión de correlación y del número K_2. Según las definiciones presentadas en el Capítulo 2, estos valores nos confirman que los sistemas son caóticos.

Para los sistemas periódico y random, no hemos encontrado una *región de escala*, por lo tanto no podemos calcular la correlación integral ni el número K_2.

Mediante la utilización del algoritmo propuesto y comparando con resultados obtenidos por otros autores, observamos que con este método se puede obtener una excelente cota inferior de la entropía de un sistema caótico.

5.2 – Aplicación del método a sistemas caóticos

Las figuras 5.1 y 5.2 muestran el gráfico de la correlación integral (C_d) como una función de ε en coordenadas log-log. Las curvas pertenecen al sistema Henon en el primer caso y al sistema de Lorentz en el segundo. Las rectas que se grafican son el resultado de la interpolación realizada por el algoritmo propuesto.

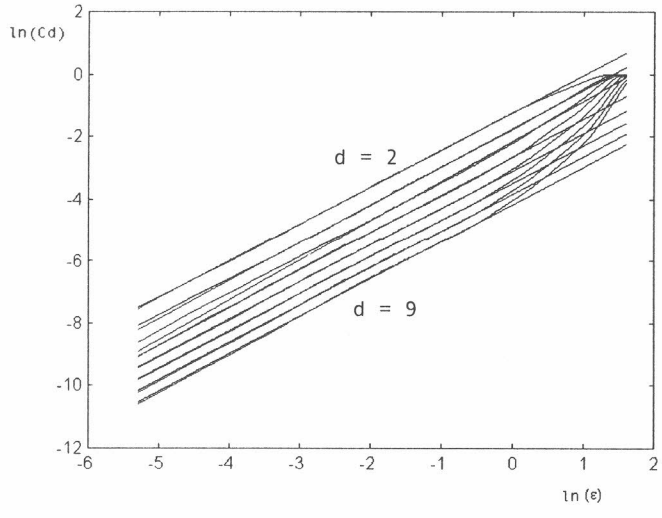


Fig. 5.1 – Método Propuesto aplicado al Sistema Henon

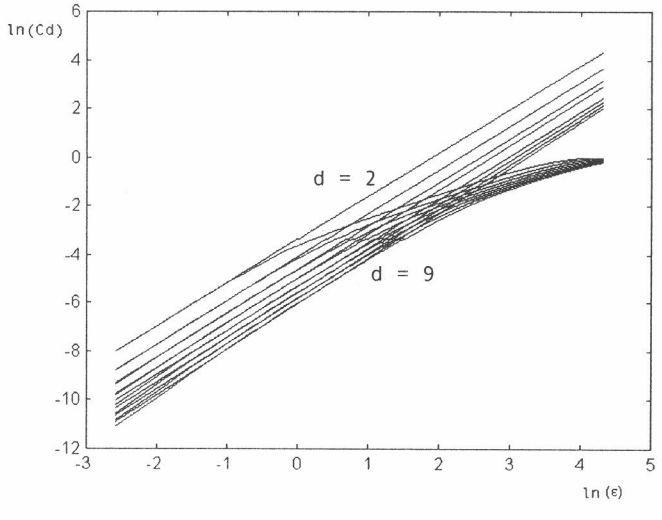


Fig. 5.2 – Método Propuesto aplicado al Sistema de Lorentz

5.3 – Aplicación del método a un sistema periódico y a un sistema random

Las figuras 5.3 y 5.4 muestran el gráfico de la correlación integral (C_d) como una función de ε en coordenadas log-log para un sistema periódico (función Seno) y para un sistema aleatorio.

Para estos sistemas el método no ha hallado una *región de escala* válida.

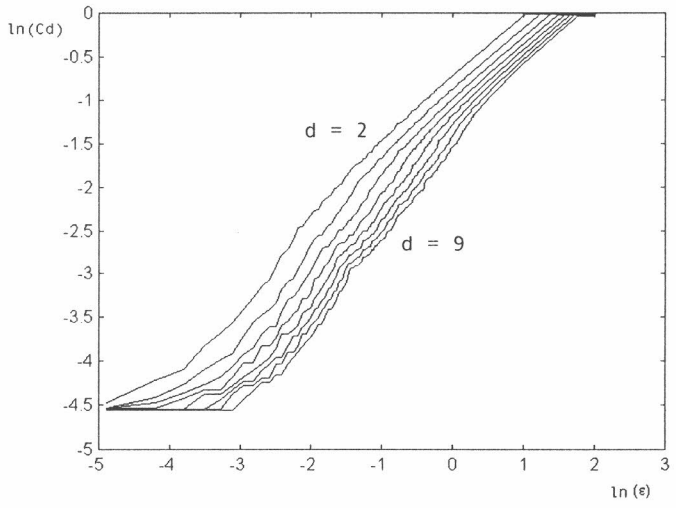


Fig. 5.3 – Método Propuesto aplicado a un Sistema Periódico

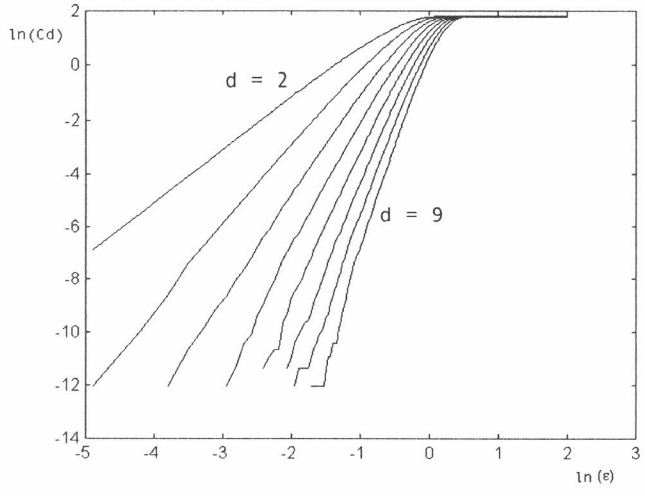


Fig. 5.4 – Método Propuesto aplicado a un Sistema Aleatorio

5.4 – Breve descripción de la *Región de escala* encontrada para los sistemas caóticos

5.4.1 – Región de escala para el sistema Henon

La *región de escala* observada para la dimensión 2 de este sistema comprende la región determinada por los valores de ε entre 0.005 y 1.28. En el gráfico de la figura 5.5 se ha detallado esta *región de escala* teniendo en cuenta que los valores representados corresponden a ln(ε). Los valores de la dimensión de correlación y de K_2 obtenidos para esta dimensión son **1,19** y **0,32** respectivamente.

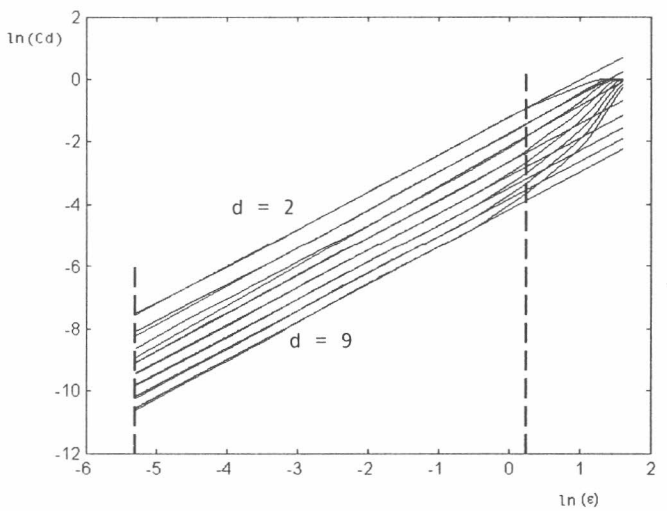


Fig. 5.5 – Sistema Henon - Región de escala obtenida para la Dimensión 2

De acuerdo a trabajos anteriores [12], es suficiente tomar *d* mayor o igual a la dimensión de correlación conocida del sistema. Por ello, se ha considerado *d* = 2 para seleccionar los resultados informados en el cuadro comparativo de la sección 5.1.

5.4.2 – Región de escala para el sistema Lorentz

La *región de escala* observada para la dimensión 2 de este sistema comprende la región determinada por los valores de ε entre 0.15 y 0.30. En el gráfico de la figura 5.6 se

ha detallado esta *región de escala* teniendo en cuenta que los valores representados corresponden a ln(ε). Los valores de la dimensión de correlación y de K_2 obtenidos para esta dimensión son **1,76** y **0,35** respectivamente.

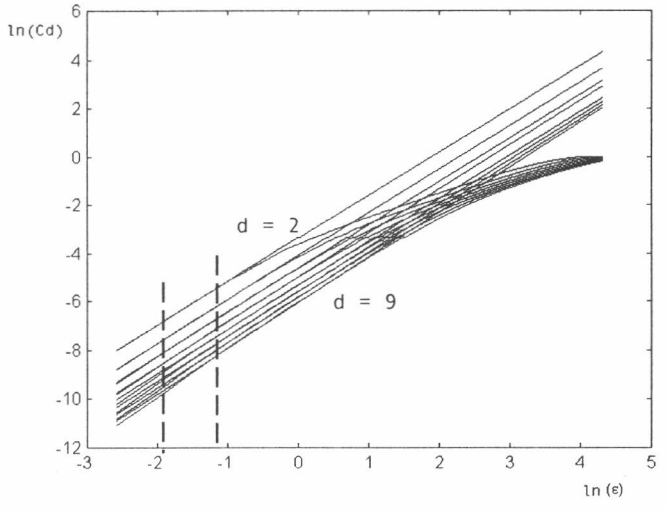


Fig. 5.6 – Sistema de Lorentz - Región de escala obtenida para la Dimensión 2

Si consideramos la dimensión 3 del sistema Lorentz, la *región de escala* se encuentra en la región determinada por los valores de ε entre 0.15 y 0.60. En el gráfico de la figura 5.7 se ha detallado esta *región de escala* teniendo en cuenta que los valores representados corresponden a ln(ε). Los valores de la dimensión de correlación y de K_2 obtenidos para esta dimensión son de **1,78** y **0,25** respectivamente.

Al igual que para el sistema Henon, para este sistema es suficiente tomar d mayor o igual a la dimensión de correlación conocida del sistema. Por ello, se ha considerado $d = 3$ para seleccionar los resultados informados en el cuadro comparativo de la sección 5.1.

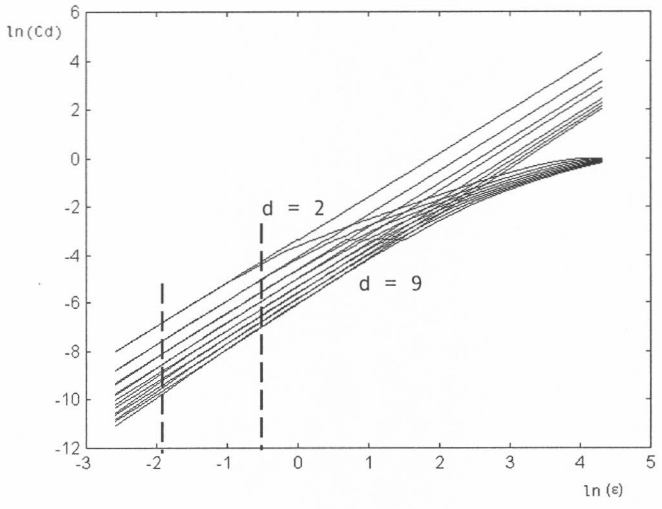


Fig. 5.7 – Sistema de Lorentz - Región de escala obtenida para la Dimensión 3

5.5 – Paralelización

El método anteriormente descrito fue implementado en lenguaje C y en PVM versión 3.4 para la parte de la creación de las tuplas, y en Matlab versión 5.2 para la parte de ventana deslizante, interpolación y distancias entre rectas.

El algoritmo fue ejecutado en una SGI Origin 200 con 4 procesadores, S.O. Irix de 64 bits y PVM 3.4 compilado y optimizado para este tipo de estación de trabajo.

Debido a que hemos paralelizado el proceso de cálculo de las tuplas, y el mismo no depende de las características de los sistemas a analizar sino de la cantidad de puntos tomados en cada muestra, hemos efectuado la medición de tiempo utilizando el sistema Henon.

Los tiempos obtenidos para las ejecuciones serial y paralela se pueden ver en la siguiente tabla:

IMPLEMENTACIÓN	TIEMPO DE EJECUCIÓN
Serial	15 horas 41 minutos
Paralela	1 hora 5 minutos

Observación: En las mediciones realizadas no se tuvo en cuenta la posible carga del sistema, lo que explicaría la gran diferencia en los tiempos de ejecución entre la implementación serial y la paralela. Si consideramos que la máquina paralela cuenta con 4 procesadores, al repartirse la carga entre los mismos, se podría esperar, en promedio, un tiempo de ejecución 4 veces menor.

La figura 5.8 muestra gráficamente la mejora lograda mediante la paralelización.

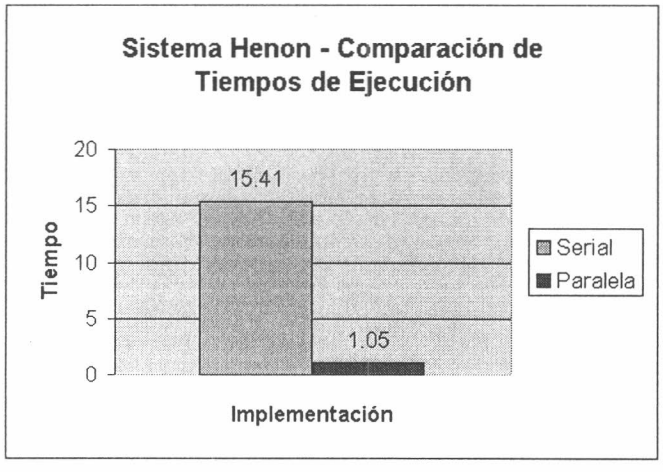


Fig. 5.8 – Sistema Henon - Comparación de Tiempos de Ejecución

o o o

Capítulo 6 – Conclusiones

El método propuesto permite calcular la dimensión de correlación y un número cercano a la entropía de Kolmogorov de un sistema dinámico basado en un conjunto de muestras. Estos parámetros caracterizan un sistema caótico. Paralelamente, calcula la dimensión de correlación a partir del mismo algoritmo. Los resultados pueden ser obtenidos de una gran cantidad de muestras en un tiempo razonable gracias a la paralelización de una de las subtareas del algoritmo. Este método puede ser usado para analizar cualquier conjunto de datos siempre que la serie de tiempo sea lo suficientemente extensa como para que el sistema se encuentre dentro de su comportamiento en estado estacionario.

Hemos tomado 4 sistemas representativos de las distintas características que se pueden presentar y se analizó el comportamiento del algoritmo para cada uno de ellos. En el caso de Henon y Lorentz, pudimos comprobar que los resultados se condicen con los obtenidos por otros autores.

Uno de los pasos fundamentales para la obtención de K_2 es la determinación de una región apropiada (*región de escala*), a partir de la cual se calcula la dimensión de correlación. En este trabajo desarrollamos un método automático para la obtención de la misma, el cual se puede aplicar a todos aquellos sistemas definidos por sus muestras.

Se analizó el algoritmo para identificar la parte que insumía la mayor proporción de uso de CPU. Se determinó que es la correspondiente al cálculo intensivo para contar las ocurrencias del sistema dentro de cada hiperesfera. Por consiguiente, se decidió paralelizar sólo esa parte, que era la que más influía sobre el tiempo total de ejecución.

Se ha observado que los valores considerados para la determinación de la *región de escala* (tamaño de las ventanas, desviación de la pendiente, coeficiente de correlación, etc.)

son muy dependientes entre sí, ya que una pequeña variación en una de ellas influye notoriamente en los resultados. Podría considerarse como el objetivo de una futura investigación la búsqueda de un método para automatizar el cálculo de dichos valores.

Podemos concluir que este método puede ser utilizado para el análisis de sistemas, no necesariamente caóticos, definidos por un conjunto de muestras.

o o o

Apéndice 1 – Algoritmo para el cálculo de la dimensión de correlación y del número K_2

A continuación, se transcribe el código fuente del algoritmo tal como puede ejecutarse en Matlab.

```
%Algoritmo que calcula la Dimensión de Correlación y la Entropía K2 de
%un sistema dinámico a partir de valores de la Integral de Correlación
%para distintos valores de Epsilon.

clear
msg = ' SISTEMA ';

%Variables a considerar para Henon

msg = strcat (msg, ' HENON');
bonpend = 0.05;
bondad = 0.0005;

%epsinc es igual a (Vmax-Vmin) * 0.00125 (Los valores de Henon van de
%-2 a 2)

epsinc = 0.005;
dim = 19;
canteps = 1000;
cantpuntos = 10000 - 50;
nom_arch = 'Eps10kf.txt';

%Variables a considerar para Lorentz

%msg = strcat (msg, ' LORENTZ');
%bonpend = 0.05;
%bondad = 0.0005;

%epsinc es igual a (Vmax-Vmin) * 0.00125 (Los valores de Lorentz van
%de -30 a 30)

%epsinc=0.075;
%dim = 10;
%canteps = 999;
%cantpuntos = 10000 - 50;
%nom_arch = 'DatosLorentzO2f_75.txt';

%Variables a considerar para Seno

%msg = strcat (msg, ' SENO');
%bonpend = 0.05;
%bondad = 0.0005;
```

```
%epsinc=0.0075;
%dim = 12;
%canteps = 999;
%cantpuntos = 640 - 50;
%nom_arch = 'CDM1507.txt';

%Variables a considerar para Random
%msg = strcat (msg, ' RANDOM');
%bonpend = 0.05;
%bondad = 0.0005;
%epsinc=0.0075;
%dim = 16;
%canteps = 999;
%cantpuntos = 640 - 50;
%nom_arch = 'RND1308_2.txt';

%Variables comunes

bondesp= (cantpuntos*cantpuntos)*0.00005;

disp (' -------')
disp (msg)
msg = 'No se hallaron rectas para las dim. ';

%Apertura y lectura de archivo con los datos y armado de la matriz de
%trabajo (b), la cual resulta de dividir los valores por N*N y tomando
%logaritmo.

f = fopen(nom_arch,'rt');
a = fscanf (f, '%f', [dim, canteps]);
valor = a;
format long;
for i=2:dim
    a(i,:) = a(i,:) / (cantpuntos * cantpuntos);
end
for i=1:dim
   for j=1:canteps
      if a(i,j) ~= 0
         b(i,j) = log(a(i,j));
      else
         b(i,j) = a(i,j);
      end
   end
end

%Variables de tamaño de las ventanas.
%El tamaño de las ventanas es el 10 por ciento de la diferencia entre
%el máximo y el mínimo valor que toma toma epsilon.

tamvenlog = b(1,canteps) - b(1,1);
tamven = 0.1 * tamvenlog;

%Variable de la cantidad de ventanas.
```

```
cantven = ceil(tamvenlog/tamven);

%Se arman las ventanas.
%Cada posición i del arreglo invent() contiene el subíndice
%correspondiente a la columna de la matriz donde está almacenado el
%valor de inicio de la ventana i.
%Cada posición i del arreglo finvent() contiene el subíndice
%correspondiente a la columna de la matriz donde está almacenado el
%valor de fin de la ventana i.

invent(1) = 1;
for j = 2:(cantven+1)

%Para obtener el extremo final de cada ventana se suma el tamaño de
%las ventanas al valor de epsilon correspondiente al inicio de la
%misma, ambos en escala logarítmica. Luego, se aplica antilogaritmo,
%se divide el resultado por el incremento de epsilon y se redondea, de
%manera de obtener la posición del extremo final dentro de los puntos
%en la curva d y/o en la matriz.

    finvent(j-1) = ceil(exp(b(1,invent(j-1)) + tamven) * (1/epsinc));
    if finvent(j-1) > canteps
        finvent(j-1) = canteps;
    end
    invent(j) = finvent(j-1);
end
for j = 1:cantven
    inventnew(j) = invent(j);
end

%Ciclo principal que se ejecuta mientras queden dimensiones y ventanas
%a considerar (v < cantven).

for d=2:dim
    v = 1;
    origenrecta(d) = 0;
    finrecta(d) = 0;
    while (v < cantven)

%Verifica si el fin de cada ventana presenta despoblación. Si es así,
%pasa a la siguiente ventana. Si no, verifica despoblación en el
%inicio, y va desplazando el inicio hasta que esté poblado.

        if valor(d, finvent(v)) < bondesp
            inventnew(v) = 0;
            v = v + 1;
        else
            t = invent(v);
            while t < finvent(v)
                if valor(d, t) < bondesp
                    inventnew(v) = 0;
                    t = t + 1;
```

```
                else
                    inventnew(v) = t;
                    t = finvent(v);
                end
            end

%Para cada ventana v, calcula el coeficiente de correlación.
%El vector x almacena los puntos x de la ventana (valores de logaritmo
%de epsilon).
%El vector y almacena los valores correspondientes a Cd para la
%dimensión d.

            x = b(1, inventnew(v):finvent(v));
            y = b(d, inventnew(v):finvent(v));
            sumxy = 0;
            sumx = 0;
            sumy = 0;
            sumx2 = 0;
            sumy2 = 0;
            n = finvent(v)-inventnew(v)+1;
            for i=1:n
                sumxy = sumxy + (x(i)*y(i));
                sumx = sumx + x(i);
                sumy = sumy + y(i);
                sumx2 = sumx2 + (x(i)^2);
                sumy2 = sumy2 + (y(i)^2);
            end
            if (((n * sumx2 - sumx^2) * (n * sumy2 - sumy^2)) == 0)

%Si no se puede calcular el coef. de correlación se asume que no hay
%recta en esta ventana

                r2 = 0;
                cero(d) = 1;
            else
                r2 = (n * sumxy - sumx * sumy)^2 / ((n * sumx2 -
                      sumx^2) * (n * sumy2 - sumy^2));
            end

%Compara el coef. de correlación con la cota establecida.

            if ((1-sqrt(r2)) < bondad)

%Calcula la recta que interpola los puntos en la ventana v y almacena
%la pendiente.

                p = polyfit (x, y, 1);
                if origenrecta(d) == 0

%No se encontró una una recta anteriormente pero sí en esta ventana
%(es la primera vez).

                    origenrecta(d) = inventnew(v);
```

```
                    pendactual = p(1);
                else
                    pendant = pendactual;
                    pendactual = p(1);

%Pregunta si la variación de las pendientes (anterior vs. actual)
%supera un porcentaje de la pendiente anterior (cota).
%Si lo hace, se asume que se terminó la recta porque las pendientes se
%comienzan a desviar.

                    if (abs(pendactual-pendant) > (pendant * bonpend))

%Se encontró la scaling region (ya no hay más rectas) y se sale del
%ciclo while
                        finrecta(d) = finvent(v-1);
                        v = cantven + 1;
                    end
                end
                v = v + 1;
            else

%No hay recta en esta ventana, según el coef. de correl.

                if origenrecta(d) == 0
                    v = v + 1;
                else

%Se encontró la scaling region (ya no hay más rectas) y se sale del
%ciclo while

                    finrecta(d)= finvent(v-1);
                    v = cantven + 1;
                end
            end

%end "if .....< bondesp"
        end

%end while
    end

    if origenrecta(d) == 0
        pend(d) = 0;
        ord(d) = 0;
        dstr = num2str(d);
        norecta(d) = 1;
        nopar(d) = 1;
        dstr = num2str(d);
        msg = strcat (msg, dstr, '-');
    else

%En caso de que haya encontrado recta en todas las ventanas, el valor
%de finrecta tiene que ser el valor de la última ventana.
```

```
        norecta(d) = 0;
        if finrecta(d) == 0
            finrecta(d) = finvent(v-1);
        end

%Se obtiene la recta que interpola los puntos en la scaling region.

        x = b(1, origenrecta(d):finrecta(d));
        y = b(d, origenrecta(d):finrecta(d));
        p = polyfit (x, y, 1);
        pend(d) = p(1);
        ord(d) = p(2);
    end

%end del ciclo principal.
end

%Se comparan las pendientes de las rectas obtenidas a fin de comprobar
%que son aprox. paralelas. Se consideran paralelas si la diferencia
%entre sus pendientes no supera una cierta cota.

bonpar = 0.1;
d = 3;
pendref = pend(2);
sonparalelas = 1;
while (d <= dim)
   if norecta(d) == 0
      if (abs(pend(d)-pendref) > bonpar)
         sonparalelas = 0;
         nopar(d) = 1;
      else
         nopar(d) = 0;
      end
    end
   d = d + 1;
end

if sonparalelas

%Se obtienen las distancias entre las rectas halladas (K2).

    xpinicial = 1;
    for d = 2:(dim-1)
        y2 = polyval ([pend(d), ord(d)], xpinicial);
        bprima = y2 + ((1/pend(d)) * xpinicial);
        xp = (bprima - ord(d+1)) / (pend(d) + 1/pend(d));
        yp = ((pend(d) - 1/pend(d)) * ((bprima -
               ord(d+1))/(pend(d)+1/pend(d))) + ord(d+1) + bprima)/2;
        DistP(d) = sqrt (((xpinicial-xp)*(xpinicial-xp)) + ((y2-
                          yp)*(y2-yp)));
        alfa = atan(pend(d));
        k2(d) = DistP(d) / cos(alfa);
```

```
        xpinicial = xp;
      end
else
   msg2 = 'Las rectas para las dim. ';
   for d = 2:dim
      if nopar(d) == 1
         dstr = num2str(d);
         msg2 = strcat (msg2, dstr, '-');
      end
   end
   msg2 = strcat (msg2, ' no son paralelas');
end
if sonparalelas
   disp (' -------')
   disp (' CAOTICO')
   disp (' -------')
   disp (' Valores de Dim. de Correlación: ')
   disp (' DIM       PENDIENTE')
   for d = 2:dim
      dstr = num2str(d);
      pendstr = num2str(pend(d));
      msg = strcat ('.. ', dstr, '........', pendstr);
      disp (msg)
   end
   disp (' -------')
   disp (' Valores de K2: ')
   disp (' DIM       DISTANCIA')
   for d = 2:dim-1
      dstr = num2str(d);
      diststr = num2str(DistP(d));
      msg = strcat ('.. ', dstr, '........', diststr);
      disp (msg)
   end
else
   disp (' -------')
   disp (' NO CAOTICO')
   disp (' -------')
   disp (msg)
   disp (' -------')
   disp (msg2)
end

%Se realiza el gráfico con los puntos y las rectas interpoladas.

xi = linspace (b(1,1), b(1,canteps), canteps);
for d = 2:dim
   yi(d,:) = polyval([pend(d), ord(d)], xi);
end

%Gráfico de d-curvas y rectas en blanco y negro
%plot (xi, yi(2,:),'k', b(1,:), b(2,:), 'k', xi, yi(3,:), 'k', b(1,:),
b(3,:), 'k', xi, yi(4,:), 'k', b(1,:), b(4,:), 'k', xi, yi(5,:), 'k',
b(1,:), b(5,:), 'k', xi, yi(6,:), 'k', b(1,:), b(6,:), 'k', xi,
```

```
yi(7,:),'k', b(1,:), b(7,:), 'k', xi, yi(8,:), 'k', b(1,:), b(8,:),
'k', xi, yi(9,:), 'k', b(1,:), b(9,:), 'k');

%Gráfico de las d-curvas en blanco y negro
plot (b(1,:), b(2,:),'k', b(1,:), b(3,:),'k', b(1,:), b(4,:),'k',
b(1,:), b(5,:),'k', b(1,:), b(6,:),'k', b(1,:), b(7,:),'k', b(1,:),
b(8,:),'k', b(1,:), b(9,:),'k');

%Gráfico de las rectas
%plot (xi, yi(2,:), xi, yi(3,:), xi, yi(4,:), xi, yi(5,:), xi,
yi(6,:), xi, yi(7,:), xi, yi(8,:), xi, yi(9,:), xi, yi(10,:) );
```

o o o

Apéndice 2 – Algoritmo paralelizado para el cálculo de la correlación integral

A2.1 - Proceso Maestro

```
#define EPSFINAL 5
#define EPSINC 0.005
#define CANTEPS 1000
#define N 10000
#define NPROC 4
#define D 20
#define H 5
#define SLAVENAME "parslave"

#include <fcntl.h>
#include <malloc.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <curses.h>
#include <stdio.h>
#include "/home/pvm/pvm3/include/pvm3.h"

int   main(int argc, char* argv[]) {

/* Definicion de variables */
      FILE *fp;
      int i, j, k, numt, msgtag;
      int n, proc, l, d, nproc, dimmax, h, canteps;
      int who, msgtype, info;
      int tids[32], mytid;
      int comparevec[H][CANTEPS], comparevecfinal[H*NPROC][CANTEPS];
      double y[N], x[N];
      double epsinc, epsfinal;
      long curtime;
      extern long GetCurTime();

/* Asignacion de constantes y apertura de archivos */
      nproc = NPROC;
      dimmax = D;
      epsinc = EPSINC;
      epsfinal = EPSFINAL;
      canteps = CANTEPS;
      h = H;
      n = N;
      fp = fopen("parh10k.txt", "w");

/* Calculo de datos segun HENON  */
      x[0] = 0;
```

```
      y[0] = 0;
      for ( j= 1; j < n; j++)
      {           x[j] = 1.4 - x[j-1]*x[j-1] + 0.3 * y[j-1];
                  y[j] = x[j-1];
                  fprintf (fp,"%f \n",x[j]);    }

/* enroll in pvm */
     mytid = pvm_mytid();

/* Lanzar procesos esclavos paralelos que calculan para distintas d's
*/
    numt=pvm_spawn(SLAVENAME, (char**)0, 0, "", nproc, tids);
    printf("NUMT es %d\n", numt);
    if( numt < nproc )
    { printf("\n Trouble spawning slaves. Aborting. Error codes
                  are:\n");
      for ( i=numt ; i<nproc ; i++ )
          printf("TID %d %d\n",i,tids[i]);
      for ( i=0 ; i<numt ; i++ )
          pvm_kill( tids[i] );
      pvm_exit();
      return(1);  }
    printf("SUCCESSFUL\n");

/* Begin User Program */

      curtime = GetCurTime();

/* Envio de datos a los procesos esclavos */
        msgtag = 3;
       for (i=0; i<nproc;i++)
      { pvm_initsend(PvmDataDefault);
        pvm_pkint(&nproc, 1, 1);
        pvm_pkint(&i, 1, 1);
        pvm_pkint(&n, 1, 1);
        pvm_pkint(&dimmax, 1, 1);
        pvm_pkint(&h, 1, 1);
        pvm_pkint(&canteps, 1, 1);
        pvm_pkdouble(&epsinc, 1, 1);
        pvm_pkdouble(&epsfinal, 1, 1);
        pvm_pkdouble(&x[0], n, 1);
        info = pvm_send(tids[i], msgtag);
        if( info == 0 )
            printf("\n Spawn exitoso par10k20\n");
              else
            printf("\n Spawn Error\n");
      }
          for (i=0; i<nproc;i++)
            printf(" %x", tids[i]);

/* Recibe de slaves */
    msgtype = 5;
    for( i=0 ; i<nproc ; i++ )
```

```
    { pvm_recv( -1, msgtype );
      pvm_upkint( &comparevec[0][0], (h*canteps), 1 );
      pvm_upkint( &proc, 1, 1 );
      pvm_upkint( &who, 1, 1 );
      printf("Recibi Proceso%d\n", proc);
      printf("Recibi Who%d\n", who);
      for( j=0 ; j<h ; j++ )
      {     l =  ((proc + 2) % nproc) + (j*nproc);
            for( k=0 ; k<canteps ; k++ )
               comparevecfinal[l][k] = comparevec[j][k];
      }
   }

/* Termina el programa, se graba archivo de salida y fin de PVM */
      printf( "Termino Pvm");
      for (d=2; d<dimmax; d++)
      {   for ( i= 1; i <= canteps; i++)
         {  if (comparevecfinal[d-2][i-1] > 0)
             fprintf (fp,"D %d LE %d C %d\n", d, i, comparevecfinal[d-
                        2][i-1]);
         }
      }
            curtime = GetCurTime() - curtime;
      fprintf (fp, "TIEMPO %ld ", curtime);
      fclose(fp);
      pvm_exit();
      return(0);
}
```

A2.2 - Proceso Esclavo

```
#define CANTX 10000
#define H 5
#define CANTEPS 1000

#include <stdio.h>
#include "/home/pvm/pvm3/include/pvm3.h"
#include <math.h>
#include <stdlib.h>

main()
{
    int mytid;
    int  i, j, master, msgtype;
    int  CantD, D, h, proc, nproc, d, m, n, N, canteps;
    double  eps, epsfinal, epsinc;
    double x[CANTX], distance;
    int comparevec[H][CANTEPS];

    /* enroll in pvm */
```

```
  mytid = pvm_mytid();

/* Receive data from master */
msgtype = 3;
pvm_recv( -1, msgtype );
pvm_upkint(&nproc, 1, 1);
pvm_upkint(&proc, 1, 1);
pvm_upkint(&N, 1, 1);
pvm_upkint(&D, 1, 1);
pvm_upkint(&h, 1, 1);
pvm_upkint(&canteps, 1, 1);
pvm_upkdouble  (&epsinc, 1, 1);
pvm_upkdouble  (&epsfinal, 1, 1);
pvm_upkdouble (&x[0], N, 1);

/* Compute Correlation Integral  */
  for (j=0; j < h; j++)
  {     for (i=0; i< canteps; i++)
              comparevec[j][i] = 0;
  }
j=0;
for ( d =2 ; d < D; d++)
{
  if ((d % nproc)== proc)
  {
  for ( n = 50; n < (N-2*d+1); n++)
  {
        for ( m = n+1 ; m < (N-1); m++)
        {
              distance = 0;
              for ( i=0; i <= (d-1); i++)
                   distance = distance+(x[n+i]-x[m+i])*(x[n+i]-
                                        x[m+i]);
              distance = sqrt(distance);
              i = 0;
              for ( eps=epsinc; eps <= epsfinal; eps=eps+epsinc)
              {
                 if ( distance <= eps)
                     comparevec[j][i] = comparevec[j][i]+2.0;
                 i = i+1;
              }
        }

  }
  j = j+1;
  }
}

/* Send result to master */
  pvm_initsend( PvmDataDefault );
  pvm_pkint( &comparevec[0][0], (h*canteps), 1 );
  pvm_pkint( &proc, 1, 1 );
  pvm_pkint( &mytid, 1, 1 );
```

```
    msgtype = 5;
    master = pvm_parent();
    pvm_send( master, msgtype );

    /* Program finished. Exit PVM before stopping */
    pvm_exit();
}
```

o o o

Bibliografía

[1] T.Parker and L.Chua, "Chaos: A Tutorial for Engineers", Proc.IEEE, Vol.75, No.8, pp.982-1008, 1987.

[2] P.Grassberger and I.Procaccia, "Characterization of Strange Attractors", Phys. Rev. Lett. 50, Vol.5, pp.346-349, 1983.

[3] J.Eckermann, S.Oliffson, D.Ruelle and S.Ciliberto, "Lyapunov exponents from time series", Phys. Rev. A, Vol.34, No.6, pp.4971-4979, 1986.

[4] P.Grassberger and I.Procaccia, "Estimation of the Kolmogorov entropy from a chaotic signal", Phys. Rev. A, Vol.28, No.6, pp .2591-2593, 1983.

[5] S.Chirravuri, S.M.Bhandarkar and D.Whitmire, "A Massively Parallel Algorithm for K_2 Entropy Computation: Case Studies of Model Systems and *In Vivo* Data", Intl. Journal of Supercomputer Appl. and High Perf. Computing, Vol.9, No.4, pp.296-311, 1995.

[6] J.Theiler, "Efficient algorithm for estimating the correlation dimension from a set of discrete points", Phys Rev. A, Vol.36, No.9, pp.4456-4462, 1987.

[7] M.Henon, Comm. Math. Phys. 50, 69, 1976.

[8] S.M.Bhandarkar, S.Chirravuri and S.Machaka, "Biomedical Applications Modeling", in High Performance Cluster Computing", Vol.2 Programming and Applications, R.Buyya Ed, Prentice-Hall Inc., Upper Saddle River, NJ, Ch.29, pp.604-624, 1999.

[9] G.A.Geist, A.Baguelin, J.Dongarra, W.Jiang, R.Manchek and V.Sunderam, "PVM: A Users' Guide and Tutorial for Networked Parallel Computing", MIT Press, 1994.

[10] V.S.Sunderam, G.A.Geist, J.Dongarra and R.Manchek., "The PVM Concurrent Computing System: Evolution, Experiences, and Trends", 1993.

[11] A.Ceriani y D.Bertaccini, "Análisis de la Paralelización de Dos Tipos de Algoritmos usando PVM", Universidad de Buenos Aires, Tesis de Licenciatura, Febrero 2001.

[12] M.Ding, C.Gregobi, E.Ott, T.Sauer and J.A.Yorke, "Plateau Onset for Correlation Dimension: When Does it Occur?", Phys. Rev. Lett. 70, Vol.25, pp.3872-3875, 1993.

[13] A.I.Mess, P.E.Rapp and L.S.Jennings, "Singular-value decomposition and embedding dimension", Phys. Rev. A, Vol.36, No.1, pp.340-346, 1987.

[14] A.M.Albano, J.Muench, C.Schwartz, A.I.Mess and P.E.Rapp, "Singular-value decomposition and the Grassberger-Procaccia algorithm", Phys. Rev. A, Vol.38, No.6, pp.3017-3026, 1988.

[15] A.Brandstater, J.Swift, H.L.Swinney, A.Wolf, J.D.Framer, E.Jen and P.J.Crutchfield, "Low-Dimensional Chaos in a Hydrodynamic System", Phys. Rev. Lett. 51, Vol.16, pp.1442-1445, 1983.

[16] M.B.Kennel, R.Brown and H.D.I.Abarbanel, "Determining Embedding Dimension for Phase Space Reconstruction Using the Method of False Nearest Neighbors", University of California, San Diego, 1992.

[17] N.Carriero and D.Gelernter, "How to Write Parallel Programs: A Guide to the Perplexed", ACM Computing Surveys, Vol.21, No.3, pp.323-358, 1989.

[18] T.S.Parker and L.O.Chua, "Practical Numerical Algorithms for Chaotic Systems", 1986

[19] E.N.Lorentz, J.Atmos, Sci.20, 130, 1963

[20] R.Devaney, "An Introduction to Chaotic Dynamical Systems", Addison-Wesley Publishing Company, Inc., 1987

[21] S.Ghorui, A.K.Das and N.Venkatramani, "A simpler and Elegant Algorithm for Computing Fractal Dimension in Higher Dimensional State Space", Pramana, Vol.54,No.2, pp.L331-L336, 2000.

o o o