Ministerio de Educación Superior
Universidad Central "Marta Abreu" de Las Villas
Facultad de Matemática, Física y Computación

Trabajo en opción por el titulo académico de
Master en Computación Aplicada

"Desarrollo e Implementación de Algoritmos Paralelos aplicados a los conceptos básicos de la Teoría de los Conjuntos Aproximados"

AUTOR

Ing. Leonardo Flavio Del Toro Melgarejo

TUTORES

Dr. Daniel Galvez Lio
Dr. Rafael Bello Perez

2006
"Año de la Revolución Energética en Cuba"

INTRODUCCIÓN

En los últimos años del siglo XX y los inicios del XXI varias ramas de las ciencias han alcanzando un dinámico desarrollo. Una de ellas, la Inteligencia Artificial, experimentó un rápido desarrollo desde la década de 1990 y aunque en su núcleo tradicional comprende el desarrollo de diversas formas de representación del conocimiento, los métodos de solución de problemas, los sistemas basados en el conocimiento, entre otros, en la actualidad se han desarrollado nuevos campos que han potenciado la aplicabilidad de esta ciencia. Entre ellos encontramos las redes neuronales artificiales, los algoritmos genéticos, los sistemas borrosos (Fuzzy Systems) y la teoría de los conjuntos aproximados (Rough Sets Theory) a los cuales algunos autores le denominan computación blanda (Soft Computing).

Cuando los sistemas de Inteligencia Artificial (IA) son aplicados a situaciones más cercanas a la realidad es necesario que ellos extraigan conclusiones (realicen inferencia) que estén menos estrechamente basados en la información conocida. Es decir, es necesario disponer de mecanismos de inferencia que permitan extraer conclusiones sugeridas pero no aseguradas (garantizadas) por sus premisas. Debido a ello resulta importante disponer de modelos computacionales con los cuales se puedan realizar inferencias considerando la problemática de la incertidumbre inherente a la información.

Uno de estos modelos lo constituye la Teoría de Conjuntos Aproximados, la que ha sido ampliamente estudiada como una técnica de descubrimiento del conocimiento. Esta teoría fue introducida por Z. Pawlak en 1982 [1] y se basa en aproximar cualquier concepto, un subconjunto del dominio, una clase, en un problema de clasificación, por un par de conjuntos “exactos”, llamados ***aproximación inferior*** y ***aproximación superior*** del concepto. El principal

objetivo del análisis de conjuntos aproximados es sintetizar aproximaciones de conceptos a partir de los datos adquiridos [2].

Los principales problemas que se pueden enfocar usando la teoría de los conjuntos aproximados incluyen reducción de datos, descubrimiento de dependencias entre datos, estimación de la significación de los datos, generación de algoritmos de decisión o control a partir de datos, clasificación aproximada de datos, descubrimiento de similaridades o diferencias en los datos, descubrimiento de patrones y descubrimiento de relaciones de causa-efecto.

Sobre la importancia que se le concede a la misma varios autores han manifestado criterios muy precisos. Por ejemplo, Lotfi A. Zadeh, de la Universidad de Berkeley, EUA, manifestó que “Los conjuntos aproximados abrieron una nueva dirección en el desarrollo de teorías sobre la información incompleta”… “Hoy la teoría de los conjuntos aproximados ha evolucionado hasta convertirse en una metodología para enfrentar una amplia variedad de problemáticas, entre ellas la incertidumbre motivada por información incompleta o imprecisa” [3]. Sankar K. Pal, de la Universidad de Varsovia, Polonia, planteó que “La teoría de los conjuntos aproximados ha emergido como otro importante enfoque matemático para el manejo de la incertidumbre cuando ésta surge de información inexacta, con ruidos o incompleta” [2]. Así, Jerzy W. Grzymala-Busse, de la Universidad de Kansas, EUA, manifestó que “La teoría de conjuntos aproximados es la mejor herramienta para modelar la incertidumbre cuando ésta se manifiesta como inconsistencia” [4] y Ron Kohavi y Brian Frasca, de la Universidad de Stanford, EUA, que “Hemos demostrado que seleccionando un subconjunto útil de rasgos para la relación de inseparabilidad [de la teoría de los conjuntos aproximados] un algoritmo de inducción basado en una simple tabla de decisión puede tener una alta precisión sobre bases de datos reales o artificiales” [6].

Mas reciente, Salvatore Greco, de la Universidad de Catania, Italia, expresó que "Esta teoría ha mostrado ser una excelente herramienta matemática para el análisis de datos" [5], W. Koczkodaj, de la Universidad de Laurentian, Canada, M. Orlowski, Universidad Tecnológica, Australia, y V. Marek, Universidad de Kentucky, EUA, que "La teoría de los conjuntos aproximados es, potencialmente, una importante herramienta en el análisis de datos con importantes aplicaciones en la minería de datos y el descubrimiento de conocimiento" [7]. Así como, Sankar K. Pal, del Instituto Estadístico de la India, expresó que "El uso de la teoría de los conjuntos aproximados para el descubrimiento de conocimiento y minar reglas es ampliamente reconocido", [8].

En particular, los conjuntos aproximados han tenido una interesante aplicación en la medicina, negocios, diseño de ingeniería, meteorología, análisis de vibraciones, análisis de conflictos, procesamiento de imágenes, reconocimiento de la voz, reconocimiento de caracteres, análisis de decisión, estudio del medio ambiente y bioinformática. Se reportan recientes aplicaciones a la industria y las investigaciones de mercado que demuestran la potencia de los conjuntos aproximados [9], [11-12], [14-16] y [5].

Teniéndola como base teórica se han desarrollado sistemas conocidos, como lo son el software para el descubrimiento de reglas LERS (Learning from Examples using Rough Sets) [17-21], creado en la Universidad de Kansas; y el sistema de análisis de datos Rosetta [22-25], implementado en la Universidad Noruega de Ciencia y Tecnología. Ambos sistemas fueron elaborados para computadoras personales empleando para ello algoritmos secuenciales.

Cualquier modelo computacional que base su funcionamiento en esta teoría debe tener en cuenta la complejidad algorítmica de los procedimientos que ella implica.

Hoy en día es común el uso de grandes bases de datos, donde la cantidad de objetos se ha incrementado. El costo computacional de los sistemas que empleen los procedimientos de esta teoría se ve afectado por un factor cuadrático [26-27] dependiente de la cantidad de objetos a procesar, los que al procesar mayores volúmenes de información exigen requerimientos superiores tanto del software como del hardware de una computadora.

De ello se deriva la posibilidad de reducir este inconveniente mediante el empleo de nuevas técnicas de programación. Una de ellas, la cual es el objeto del presente estudio, la constituyen las técnicas paralelas, también conocida por computación paralela. Cada vez más, el proceso paralelo es visto como el único método rentable para lograr soluciones rápidas a los problemas que involucra el procesamiento de grandes volúmenes de datos. El surgimiento de computadoras paralelas baratas con multiprocesamiento (más de un procesador) así como la expansión de los clusters de computadoras ha permitido generalizar la aplicación de estos métodos paralelos, convirtiéndose en el estándar de software para estas plataformas, lo cual propiciará un incremento substancial del software paralelo en este siglo.

Las aplicaciones intensivas de datos, tales como el procesamiento de transacciones, recuperación de información, minería de datos y los servicios multimedia han proporcionado un nuevo desafío a la actual generación de plataformas paralelas. El surgimiento de nuevas áreas como la bioinformática y la nanotecnología tienen implicaciones para los algoritmos y el desarrollo de sistemas, mientras los cambios en las arquitecturas, programando modelos y aplicaciones, tienen implicaciones para hacer a las plataformas paralelas más disponibles a los usuarios en la forma de servicios basados en “grid” (malla).

Si bien desde hace una década atrás los modelos de programación se basaban en interfaces de aplicación de programas (API) personalizados para la mensajería y el paralelismo basado en lazos, los modelos actuales estandarizan

estas API a través de las diferentes plataformas. Las bibliotecas de paso de mensajes como PVM y MPI, las bibliotecas de hilos, similares a los hilos de POSIX, y los modelos basados en directivas como OpenMP son ampliamente aceptados como estándares y se han hecho portables a una variedad de plataformas [28].

El problema de la implementación paralela de métodos de la Inteligencia artificial para ganar eficiencia en su implementación ha sido ya explorado como se ilustra en los trabajos [29-33]. La implementación paralela de estos métodos usualmente obliga a una reformulación de los algoritmos, para lo cual se pueden explorar diversas alternativas de paralelización. Si consideramos la importancia que se le atribuye a la Teoría de los Conjuntos Aproximados y las ventajas que proporcionan las técnicas paralelas de programación se pudieran desarrollar herramientas para clasificación aproximada de datos que empleen algoritmos paralelos.

Problema de Investigación:

¿Será posible obtener un algoritmo paralelo para implementar los principales conceptos de la teoría de los conjuntos aproximados que mejore el rendimiento de versiones secuenciales?

Hipótesis:

Sobre la teoría de los Conjuntos Aproximados (Rough Sets) se han desarrollado algoritmos secuenciales que han implementado los principales conceptos de esta teoría [17-25]. Cuando estos Sistemas de Información son muy robustos el costo computacional de las implementaciones secuenciales se eleva. Dadas estas circunstancias es posible emplear las ventajas que brindan las técnicas paralelas, las cuales permiten dividir el problema en sub-problemas de menor complejidad.

Si aplicamos las técnicas de paralelización a la teoría de los Conjuntos Aproximados, podríamos obtener implementaciones paralelas que mejoren el rendimiento de las aplicaciones secuenciales.

Objetivo general:

Aplicar técnicas de paralelización al algoritmo secuencial que permite implementar los conceptos básicos de la teoría de los Conjuntos Aproximados.

Objetivos Específicos:

1. Analizar las bondades que ofrecen las técnicas paralelas y las ventajas que brinda un clusters de computadoras para desarrollar aplicaciones con estas características.
2. Determinar las etapas donde es posible aplicar técnicas de paralelización.
3. Proponer diferentes alternativas de algoritmos paralelos para obtener los principales conceptos de esta teoría.
4. Evaluar las diferentes alternativas de algoritmos paralelos.

Valor Científico

Se proponen alternativas de algoritmos paralelos, para la obtención de los principales conceptos de la teoría de los Conjuntos Aproximados.
Obtención de una aplicación sobre un cluster de computadoras que implemente los conceptos básicos de la teoría de los Conjuntos Aproximados.

Valor Práctico:

Se brinda una herramienta práctica paralela que permite medir el grado de consistencia de un Sistema de Información, presentando rendimientos superiores a su equivalente secuencial.

Para cumplimentar con los objetivos trazados, la presente tesis de maestría está dividida en 4 capítulos.

En el primero se introducen los principales conceptos de la Teoría de los Conjuntos Aproximados estableciendo un grupo de medidas para realizar la inferencia.

En el segundo capítulo se trataran aspectos relacionados con la modelación analítica de aplicaciones paralelas, así como las métricas de rendimiento para un sistema con tales características.

.

En el tercer capítulo se presentan los algoritmos paralelos diseñados para los principales conceptos de la Teoría de los Conjuntos Aproximados, obteniendo las expresiones de sus complejidades computacionales, así como algunas métricas de rendimiento que caracterizan a un sistema paralelo.

En el último capítulo se presentan los resultados experimentales que sobre determinado sistema de información brindan las aplicaciones paralelas basadas en los algoritmos presentados con anterioridad realizando comparaciones con su similar secuencial.

CAPÍTULO I – Elementos Básicos de la Teoría de los Conjuntos Aproximados

En 1982 el profesor Zdzislaw Pawlak publicó un artículo en el cual presentó los conjuntos aproximados inferior y superiormente (rough sets) [1], abriendo una nueva dirección en el desarrollo de teorías sobre la información incompleta.

Actualmente la teoría de los conjuntos aproximados ha evolucionado hasta convertirse en una metodología para enfrentar una amplia variedad de problemáticas, entre ellas la incertidumbre motivada por información incompleta o imprecisa, Lotfi A. Zadeh en [3]. En particular para enfrentar situaciones de incertidumbre motivadas por inconsistencias.

Según lo expresado por Grzymala-Busse en [4], la teoría de conjuntos aproximados es la mejor herramienta para modelar la incertidumbre cuando ésta se manifiesta como inconsistencia. A la vez que se relaciona con la teoría de los conjuntos borrosos (fuzzy sets) y la teoría de evidencia en el sentido de que todas tienen que ver con los problemas de vaguedad e incertidumbre [34], y [35].

1.1) La teoría de conjuntos aproximados, inferior y superiormente.

La filosofía de los conjuntos aproximados se basa en la suposición de que todo objeto de un universo U tiene asociado una cierta cantidad de información (datos y conocimiento) expresada por medio de algunos atributos que permiten describirlo.

El conocimiento sobre el modelo se expresa mediante un Sistema de Decisión (definido por un atributo de decisión). Un sistema como este puede presentar inconvenientes basado en la redundancia presente en al menos dos formas; primero, el mismo objeto u objetos idénticos (según alguna relación) pueden aparecer varias veces o, segundo, algunos de los atributos que describen los objetos del universo son superfluos.

Informalmente se pueden definir los conjuntos aproximados de la forma siguiente:

Los objetos que tienen la misma descripción son inseparables (similares) con respecto al conjunto de atributos considerados. Esta relación de inseparabilidad constituye la base matemática de la teoría, y la misma induce una partición del universo U en bloques de objetos inseparables. Cualquier subconjunto X de este universo U se puede expresar en términos de estos bloques de forma exacta o aproximada. En el último caso el conjunto X se puede caracterizar por dos conjuntos ordinarios denominados *aproximación inferior* y *aproximación superior*.

La aproximación inferior de X está formada por todos los bloques que son subconjuntos de X, y la aproximación superior consiste de todos los bloques que tienen una intersección no vacía con X. A la aproximación inferior de X pertenecen los objetos que con seguridad (certeza) pertenecen a X, y a la aproximación superior pertenecen los objetos que pudieran pertenecer a X.

En el conjunto diferencia de ambos conjuntos están los objetos sobre los cuales no se puede determinar con certeza su pertenencia a X. La cardinalidad de este conjunto diferencia se puede usar como una medida de la vaguedad de la información sobre X.

1.2) Definiciones sobre un Sistema de Información.

Diversos modelos computacionales operan sobre colecciones de datos, en cada caso esta colección tiene sus características, sobre todo organizativas, y recibe una denominación particular. En el caso de la Teoría de los conjuntos aproximados la estructura de información básica es el Sistema de información.

Definición 1.1: (Sistema de Información)

Sea un conjunto de atributos $A = \{A_1, A_2, ..., A_n\}$ y un conjunto U no vacío llamado universo de ejemplos (objetos, entidades, situaciones o estados, etc.) descritos usando los atributos A_i. Al par (U, A) se le denomina *Sistema de información*.

Suponga que se desea determinar el estado general de salud de una población compuesta de 6 pacientes. En otras palabras, se desea clasificar si los pacientes

están enfermos con Gripe o no. Para ello, un experto en medicina puede llegar a una conclusión basado en un grupo de síntomas conocidos, los cuales pueden indicarse por la presencia de: Dolor de cabeza, Dolor muscular y Temperatura corporal. Esta situación puede describirse por medio de un Sistema de Información como el descrito en la Tabla 1.1 donde su universo (U) está formado por seis objetos (pacientes: P1 a P6), los cuales se describen por el conjunto de atributos:

A = { (A_1) Dolor de cabeza, (A_2) Dolor muscular, (A_3) Temperatura, (A_4) Gripe }

Paciente	***Dolor de cabeza***	***Dolor muscular***	***Temperatura***	***Gripe***
P1	no	si	alta	**Si**
P2	si	no	alta	**Si**
P3	si	si	muy alta	**Si**
P4	no	si	normal	**No**
P5	si	no	alta	**No**
P6	no	si	muy alta	**Si**

Tabla 1.1: Ejemplo de Sistema de Información.

Si a cada atributo A_i se le asocia un dominio V_i, se puede construir una función f: U x A $\rightarrow$ V, tal que f (x, A_i) $\in$ V_i para cada $A_i \in A$, $x \in U$, a la que se le denomina función de información.

Los dominios de los atributos (V_1:A_1,…, V_4:A_4) del sistema de información de la Tabla 1.1 son V_1={no, si}, V_2={no, si}, V_3={normal, alta, muy alta}, y V_4={No, Si}.

Se dice que un atributo $A_i \in A$ *separa* o distingue un objeto "x" de otro "y" si y solo si:

Separa (A_i, x, y) $\Leftrightarrow$ f(x, A_i) $\neq$ f(y, A_i) (1.1)

Por ejemplo el atributo *Dolor de cabeza* separa los objetos P1 y P2, pero no separa los objetos P1 y P4. En este ultimo caso se dice que **x** e **y** son inseparables respecto al atributo *Dolor de cabeza*.

Definición 1.2: (Sistema de decisión)

Si a cada elemento de U se le agrega un nuevo atributo **d** llamado decisión indicando la decisión tomada en ese estado o situación, entonces se obtiene un *Sistema de decisión* (U, $A \cup \{d\}$, donde $d \notin A$).

El valor de la decisión puede representar un número de la clase en la cual clasificar el objeto, mientras que en un sistema de control la decisión significa una acción que se debe ejecutar en el estado descrito por el objeto, entre otras alternativas de interpretación. La Tabla 1.1 es un ejemplo de Sistema de decisión, si consideramos al atributo ***Gripe*** como *atributo de decisión*.

El atributo de decisión **d** induce una partición del universo de objetos U. Sea V_d el conjunto de enteros {1,...m}, entonces $\{X_1,...,X_m\}$ es una colección de clases de equivalencias, llamadas clases de decisión, donde dos objetos pertenecen a la misma clase si ellos tienen el mismo valor para el atributo decisión.

$$X_i = \{x \in U : d(x)=i\} \quad (1.2)$$

Para el sistema de información de la tabla 1.1 las clases de decisión son:

{ {P1, P2, P3, P6} , {P4, P5} **}**.

Gripe = Si Gripe = No

Usualmente son estas clases de decisión las que se quieren analizar como conjuntos aproximados.

Definición 1.3: (Consistencia del sistema de decisión)

Cuando dos objetos inseparables, según un subconjunto de atributos **B**, pertenecen a clases de decisión diferentes se dice que el Sistema de decisión es *inconsistente*, en otro caso es *consistente*.

En el sistema de información de la Tabla 1.1 para *B={Dolor de cabeza, Dolor muscular}*, P1 y P4 son inseparables, pero P1 pertenece a la clase Gripe=Si, y P4 pertenece a la clase Gripe=No, lo cual indica que este sistema de información es *inconsistente* para la relación establecida por ese conjunto *B*.

Se define entonces la función $\delta_B(x)$, la cual da la relación de clases de decisión a las cuales pertenecen los objetos inseparables del objeto x según B ($I_B(x)$):

$$\delta_B(x) = \{v \in V_d : \exists y \in I_B(x) \text{ tales que } d(y)=v \} \quad (1.3)$$

Nótese que un sistema de decisión es consistente, si y solo si, el conjunto $\delta_B(x)$ es un conjunto unario para todo $x \in U$.

Por ejemplo para B={Dolor de cabeza, Dolor muscular}, y el objeto P1 se tiene que:

$\delta_B(P1)$={Si, No}, lo cual implica que $|\delta_B(P1)|>1$ y por tanto este sistema de decisión es inconsistente.

Los sistemas de decisión pueden construirse con más de un atributo de decisión, es decir, se tiene el conjunto de atributos de decisión D. En este caso el sistema se define por (U, $A \cup D$}.

1.3) Definiendo los conjuntos aproximados superior e inferior.

La noción de inseparabilidad (indiscernibility) es fundamental en la teoría de los conjuntos aproximados, es el objeto matemático de partida de esta teoría [3], y [36]. Los objetos que son caracterizados por la misma información son inseparables desde la perspectiva de la información disponible. Informalmente, dos objetos en una tabla de decisión son inseparables si no se puede distinguir entre ellos sobre la base de un conjunto de atributos dados. Por eso, la inseparabilidad es una función del conjunto de atributos bajo consideración.

Definición 1.4: (Relación de inseparabilidad):

A cada subconjunto de atributos B de A ($B \subseteq A$) está asociada una relación binaria de inseparabilidad denotada por I_B, la cual es el conjunto de pares de objetos que son inseparables uno de otro por esa relación.

$$I_B = \{ (x,y) \in UxU : f(x,A_i) = f(y,A_i) \text{ para todo } A_i \in B\} \quad (1.4)$$

Por ejemplo para *B={Dolor de cabeza, Dolor muscular}*, y el sistema de información de la tabla 1.1 se tiene I_B= { (P1,P4), (P1,P6), (P4,P6), (P2,P5) }.

Si (x ,y) $\in I_B$ se dice que los objetos *x* e *y* son inseparables según B (B-inseparables).

Una relación de inseparabilidad (indiscernibility relation) que sea definida a partir de formar subconjuntos de elementos de U que tienen igual valor para un subconjunto de atributos B de A ($B \subseteq A$) es una relación de equivalencia. Es decir, es una relación binaria $R \subseteq UxU$ que es reflexiva, simétrica y transitiva.

La clase de equivalencia de un elemento *x* de U es el conjunto de todos los elementos *y* de U tal que xRy; es decir, los elementos de U que son similares a x considerando los atributos contenidos en B. Una relación de equivalencia induce una partición de el universo U. Por B(x) se denota al conjunto de todos los $y \in U$ tal que yRx; es decir, el elemento de la partición al que pertenece el objeto x.

Por ejemplo para *B={Dolor de cabeza, Dolor muscular}*, y el sistema de información de la tabla 1.1 se tiene:

B(P1)= { P1, P4, P6}

B(P2)= { P2, P5 }

Sea el sistema de información (U, A), y los conjuntos $B \subseteq A$ y $X \subseteq U$. Se puede aproximar X usando solamente la información contenida en B construyendo dos conjuntos llamados aproximación inferior (B_*) y aproximación superior (B*) respectivamente del conjunto X para la relación B. Un conjunto aproximado es cualquier subconjunto $X \subseteq U$ definido a través de sus aproximaciones inferior y superior.

Definición 1.5: (Aproximaciones de un conjunto)

La aproximación inferior de un conjunto (con respecto a un conjunto dado de atributos) se define como la colección de objetos cuyas clases de equivalencia están contenidas completamente en el conjunto; mientras que la aproximación superior se define como la colección de objetos cuyas clases de equivalencia están al menos parcialmente contenidas en el conjunto. Formalmente,

$B_*(X)= \{x \in U \mid B(x) \subseteq X\}$ (1.5)

$B^*(X)= \{x \in U \mid B(x) \cap X \neq \phi\}$ (1.6)

Los elementos de $B_*(X)$ son todos y solamente aquellos objetos del universo U que pertenecen a las clases de equivalencia generadas por la relación I_B contenidas en X; mientras que los elementos de B*(X) son todos y solamente aquellos objetos de U que pertenecen a las clases de equivalencia generadas

por la relación de inseparabilidad conteniendo al menos un objeto x perteneciente a X. A partir de ellos se define la región límite de X para la relación de equivalencia B:

Definición 1.6: (Región límite)

$BN_B(X) = B^*(X) - B_*(X)$ (1.7)

Si el conjunto BN_B es vacío entonces el conjunto X es exacto respecto a la relación de equivalencia B. En caso contrario, $BN_B(X) \neq \phi$, el conjunto X es inexacto o aproximado con respecto a B. El principal objetivo de esta teoría es la síntesis de aproximaciones para los conceptos.

1.4) Medidas para realizar inferencias.

Sobre la base del conocimiento en B los objetos que pertenecen a $B_*(X)$ pueden ser clasificados con certeza como miembros de X, los objetos miembros de $B^*(X)$ pueden ser solamente clasificados como posibles miembros de X. Los elementos en BN_B no pueden ser clasificados con absoluta certeza como miembros del conjunto X. Los elementos del conjunto U-$B^*(X)$ con certeza no pertenecen a X.

Por eso, los conjuntos aproximados pueden ser vistos como un modelo matemático de los conceptos vagos, es decir, aquellos en los que los elementos del universo no pueden ser clasificados con certeza como elementos o no del concepto. Las principales medidas construidas a partir de los conjuntos aproximados son las siguientes.

Definición 1.7: (Precisión de la aproximación)

La *vaguedad* del concepto o conjunto X, también llamada *precisión de la aproximación* (*accuracy*), con respecto a la relación B se puede caracterizar matemáticamente por el coeficiente:

$$\alpha_B(X) = \frac{|B_*(X)|}{|B^*(X)|} \qquad (1.8)$$

Esta magnitud mide el grado de perfección o integridad del conocimiento sobre el conjunto X considerando los atributos incluidos en la relación de equivalencia (B) donde $|X|$ denota la cardinalidad del conjunto X, y $0 \leq \alpha_B(X) \leq 1$.

Si $\alpha_B(X)=1$, el conjunto X será *duro* o *exacto* con respecto a la relación de equivalencia B, mientras que si $\alpha_B(X)<1$, el conjunto X es *aproximado* o *vago* con respecto a B.

Definición 1.8: (Calidad de la aproximación)

La calidad de la aproximación de X por medio de los atributos en B se define por:

$$\gamma_B(X) = \frac{|B_*(X)|}{|X|} \qquad (1.9)$$

Esta medida de calidad representa la frecuencia relativa de los objetos correctamente clasificados según B, donde $0 \leq \alpha_B(X) \leq \gamma_B(X) \leq 1$.

Definición 1.9: (Calidad de la clasificación)

Una generalización de esta medida para medir la calidad de la aproximación del Sistema de Información según la relación B se define por:

$$\gamma_B(Y) = \frac{\sum_{i=1}^{n} |B_* Yi|}{|U|} \qquad (1.10)$$

y se le denomina *calidad de la clasificación*.

Esta medida de calidad representa la frecuencia relativa de los objetos del universo U correctamente clasificados por medio de los atributos en B.

Empleando el concepto de inseparabilidad se puede definir la función de pertenencia a conjuntos aproximados. Si el concepto o conjunto X es vago sus contornos no están rigurosamente definidos por lo que surge la incertidumbre relacionada con la cuestión de la pertenencia de los elementos al conjunto.

Definición 1.10: (Función de pertenencia aproximada)

La función de pertenencia de un elemento **x** a un conjunto **X** según la relación de equivalencia **B** se define como:

$$\mu_X^B(x) = \frac{|X \cap B(x)|}{|B(x)|} \qquad (1.11)$$

La pertenencia del objeto P1 al conjunto *Gripe=Si* según el sistema de información de la tabla 1.1 es:

$$\mu_{Gripe=Si}^B(P1) = \frac{|\{P1,P2,P3,P6\} \cap \{P1,P4,P6\}|}{\{P1,P4,P6\}} = \frac{2}{3} = 0.66$$

Obviamente el grado de pertenencia está en el intervalo [0,1]. La función de pertenencia puede ser comprendida como un coeficiente de la certidumbre con la que un elemento x pueda ser miembro del conjunto X. Esta medida cuantifica el grado de correspondencia relativa entre el conjunto X y la clase de equivalencia a la cual pertenece x. Se puede interpretar análogamente a la probabilidad condicional y entenderse como el grado de certidumbre de que el elemento x pertenezca al conjunto X.

Definición 1.11: (Aspereza del conjunto aproximado)

$$\rho_B(X) = 1 - \alpha_B(X) \qquad (1.12)$$

Nótese que un conjunto duro tiene un grado de aspereza 0, es decir, usando los atributos considerados en la relación de equivalencia se tiene un conocimiento completo del conjunto; mientras que un conjunto totalmente vago tendrá aspereza igual a 1.

Aunque la teoría de los Conjuntos Aproximados tiene un conjunto de extensiones las mismas no serán abordadas en este capítulo al no constituir un objetivo de esta tesis.

1.5) Ejemplo de estudio de un sistema de información usando la teoría de los conjuntos aproximados.

Sea el sistema de decisión de la tabla 1.1

En el mismo U = {P1, P2, P3, P4, P5, P6}, y el conjunto de atributos A = {Dolor de cabeza, Dolor muscular, Temperatura, Gripe}, donde el atributo d={Gripe} es el atributo de decisión.

Los dominios de los atributos son: V1 = {si, no}, V2={si, no}, V3={normal, alta, muy alta}, y Vd={Si, No}.

Si Vd define las clases a las que pertenecen los objetos tenemos:

Clase (Si) = {P1, P2, P3, P6}

Clase (No) = {P4, P5}.

Considerando la relación B = {Dolor de cabeza, Dolor Muscular, Temperatura} se obtienen los resultados siguientes:

La partición inducida por B es: {{P1}, {P2, P5}, {P3}, {P4}, {P6}}

Las aproximaciones para el conjunto (**Gripe=Si** {P1, P2, P3, P6}) son:

B_* (Gripe=Si) = {P1, P3, P6} B^* (Gripe=Si) = {P1, P2, P3, P5, P6}

α_B (Gripe=Si) = 3/5 = 0.6 γ_B (Gripe=Si) = 3/4 = 0.75

Luego la clase Gripe=Si es un conjunto aproximado, con una precisión de la aproximación de α_B (Gripe=Si) = 0.6 y una calidad de γ_B (Gripe=Si) = 0.75.

Las aproximaciones para el conjunto (**Gripe=No** {P4, P5}) son:

B_* (Clase=No) = {P4} B^* (Clase=Si) = {P2, P4, P5}

α_B (Clase=No) = 1/3 = 0.33 γ_B (Gripe=No) = 1/2 = 0.5

Luego la clase Gripe=No es un conjunto aproximado, con una precisión de la aproximación de αB(Clase=No) = 0.33, y una calidad de γB(Gripe=No) = 0.5.

Como se observa en este ejemplo el criterio de selección definido por la relación B clasifica al sistema de forma aproximada por lo cual este criterio es finalmente quien decide con que calidad se puede clasificar a un Sistema de Información.

Por otro lado, aquel conjunto mínimo de atributos, presentes en B, que clasifiquen correctamente a un sistema de información conforman un reducto, el

cual es también uno de los objetivos fundamentales de la teoría de los Conjuntos Aproximados Extendida.

CAPÍTULO II - Computación Paralela.

Actualmente la capacidad de integración y el abaratamiento de las tecnologías permiten que casi cualquier institución ya sea docente, científica, empresarial, entre otras, pueda contar con una capacidad de cómputo antes inimaginable para las tareas que necesita. Se prevé que la capacidad de integración llegue a un techo tecnológico, en el cual se necesite un nuevo paradigma para poder seguir incrementando la capacidad de procesamiento de las máquinas. Uno de esos paradigmas es el procesamiento paralelo.

En la actualidad, la programación paralela se ha convertido en una poderosa herramienta que permite dar una solución a grandes problemas costosos computacionalmente. La posibilidad de disminuir el tiempo de ejecución en la solución de estos problemas y aumentar la dimensión del problema que puede resolverse, se han convertido en sus objetivos fundamentales.

Por *procesamiento paralelo* se entiende la capacidad de utilizar varios elementos de proceso para ejecutar diferentes partes del mismo programa simultáneamente.

La resolución de problemas mediante procesamiento paralelo no es nueva, está basada en el viejo y conocido método de *divide y vencerás* utilizado para resolver problemas de carácter computacional.

2.1) Reseña Histórica de la Computación Paralela.

Desde 1955 personas como Gene Amdahl han investigado en el campo de arquitecturas paralelas obteniendo un grupo de parámetros que al medir su comportamiento optimizaban dichas arquitecturas permitiendo que la relación costo-rendimiento aumentase. Empresas como IBM, DEC y desde luego muchas otras organizaciones como el MIT, se interesan en la computación paralela desde las décadas de los 50-60, y de hecho siguen investigando y obteniendo

resultados en la actualidad, hasta el punto en que prácticamente todos los ordenadores que existen actualmente en el mercado explotan de una u otra manera soluciones paralelas.
En la década de los 80, la compartición de recursos a través de las redes de computadoras hizo posible un nuevo planteamiento para aprovechar no solo recursos como capacidad de almacenamiento o capacidad de impresión, sino para utilizar ciclos de CPU de otras máquinas conectadas a la red (los llamados multicomputadores). En los 70 y a primeros de los 80, personas como Bruce J. Nelson expusieron trabajos teóricos de cómo se podía utilizar mediante software esta capacidad de procesamiento paralelo que hasta ahora estaba relegada principalmente al hardware, limitándose el software a aprovecharlo mediante técnicas de programación explícita. En los años 90 con la expansión de las redes, la posibilidad de acceso a estas arquitecturas dejó de ser un privilegio de las entidades propietarias de los mismos, extendiéndose a miles de usuarios que remotamente podían ejecutar sus aplicaciones.

Esta década también se caracterizó por grandes avances en la tecnología del microprocesador la cual aún se extiende hasta la actualidad, elevando la frecuencia de procesamiento desde 40 Mhz (MIPS R3000, 1988) hasta mas de 3.0 Ghz (Pentium 4, 2005). Este hecho implica que el número de instrucciones por ciclo (CPI) se ha incrementado grandemente, lo cual se traduce en un incremento de la razón de ejecución de operaciones de punto flotante por segundo (FLOPS) [28]. Estas máquinas poderosas pueden desarrollar varios miles de millones de operaciones por segundo.

Las supercomputadoras tradicionales emplean procesamiento en paralelo, las cuales contienen arreglos de microprocesadores ultrarrápidos que trabajan en sincronía para resolver problemas muy complejos. Los fabricantes de supercomputadoras como Cray, IBM, Silicon Graphics, entre otros, producen modelos con diseños especiales y cuestan decenas de millones de dólares, precios que van más allá de los presupuestos de inversión de los grupos de investigación [37].

En los últimos años, el personal académico de diversas universidades y centros de investigación ha construido sus propias supercomputadoras conectando computadoras personales (PC) y desarrollando software para enfrentar tales problemas dando lugar a los *clusters de computadoras*. En 1994, se integró el primer cluster de PC en el Centro de Vuelos Espaciales Goddard de la NASA, para resolver problemas computacionales que aparecen en las ciencias de la Tierra y el Espacio. Los pioneros de este proyecto fueron Thomas Sterling, Donald Becker y otros científicos de la NASA. El cluster de PC desarrollado tuvo una eficiencia de 70 megaflops (MFLOPS - millones de operaciones de punto flotante por segundo). Los investigadores de la NASA le dieron el nombre de *Beowulf* a este cluster, en honor del héroe de las leyendas medievales, quien derrotó al monstruo gigante Grendel.

En 1996, le siguieron a este proyecto de la NASA otros dos nuevos. Uno de ellos es el proyecto *Hyglac* desarrollado por investigadores del Instituto Tecnológico de California (CalTech) y el Laboratorio de Propulsión Jet (JPL), y el otro, el proyecto *Loki* construido en el Laboratorio Nacional de Los Alamos, en Nuevo México. Cada cluster se integró con 16 microprocesadores Intel Pentium Pro y tuvieron un rendimiento sostenido de más de un gigaflop con un costo menor a $50,000 dólares.

En este nuevo siglo se ha extendido el uso de los clusters usando la Internet. Con ello aparece una nueva terminología, *Computación **GRID***. Esta constituye el procesamiento distribuido llevada a un nivel multi organizacional y se distingue de la computación distribuida por enfocarse en el compartimiento de recursos a gran escala, aplicaciones innovadoras y alto desempeño. En otras palabras, un “Grid” es una infraestructura de hardware y software que provee acceso de bajo costo a recursos computacionales de alto nivel de forma confiable, consistente y transparente [38].

Típicamente los sistemas computacionales paralelos están conformados por cientos y hasta miles de procesadores de distintos fabricantes, entre ellos encontramos a *BlueGene/L* (eServer Blue Gene Solution) con 65536

procesadores IBM, el que es capaz de desarrollar una potencia de cálculo de hasta 136800 gigaflops (GFLOPS) [39]. Una lista con los primeros 15 sistemas de cómputo más potentes se muestra en el Anexo -1.

2.2) Clasificación de los Sistemas Paralelos.

Según el número de procesadores presentes en un sistema de cómputo, actualmente se conocen dos tipos de clasificaciones. La primera de ellas fue propuesta por Michael J. Flynn en 1966, a la cual se le denomina *clasificación clásica*, la cual se aplicó a sistemas con uno o varios procesadores. La segunda, es una *clasificación moderna* en la que sólo se consideran los sistemas con más de un procesador.

2.2.1) Clasificación de *Flynn*

Flynn publicó su primera propuesta en 1966 y posteriormente en 1970. Esta taxonomía se basa en el flujo que siguen los datos dentro de una computadora y de las instrucciones que operan sobre esos datos. Flynn hace una clasificación en 4 grandes categorías:

- **SISD (Single Instruction stream, Single Data stream)**

  Los sistemas de este tipo se caracterizan por tener un único flujo de instrucciones sobre un único flujo de datos, es decir, se ejecuta una instrucción detrás de otra. Este es el concepto de arquitectura serie de *von Neumann* donde, en cualquier momento, sólo se ejecuta una única instrucción. Un ejemplo de estos sistemas son las máquinas secuenciales convencionales (Anexo-2a).

- **SIMD (Single Instruction stream, Multiple Data stream)**

  Estos sistemas tienen un único flujo de instrucciones que operan sobre múltiples flujos de datos. Ejemplos de estos sistemas los tenemos en las máquinas vectoriales con hardware escalar y vectorial.

  El procesamiento es síncrono, la ejecución de las instrucciones sigue siendo secuencial como en el caso anterior y todos los elementos realizan una

misma instrucción pero sobre una gran cantidad de datos. Por este motivo existirá concurrencia de operación, considerándose a esta clasificación el origen de la máquina paralela. El funcionamiento de este tipo de sistemas es el siguiente: La Unidad de Control envía una misma instrucción a todas las unidades de proceso (ALU). Las unidades de proceso operan sobre datos diferentes pero con la misma instrucción recibida (Anexo-2b).

Existen dos alternativas distintas que aparecen después de realizarse esta clasificación:

Arquitectura Vectorial con segmentación: Una CPU única particionada en unidades funcionales independientes trabajando sobre flujos de datos concretos.

Arquitectura Matricial (matriz de procesadores): Varias ALU idénticas a las que el procesador de instrucciones asigna una única instrucción pero trabajando sobre diferentes partes del programa.

- **MISD (Multiple Instruction stream, Single Data stream)**

Este tipo de sistemas no ha tenido implementación hasta hace poco tiempo. En ellos múltiples instrucciones operan sobre un único flujo de datos.

Los sistemas MISD se contemplan de dos maneras distintas:

- Varias instrucciones operando simultáneamente sobre un único dato.
- Varias instrucciones operando sobre un dato cuyo resultado será la entrada para la siguiente etapa. Se trabaja de forma segmentada y todas las unidades de proceso pueden trabajar de forma concurrente.

Ejemplos de estos tipos de sistemas son los arrays (arreglos) sistólicos o arrays de procesadores. También podemos encontrar aplicaciones de redes neuronales en máquinas masivamente paralelas (Anexo-2c).

- **MIMD (Multiple Instruction stream, Multiple Data stream)**

Estos sistemas empezaron a utilizarse a principios de los 80 y en ellos múltiples instrucciones operan sobre múltiples datos. Son sistemas con memoria compartida que permiten ejecutar varios procesos simultáneamente

(sistema multiprocesador). Cuando las unidades de proceso reciben datos de una memoria no compartida estos sistemas reciben el nombre de MULTIPLE SISD (MSISD). En arquitecturas con varias unidades de control (MISD y MIMD), existe otro nivel superior con una unidad de control que se encarga de controlar todas las unidades de control del sistema. Ejemplo de estos sistemas son las máquinas paralelas actuales (Anexo-2d).

2.2.2) Caracterización de las redes de Interconexión.

Existen varias arquitecturas para las computadoras paralelas, por lo cual es importante conocer sobre cual de ellas se ejecutarían nuestros programas.

El sistema de interconexión debe ser rápido y en dependencia del tipo de conexión puede clasificarse como completa o incompleta.

- **Sistema de interconexión incompleto**
  - Bus común.
  - Red de barras cruzadas (crossbar).
  - Red multietapa.
- **Sistema de interconexión completo (Anexo 3):**
  - Red Estrella (star).
  - Red Completamente conectada
  - Red Malla (mesh).
  - Red Hipercubo (hypercube).
  - Red anillo (ring).

En dependencia de la red de interconexión que empleen los procesadores para comunicarse, estas redes se clasifican en *estáticas* o *dinámicas*. Estas se pueden caracterizar usando diferentes criterios acerca del costo y el rendimiento que puede brindar la misma.

Redes de Interconexión Estáticas

Una red estática es aquella cuya topología queda definida de manera definitiva y estable durante la construcción de la máquina paralela [28]. La red simplemente

une los diversos elementos de acuerdo a una configuración dada. En las redes de interconexión estáticas existen enlaces permanentes de conexión.
Se utiliza sobre todo en el caso de los multicomputadores para conectar los diversos procesadores que posee la máquina. Por la red sólo circulan los mensajes entre procesadores, por lo que se dice que la red presenta un *acoplamiento débil*. En general, en las redes estáticas se exige poca carga a la red.

Se define como ***diámetro*** de una red a la distancia entre dos nodos de procesamiento cualesquiera y a su vez esta distancia se define como el camino más corto (menor número de enlaces) entre dichos nodos. Por ejemplo, en una red de tipo estrella el diámetro es 2 (Figura 2.1), mientras que para una tipo anillo es *p/2*, donde p es el número de nodos de procesamiento y para una red basada en una malla 2-D (bidimensional) es $2(\sqrt{p}-1)$ entre los dos nodos que están las esquinas opuestas diagonalmente.

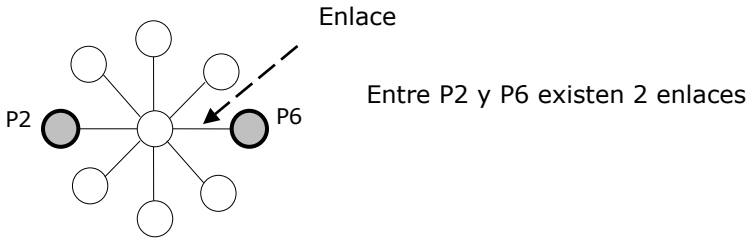


Figura 2.1: Diámetro de una red estrella de 9 nodos.

La ***conectividad*** de una red es una medida de la multiplicidad de los caminos entre dos nodos cualesquiera. Se prefiere una red con una alta conectividad, la cual posee una baja contención para los recursos de comunicación, por lo cual la retención del canal de comunicación por cualquier nodo debe ser mínima. Esta medida se puede ver como el mínimo número de arcos que deben ser removidos de la red para particionarla en dos redes desconectadas. Por ejemplo para un anillo (Figura 2.2) o una malla 2-D el número de arcos es 2.

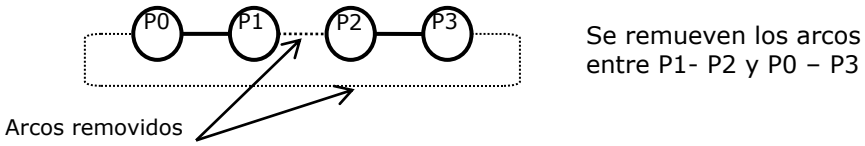


Figura 2.2: Conectividad de un anillo de 4 nodos.

El ***ancho de una bisección*** de una red es definida como el número mínimo de enlaces de comunicación que deben ser eliminados para particionar una red en dos mitades iguales. Así, para una malla 2-D deben eliminarse $\sqrt{p}$ enlaces (Figura 2.3).

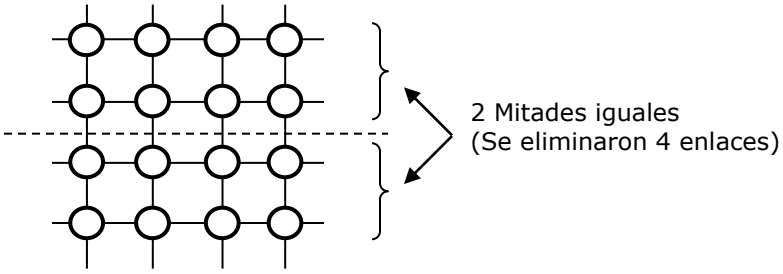


Figura 2.3: Ancho de una bisección para una malla 2-D para p =16 nodos.

Para evaluar el ***costo*** de una red pueden usarse varios criterios. Uno de ellos considera el número de enlaces de comunicación o el número de "cables" requeridos por la red. Así, para una red en estrella el costo es p-1 y para un hipercubo es (*p* log *p*)/2 enlaces. En la siguiente figura 2.4 se analiza el costo para un hipercubo de 8 nodos.

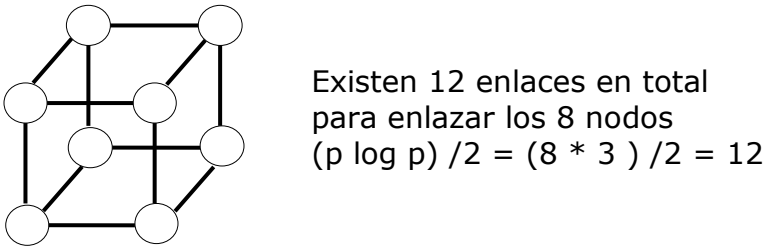


Figura 2.4: Costo de un hipercubo para p = 8 nodos.

En la siguiente tabla 2.1 se muestra un resumen de estos parámetros para algunas de las topologías de redes estáticas (***p*** indica el número de procesadores) [28].

Red	Diámetro	Ancho de la Bisección	Arcos de Conectividad	Costo (número de enlaces)
Estrella	2	1	1	p-1
Arbol binario completo	2 log ((p+1)/2)	1	1	p-1
Arreglo lineal	p-1	1	1	p-1
Malla 2-D	$2(\sqrt{p}-1)$	$\sqrt{p}$	2	$2*(p-\sqrt{p})$
Hipercubo	log p	p/2	log p	(p log p) /2

Tabla 2.1: Características de algunas topologías de redes estáticas.

Redes de Interconexión Dinámicas

Una red dinámica es una red cuya topología puede variar durante el curso de la ejecución de un programa paralelo o entre dos ejecuciones de programas. La red está constituida por elementos materiales específicos, llamados commutadores o *switches*.

Las redes dinámicas se utilizan sobre todo en los multiprocesadores. En este caso, la red une los **p** procesadores a los **M** bancos de memoria. Cualquier acceso de un procesador a la memoria (acceso a datos o a instrucciones) debe pasar a través de la red, por lo se dice que la red tiene un *acoplamiento fuerte*. La red debe poseer un desempeño extremadamente bueno para no demorar demasiado a los procesadores que acceden a memoria. Ya que el envío de mensajes se hace a través de elementos de conmutación existe una sobrecarga (overhead) introducido por el modo de funcionamiento del proceso de conmutación. Esto hace pensar en la necesidad de considerar a los nodos como elementos de conmutación, además de ser los elementos de procesamiento.

De esta manera el ***diámetro*** de la red será definido como la distancia máxima entre cualquier par de nodos (de procesamiento o de conmutación), lo cual es un indicativo del retardo máximo que afecta a un mensaje durante el proceso de comunicación entre estos nodos.

La ***conectividad*** de un nodo se define como el número mínimo de nodos que deben fallar (removidos de la red) para fragmentar la red en dos partes.

De manera similar los ***arcos de conectividad*** pueden ser definidos como el número mínimo de arcos que deben fallar (removidos de la red) para fragmentar la red en dos partes inalcanzables.

El ***ancho de la bisección*** debe considerar cualquier posible particionamiento de los **p** elementos de procesamiento en dos partes iguales. Para cada *partición* se debe seleccionar un particionamiento dirigido de los nodos de conmutación de forma tal que el número de arcos intersectados por esta *partición* sea minimizado.

2.3) Costo de la comunicación en máquinas paralelas.

La mayor fuente de sobrecarga (overhead) en la ejecución de un programa paralelo se origina por la comunicación de la información entre elementos de procesamiento. El costo de la comunicación depende de una variedad de características incluidas en los modelos semánticos de programación, la topología de la red, la manipulación de datos y el enrutamiento de los mismos, así como los protocolos de software asociados [28].

2.3.1) Costo del Paso de Mensajes en Maquinas Paralelas.

El tiempo total que toma la comunicación de un mensaje entre dos nodos en una red lo constituyen: la suma del tiempo para preparar el mensaje para la transmisión mas el tiempo que le toma al mensaje durante su paso por la red hacia su destino.

Los principales parámetros que determinan la latencia de la comunicación son los siguientes:

Definición 2.1: *tiempo de inicio* (***startup time***)(t_s)

Constituye el tiempo requerido para manipular un mensaje en los nodos transmisores o receptores. Este incluye el tiempo para preparar el mensaje (adicionar el encabezado, la información para la corrección de errores, etc.), el

tiempo para ejecutar el algoritmo de enrutamiento, y el tiempo para establecer una interface entre el nodo local y el enrutador. Este retardo ocurre una sola vez para la transferencia de un mensaje simple (sin datos).

Definición 2.2: *tiempo de salto* (***Per-hop time***) (t_h)

Después que un mensaje abandona el nodo, le toma un tiempo finito para alcanzar el próximo nodo en su camino. Al tiempo que le toma al encabezado del mensaje para viajar entre dos nodos directamente conectados en la red se le llama *tiempo de salto*. También se le conoce como latencia del nodo (***latency node***). Este tiempo se encuentra directamente relacionado con la latencia del elemento conmutador de rutas (caminos), el cual determina el canal o buffer de salida por donde será enviado el mensaje.

Definición 2.3: *tiempo de transferencia por palabra (**Per-word transfer time**)(t_w)*

Si el ancho de banda de un canal es ***r*** *palabras por segundo (palabra/seg)*, entonces a cada palabra le toma un tiempo $t_w = 1/r$ para atravesar el enlace. A este tiempo se le llama tiempo de transferencia por palabra. Este tiempo incluye la red así como la sobrecarga en el proceso de almacenamiento en los buffers.

Típicamente, para el envío de mensajes en las computadoras paralelas se emplean dos técnicas, enrutamiento basado en el almacenamiento y envío de mensajes (*store and forward routing*) y el enrutamiento basado en divisiones a tamaños fijos del mensaje (*cut - through routing*).

En el primer caso cada nodo receptor envía el mensaje al siguiente nodo en el enlace, solo después de haberlo recibido y almacenado totalmente, lo cual implica un pobre uso de los recursos de comunicación.

El tiempo de comunicación que toma enviar un mensaje de tamaño ***m*** para esta técnica se define:

$$t_{comm} = t_s + (mt_w + t_h)\,L$$

(m es el tamaño del mensaje y L es el número de enlaces)

En las computadoras modernas el tiempo por saltos (t_h) es muy pequeño y a su vez es mucho menor que mt_w por lo cual se puede ignorar, quedando:

$$t_{comm} = t_s + mLt_w \qquad (2.1)$$

En el segundo caso, un mensaje es dividido en unidades más pequeñas de tamaño fijo, las cuales son enviadas por el mismo camino del enlace en una secuencia establecida. De esta manera se eliminan las sobrecargas asociadas con la corrección y eliminación de errores.

$$t_{comm} = t_s + Lt_h + t_w m \quad (2.2)$$

Con el objetivo de minimizar el costo de la transferencia de los mensajes en la ecuación (2.2) se deben cumplir con los siguientes principios:

a) El envío de grandes mensajes en lugar de varios pequeños. Esto implica eliminar el costo (t_s) por cada pequeño mensaje, por lo que se amortiza la latencia de inicio. En plataformas típicas, como los clusters y máquinas de paso de mensajes $t_s >> t_h$ y también $t_s >> t_w$.
b) Minimizar el volumen de los datos. Esto implica reducir el volumen de datos a comunicar como sea posible, disminuyendo el pago por sobrecarga (overhead) en términos de transferencia por palabras (t_w).
c) Minimizar la distancia en la transferencia de datos. Esto implica minimizar el número de saltos que un mensaje debe realizar para llegar a su destino.

De estos tres principios resulta evidente que los dos primeros son relativamente fáciles de lograr, no así el último, debido a alguna de las siguientes problemáticas:

- En muchas bibliotecas de paso de mensajes (MPI), el control del programador para colocar (mapping) los procesos en los procesadores es limitado.
- En muchas arquitecturas, el enrutamiento se hace de forma aleatoria (dos pasos). Primeramente el mensaje es enviado a un nodo cualquiera (de forma aleatoria) y a partir de él (nodo intermedio) al nodo destino.
- El tiempo de salto (t_h) se considera como latencia de inicio (t_s) para pequeños mensajes o como un componente por palabra (mt_w) para grandes mensajes.

A partir de estos puntos, se define un modelo de costo simple en el cual el costo de la transferencia de un mensaje entre dos nodos de una red, está dado por:

$$t_{comm} = t_s + t_w m \quad (2.3)$$

De la ecuación anterior (2.3) se observa que el tiempo de transferencia por palabra (t_w) es uno de los factores con mayor significado, el cual aparece unido al tamaño del mensaje a transmitir. Típicamente este factor ($t_w m$) tiende a ser considerado el más importante en el costo de la comunicación, por lo cual las computadoras paralelas deben poseer redes de interconexión con altos anchos de banda y baja latencia. En el Anexo 4 se pueden encontrar las tendencias actuales y futuras de estas redes de interconexión.

2.3.2) Principios de Diseño de Algoritmos Paralelos

El desarrollo de algoritmos es uno de los componentes críticos para la solución de problemas. Un *algoritmo secuencial* consiste de una secuencia de pasos básicos para darle solución a un problema determinado en una computadora típica.

De manera similar un *algoritmo paralelo* nos indica como solucionar un problema dado, usando múltiples procesadores. Este presenta una dimensión añadida, como lo es la concurrencia de procesos y los diseñadores deben especificar una serie de pasos que deben ser ejecutados simultáneamente [28].

Con el objetivo de minimizar el costo de los programas paralelos se deben seguir varios principios, entre los cuales tenemos:

- Identificar porciones del problema que pueden ser ejecutados de forma concurrente.
- Distribuir las partes concurrentes del algoritmo en múltiples procesos que se ejecutan en paralelo.
- Distribuir los datos de entrada, salida e intermedios asociados con el programa.
- Administrar el acceso a los datos compartidos por múltiples procesadores.
- Sincronizar los procesadores en las distintas etapas de ejecución del programa paralelo.

De cualquier manera, es prácticamente imposible aplicar todos estos principios, por lo cual en dependencia de la arquitectura empleada y del paradigma de programación paralela empleado, solo algunas de estas combinaciones llegan a obtener buenos rendimientos.

Para lograrlo debemos tener en cuenta varios términos entre los cuales tenemos a los siguientes:

- *Descomposición*: Consiste en dividir un proceso de cálculo en pequeñas partes, algunas de las cuales o todas pueden ejecutarse en paralelo.
- *División por Tareas*: Las *tareas* constituyen unidades de cálculo definidas por el programador en las cuales se subdivide el procesamiento principal. Estas pueden tener un tamaño arbitrario y se convierten en unidades indivisibles de computación.
- *Dependencia de Grafos*: Algunas tareas pueden usar datos producidas por otras por lo cual pudieran tener que esperar por los mismos para terminar su procesamiento, ya que de lo contrario aparecerían errores.
- *Granularidad:* Es determinada por el número y tamaño de las tareas en las cuales se descompone un problema. Una descomposición que produzca un gran número de pequeñas tareas se le llama granularidad fina y por el contrario si tenemos pocas tareas de gran tamaño se le llama granularidad gruesa.
- *Grado de la concurrencia:* Establece el máximo número de tareas que pueden ser ejecutadas simultáneamente para un programa paralelo en un tiempo dado. En algunos casos el máximo grado de concurrencia es menor que el número total de tareas debido a la dependencia entre las mismas.

Para garantizar una descomposición de un problema en tareas concurrentes deben emplearse diferentes técnicas entre las cuales encontramos a las *técnicas de descomposición recursivas*, de *descomposición de datos* y *descomposición exploratoria.*

2.4) Modelación Analítica de Programas Paralelos.

Un algoritmo secuencial es evaluado usualmente en términos del tiempo de ejecución, expresado como una función del tamaño de sus datos de entrada.

Por otro lado el tiempo de ejecución de un algoritmo paralelo no solamente depende del tamaño de sus datos de entrada sino también del número de elementos de procesamiento usados, la velocidad del cómputo realizado y de la velocidad de la comunicación interprocesos.

Es por esa causa que un algoritmo paralelo no puede evaluarse de forma independiente de la arquitectura empleada sin tener alguna pérdida en la precisión, por lo cual se define a un *sistema paralelo* como la combinación de un algoritmo y la arquitectura paralela sobre el que se implementa el mismo.

Para medir el desempeño de un sistema paralelo se emplean un número de medidas. Estas medidas no solo se basan en el tiempo que toma darle solución a un problema dado en una arquitectura paralela dada, sino también, cuanto mas rápido es este algoritmo respecto a una versión secuencial (serie).

2.4.1) Fuentes de Sobrecarga (OverHead) en programas paralelos.

Se espera que al usar dos elementos de procesamiento, un programa se ejecute dos veces más rápido, lo cual es raro debido a una variedad de sobrecargas asociadas con el paralelismo.

Adicionalmente al cómputo esencial (similar al que podría realizar un programa secuencial) un programa paralelo también invierte tiempo en la comunicación interprocesos, en espera (idling) y en exceso (excess) de cómputo. La siguiente figura 2.5 se muestra un comportamiento típico de un programa paralelo.

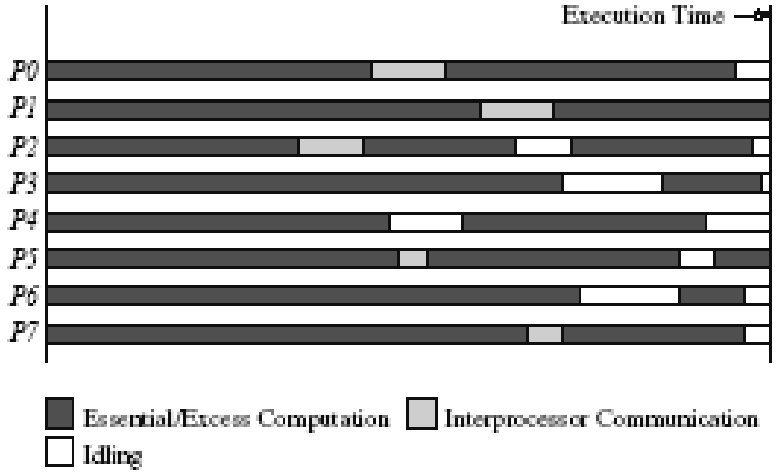


Figura 2.5: Perfil de la ejecución de un programa paralelo sobre 8 procesadores

Comunicación Interprocesador (Interprocessor Communication): Constituye el tiempo empleado para la comunicación de datos entre los elementos de procesamiento.

En espera (Idling): Esta espera se debe a varias razones entre las cuales tenemos: desbalance de carga, sincronización entre procesos, así como la presencia de componentes seríales en el programa.

Exceso de cómputo (Excess Time): Constituye la sobrecarga (overhead) que resulta de la diferencia en el desempeño de cómputo en que incurre un programa paralelo con respecto al mejor programa secuencial. Este aparece cuando existen operaciones secuenciales que en términos prácticos no deben paralelizarse.

Esencial (Essential): Constituye el tiempo añadido por el tipo de sistema operativo, lenguaje de programación empleado, etc., el cual está presente en algoritmos secuenciales como en paralelos.

Ya que varios algoritmos paralelos pueden incurrir en sobrecargas para la solución del mismo problema es importante cuantificar estas pérdidas para poder encontrar cual de ellos es el más apropiado. Para ello se emplea un grupo de métricas que se abordan en el siguiente epígrafe.

2.4.2) Métricas para el desempeño de Programas Paralelos.

Al lapso de tiempo transcurrido desde el comienzo, hasta el final de la ejecución de un programa que se realiza en una máquina secuencial, se le conoce como tiempo de ejecución serie (*serial runtime*) y se denota como Ts. Por otro lado, al lapso de tiempo que transcurre desde que comienza la ejecución de un programa paralelo hasta que el último de los elementos de procesamiento termina, se le conoce como tiempo de ejecución paralelo (*parallel runtime*) y se denota como Tp.

Definición 2.4: *Función de sobrecarga* **(Overhead Function)** (T_0)

Expresa la sobrecarga en que incurre un programa paralelo, en base al tiempo total colectivo que emplean **p** elementos de procesamiento con respecto al tiempo que emplea la mejor versión secuencial conocida. El tiempo total empleado en la solución de un problema determinado por **p** elementos de procesamiento es **pTp**.

La expresión que describe esta sobrecarga es: $\boldsymbol{T_0 = pTp - Ts}$ (2.8)

Definición 2.5: G*anancia del programa paralelo* **(Speedup)** (S)

Expresa la ganancia obtenida al paralelizar una aplicación dada sobre la implementación paralela. Define la razón del tiempo empleado para la solución de un problema en un elemento de procesamiento **p**, entre el tiempo empleado para resolver el mismo problema en una computadora paralela con **p** elementos de procesamiento idénticos.

La expresión que describe esta ganancia es: $\boldsymbol{S = \frac{Ts}{Tp}}$ (2.9)

Definición 2.6: *Eficiencia* **(Efficiency)** (E)

Expresa en que medida los elementos de procesamiento han sido usados al máximo de su capacidad (100%) durante la ejecución de un algoritmo paralelo. Esta medida muestra la fracción de tiempo en que un elemento de procesamiento es totalmente usado.

La expresión que describe esta eficiencia es: $\boldsymbol{E = \frac{S}{p}}$ (2.10)

Definición 2.7: *Costo* **(Cost)** (C)

Refleja la suma del tiempo que cada elemento de procesamiento emplea en la solución de un problema.

La expresión que describe el costo de un sistema paralelo es el siguiente:

$$C = pTp \qquad (2.11)$$

Es importante hacer notar que el comportamiento de un algoritmo puede cambiar notablemente para diferentes entradas (por ejemplo, para un problema de ordenamiento si los datos ya están ordenados). De hecho, para muchos programas el tiempo de ejecución es en realidad una función de la entrada específica, y no sólo del tamaño de ésta.

Debido a ello suelen estudiarse tres casos para un mismo algoritmo: *caso peor*, *caso mejor* y *caso medio*.

El caso mejor corresponde a la traza (secuencia de sentencias) del algoritmo que realiza menos instrucciones, el cual es **poco representativo.**

El caso peor corresponde a la traza del algoritmo que realiza más instrucciones, por lo cual es el más apropiado para indicar la medida de complejidad de un algoritmo: En este se obtiene una **cota superior** del tiempo de ejecución para cualquier tamaño de la entrada.

El caso medio, corresponde a la traza del algoritmo que realiza un número de instrucciones igual a la esperanza (media) matemática de la variable aleatoria definida por todas las posibles trazas del algoritmo para un tamaño de la entrada dado, con las probabilidades de que éstas ocurran para esa entrada. Si bien este es el caso ideal su procesamiento puede ser muy difícil.

El costo de un algoritmo secuencial o paralelo se expresa usando una *notación asintótica*, la cual constituye una notación matemática que permite analizar el comportamiento de las funciones en el límite o su tasa de crecimiento. Estas notaciones se usan para evaluar el comportamiento de un algoritmo con valores de datos de entrada grandes.

Entre las notaciones más conocidas de este tipo tenemos:

a. **Notación** Θ (theta): Expresa el orden exacto de la función (caso medio).
b. **Notación** O (o mayúscula): Expresa la cota superior (peor caso).
c. **Notación** Ω (omega): Expresa la cota inferior (mejor caso).

Un sistema paralelo tiene un *costo óptimo*, si el costo de la solución de un problema en esta arquitectura tiene el mismo crecimiento asintótico (en términos de Θ) como una función del tamaño de los datos de entrada, de manera similar al algoritmo secuencial mas rápido conocido en un único elemento de procesamiento.

De esta manera la eficiencia de un sistema paralelo con un costo óptimo tendría una eficiencia del orden $\Theta(1)$.

CAPÍTULO III- Algoritmos Secuenciales y Paralelos empleados en la solución de los principales conceptos de la Teoría de los Conjuntos Aproximados.

En la actualidad, la solución de un problema puede considerarse una forma de razonamiento muy compleja que requiere la generación y asimilación de nuevas herramientas. Típicamente existen cuatro vías fundamentales para encontrar la solución a un problema dado: (a) la aplicación de una fórmula explícita que da la solución, (b) el uso de una definición recursiva, (c) el uso de un algoritmo que converge a la solución o (d) la aplicación de otros procesos, en particular los de prueba y error. Siempre que sea posible el primer método es el mejor. Por ejemplo, la complejidad de encontrar los ceros de un polinomio de segundo grado es constante, un **O**(1).

Por el contrario el método recursivo es computacionalmente muy costoso. Un buen algoritmo debe ser capaz de resolver un problema en tiempo polinomial, lo cual significa que su complejidad en el peor caso está acotada superiormente por un polinomio sobre el tamaño de su entrada. Las tres primeras vías son las usadas en los programas de computadoras tradicionales; ellos incluyen el desarrollo de un modelo matemático, o lógico, adecuado que describe un dominio o una parte de él. La solución al problema se deriva del uso de ese modelo. Con la existencia de este modelo se puede hallar una secuencia estructurada de pasos que garantiza encontrar una solución al problema en una longitud finita de tiempo. Tales procedimientos son llamados algoritmos y ellos forman la base para sistemas de software convencional.

La cuarta vía es la utilizada para problemas en los cuales no hay una solución algorítmica conocida o es tan compleja que no es posible una implementación computacional conocida práctica, como es el caso del problema del Viajero vendedor. Muchos de los problemas que carecen de una solución algorítmica son resueltos por los humanos a pesar de su capacidad de procesamiento “inferior”.

Existen problemas combinatorios para los cuales no se conocen algoritmos de resolución que no sean otros que aquellos que producen una explosión del tiempo de cálculo (exponencialmente) al aumentar el tamaño del problema.

Dada la importancia actual que revisten los conjuntos aproximados se han desarrollado varios algoritmos que dan solución a esta problemática. Como ya hemos planteado estos algoritmos analizan el comportamiento de un sistema de información, específicamente cuan eficientemente es capaz de clasificar un atributo o un conjunto de ellos a este sistema. Este análisis ya se ha planteado en la literatura científica, donde se ha proporcionado un algoritmo que converge a la solución del mismo, el cual es empleado por varias aplicaciones secuenciales. Por el contrario el empleo de nuevas técnicas, como las de procesamiento paralelo, aún no se han desarrollado intensivamente.

Como parte del presente trabajo se analizan las soluciones secuenciales abordadas en la literatura llegando a proponer una solución secuencial para el cálculo de los principales conceptos de esta teoría.
Ya que en las soluciones computacionales basadas en métodos secuenciales, al elevarse la dimensión de los datos su costo se eleva también en alguna medida, se proponen varios algoritmos basados en soluciones paralelas, los cuales aplican este tipo de técnicas en algunas o en todas las etapas que abarca el proceso de solución de los conceptos expuestos en el capítulo anterior.
Estos algoritmos, como se mostrara en el presente capítulo, serán implementados de manera paralela para lograr reducir el tiempo de su ejecución.

3.1) Aspectos computacionales de la Teoría de conjuntos aproximados.

Como todo modelo computacional, al estudiar la Teoría de los conjuntos aproximados es necesario analizar la complejidad de los procedimientos de la misma.

Entre estos procedimientos se tiene al proceso de construcción de las *clases de equivalencia* como una etapa muy costosa dentro del proceso solución según han demostrado varios estudios. Por ejemplo Bell, D. y Guan así como Deogun, J.S han demostrado en [26-27], que el procedimiento empleado para construir una relación de equivalencia (basado en un criterio de selección) presenta una complejidad computacional **O**(lm^2), donde ***l*** es el número de atributos (rasgos) que describen a los objetos y ***m*** es la cantidad de objetos en el universo.

Así también, la complejidad computacional de encontrar el conjunto B*(X) (*aproximación superior del conjunto X*) o el conjunto B$_*$(X) (aproximación inferior del conjunto X) es **O**(lm^2) [26-27].

3.2) Algoritmo Secuencial empleado en la solución de los principales conceptos de la Teoría de los Conjuntos Aproximados

En la actualidad se conocen algunos sistemas profesionales basados en esta teoría. Por ejemplo, el software para el descubrimiento de reglas LERS (Learning from Examples using Rough Sets) [20-24], y el sistema de análisis de datos Rosetta [25-28]. Ambos sistemas proveen sus soluciones abarcando algunos de los principales conceptos, empleando para ello soluciones secuenciales.

A partir de los contenidos expuestos en el capítulo anterior se puede afirmar que el proceso de solución de los principales conceptos de la teoría de los conjuntos aproximados consta de 3 etapas fundamentales:

1ra Etapa: Procesamiento de los datos (1).

2da Etapa: Construcción de las clases equivalentes a partir de un criterio dado (2).

3ra Etapa: Construcción de la aproximación inferior y superior del conjunto (3).

En la primera de ellas (1) se *introducen los datos (objetos)* pertenecientes al ***sistema de información y*** *se procesan las clases a las que pertenecen los mismos*. Estos pueden ser presentados en forma de un archivo de texto, donde cada línea del mismo constituye un objeto de ese sistema o desde una estructura especial en la memoria del ordenador.
La etapa siguiente (2) persigue el objetivo de *conformar las clases de equivalencia* a la que pertenece cada objeto. Aquí es importante recordar el papel que juega el ***criterio*** seleccionado para conformar dichas clases. Este proceso implica un proceso de búsqueda y construcción de una nueva clase de equivalencia en caso de no encontrarla. Esta etapa se encuentra estrechamente sincronizada con la anterior, ya que este procesamiento se realiza para cada objeto introducido desde la etapa (1). A su vez constituye el punto de partida para la etapa (3) donde se encuentran ya construidas todas las clases de equivalencia.

La última etapa (3) permite dar solución a los principales conceptos de la teoría: *aproximaciones superior e inferior de un conjunto, precisión y calidad de la clasificación*, entre otros y se caracteriza por el proceso de **construcción de las aproximaciones**. Esta etapa se realiza en dependencia del número de valores diferentes que presente el atributo de clase a la pertenece cada objeto, típicamente define el conjunto que se quiere aproximar. En la mayoría de los sistemas de información lo constituye el último atributo.

Con el objetivo de validar las afirmaciones planteadas anteriormente [26-27] el presente trabajo parte de desarrollar una versión secuencial, cuyo algoritmo es el que se muestra en la figura 3.1.
Para ello consideremos un sistema de Información (SInf), con los siguientes datos:

***m*:** Total de objetos en el Sistema de Información (SInf)

n: Un objeto de ***l*** rasgos (***l_1 ... l_k***)

CE: Clase de Equivalencia de acuerdo a un criterio (definida por rasgos)

k: Clase a la que pertenece el objeto n (***l_k***)

K: Total de clases presentes en el SInf

Nk: Total de objetos en el Sistema de Información (S.I) que pertenece a la clase k.

apx_inf: Número de objetos que en una CE pertenecen a la aproximación inferior

apx_sup: Número de objetos que en una CE pertenecen a la aproximación superior

```
1. program rset_sec
2. array_class := Φ;    {Inicializa el arreglo de clases }
3. array_CE := Φ;  {Inicializa el arreglo de clases equivalentes}
4. Para cada objeto en el Sinf (m)
5.    leer n;              {leer el objeto n del SInf}
6.    procesar la información de clase de n; {Función info_clase}
7.    construir la CE para el objeto n; {Función const_ceq}
8. construir las aproximaciones de clases {Función const_aprox}
9. end_rset_sec;

function info_clase  {Función que maneja las clases del SInf}
1.  Si no está la clase (k) de n en array_class
2.    entonces almacenarla e inicializar el contador de objetos
                de esa clase (Nk) en 1
3.  sino incrementar el contador de objetos de la clase de n (Nk)
4.  end_function;

function const_ceq   {Función que construye una CE a partir de n
                      y el conjunto de rasgos indicados de n}
1.  Si no existen CE en array_CE
2.     entonces
3.        construir una CE con la información de n y
4.        adicionarla a array_CE
5.  sino
6.     Para cada CE en array_CE
7.        Si los valores del criterio de equivalencia de n
             coinciden con los valores de la CE actual
8.        entonces adicionar n a la CE actual
```

```
9.        sino construir una CE con la información de n y
10.            adicionarla a array_CE
11. end_function;

function const_aprox  {Función que construye las aproximaciones}
1.  t:= 0;       {número de objetos en la apx_inf}
2.  Para cada clase (k) diferente del SInf
3.     apx_inf:=0;
4.     apx_sup:=0;
5.     Para cada clase de equivalencia (CE)
6.        r:= número de objetos en la CE con clase igual a k;
7.        Si r ≠ 0
8.           Si r == cantidad de objetos en la CE (tobj_CE)
9.              entonces apx_inf:= aprox_inf+tobj_CE;
10.          apx_sup:= aprox_sup+tobj_CE;
11.    precisión:= apx_inf/apx_sup;
12.    calidad:= apx_inf/Nk;
13.    t:= t+apx_inf;
14. calidad_SInf:= t/m;     {Calidad de la clasificación}
15. end_function;
```

Figura 3.1: Algoritmo Clásico Secuencial.

Este algoritmo consta de un bloque principal (***rset_sec***) y tres bloques auxiliares (funciones ***info_clase***, ***const_ceq*** y ***const_aprox***).

El descubrimiento de cada clase (k) distinta presente en SInf (línea 6 de ***rset_sec***) se determina examinando cada objeto n. Este proceso (definido en ***info_clase)*** busca la presencia de la clase (k) de un objeto n en el arreglo de K clases clases (*array_class*) lo cual implica una complejidad del orden **O(*K*)**. Este proceso se encarga además de almacenar el tipo de clase de n, llevando a su vez un control del número de objetos pertenecientes a cada una de ellas, donde ambas operaciones aportan una complejidad computacional **O**(***1***).

$$\mathbf{O}(\boldsymbol{K}) + \mathbf{O}(\boldsymbol{1}) + \mathbf{O}(\boldsymbol{1}) \approx \mathbf{O}(\boldsymbol{K}) \qquad (3.2.1)$$

Ya que este descubrimiento de las clases presentes en SInf se le realiza a cada objeto presente en el mismo (***m***) su complejidad computacional será un **O**(***Km***).

$$\mathbf{O}(\boldsymbol{K}) * \mathbf{O}(\boldsymbol{m}) \approx \mathbf{O}(\boldsymbol{Km}) \qquad (3.2.2)$$

En el caso de que cada objeto n pertenezca a una clase k distinta (**K** = **m**) entonces podemos esperar una complejidad computacional máxima del orden de un $\mathbf{O(m^2)}$.

$$\mathbf{O}(\boldsymbol{m}) * \mathbf{O}(\boldsymbol{m}) \approx \mathbf{O}(\boldsymbol{m^2}) \qquad (3.2.3)$$

A su vez, el proceso de construcción de las clases de equivalencia (línea 7 de ***rset_sec***) está afectado por la cantidad de objetos presentes en el SInf (***m***). A su vez este proceso (definido en ***const_ceq***) añade la complejidad inherente al proceso de búsqueda de la clase de equivalencia a la que pertenecería un objeto n de ***l*** rasgos. Es decir que si existe un número ***c*** de clases equivalentes, donde comparar cada objeto n (líneas 6 a 10), implicaría una complejidad computacional definida por un **O**(***cl***).

Finalmente el proceso de construcción y adición de una nueva clase se realizaría con un costo unitario cada una (**O**(***1***)) con lo cual este proceso presenta finalmente una complejidad **O**(***cl***) en ***const_ceq***:

$$\mathbf{O}(cl) + \mathbf{O}(1) + \mathbf{O}(1) \approx \mathbf{O}(cl) \qquad (3.2.4)$$

a la cual ahora se le añade el costo de procesar ***m*** objetos. De acuerdo a ello el proceso de construcción de las clases equivalentes para un SInf de ***m*** objetos (líneas 4 a 7) presentará una complejidad computacional **O**(***mcl***):

$$\mathbf{O}(cl) * \mathbf{O}(m) \approx \mathbf{O}(mcl) \qquad (3.2.5)$$

Si como resultado del criterio de selección, cada objeto pertenece a una clase de equivalencia distinta tendremos entonces que ***c*** = ***m***, por lo que la máxima complejidad computacional que podemos esperar es un **O(lm^2)**.

$$\mathbf{O}(ml) * \mathbf{O}(m) \approx \mathbf{O}(lm^2) \qquad \textbf{(3.2.6)}$$

La última etapa referida a la construcción de las aproximaciones de las clases (línea 8 de ***rset_sec***) se realizará tantas veces (***K***) como a clases distintas (***k***) pertenezca cada objeto del sistema de información. Este proceso (definido en ***const_aprox***) debe recorrer todas las clases de equivalencia formadas en la etapa anterior buscando la existencia de la clase k en cada objeto n (de ***l*** rasgos) presente en las mismas. Si consideramos a ***c*** como el total de clases de equivalencia podemos concluir que la complejidad computacional de esta etapa es un **O**(**Klc**).

$$\mathbf{O}(K) * \mathbf{O}(cl) \approx \mathbf{O}(Kcl) \qquad (3.2.7)$$

De manera similar, si $c = m$ entonces su complejidad computacional sería ahora un **O(Kml)**.

$$\mathbf{O}(K) * \mathbf{O}(ml) \approx \mathbf{O}(Kml) \qquad (3.2.8)$$

En el caso de que cada objeto n pertenezca a una clase k distinta ($\mathbf{K} = \mathbf{m}$) entonces podemos esperar una complejidad computacional máxima del orden de un $\mathbf{O(lm^2)}$.

$$\mathbf{O}(m) * \mathbf{O}(ml) \approx \mathbf{O}(lm^2) \qquad \textbf{(3.2.9)}$$

De acuerdo a 3.2.6 y 3.2.9 este algoritmo concuerda con los resultados expuestos en [26-27].

De esta manera se puede concluir que el proceso para determinar los principales conceptos de la teoría de los conjuntos aproximados tendrá una complejidad computacional de un $\mathbf{O(m^2(2l+1))}$.

$$\mathbf{O}(m^2) + \mathbf{O}(lm^2) + \mathbf{O}(lm^2) \approx \mathbf{O}(m^2) + \mathbf{O}(2lm^2) \approx \mathbf{O}\,(m^2(2l+1)) \qquad \textbf{(3.2.10)}$$

En la tabla 3.1 se resumen los costos por etapas que presenta este algoritmo secuencial.

Etapa	Costo
1- Procesamiento de los datos (clases)	$\mathbf{O}(m^2)$
2- Construcción de CE	$\mathbf{O}(lm^2)$
3- Construcción de las aproximaciones	$\mathbf{O}(lm^2)$
Cálculo de los principales conceptos (total)	$\mathbf{O(2m^2l + m^2)}$.

Tabla 3.1: Complejidad computacional por etapas del Algoritmo Clásico Secuencial.

Basado en este algoritmo se construyó una aplicación, cuyos resultados se exponen en [40].

Otro elemento importante que incidirá en el rendimiento de este algoritmo es el tiempo que consumiría cada etapa, el cual dependerá del tipo de arquitectura donde se ejecute el mismo, así como del lenguaje de programación empleado.

Así, el tiempo empleado en la etapa (2), (***t_CE***: tiempo empleado en la construcción de las clases de equivalencia a partir de un criterio especifico),

abarca los tiempos de búsqueda de la CE (**t_bce)**, el tiempo de actualización de la misma (**t_ace**) y el tiempo de construcción si fuera necesario (**t_cce**).

Es decir, $t\_CE = t\_bce * t\_ace$ ó $t\_CE = t\_bce * t\_cce) \approx lm^2$ (3.2.12)

Por otro lado en la etapa (3) (***t_aprox***: tiempo empleado en la solución de los principales conceptos), abarca un tiempo similar a **t_bce** al examinar el tipo de clase (k) mas el tiempo de cálculo de los principales conceptos (**t_pc**).

$$t\_aprox = t\_bce + t\_pc \approx lm^2 \qquad (3.2.13)$$

Y finalmente en la etapa (1) (***t_proc_clases***: tiempo empleado en el procesamiento de las clases será:

$$t\_proc\_clases \approx m^2$$

Así el tiempo total serie es: $Tsec = t\_CE + t\_aprox + t\_proc\_clases$ donde:

$$Tsec = \mathbf{m^2(2l+1)} \qquad \mathbf{(3.2.15)}$$

3.3) Propuestas de Algoritmos Paralelos alternativos a esta problemática.

Ya que el objetivo fundamental de este trabajo es obtener algoritmos paralelos que den solución a los principales conceptos de esta teoría aprovechando las ventajas de las técnicas paralelas se realizó un estudio encaminado a descubrir en cuales de las 3 etapas del proceso descritas en el capítulo anterior es posible aplicar dichas técnicas.

Como resultado de este estudio se desarrollaron 4 algoritmos (denominados A, B, C y D), los cuales aplican técnicas paralelas en algunas o en todas las etapas tal como se muestra en la siguiente tabla:

Algoritmo	**Etapa – 1** Procesamiento de los Datos	**Etapa – 2** Construcción de las Clases de equivalencia	**Etapa – 3** Calculo de los principales conceptos
A	Si	Si	Si
B	Si	Si	Si
C	No	No	Si
D	Si	Si	Si

Tabla 3.2: Empleo de técnicas paralelas por algoritmo

Cada algoritmo presenta características propias excepto el A y el B los cuales son prácticamente similares, incluyendo el segundo una pequeña diferencia con respecto al A en la etapa 2, lo que se tradujo en una ligera mejoría. Cada algoritmo desarrollado presenta mejores rendimientos con respecto a su versión anterior partiendo de las dificultades presentadas por su precedente.

Ellos han sido concebidos para implementarse sobre un clusters de computadoras, por los beneficios que brinda esta arquitectura, tal como se expuso en el capítulo 2 y a su vez se basan en el modelo maestro/esclavo ("Master-Slave") con el ánimo de sincronizar y dirigir cada proceso desde un procesador de manera centralizada. *La red de interconexión* presente en un cluster se considerará *totalmente conectada*. A su vez para el desarrollo de los mismos se tuvo en cuenta el paralelismo de los datos, debido a que se debe procesar un gran volumen de información contenido en una base de datos o Sistema de Información, donde resulta característico el trabajo con estructuras de datos muy regulares repitiendo una misma acción sobre cada elemento de la estructura. En los epígrafes siguientes se exponen las principales características de estos algoritmos.

3.3.1) Algoritmo Paralelo "A"

Este algoritmo constituyó el primer algoritmo paralelo desarrollado para dar solución a los principales conceptos de la teoría de los conjuntos aproximados, convirtiéndose en la base para el desarrollo del resto de los algoritmos. Fue concebido para emplear técnicas paralelas en las 3 etapas fundamentales del proceso y emplea el modelo "Master-Slave". En este algoritmo el procesador maestro (principal), es el encargado de procesar y distribuir los objetos del Sistema de información (SInf) (etapa-1) de acuerdo a una distribución propia persiguiendo como objetivo que cada procesador esclavo ("slave") procese una cantidad similar de objetos del SInf, mientras que los procesadores esclavos son los encargados de construir las clases de equivalencia a la par que reciben cada

objeto. Debido a esta distribución existe una alta probabilidad que existan clases de equivalencia similares en los distintos procesadores esclavos.

Posteriormente se mezclaron todas las clases de equivalencias obtenidas en cada procesador, quedando el resultado de dicho proceso en el procesador 1. A partir de este momento, con el objetivo de preparar las condiciones para realizar el cálculo de la aproximación, se difunde la estructura de datos presente en el procesador 1, hacia todos los restantes procesadores. Finalmente se distribuyó el proceso de cálculo de la aproximación, creando en cada procesador el archivo solución correspondiente.

En esta implementación, el uso de los procesadores se mantiene durante todas las etapas del proceso prácticamente. Es típico el uso de un algoritmo de mezcla, donde a partir de los procesadores pares se comienzan a distribuir los datos a sus respectivos impares, repitiendo este proceso m veces, donde $m = \log_2(P-1)$, y P es el número de procesadores. La complejidad de este algoritmo es $\log_2(P-1)$. Este algoritmo incluye un proceso de reducción que decrementa la cantidad de procesadores que participan en el proceso de mezcla en cada iteración.

El algoritmo que describe este procesamiento es el que se describe en los anexos 5 al 8, según la etapa analizada (tabla 3.2), asumiendo un SInf con características similares al definido para el algoritmo clásico secuencial y una red de **P** procesadores ($p_0..p_{p-1}$).

De manera similar, tanto el algoritmo “A” como el resto “B, C y D” han sido desarrollados para implementarlos en un cluster de computadoras (topología física completamente conectada) el cual presenta un diámetro equivalente a 1.

3.3.1.1) Análisis de la complejidad y cálculo del tiempo paralelo

Ya que este algoritmo emplea el esquema maestro-esclavo, el análisis de la complejidad computacional de los procesos involucrados, debe realizarse de manera independiente por cada etapa presente en el algoritmo.

Etapas 1 y 2: Distribución de los objetos del Sinf y Construcción de las CE de acuerdo al criterio establecido.

Procesador Maestro (p0):

Al determinar p0 la cantidad de datos ($m^* = \lceil m/P-1 \rceil)+1$, como máximo) que debe emplear cada procesador esclavo, el proceso de enviar este valor le tomaría un tiempo $T_{0m^*} = t_s + t_w m_1$, para cada uno de ellos, siendo m_1 el tamaño del valor a enviar (un número entero). Para P-1 procesadores involucrados el tiempo paralelo empleado será $T_{0m^*}=(P-1)(t_s + t_w m_1)$. Ya que m_1 es *constante* no se considera en la expresión del tiempo.

De esta manera $\mathbf{T_0p_{m^*}=(P-1)(t_s+t_w)}$ y el costo asociado a esta operación (líneas 3 a 5 ***rset_parA***) tendría una complejidad computacional del orden **O(P-1)**. (3.3.1.1)

El proceso de envío de cada objeto del SInf (m) a los (P-1) procesadores esclavos correspondientes (línea 7) que se describe en las líneas de la 6 a la 9, le tomaría un tiempo $T_{0n}=t_s + t_w m_2$ por cada objeto n, dependiendo m_2 del número de atributos (**l**) de n. Esto implica un tiempo total paralelo para m objetos definido por: $\mathbf{T_0p_m=m(t_s+t_wl),}$ con una complejidad computacional del orden **O(m)*O(l)** = **O(lm)**. (3.3.1.2)

Otra de las funciones de este procesador es realizar el proceso de descubrimiento de cada clase (k) distinta presente en SInf (línea 10) de manera similar al algoritmo secuencial, examinando cada objeto n. Al tiempo empleado en este proceso se le llama T_{0c_k}.

Este proceso (definido en ***info_clase***) se le realiza a cada objeto presente en el SInf (***m***) el cual presenta una complejidad computacional de un **O**(***Km***) de acuerdo a 3.2.2 y pudiera ser como máximo de un orden **O**($\boldsymbol{m^2}$) según 3.2.3.

Así $\mathbf{T_0c_k}$ puede ser definido de un orden $\mathbf{O(m^2).}$ (3.3.1.3)

Por razones de simplificación del análisis consideraremos al factor P-1 (procesadores esclavos) como **P** (todos los procesadores). De la misma manera el particionamiento del SInf (**m**) (m/P+1) se considerará **m/P**.

De esta forma el tiempo empleado por el procesador p0 es:

$$\mathbf{T_0 = T_0p_{m^*}+ T_0p_m+T_0c_k= P(t_s +t_w) + m(t_s +t_wl) + m^2} \quad \mathbf{(3.3.1.4)}$$

De acuerdo a 3.3.1.1, 3.3.1.2 y 3.2.3 la máxima complejidad computacional esperada para esta etapa en p0 será: **O(P)** + **O(lm)** + **O**($\mathbf{m^2}$)

Típicamente m >> P, por lo que la complejidad sería:

O(ml) + **O**($\mathbf{m^2}$) **≈ O**($\mathbf{m^2}$) **(3.3.1.5)**

Procesadores Esclavos:

En un esquema maestro-esclavo las operaciones de intercambio realizadas entre el procesador maestro y el esclavo se consideran únicamente en uno de los dos extremos (para este análisis se considerará en el maestro). Con el arribo de cada dato (n*) se comienza el proceso de construcción de la CE correspondiente para el mismo (Función ***const_ceq***), la cual se realiza en un tiempo serie, con una complejidad computacional **O(lc*)** por cada objeto donde **c*** es el número de CE formadas en el procesador esclavo correspondiente y **l** es el número de atributos de n*. Para una cantidad de datos m/P el tiempo empleado en la CE será: T_{eCE}= (m/P)lc*.

Si cada objeto **n**, por sus rasgos (**l**) es diferente entonces c* = m/P. Por lo cual

$\mathbf{T_{eCE} = (m/P)\ l\ (m/P) \approx (m/P)^2 l}$. **(3.3.1.6)**

Lo que implica una complejidad **O(($\mathbf{m/P)^2 l}$)**. **(3.3.1.7)**

Etapa 2.1: Mezcla de las CE hacia el procesador 1.

En esta solo intervienen los procesadores esclavos, donde esta etapa se repetirá ***nm*** veces (nm = $\log_2$(P)) (líneas 17 a la 31). En cada una de estas etapas interviene además un proceso especial encargado de reducir la cantidad de procesadores participantes en una razón (P-1)/2 + 1 como mínimo (Proceso de reducción) por cada vez (Anexo 9). Cada mezcla involucra el envío o recepción en los procesadores correspondientes de las CE (c*) definidas en array_CE*. Este proceso involucra el envío de un bloque de tamaño m_3, por lo que el tiempo asociado a este sería $T_{ep_{envio/recep}} = t_s + t_w m_3$, donde m_3 puede ser a lo sumo m/P (c*).

De esta forma $\mathbf{T_{ep_{envio/recep*}} = t_s + (m/P)t_w}$, lo que implica una complejidad **O(m/P)** por mezcla.

Ya que se realizan $\log_2 P$ mezclas, el tiempo empleados en todas las etapas de envío-recepción será: $\mathbf{T_{ep_{envio/recep}} = \log_2 P(t_s + (m/P)t_w)}$. **(3.3.1.8)**

Lo que implica una complejidad $\mathbf{O(\log_2 P(m/P))}$. **(3.3.1.9)**

Este proceso implica además la búsqueda, construcción o actualización de cada CE recibida ($\mathbf{c_r^*}$) en el procesador (Función ***mezc_CE***), respecto a las CE existentes en el mismo ($\mathbf{c^*}$) lo cual se realiza de forma serie en cada procesador.

Ya que $\mathbf{c_r^* = c^* = m/P}$ en el caso extremo, tenemos que para CE formadas por objetos de **l** atributos, el tiempo empleado será: $\mathbf{T_{e\_mezcla} = (m/P)^2 l}$. **(3.3.1.10)**

Este proceso implica una complejidad del orden $\mathbf{O((m/P)^2 l)}$. **(3.3.1.11)**

Así, de 3.3.8 y 3.3.10 tenemos que:

$$\mathbf{T_{ep_{mezcla/reducción}} = \log_2(P)[t_s+(m/p)t_w] + (m/p)^2 l} \qquad \mathbf{(3.3.1.12)}$$

Con una complejidad según 3.3.1.9 y 3.3.1.11 del orden:

$$\mathbf{O(\log_2 P(m/P)) + O((m/P)^2 l) \approx O((m/P)^2 l)} \qquad \mathbf{(3.3.1.13)}$$

<u>Etapa 2.2:</u> Difusión de las CE hacia todos los procesadores.

Esta etapa persigue que todos los procesadores presentes tengan una copia idéntica de las *CE* formadas. Para ello desde el procesador p1 deben difundirse todas sus *CE* (one-to-all-broadcast), lo cual implica el envío de un bloque de tamaño m_4 (array_CE) (líneas 34 a 36), cuyo tamaño como máximo será **m**.

El tiempo paralelo empleado será: $\mathbf{T_{ep_{dif\_CE}} = t_s + mt_w}$. **(3.3.1.14)**

Así, su complejidad será un $\mathbf{O(m)}$. **(3.3.1.15)**

<u>Etapa 3:</u> Construcción de las Aproximaciones.

Como la información de las k clases presentes en el SInf se encuentran en el procesador p0, este debe difundirlas al resto (one-to-all-broadcast), lo cual implica el envío de un bloque de tamaño m_5 (*array_class*) (líneas 37 a 39) cuyo tamaño como máximo será **m**.

El tiempo paralelo empleado será: $\mathbf{T_{ep_{dif\_K}} = t_s + mt_w}$. **(3.3.1.16)**

Así, su complejidad será un $\mathbf{O(m)}$. **(3.3.1.17)**

Una vez terminada esta difusión, cada procesador calcula las aproximaciones correspondientes a sus clases (Función ***const_aprox_par***) (línea 40). Este

proceso se realizará en cada procesador, **K/P** veces como máximo, sobre todas las CE, cuyo número será como máximo **m** en un tiempo paralelo empleado será: $\mathbf{T_{aprox\_k} = m(K/P).}$

Para el peor caso cada objeto pertenece a una clase distinta (K=m), por lo cual:

$$\mathbf{T_{aprox\_k} = m^2/P.} \qquad \textbf{(3.3.1.18)}$$

Así, su complejidad será un $\mathbf{O(m^2/P)}$ **(3.3.1.19)**

Finalmente para determinar la calidad de clasificación del criterio empleado para construir las CE, se realiza una operación de reducción (Suma) paralela (all_to_one_reduction), donde el procesador p0 recibe la información de los objetos presentes en la aproximación inferior (t) y determina el parámetro de calidad. Este proceso tomaría un tiempo paralelo $\mathbf{T_0p_{cal} = P(t_s+t_w)}$. **(3.3.1.20)**

La complejidad computacional presente en el algoritmo paralelo A comprende a las etapas 1 y 2 (distribución de los objetos y construcción de las CE según el criterio establecido; 3.3.1.5 y 3.3.1.7), 2.1 (mezcla y reducción de las CE; 3.3.1.9 y 3.3.1.13), 2.2 (difusión de las CE; 3.3.1.15) y 3 (construcción de las aproximaciones; 3.3.1.17 y 3.3.1.19).

Esto es:

$$\mathbf{O(m^2)+ O((m/P)^2l)+O(\log_2 P(m/P))+O((m/P)^2l)+ O(m)+O(m)+O(m^2/P))}$$

Descartando los $\mathbf{O(m)}$ quedaría:

$$\mathbf{O(m^2)+ O((m/P)^2l)+O(\log_2 P(m/P))+O((m/P)^2l)+ O(m^2/P))}$$

Como la complejidad del $\mathbf{O(\log_2 P(m/P))}$ es mucho menor que las demás la complejidad queda reducida a:

$$\mathbf{O(m^2)+O(2[(m/P)^2l])+ O(m^2/P)} \qquad \textbf{(3.3.1.21)}$$

Por otro lado el tiempo total paralelo empleado será:

$$\mathbf{Tp_A = T_0 + T_{eCE} + Tep_{mezcla/reducción} + Tep_{dif\_CE} + Tep_{dif\_K} + T_{aprox\_k} + T_0p_{cal}}$$

$$\mathbf{Tp_A=\{P(t_s+t_w)+m(t_s+t_wl)+m^2\}+\{(m/P)^2l\}+\{\log_2(P)[ts+(m/P)tw]+(m/P)^2\}+\{t_s+mt_w\}+}$$
$$\mathbf{\{t_s+mt_w\}+\{m^2/P)\}+\{P(t_s+t_w)\}}$$

Reagrupando los términos secuenciales y paralelos:

$$\mathbf{Tp_A=\{m^2+(m/P)^2l+(m/P)^2+m^2/P)\}+\{P(t_s+t_w)+m(t_s+t_wl)+\log_2(P)[ts+(m/P)tw]+}$$
$$\mathbf{t_s+mt_w+t_s+mt_w+P(t_s+t_w)\}}$$

$$\mathbf{Tp_A = \{m^2+(m/P)^2l+(m/P)^2+m^2/P)\}+\{t_s(2P+m+\log_2(P)+2)\}+ \{t_w(2P+ml+ \log_2(P)m/P)+2m)\}} \quad \textbf{(3.3.1.22)}$$

3.3.1.2) Análisis de algunas métricas de rendimiento

Dos de las métricas mas importantes que miden el rendimiento de un programa paralelo son S (Speedup) y E (Eficiencia), ambas definidas en el capítulo 2.

S expresa la ganancia obtenida al paralelizar una aplicación dada (serie) sobre la implementación paralela.

La expresión que describe esta ganancia es: $S = \frac{Ts}{Tp}$

De acuerdo a 3.2.15, $\boldsymbol{Tsec} = \mathbf{2m^2l + m^2}$ siendo este el valor de Ts.

Y, $\mathbf{Tp_A = \{m^2+(m/P)^2l+(m/P)^2+m^2/P)\}+\{t_s(2P+m+\log_2(P)+2)\}+ \{t_w(2P+ml+ \log_2(P)m/P)+2m)\}}$

Por lo cual:

$$S_A = \frac{Ts}{Tp_A} = \frac{2m^2 l + m^2}{\{m^2 + \left(\frac{m}{P}\right)^2 l + \left(\frac{m}{P}\right)^2 + \left(\frac{m^2}{P}\right)\} + \{ts(2P + m + \log_2 P + 2)\} + + \{t_w(2P + ml + \left(\frac{m}{P}\right)\log_2 P + 2m\}}$$

Reduciendo esta expresión adecuadamente y analizando el comportamiento de la misma cuando ***m*** tiende al infinito (∞), nos queda:

$$S_A = \frac{Ts}{Tp_A} = \frac{2P^2}{\frac{P^2 + P + 1}{l} + 1} < 1 \qquad \textbf{(3.3.1.22)}$$

De esta expresión se observa que no existe ganancia sobre la versión serial.

E expresa en que medida los elementos de procesamiento han sido usados al máximo de su capacidad (100%) durante la ejecución de un algoritmo paralelo.

La expresión que describe esta eficiencia es:

$$E_A = \frac{S_A}{P} = \frac{\frac{2P^2}{\frac{P^2+P+1}{l}+1}}{P} = \frac{2P}{\frac{P^2+P+1}{l}+1} \approx 0 \qquad \textbf{(3.3.1.24)}$$

De las expresiones anteriores se observa que este algoritmo paralelo A no mejora el rendimiento respecto a su similar secuencial.

3.3.2) Algoritmo Paralelo "B"

Con el objetivo de encontrar un algoritmo más eficiente que el anterior se propuso un algoritmo similar al anterior (A) al que solo se le realizaron modificaciones al proceso de mezcla con el objetivo de reducir su costo (Anexo 10).

En el algoritmo empleado en esta etapa los procesadores pares comienzan a distribuir los datos a sus respectivos impares de manera similar a la variante anterior, lo que este proceso solo se realiza una sola vez. Posteriormente se realiza el proceso de mezcla que tiene como característica que solo participan los procesadores impares, repitiéndose el proceso de mezcla m-1 veces (nm=log(P-1)-1). Este proceso de mezcla también incluye el proceso de reducción que limita la cantidad de procesadores participantes en cada iteración. El algoritmo que describe la etapa de mezcla y reducción modificada se muestra en el anexo 11.

3.3.2.1) Análisis de la complejidad y cálculo del tiempo paralelo

<u>Etapa 2.1:</u> Mezcla de las CE hacia el procesador 1.

En esta solo intervienen los procesadores esclavos, donde esta etapa se repetirá ***nm-1*** veces ($\log_2(P)$ -1)) (líneas 1 a la 14). En cada una de estas etapas interviene además un proceso de especial encargado de reducir la cantidad de procesadores participantes (Pe) en una razón de P-2 como mínimo (Proceso de reducción) por cada vez.

Cada mezcla involucra el envío o recepción en los procesadores correspondientes de las CE (c*) definidas en array_CE*. Este proceso involucra el envío de un bloque de tamaño m_3, por lo que el tiempo asociado a este sería $T_ep_{envio/recep}= t_s + t_w m_3$, donde m_3 puede ser a lo sumo m/P (c*).

De esta forma $\mathbf{T_ep_{envio/recep^*}= t_s + (m/P)t_w}$, lo que implica una complejidad **O(m/P)** por mezcla.

Ya que se realizan $\log_2 P-1$ mezclas, el tiempo empleado en todas las etapas de envío-recepción será: $\mathbf{T_ep_{envio/recep}= [(\log_2 P)-1](t_s + (m/P)t_w)}$. **(3.3.2.1)**

Lo que implica una con una complejidad $\mathbf{O([(\log_2 P)-1](m/P))}$. **(3.3.2.2)**

Este proceso implica además la búsqueda, construcción o actualización de cada CE recibida ($\mathbf{c_r^*}$) en el procesador (Función ***mezc_CE***), respecto a las CE existentes en el mismo (**c***) lo cual se realiza de forma serie en cada procesador.

Ya que $\mathbf{c_r^*=c^*=m/P}$ en el caso extremo, tenemos que para CE formadas por objetos de **l** atributos, el tiempo empleado será: $\mathbf{T_{e\_mezcla}= (m/P)^2 l}$. **(3.3.2.3)**

Este proceso implica una complejidad del orden $\mathbf{O((m/P)^2 l)}$. **(3.3.2.4)**

Así, de 3.3.2.1 y 3.3.2.3 tenemos que:

$$\mathbf{T_ep_{mezcla/reducción} = [\log_2(P)-1][ts+(m/p)tw] + (m/p)^2 l} \quad \textbf{(3.3.2.5)}$$

Con una complejidad según 3.3.2.2 y 3.3.2.4 del orden:

$$\mathbf{O([(\log_2 P)-1](m/P)) + O((m/P)^2 l) \approx O((m/P)^2 l)} \quad \textbf{(3.3.2.6)}$$

Como el resto de las etapas son similares al algoritmo A su tiempo paralelo será:

$$\mathbf{Tp_B= \{m^2+(m/P)^2 l+(m/P)^2+m^2/P)\}+\{t_s(2P+m+[(\log_2(P))-1]+2)\}+}$$

$$\mathbf{\{t_w(2P+ml+ [(\log_2(P))-1]m/P+2m)\}} \quad \textbf{(3.3.2.7)}$$

Realizando un análisis de complejidad similar al realizado en el capítulo anterior encontramos que su complejidad es la misma que la expresada en 3.3.1.21.

Por lo cual la complejidad computacional presente en el algoritmo B es:

$$\mathbf{O(m^2)+O(2[(m/P)^2 l])+ O(m^2/P)} \quad \textbf{(3.3.2.8)}$$

3.3.2.2) Análisis de algunas métricas de rendimiento

S expresa la ganancia obtenida al paralelizar una aplicación dada (serie) sobre la implementación paralela.

La expresión que describe esta ganancia es: $S = \frac{Ts}{Tp}$

De acuerdo a 3.2.15, ***Tsec*** = $\mathbf{2m^2l + m^2}$ siendo este el valor de Ts.

Y, $\mathbf{Tp_B = \{m^2+(m/P)^2l+(m/P)^2+m^2/P)\}+\{t_s(2P+m+[(\log_2(P))-1]+2)\}+}$

$\mathbf{\{t_w(2P+ml+[(\log_2(P))-1]m/P+2m)\}}$

$$S_B = \frac{Ts}{Tp_B} = \frac{2m^2 l + m^2}{\{m^2 + \left(\frac{m}{P}\right)^2 l + \left(\frac{m}{P}\right)^2 + \left(\frac{m^2}{P}\right)\} + \{t_s(2P + m + [(\log_2 P) - 1] + 2)\} + \\ + \{t_w(2P + ml + \left(\frac{m}{P}\right)[(\log_2 P) - 1] + 2m\}}$$

Reduciendo esta expresión adecuadamente y analizando el comportamiento de la misma cuando ***m*** tiende al infinito (∞), nos queda:

$$S_B = \frac{Ts}{Tp_B} \approx \frac{2P^2}{\frac{P^2 + P + 1}{l} + 1} \quad \mathbf{< 1} \qquad \mathbf{(3.3.2.9)}$$

De esta expresión se observa que no existe ganancia sobre la versión serial y su comportamiento es similar al algoritmo A.

E expresa en que medida los elementos de procesamiento han sido usados al máximo de su capacidad (100%) durante la ejecución de un algoritmo paralelo.

La expresión que describe esta eficiencia es:

$$E_B = \frac{S_B}{P} = \frac{\frac{2P^2}{\frac{P^2 + P + 1}{l} + 1}}{P} = \frac{2P}{\frac{P^2 + P + 1}{l} + 1} \quad \mathbf{\approx\ 0} \qquad \mathbf{(3.3.2.10)}$$

De las expresiones anteriores se observa que este algoritmo paralelo B tampoco mejora el rendimiento respecto a su similar secuencial.

3.3.3) Algoritmo Paralelo "C"

Este algoritmo solo emplea técnicas paralelas en la tercera etapa con el objetivo de eliminar el costo que aporta la etapa de mezcla y reducción, la cual es la más costosa de todas.

Con ello se minimiza el costo de envío de datos a los distintos procesadores de la red virtual, eliminando el paralelismo en las 2 primeras etapas del proceso. En esta implementación el procesamiento de los datos y la construcción de las clases de equivalencia serán realizados por un solo procesador (p0), empleando algoritmos secuenciales (Figura 3.1). Ya que el procesador p0 construyó todas las *CE* posibles a partir del criterio seleccionado (Etapas 1 a 2.1) de manera serie, para emplear el paralelismo en las últimas etapas se debe comenzar por difundirlas (one_to_all_broadcast) al resto de los procesadores.

Finalmente a cada procesador (incluyendo al procesador maestro p0), se les distribuyó el proceso de cálculo de la aproximación, creando cada uno de ellos el archivo solución correspondiente (Etapa 3). Al algoritmo que interviene en la etapa 2.2 se le realizó una ligera modificación y es el que se muestra en la Figura 3.4.

Etapa 2.2: Difusión de las CE hacia todos los procesadores.

```
{Inicio del proceso de Difusión}
1. Si es el procesador p0 {Procesador con idp=0}
2.   enviar sus CE al resto de los procesadores;  {array_CE}
            {One to all BCast}
3.   sino recibir las CE del procesador p0
         {Resto de los procesadores};
{Fin del proceso de Difusión}
```

Figura 3.4: Etapa de difusión de las CE modificada, Algoritmo Paralelo (Variante C).

Una vez que cada procesador tiene todas las *CE*, el procesador p0 debe enviar la información de las clases presentes en el SInf. Para ello emplea también una operación de difusión (líneas 37 a la 39 de la etapa 3 en los algoritmos A y B). Posteriormente se construyen las aproximaciones de las clases correspondientes a cada procesador. Estos dos procesos son similares a los empleados en los algoritmos anteriores.

3.3.3.1) Análisis de la complejidad y cálculo del tiempo paralelo

Etapas 1 y 2: Procesamiento de los objetos del Sinf y Construcción de las CE de acuerdo al criterio establecido.

En este algoritmo el procesamiento de la información (Sinf) y la construcción de las *CE* es realizado por el procesador p0, empleando algoritmos seriales o secuenciales.

De la tabla 3.1, podemos observar que la complejidad computacional de estos procesos es: $\mathbf{O(m^2) + O(lm^2)}$. **(3.3.3.1)**

Así el tiempo empleado en estos procesos es: $\mathbf{Ts_0 = m^2 + lm^2}$. **(3.3.3.2)**

Los procesos de difusión de las CE (etapa 2.2), de las distintas k clases presentes en el SInf, así como el proceso de construcción de las aproximaciones (etapa 3) se realizarán en paralelo.

Esta etapa persigue que todos los procesadores presentes tengan una copia idéntica de las *CE* formadas. Para ello desde el procesador p0 deben difundirse todas sus *CE* (one-to-all-broadcast), lo cual implica el envío de un bloque de tamaño m_4 (array_CE) (líneas 34 a 36), cuyo tamaño como máximo será **m**.

En esta el tiempo paralelo empleado será: $\mathbf{T_0p_{dif\_CE} = t_s + mt_w}$. **(3.3.3.3)**

Así, su complejidad será un $\mathbf{O(m)}$. **(3.3.3.4)**

Etapa 3: Construcción de las Aproximaciones.

Como la información de las k clases presentes en el SInf se encuentran en el procesador p0, este debe difundirlas al resto (one-to-all-broadcast), lo cual implica el envío de un bloque de tamaño m_5 (*array_class*) cuyo tamaño como máximo será **m**.

El tiempo paralelo empleado en esta difusión será: $\mathbf{T_ep_{dif\_K} = t_s + mt_w}$. **(3.3.3.5)**

Así, su complejidad será un $\mathbf{O(m)}$. **(3.3.3.6)**

Una vez terminada esta difusión, cada procesador calcula las aproximaciones correspondientes a sus clases (Función ***const_aprox_par***) (línea 40). Este proceso se realizará en cada procesador, **K/P** veces como máximo, sobre todas

las CE, cuyo número será como máximo **m** en un tiempo paralelo empleado cuya expresión será: $\mathbf{T_{aprox\_k} = m(K/P).}$

Para el peor caso donde cada objeto pertenece a una clase distinta (K=m),

$$\mathbf{T_{aprox\_k} = m^2/P.} \qquad \textbf{(3.3.3.7)}$$

Así, su complejidad será un $\mathbf{O(m^2/P)}$ **(3.3.3.8)**

Finalmente para determinar la calidad de clasificación del criterio empleado para construir las CE, se realiza una operación de reducción (Suma) paralela (all_to_one_reduction), donde el procesador p0 recibe la información de los objetos presentes en la aproximación inferior (t) y determina el parámetro de calidad.

Este proceso tomaría un tiempo paralelo $\mathbf{T_0p_{cal} = P(t_s+t_w)}$. **(3.3.3.9)**

Y su complejidad será del orden: $\mathbf{O(m)}$ **(3.3.3.10)**

La complejidad computacional de esta etapa 3 abarca las comprendidas en las expresiones 3.3.3.1, 3.3.3.4, 3.3.3.6, 3.3.3.8 y 3.3.3.10, que son:

$$\mathbf{O(m^2) + O(lm^2)+ O(m)+ O(m) +O(m^2/P) + O(P)}$$

Descartando las complejidades $\mathbf{O(m) + O(P)}$ por ser de orden inferior a las demás nos queda:

$$\mathbf{O(m^2) + O(lm^2) + O(m^2/P)} \qquad \textbf{(3.3.3.11)}$$

Finalmente el tiempo consumido por este algoritmo (C) involucra a los tiempos definidos en 3.3.3.2, 3.3.3.3, 3.3.3.5 y 3.3.3.7.

$$\mathbf{Tp_c = Ts_0 + T_0p_{dif\_CE} + T_ep_{dif\_K} + T_{aprox\_k} + T_0p_{cal}}$$

$$\mathbf{Tp_c = m^2 + lm^2 + 2(t_s+mt_w) + P(t_s+t_w)} \qquad \textbf{(3.3.3.10)}$$

3.3.3.2) Análisis de algunas métricas de rendimiento

S expresa la ganancia obtenida al paralelizar una aplicación dada (serie) sobre la implementación paralela.

La expresión que describe esta ganancia es: $S = \frac{Ts}{Tp}$

De acuerdo a 3.2.15, $\boldsymbol{Tsec} = \mathbf{2m^2l + m^2}$, siendo este el valor de Ts.

Y, $\mathbf{Tp_c = m^2 + lm^2 + 2(t_s+mt_w) + P(t_s+t_w) = m^2 + lm^2 + t_s(2+P) + t_w(2m+P)}$

Por lo cual:

$$S_C = \frac{Ts}{Tp_C} = \frac{2m^2l + m^2}{m^2 + m^2l + t_s(2+P) + t_w(2m+P)} \qquad \textbf{(3.3.3.10)}$$

Reduciendo esta expresión adecuadamente y analizando el comportamiento de la misma cuando ***m*** tiende al infinito (∞), nos queda:

$$S_C = \frac{Ts}{Tp_C} = \frac{2P + \frac{P}{\boldsymbol{l}}}{P + \frac{P}{\boldsymbol{l}}} = \frac{P + 2P\boldsymbol{l}}{P + P\boldsymbol{l}} \textbf{ > 1} \qquad \textbf{(3.3.3.11)}$$

De esta expresión se observa que existe una pequeña ganancia sobre la versión serial y mejora significativamente su ganancia respecto a los algoritmos anteriores.

E expresa en que medida los elementos de procesamiento han sido usados al máximo de su capacidad (100%) durante la ejecución de un algoritmo paralelo.
La expresión que describe esta eficiencia es:

$$E_C = \frac{S_C}{P} = \frac{\frac{P + 2P\boldsymbol{l}}{P + P\boldsymbol{l}}}{P} = \frac{P + 2P\boldsymbol{l}}{P^2 + P^2\boldsymbol{l}} \textbf{ > 0} \qquad \textbf{(3.3.3.12)}$$

Por lo que se observa una mejoría con la influencia al concentrar la mayoría de los procesos seriales costosos en un solo procesador, pero no realiza un empleo de las técnicas paralelas en todas las etapas del proceso.

3.3.4) Algoritmo Paralelo "D"

Como se ha tratado en el presente capítulo, el costo computacional de los algoritmos paralelos se hace significativo en las etapas de reducción y mezcla (Algoritmo A y B), y no tanto en la comunicación entre los distintos procesadores realizada en las etapas de difusión y obtención de los resultados finales. Por otra parte el algoritmo C no emplea técnicas paralelas en todas las etapas del proceso. Debido a estos inconvenientes se desarrolló un nuevo algoritmo que minimizará las dificultades de las dos primeras variantes y empleará técnicas de procesamiento paralelo en todo el proceso a expensas de aumentar los

procesos de comunicación entre los procesadores. A pesar de este último inconveniente no es significativo este costo adicional frente a los procesos secuenciales que aún están presentes en esta nueva propuesta de algoritmo, a la cual se le denominó variante D.

Esta nueva variante D emplea algoritmos paralelos en 2 etapas particulares a las cuales se les denominó D1 y D2.

En la etapa D1 se construyen clases de equivalencia (*CE*) conformadas según el criterio establecido, donde el procesador p0 difunde cada objeto hacia el procesador en el que esté la clase de equivalencia a la que el mismo pertenece.

De esta manera se eliminan la etapa de mezcla y reducción de las versiones A y B, y se distribuye la construcción de las clases de equivalencia con respecto a la variante C en la que solo participaba el procesador p0. Como resultado de esta etapa se encuentran distribuidas en cada uno de los procesadores (excepto en p0) las clases de equivalencia.

Seguidamente en la etapa D2 se procede a construir las aproximaciones. Esta segunda etapa la inicia p0 ordenando a cada procesador el cálculo de la aproximación para la clase indicada. Una vez que cada procesador realiza los cálculos, envía sus resultados, en este caso la aproximación superior e inferior parcial del conjunto al procesador p0, el que se encargará de procesarlos para obtener la aproximación superior e inferior total para la clase en cuestión, así como sus parámetros de precisión y calidad de la clasificación.

Finalmente p0 crea el archivo solución determinando la calidad de clasificación del SInf según el criterio de equivalencia establecido.

El algoritmo que describe este proceso es el que se muestra en los anexos 12, 13 y 14.

3.3.4.1) Análisis de la complejidad y cálculo del tiempo paralelo

Para realizar el análisis de este algoritmo analizaremos las 2 etapas fundamentales que intervienen en el mismo de forma independiente, como se realizó para el resto de los algoritmos.

<u>Etapa D1:</u> Procesamiento de la información y construcción de las CE.

<u>Procesador Maestro (p0):</u>

Este procesador inicialmente notifica la cantidad de objetos presentes en el SInf (m*) al resto de los procesadores esclavos para lo cual le tomaría un tiempo $T_0p_{m^*} = t_s + t_w m_1$ (one_to_all_broadcast). Ya que m_1 es *constante* no se considera en la expresión del tiempo.

De esta forma: $\mathbf{T_0p_{m^*} = t_s + t_w}$ (3.3.4.1)

Lo cual presenta una complejidad del orden $\mathbf{O(1)}$. (3.3.4.2)

El proceso de envío de cada objeto del SInf (m) a los procesadores esclavos correspondientes (línea 7) que se describe en las líneas de la 6 a la 11, le tomaría un tiempo $T_{0n}=t_s + t_w m_2$ por cada objeto n, dependiendo m_2 del número de atributos (**l**) de n.

Esto implica un tiempo total paralelo para m objetos definido por la expresión: $\mathbf{T_0p_m=m(t_s+t_wl)}$. (3.3.4.3)

Con una complejidad del orden: $\mathbf{O(m)*O(l)} = \mathbf{O(lm)}$. (3.3.4.4)

Otra de las funciones de este procesador es realizar el proceso de descubrimiento de cada clase (k) distinta presente en SInf (línea 8) de manera similar al algoritmo secuencial, examinando cada objeto n. Al tiempo empleado en este proceso se le llama T_0c_k.

Este proceso (definido en ***info_clase***) se le realiza a cada objeto presente en el SInf (***m***) el cual presenta una complejidad computacional de un $\mathbf{O}(\boldsymbol{Km})$ de acuerdo a 3.2.2 y pudiera ser como máximo de un orden $\mathbf{O}(\boldsymbol{m^2})$ según 3.2.3.

Así, $\mathbf{T_0c_k=m^2}$ (3.3.4.5) y puede ser definido de un orden $\mathbf{O(m^2).}$ (3.3.4.6)

El envío de cada objeto n produce que cada esclavo busque su presencia en alguna de sus CE y notifique el resultado de su búsqueda al procesador p0. Este proceso permite al procesador conocer de la existencia de la CE a la que pertenece n por medio de una operación de reducción (all_to one_reduce) empleando un tiempo: $T_0p_{existCE} = P(t_s+t_w m_3)$, donde m_3 es una constante por lo que $\mathbf{T_0p_{existCE} = P(t_s+t_w)}$. (3.3.4.7)

La complejidad asociada al mismo será entonces de un orden: $\mathbf{O(P)}$ (3.3.4.8)

Para evitar que existan finalmente CE repetidas en cada procesador se implementa un mecanismo de comunicación entre el procesador maestro y el esclavo que debe construir la CE a la que pertenece n. Esta tendría lugar cuando ningún procesador esclavo notifique la presencia de la CE a la que pertenece n según el criterio establecido por lo que el procesador p0 le notificaría al procesador indicado, según una distribución especifica, que adicione o construya la CE para el objeto n difundido anteriormente.

El tiempo que toma el envío de esta notificación será: $T_{0notifCE^*} = t_s + t_w m_4$, ($m_4$ es una constante) donde para el peor caso (todos los objetos de n son distintos) existirá una CE diferente en todo momento en los procesadores esclavos.

Basado en esta situación deberán enviarse m mensajes en total, siendo ahora el tiempo empleado: $\mathbf{T_{0notifCE} = m(t_s + t_w)}$, (3.3.4.9) con una complejidad del orden $\mathbf{O(m/P)}$. (3.3.4.10)

De esta forma el tiempo empleado por el procesador p0 es:

$$\mathbf{T_{0D1} = T_0p_{m^*} + T_0p_m + T_0c_k + T_0p_{existCE} + T_{0notifCE}}$$

$$\mathbf{T_{0D1} = t_s + t_w + m(t_s + t_w l) + m^2 + P(t_s + t_w) + m(t_s + t_w)} \quad \mathbf{(3.3.4.11)}$$

De acuerdo a 3.3.4.2, 3.3.4.4, 3.3.4.6, 3.3.4.8 y 3.3.4.10 la máxima complejidad computacional esperada para esta etapa en p0 será:

$$\mathbf{O(1) + O(lm) + O(m^2) + O(mP) + O(m/P) \approx O(m^2)} \quad \mathbf{(3.3.4.12)}$$

Procesadores Esclavos:

Con cada difusión de los **m** objetos del SInf los procesadores esclavos deben realizar una búsqueda y adicionalmente una posible actualización de sus CE si alguna de ellas contiene ya al objeto **n** enviado.

Este proceso de búsqueda será realizado en un tiempo serie definido por $T_e p_{busqCE}$, considerando que a lo sumo habrá **m/P** CE en cada procesador.

Así, $\mathbf{T_e p_{busqCE} = (m/P)^2 l}$, (3.3.4.13) adicionando una complejidad computacional del orden $\mathbf{O((m/P)^2 l)}$. (3.3.4.14)

Considerando que cada objeto es diferente habrá que construir una nueva CE por cada uno de ellos, adicionando como resultado de las notificaciones **m/P**

nuevas CE. El tiempo empleado en este proceso es: $\mathbf{T_{e}p_{constCE} = m/P}$, (3.3.4.15) realizándose con una complejidad $\mathbf{O(m/P)}$. (3.3.4.16)

De esta forma el tiempo empleado por los procesadores esclavos sera:

$$\mathbf{T_{eD1} = T_{e}p_{busqCE} + T_{e}p_{constCE} = (m^2l)/P + m/P} \quad \mathbf{(3.3.4.17)}$$

De acuerdo a 3.3.4.14 y 3.3.4.16 la máxima complejidad computacional esperada para esta etapa en p0 será:

$$\mathbf{O((m^2l)/P).+ O(m/P) \approx O((m^2l)/P)} \quad \mathbf{(3.3.4.18)}$$

<u>Etapa D2:</u> Construcción de las Aproximaciones.

<u>Procesador Maestro (p0):</u>

Al concluir la etapa precedente cada procesador tiene a lo sumo **m/P** CE distintas. Por lo que el procesador p0 debe difundir el total de clases **(K)** presentes en el SInf (a lo sumo **K=m**) para que todos los procesadores esclavos participen en el cálculo de las aproximaciones.

Para ello previamente el procesador p0 indica al resto el número de clases presentes (K). Para ello emplea un tiempo $T_0p_{K^*} = t_s+t_wm_5$, siendo m_5 un valor constante.

Así, $\mathbf{T_0p_{difK^*} = t_s+t_w}$, (3.3.4.19) el cual tiene una complejidad de orden $\mathbf{O(1)}$. (3.3.4.20)

Una vez que se han notificado el total de clases presentes (a lo sumo m) se comienza a difundir cada clase por el procesador p0 al resto de los esclavos.

Este proceso de difusión implicaría un tiempo $T_0p_{K^*}= m(t_s+t_wm_6)$, donde m_6 es un valor constante, por lo cual $\mathbf{T_0p_K = m(t_s+t_w)}$, (3.3.4.21) con una complejidad $\mathbf{O(m)}$ (3.3.4.22).

De esta forma el tiempo empleado por el procesador p0 en la etapa D2 es:

$$\mathbf{T_{0D2} = T_0p_{difK^*} + T_0p_K = t_s+t_w + m(t_s+t_w)}. \quad \mathbf{(3.3.4.22)}$$

De acuerdo a 3.3.4.20 y 3.3.4.22 la máxima complejidad computacional esperada para esta etapa en p0 será:

$$\mathbf{O(1) + O(m) \approx O(m)} \quad \mathbf{(3.3.4.23)}$$

Procesadores Esclavos:

Por cada clase recibida desde el procesador p0 se realizará la construcción de las aproximaciones. Para este proceso cada procesador esclavo debe recorrer a los sumo **m/P** CE, lo cual implica un tiempo $\mathbf{T_{eD2}= (m/P)^2 l}$, **(3.3.4.24)** con una complejidad del orden $\mathbf{O((m/P)^2 l)}$ **(3.3.4.25).**

Finalmente el tiempo consumido por este algoritmo (D) involucra a los tiempos definidos en 3.3.4.11, 3.3.4.17, 3.3.4.22 y 3.3.4.24. .

$$\mathbf{Tp_D = T_{0D1} + T_{eD1} + T_{0D2} + T_{eD2}}$$

$$\mathbf{Tp_D = t_s + t_w + m(t_s + t_w l) + m^2 + P(t_s + t_w) + m(t_s + t_w) + (m/P)^2 l + m/P + t_s + t_w + m(t_s + t_w) + (m/P)^2 l}$$

$$\mathbf{Tp_D = m^2 + 2(m/P)^2 l + m/P + t_s(2 + 3m + P) + t_w(2 + ml + P + 2m)} \quad \mathbf{(3.3.4.26)}$$

Con una complejidad presente en el algoritmo D según 3.3.4.12, 3.3.4.18, 3.3.4.23 y 3.3.4.25 del orden: $\mathbf{O(m^2) + O(2(m/P)^2 l) + O(m)}$. Descartando la complejidad O(m) quedaría finalmente:

$$\mathbf{O(m^2) + O(2(m/P)^2 l)} \quad \mathbf{(3.3.4.27)}$$

3.3.4.2) Análisis de algunas métricas de rendimiento

S expresa la ganancia obtenida al paralelizar una aplicación dada (serie) sobre la implementación paralela.

La expresión que describe esta ganancia es: $S = \frac{Ts}{Tp}$

De acuerdo a 3.2.15, $\boldsymbol{Tsec} = \mathbf{2m^2 l + m^2}$ siendo este el valor de Ts.

Y, $\mathbf{Tp_D = m^2 + 2(m/P)^2 l + m/P + t_s(2 + 3m + P) + t_w(2 + ml + P + 2m)}$

Por lo cual:

$$S_D = \frac{Ts}{Tp_D} = \frac{2m^2 l + m^2}{m^2 + \frac{2m^2 \boldsymbol{l}}{\boldsymbol{P}^2} + \frac{m}{P} + \boldsymbol{t_s}(2 + 3\boldsymbol{m} + \boldsymbol{P}) + \boldsymbol{t_w}(2 + \boldsymbol{ml} + \boldsymbol{P} + 2\boldsymbol{m})} \quad \mathbf{(3.3.4.28)}$$

Reduciendo esta expresión adecuadamente y analizando el comportamiento de la misma cuando ***m*** tiende al infinito (∞), nos queda:

$$S_D = \frac{Ts}{Tp_D} = \frac{2(P^2 \boldsymbol{l} + P^2)}{P^2 + 2\boldsymbol{l}} \quad \mathbf{(3.3.3.29)}$$

De esta expresión se observa que existe una ganancia sobre la versión serial y mejora significativamente su ganancia respecto a los algoritmos anteriores.

E expresa en que medida los elementos de procesamiento han sido usados al máximo de su capacidad (100%) durante la ejecución de un algoritmo paralelo.

La expresión que describe esta eficiencia es:

$$E_D = \frac{S_D}{P} = \frac{\dfrac{2\left(P^2\boldsymbol{l} + P^2\right)}{P^2 + 2\boldsymbol{l}}}{P} = \frac{2\left(P^2\boldsymbol{l} + P^2\right)}{P^2 + P2\boldsymbol{l}} \quad \mathbf{> 0} \qquad \mathbf{(3.3.3.11)}$$

Por lo que se observa existe una mejoría con la influencia de emplear técnicas paralelas en todo el proceso distribuyendo los procesos seriales entre todos los procesadores.

CAPÍTULO IV - Análisis y evaluación de los resultados brindados por los algoritmos propuestos.

En el presente capítulo se abordarán los resultados obtenidos por cada uno de los algoritmos, los cuales fueron implementados sobre el cluster de computadoras del Centro de Estudios Informáticos (CEI) de la Universidad Central "Marta Abreu" de Las Villas.

4.1) Análisis comparativo de los algoritmos desarrollados.

En el capítulo anterior se obtuvieron las expresiones que describen las complejidades de cada algoritmo y se determinaron algunas de sus métricas de rendimiento.

En la tabla 4.1 se muestra el análisis de las complejidades presentes en cada etapa para cada uno de los algoritmos tratados.

Algoritmo	**Complejidad Computacional Máxima (O)**		
	E1: Procesamiento de las clases	**E2:** Construcción y/o Mezcla de CE	**E3:**Calculo de las Aproximaciones
Secuencial	$O(m^2)$	$O(m^2l)$	$O(m^2l)$
A	$O(m^2)$	$O(2[(m/P)^2l])$	$O(m^2/P)$
B	$O(m^2)$	$O(2[(m/P)^2l])$	$O(m^2/P)$
C	$O(m^2)$	$O(m^2l)$	$O(m^2/P)$
D	$O(m^2)$	$O(m/P)^2l$	$O(m/P)^2l$

Tabla 4.1 Complejidades computacionales por etapas

En ella podemos observar que el procesamiento de las clases (**E1**) no se paralelizó en ninguno de los algoritmos por lo cual este factor es uno de los que mas incide en las expresiones de los algoritmos paralelos, aunque en casos prácticos el número de clases distintas a las que pertenecen los objetos de un Sistema de Información (SInf) es mucho menor que el número total de objetos.

La etapa correspondiente a la construcción de Clases de Equivalencia (CE) (**E2**) se paralelizó en los algoritmos A, B y D, mostrando resultados superiores en el algoritmo D, al necesitar las propuestas A y B de una etapa de mezcla adicional para evitar la existencia de CE duplicadas, con la presencia de un número alto de comunicaciones entre los procesadores. Esta etapa no se paralelizó en la variante C con el objetivo de reducir su complejidad.

La ultima de las etapas, correspondiente al cálculo de las aproximaciones (**E3**), se paralelizó en todos los algoritmos. Esta presenta una complejidad similar para las variantes paralelas A, B y C, donde se emplea el mismo procedimiento que debe recorrer volúmenes equivalentes de CE en todos los procesadores. El mejor algoritmo para esta etapa fue la variante D, ya que solo debe recorrer un volumen de CE mucho menor al encontrarse las mismas distribuidas en todos los procesadores esclavos formando subconjuntos del número de CE totales presentes en las propuestas anteriores.

En la siguiente tabla 4.2 se resumen las complejidades computacionales totales presentes en cada algoritmo.

Algoritmo	Complejidad Computacional Máxima (O)
Secuencial	$O(2m^2l+m^2)$
A	$O(m^2)+O(2[(m/P)^2l])+ O(m^2/P)$
B	$O(m^2)+O(2[(m/P)^2l])+ O(m^2/P)$
C	$O(m^2) + O(lm^2) + O(m^2/P)$
D	$O(m^2) + O(2(m/P)^2l)$

Tabla 4.2 Complejidades computacionales de los algoritmos propuestos

En el presente resumen se puede observar que solo el algoritmo D presenta una complejidad inferior respecto a la variante secuencial. A su vez no existe

diferencia entre las variantes A y B pues presentan algoritmos similares donde el hecho de eliminar una etapa de mezcla en el algoritmo B no logró mejoras significativas.

Estas diferencias también se observan a través de las distintas expresiones obtenidas de la ganancia y eficiencia de cada algoritmo las cuales se muestran en la tabla 4.3.

Algoritmo	Ganancia (S)	Eficiencia (E) (0< E <1)
A	$\frac{2P^2}{\frac{P^2+P+1}{l}+1} < \mathbf{1}$	$\frac{2P}{\frac{P^2+P+1}{l}+1} \approx \mathbf{0}$
B	$\frac{2P^2}{\frac{P^2+P+1}{l}+1} < \mathbf{1}$	$\frac{2P}{\frac{P^2+P+1}{l}+1} \approx \mathbf{0}$
C	$\frac{P+2Pl}{P+Pl} > \mathbf{1}$	$\frac{P+2Pl}{P^2+P^2l} > \mathbf{0}$
D	$\frac{2(P^2l+P^2)}{P^2+2l} >> \mathbf{1}$	$\frac{2(P^2l+P^2)}{P^3+P2l} >> \mathbf{0}$

Tabla 4.3 Complejidades computacionales de los algoritmos propuestos

En ella podemos observar que los algoritmos A y B no ofrecen ganancia alguna sobre la versión secuencial debido al proceso de mezcla y reducción de CE, donde se intercambia un gran volumen de información entre los procesadores esclavos aumentando el número de mensajes con bloques de información de tamaños grandes. Ello implica que su eficiencia es nula para ambos algoritmos.

Por otro lado el algoritmo C presenta una ligera ganancia respecto a la versión secuencial al emplear técnicas paralelas en la última etapa del proceso aunque su eficiencia es baja por presentar grandes volúmenes de procesos secuenciales en sus dos primeras etapas.

Por último el algoritmo D presenta una ganancia significativamente superior a la versión secuencial empleando técnicas paralelas en todo el proceso, lo cual implica una eficiencia superior respecto a su versión precedente C.

Otro aspecto importante a considerar en las expresiones anteriores es la dimensión de ***I*** (número de atributos de los objetos del SInf) la cual puede influir significativamente en el comportamiento de los algoritmos.

4.2) Análisis comparativo aplicaciones desarrolladas basados en los algoritmos propuestos.

Partiendo de los algoritmos desarrollados se implementaron sus correspondientes aplicaciones sobre una arquitectura monoprocesador (algoritmo secuencial) y sobre el cluster de computadoras del CEI (algoritmos A, B, C y D). Estas tienen la característica de ser programadas en el lenguaje C estándar.

Para validar los resultados de las aplicaciones implementadas basadas en estos algoritmos, se empleó una base de datos de información biológica usada por el Grupo Multidisciplinario de Bioinformática de la UCLV, la cual tiene mas de 15 070 artículos y fue construida a partir de la base de datos Integrated Sequence-Structure Database (ISSD) [41]. Los datos representan cadenas que describen secuencias de aminoácidos. En la base de datos se reconocen 8 posibles estructuras secundarias: H (alpha helix), B (residue in isolate beta-bridge), E (extended strand), G (3-helix), I (S helix), P (polyprotine type II helix), T (hydrogen bonded turn), S (bend), las cuales constituyen el atributo clase.

Para esta base de datos se usó como criterio de inseparabilidad, en el proceso de construcción de las CE, la longitud de las cadenas y el valor. La relación B sobre este sistema de decisión definió una partición formada por 6968 Clases de Equivalencia. El resultado del proceso de construcción de la aproximación

inferior y superior de cada una de las estructuras secundarias se muestra en la tabla 4.3.

	Estructura secundaria							
Aproximaciones	**B**	**E**	**G**	**H**	**I**	**P**	**S**	**T**
Inferior (B_*)	0	2508	601	1951	4	634	397	623
Superior (B^*)	4218	4751	898	1953	4	865	8298	8642
Precisión de la aproximación (α)	0.0000	0.5279	0.6693	0.9990	1.0000	0.7329	0.0478	0.0721
Calidad de la aproximación (γ)	0.0000	0.8577	0.8056	0.9995	1.0000	0.8476	0.1007	0.1606

Tabla 4.3: Cantidad de objetos en los conjuntos B_*, B^*, el valor de (α) y (γ) de cada estructura secundaria

A partir de estos resultados se observa que según la relación de equivalencia (B) establecida, solo para la clase I (α=1) y prácticamente también para la clase H (α =0.9990) el conjunto X es duro o exacto, y es aproximado para el resto de las clases, de las cuales solo G, P y E muestran resultados satisfactorios. Esta relación de equivalencia logra clasificar correctamente a los objetos de las clases H (γ=0.9995) e I (γ=1) y con un nivel aceptable a los objetos de las clases G, P y E.

No obstante la calidad de la clasificación, según la relación de equivalencia (B) establecida para este Sistema de Información, es: γ_B = 0.4458, lo cual indica que B no clasifica a este sistema de información (SInf) de una manera correcta o deseada.

4.2.1) Análisis comparativo de las soluciones empleadas.

Para establecer un análisis comparativo del costo, todas las implementaciones fueron ejecutadas sobre el cluster de computadoras del CEI, el cual está conformado por 1 Servidor Dual Pentium III, 1GHz, RAID de 5 discos, 256 MB

de memoria y 12 estaciones Pentium III, 256 MB, 800 MHz basando su comunicación en una red privada Fast Ethernet (100 Mb/s). El Software empleado se basa en Sistema Operativo Linux Gentoo 2004.3, usando el kernel 2.6.9, gcc 3.3.5, glibc 2.3.4 y mpich 1.2.5.2. Los algoritmos empleados fueron programados usando el estándar de programación MPI [42].

El resultado de la ejecución de estos algoritmos usando desde 2 hasta 10 nodos se muestra en anexo 15 a.

De ellos observamos que solo el algoritmo C y D logran los mejores tiempos, pero para el primero su tiempo comienza a empeorar con el empleo de un mayor número de procesadores desde el mismo comienzo del proceso.

Sin embargo para el algoritmo D el tiempo de ejecución mejora desde el inicio del proceso con el aumento del empleo de mayor cantidad de procesadores. Este comportamiento se mantiene hasta emplear 5 procesadores, a partir del cual comienza a estabilizarse (la disminución del tiempo no es significativa) hasta emplear 7 procesadores, con lo cual se logra el mejor desempeño. A partir de 8 procesadores en adelante comienza a aumentar nuevamente el tiempo empleado. Este comportamiento puede observarse en el anexo 15 b, describiendo una especie de meseta, hecho este referenciado en la “Ley de Amdahl”.

A su vez el algoritmo secuencial implementado logró un tiempo de ejecución de 17.162 seg.

Por otro lado solo el algoritmo D logró una ganancia significativa. Esta llega casi a duplicar (1.972) el tiempo de ejecución con respecto a la variante secuencial para 7 procesadores [43], mientras que los algoritmos A y B no presentan ninguna ganancia y en el algoritmo C, que si presenta una ligera ganancia con pocos procesadores, esta cae desde el comienzo (Anexo 16).

Al analizar la eficiencia se observa que en todos los algoritmos, excepto el D, tiende a disminuir bruscamente con el aumento del número de procesadores. Solo el algoritmo D esta cae suavemente logrando ser de un **28%** cuando se obtiene la máxima ganancia (Anexo 17).

A pesar de haber conseguido parámetros de ganancia y eficiencia satisfactorios en la versión paralela basada en el algoritmo D, esta puede mejorarse aun más si se realizara un análisis de balance que permita emplear al máximo el número de procesadores empleados, comenzando por paralelizar también el proceso de descubrimiento de las clases presentes en el Sistema de Información, que adiciona un término cuadrático (m^2) como se observa en la expresión 3.3.4.26.

CONCLUSIONES

Con el presente trabajo se obtiene como resultado el desarrollo e implementación de algoritmos paralelos al aplicar técnicas de paralelización sobre el algoritmo secuencial que calcula los principales conceptos de esta teoría, permitiendo minimizar el tiempo empleado al procesar Sistemas de Información voluminosos.

Los principales resultados obtenidos en el presente trabajo son:

- Se desarrolló un algoritmo secuencial y se implementó una aplicación basado en el mismo que permitió validar los resultados expuestos en la bibliografía científica.

- Del estudio del algoritmo secuencial se determinaron las tres etapas en que es posible aplicar técnicas de paralelización.

- Se empleó el modelo de paralelización basado en los datos que permitió subdividir el volumen de información entre varios procesadores que ejecutan un código similar en paralelo.

- Se desarrollaron 4 algoritmos paralelos, referenciados como A, B, C y D que dan solución a esta problemática y se implementaron sobre un cluster de computadoras.

- El algoritmo D fue el que mejores resultados presentó en sus parámetros de ganancia, eficiencia y tiempo de ejecución comparándolo con la versión secuencial al procesar el Sistema de Información *Integrated Sequence-Structure Database* (ISSD) empleado por el Grupo Disciplinario de Bioinformática de la UCLV.

RECOMENDACIONES

- Continuar analizando la posibilidad de mejorar el algoritmo D.

- Extender este algoritmo para procesar cualquier tipo de Sistema de Información y abarque las extensiones de la Teoría de los Conjuntos Aproximados.

- Desarrollar una librería con estos algoritmos para que puedan ser reusados en otras aplicaciones del grupo.

REFERENCIAS BIBLIOGRÁFICAS

[1] Pawlak, Z.. Rough sets. International Journal of Information & Computer Sciences 11, 341-356, 1982.

[2] Pal, S.K. and Skowron, A. (Eds).. Rough Fuzzy Hybridization: a new trend in decision-making. Springer-Verlasg, 1999.

[3] Orlowska, E. (Ed.).. Incomplete Information, Rough sets analysis. Physica-Verlag. 1998.

[4] Grzymala-Busse, J.W.. Managing uncertainty in machine learning from examples. Proceedings of the Workshop Intelligent Information Systems III, 6-10 june, Poland, 1994.

[5] Greco, S. Et al.. Rough sets theory for multicriteria decision analysis. European Journal of Operational Research 129, pp. 1-47, 2001.

[6] Kohavi, R. and Frasca, B.. Useful feature subsets and Rough set Reducts. Proceedings of the Third International Workshop on Rough Sets and Soft Computing. 1994.

[7] Koczkodaj, W.W. et al.. Myths about Rough Set Theory. Comm. of the ACM, vol. 41, no. 11, nov. 1998.

[8] Pal, S.K. et al.. Web mining in Soft Computing framework: Relevance, State of the art and Future Directions. IEEE Transactions on Neural Networks, 2002.

[9] Carlin, U. et al.. Rough sets analysis of patients with suspected acute appendicitis. In [10], pp. 1528-1533, 1998.

[10] Bouchon-Meunier, B. and Yager, R.R.. Proc. Seventh Conf. On Information Processing and Management of Uncertainty in Knowledge-based systems (IPMU'98), Paris, Francia, Edisions EDK Paris, 1998.

[11] Furuta, H. et al.. Extraction method based on rough set theory of rule-type knowledge from diagnosis cases of slope-failure danger levels. In [13] pp. 178-192, 1998.

[12] Mrozek, A. and Skabek, K.. Rough sets in economic applications. In [13], pp. 238-271, 1998.

[13] Polkowski, L. and Skowron, A. (Eds.). Rough sets in Knowledge Discovery 1: Applications, Case Studies and Software Systems. Physica-Verlag, 1998.

[14] Czyzewski, A. and Krolikowski, R.. Applications of fuzzy logic and rough sets to audio signal enhancement. In [2], pp. 397-409, 1999.

[15] Peters III, F.J. and Ramanna, S.. A rough set approach to assessing software quality: Concepts and rough Petri net models. In [Pal99], pp. 349-380, 1999.

[16] Tay, F.E. and Shen, L.. Economic and financial prediction using rough set model. European Journal of Operational Research 141, pp. 641-659, 2002.

[17] Dean, J. S. and Grzymala-Busse, J. W. An overview of the learning from examples module LEM1. Deparment of Computer Science, University of Kansas, TR-82, 1988.

[18] Grzymala-Busse, J.W.. LERS- A system for learning from examples based on rough sets. In Slowinski, R. (Ed.) Intelligent Decision Support, Handbook of Applications and Advances of the Rough Sets Theory. Kluwer Academic Publishers, Dordrecht, Boston, London, 3-18, 1992.

[19] Grzymala-Busse, J.W.. The rule induction systems LERS Q: a version for personal computers. In Proceedings of the International Workshop on Rough Sets and Knowledge Discovery, p. 511, 1993.

[20] Grzymala-Busse, J.W.. A new version of the rule induction system LERS. Fundamenta Informaticae 31, pp. 27-39, 1997.

[21] Grzymala-Busse, J.W.. Applications of the rule Induction System LERS, Rough Sets in Knowledge Discovery. (Ed) L. Polkowski and A. Skowron, Physica-Verlag, 1998.

[22] Ohrn, A.. ROSETTA Technical Reference Manual. Department of Computer and Information Science, Norwegian University of Science and Technology, Trondheim, Norway.

[23] The Rosetta WWW homepage: http://www.idi.ntnu.no/~aleks/rosetta/, 2002.

[24] Ohrn, A. and Komorowski, J.. Rosetta: A rough set toolkit for analysis of data. In Proc. Third Int. Join Conference on Information Science, Durham, NC, USA, march 1-5, vol. 3, pp. 403-407. 1997.

[25] Ohrn, A. et al.. Rosetta Part I: System Overview. Technical Report Dept. of Computer and Information Science, Norwegian University of Science and Technology, Trondheim, Norway, 1997.

[26] Bell, D. and Guan, J.. Computational methods for rough classification and discovery. Journal of ASIS 49, 5, pp. 403-414. 1998.

[27] Deogun, J.S. et al.. Feature selection and effective classifiers. Journal of ASIS 49, 5, pp. 423-434. 1998.

[28] Grama, A., Gupta A., Karypis, G. and Kumar, V. Introduction to Parallel Computing, Second Edition. Ed. Adisson Wesley, 2003.

[29] Grama, A. y Kumar, V.. A survey of parallel search algorithms for discrete optimizacion problems. url="citeseer.nj.nec.com/et93survey.html", 1993.

[30] Toward scalable and parallel inductive learning: a case study in splice junction prediction. Working notes of Workshop on Machine Learning and Molecular Biology, 1994.

[31] Provost, F y otros. A survey of methods for scaling up inductive algorithms. Data mining and Knowledge Discovery, vol. 3, no. 2, 131-169. 1999.

[32] Wooley, B y otros. Machine learning using clusters of computer. url="citeseer.nj.nec.com/295989.html". 2000.

[33] Wooley, B y otros. Scaling the data minino step in Knowledge discovery using oceanography data. Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, pp. 85-92, 2000.

[34] Pawlak, Z. et al.. Rough sets. Comm. of ACM, 38(11), pp. 89-95, nov. 1995.

[35] Pawlak, Z.. Vagueness and uncertainty: a rough set perspective. Computational Intelligence vol. 11, no. 2, 1995.

[36] Komorowji, J. et al.. A Rough set perspective on Data and Knowledge. In The Handbook of Data Mining and Knowledge Discovery, W. Klosgen and J. Zytkow (Eds.). Oxford University Press. 1999.

[37] http://clusters.fisica.uson.mx/**,** 2005.

[38] Foster, I. and Kesselman, C. The Grid: Blueprint for a New Computing Infrastructure, 1998.

[39] http://www.top500.org, 2006.

[40] Del Toro, Leonardo y Bello Pérez, Rafael. Una aplicación de los conjuntos aproximados en el análisis de la consistencia de datos en bioinformática, Memorias del VIII Congreso Internacional de Matemática y Computación (COMPUMAT 2003), ISSN – 17286042.

[41] Del Toro, Leonardo, Galvez Lio, Daniel y Bello Pérez, Rafael. Implementación sobre un Cluster de PCs de los conceptos básicos de la teoría de los conjuntos aproximados**,** Memorias de la X Convención Internacional Informática 2004, I Simposio Cubano de Inteligencia Artificial, ISBN-959-237-117-2.

[42] http://www.mpi-softtech.com, 2006

[43] Del Toro, Leonardo, Galvez Lio, Daniel y Bello Pérez, Rafael. Propuesta de Paralelización de la teoría de los conjuntos aproximados (rough sets), Memorias de la XI Convención Internacional, Informática 2005, (VIII Congreso de Nuevas Tecnologías y Aplicaciones Informáticas), ISBN: 959-7164-87-6.

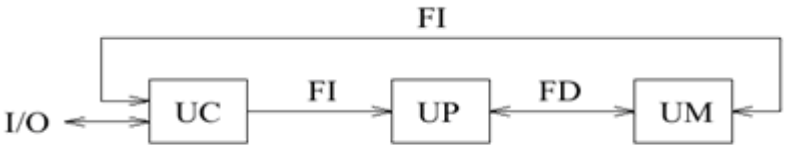
Anexo-1: Los primeros 15 sistemas de cómputo

(http://www.top500.org)

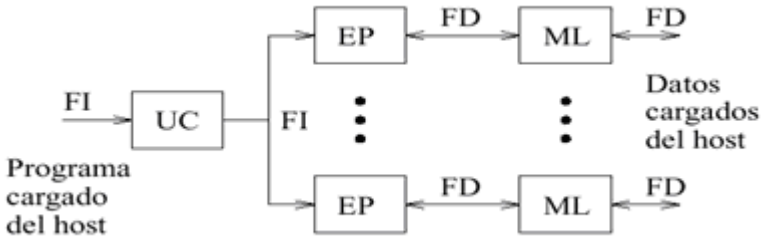
Rank	**Site** **Country/Year**	**Computer / Processors** **Manufacturer**	**R_{max}** **R_{peak}**
1	DOE/NNSA/LLNL United States/2005	*BlueGene/L* **eServer Blue Gene Solution** / 65536 / IBM	136800 183500
2	IBM Thomas J. Watson Research Center, United States/2005	*BGW* **eServer Blue Gene Solution** / 40960 / IBM	91290 114688
3	NASA/Ames Research Center/NAS United States/2004	*Columbia* **SGI Altix 1.5 GHz, Voltaire Infiniband** / 10160 / SGI	51870 60960
4	The Earth Simulator Center Japan/2002	**Earth-Simulator** / 5120 / NEC	35860 40960
5	Barcelona Supercomputer Center, Spain/2005	*MareNostrum* **JS20 Cluster, PPC 970, 2.2 GHz, Myrinet** 4800 / IBM	27910 42144
6	ASTRON/University Groningen, Netherlands/2005	**eServer Blue Gene Solution** / 12288 / IBM	27450 34406.4
7	Lawrence Livermore National Laboratory, United States/2004	*Thunder* **Intel Itanium2 Tiger4 1.4GHz - Quadrics** 4096 / California Digital Corporation	19940 22938
8	Computational Biology Research Center, AIST, Japan/2005	**eServer Blue Gene Solution** / 8192 / IBM	18200 22937.6
9	Ecole Polytechnique Federale de Lausanne, Switzerland/2005	**eServer Blue Gene Solution** / 8192 / IBM	18200 22937.6
10	Sandia National Laboratories United States/2005	**Red Storm, Cray XT3, 2.0 GHz** / 5000 Cray Inc.	15250 20000
11	Oak Ridge National Laboratory United States/2005	**Cray XT3, 2.4 GHz** / 3748 / Cray Inc.	14170 17990
12	Los Alamos National Laboratory United States/2002	*ASCI Q* **ASCI Q - AlphaServer SC45, 1.25 GHz** 8192 / Hewlett-Packard	13880 20480
13	Lawrence Livermore National Laboratory, United States/2005	**eServer pSeries p5 575 1.9 GHz** / 2048 IBM	13090 15564.8
14	Virginia Tech United States/2004	*System X* **1100 Dual 2.3 GHz Apple XServe/Mellanox Infiniband 4X/Cisco GigE** / 2200 Self-made	12250 20240
15	Japan Atomic Energy Research Institute, Japan/2005	**SGI Altix 3700 Bx2, 1.6 GHz, NUMALink** / 2048 / SGI	11814 13107

Nota: **R_{max}** y **R_{peak}** se expresan en GFlops.

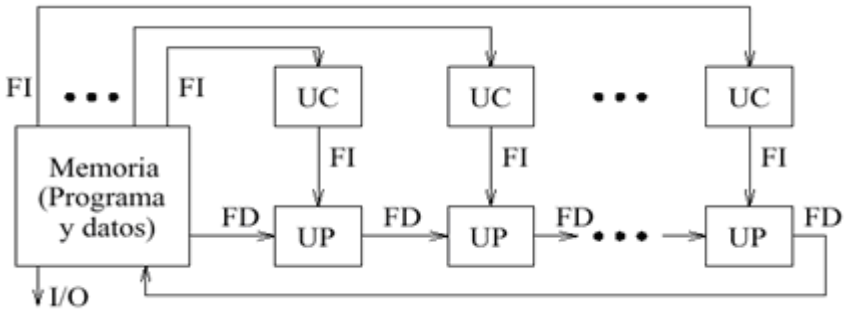
Anexo-2: Categorías de Flynn



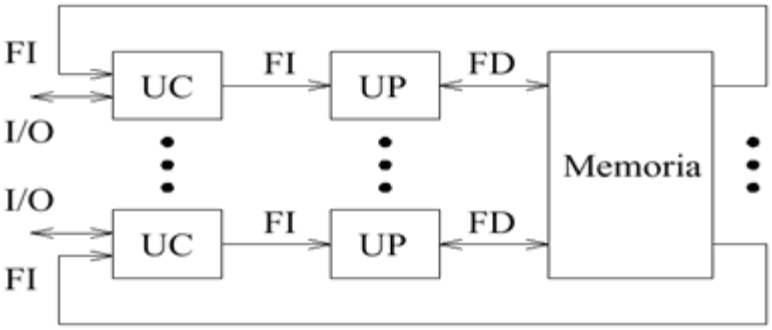
a) SISD (monoprocesador)



b) SIMD (procesador matricial)



c) MISD (array sistólico)



d) SIMD (multiprocesador)

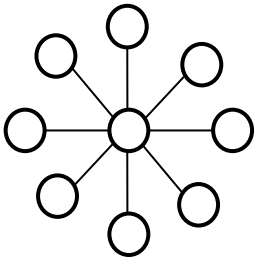
NOTA:

UC: Unidad de Control

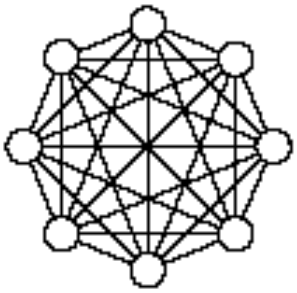
I/O: Operaciones de Entrada/Salida

UP: Unidad de Procesamiento

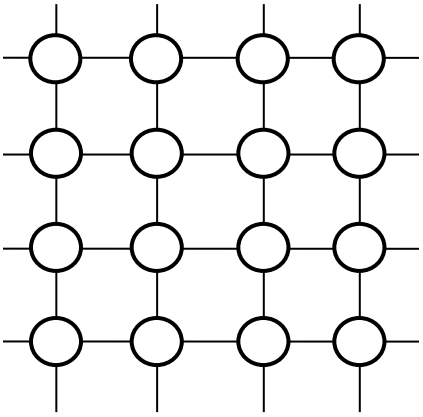
Anexo-3: Sistemas de interconexión completos



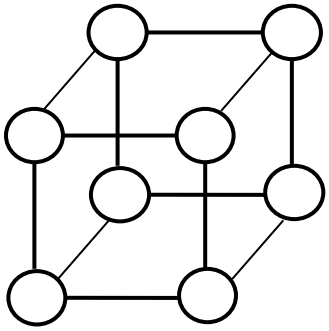
a) Estrella de 9 nodos



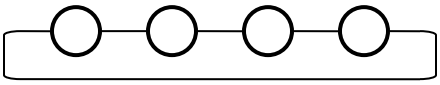
b) Red completamente conectada de 8 nodos



c) Malla 2D



d) Hipercubo de 8 nodos



e) Anillo con 4 nodos

Anexo-4: Tendencias Actuales y Futuras de las Redes de Interconexión.

Tipo de Red	Características
Fast Ethernet (viejo estándar)	Posee un ancho de banda de 100 Mb/s. El nodo master puede usar la tecnología Gigabit.
Gigabit Ethernet (nuevo estándar)	Posee un gran ancho de banda (1Gb/s), con una latencia aceptada de 50 µs. • Sus adaptadores no son costosos. • Gran densidad de puertos (48). • Para densidades de puertos >48 aun son muy costosos.
Myrinet 2000 (estándar de alta velocidad)	Tiene un ancho de banda de 2Gb/s, con una latencia de 5-6 µs. • Escalable (en switches con 128 puertos, en cascada para >128 puertos) • Open Source Drivers /MPI • Es costosa
Dolphin SCI (niche product)	Tiene un ancho de banda de 2Gb/s, con una latencia baja (4 µs). • No usa switches. • Presenta un cableado muy complicado para las topologías torus 2D/3D. • Posee una escalabilidad limitada (máximo de 8 nodos en un anillo). • MPI version closed source. • Sus precios son similares a Myrinet.
Infiniband (nuevo estándar de alta velocidad	Tiene un ancho de banda de 10Gb/s, con una latencia de 6-7 µs. • Es totalmente escalable (hasta 96 puertos de switches, en cascada para >96 puertos). • Open Source Drivers /MPI • Sus precios son similares a Myrinet. • Constituye un estándar abierto.
Quadrics	Tiene un ancho de banda de 2.5Gb/s, con una latencia de 2 µs. • Es un sistema propietario desarrollado para SMP.

Nota:

Mb/s: Megabit por segundo (1×10^6 seg)

Gb/s: Gigabit por segundo (1×10^9 seg)

µs: microsegundo (10^{-6} seg)

switches: Elementos de conmutación para redes

Anexo-5: Algoritmo Paralelo "A"

Etapas 1 y 2: Distribución de los objetos del Sinf y Construcción de las CE de acuerdo al criterio establecido.

```
 1. program rset_parA   {Algoritmo paralelo A}
 2. Si p==0 {procesador 0 o Nodo Maestro}
 3.   array_class := Φ;   {Inicializa el arreglo de clases}
 4.   determinar la cantidad de objetos (m*) a enviar a cada
            procesador;       {m*:=(m/P-1)) o m*:=(m/P-1))+1}
 5.   enviar m* al procesador correspondiente;
 6.   Para cada objeto en el Sinf (m)
 7.     determinar a que procesador p se le enviará cada n;
       {p:=(n %(P-1)+1, considerando al primer objeto n0}
 8.     leer n;           {leer el objeto n del SInf}
 9.     enviar n al procesador p;
 10.    procesar la información de clase de n;
                         {Función info_clase}
{Fin del trabajo del procesador master}

 11. sino                {Procesadores esclavos}
 12.  array_CE := Φ;     {Inicializar el arreglo de CE}
 13.  recibir la cantidad de objetos (m*) del nodo maestro;
 14.  Para cada objeto (m*)
 15.    recibir el objeto n correspondiente;
 16.    construir la CE para el objeto n; {Función const_ceq}
{Fin de la construcción de las CE en los nodos esclavos}
```

Anexo-6: Algoritmo Paralelo "A"

Etapa 2.1: Mezcla de las CE hacia el procesador p1

{Inicio de la etapa de Mezcla}

```
17.  Pe:= número de procesadores esclavos empleados;
18.  nm:= log2(Pe);     {Determina el número de mezclas
     necesarias}
19.  Para cada valor de nm
20.    Si el identificador del procesador (idp) es menor o igual
         al número de procesadores esclavos (Pe)
21.      Si el procesador (p) es par        {p%2==0}
22.        enviar sus CE al procesador impar precedente;    {p-1}
23.      sino      {procesadores impares}
24.        recibir las CE del procesador siguiente (CE_r);  {p+1}
25.        mezclar las CE_r con sus CE; {Función mezc_CE)

26.      {Inicio de la etapa de Reducción}
27.      Si el procesador (p) es impar {p%2≠0}
28.        enviar sus CE al procesador cuyo idp sea igual a
           (p div 2)+1;  {div: División entera}
29.      sino
30.        recibir las CE (CE_r)del procesador cuyo idp sea igual
           a 2*p-1;
31.        mezclar las CE_r con sus CE; {Función mezc_CE)
 {Fin de la etapa de Reducción}

32.    Si Pe es par, Pe:= Pe/2;   {Reduciendo los nodos}
33.    sino Pe:= Pe/2 + 1;        {participantes}
```

{Fin de la etapa de Mezcla, el nodo 1 tiene todas las CE posibles a construir de acuerdo al criterio seleccionado}

{Fin del trabajo de los procesadores esclavos}

Anexo-7: Algoritmo Paralelo "A"

a) Etapa 2.2: Difusión de las CE hacia todos los procesadores.

```
{Inicio del proceso de Difusión}
 34.  Si es el procesador p1 {Procesador con idp=1}
 35.     enviar sus CE al resto de los procesadores;
                {One to all BCast}
 36.  sino recibir las CE del procesador p1 {Resto de los
           procesadores incluyendo al procesador master p0};
{Fin del proceso de Difusión}
```

b) Etapa 3: Construcción de las Aproximaciones.

```
{Difusión de las k clases presentes en el Sistema de Información
(Sinf) desde el procesador p0}

 37. Si p==0 {procesador 0 o Nodo Maestro}
 38.  enviar la información de las clases presentes en el Sinf
      (arreglo de clases array_class); {One to All Bcast}
 39. sino recibir la información de clases;
          {Resto de los procesadores}
{Fin de la difusión de CE}

 40. construir las aproximaciones de clases
                          {Función const_aprox_par}
 41. Si idp ≠ 0        {Todos los procesadores excepto el master
     (p0)}
 42.    enviar el total de objetos en la aproximación inferior (t)
        al procesador p0;     {All to One Reduce, Addition}
 43. sino               {Procesador master (p0)}
 44.   recibir t de cada procesador y determinar el total de
       objetos que del SInf se encuentran en la aproximación
       superior (T);              {All to One Reduce, Addition}
 45.   calidad_SInf:= T/m;        {Calidad de la clasificación}
 46. end_program rset_parA
 {Fin del algoritmo paralelo A}
```

Anexo-8: Algoritmo Paralelo "A"

Funciones auxiliares

```
function mezc_CE    {Función que actualiza o añade una nueva CE a
partir las CE recibidas (CE_r)}
1. Para cada CE_r
2. nce_r:= número de objetos en la CE_r;
3.   Para cada CE en array_CE
4.     Si los valores del criterio de equivalencia de uno de los
          objetos presentes en CE_r coinciden con los valores de la
          CE actual
5.         adicionar cada objeto (nce_r) a la CE actual;
6.   adicionar CE_r a array_CE;
7. end_function;
```

```
function const_aprox_par {Función que construye las
aproximaciones}
16. t:= 0;      {número de objetos en la apx_inf}
17. Para cada clase (k) diferente del SInf
18.    pos:= posición de k en el arreglo de clases (array_class);
19.    c:= procesador que debe construir la aproximación de
           la clase k {c:=pos%P)};
20.    Si el identificador del procesador (idp) es igual a c
21.      apx_inf:=0;
22.      apx_sup:=0;
23.      Para cada clase de equivalencia (CE)
24.         r:= número de objetos en la CE con clase igual a k;
25.         Si r ≠ 0
26.            Si r == cantidad de objetos en la CE (tobj_CE)
27.               entonces apx_inf:= aprox_inf+tobj_CE;
28.          apx_sup:= aprox_sup+tobj_CE;
29.      precisión:= apx_inf/apx_sup;
30.      calidad:= apx_inf/Nk;
31.      t:= t+apx_inf;
32. end_function;
```

Anexo-9: Algoritmo Paralelo "A"

Etapas de Mezcla y Reducción (para 10 nodos esclavos)

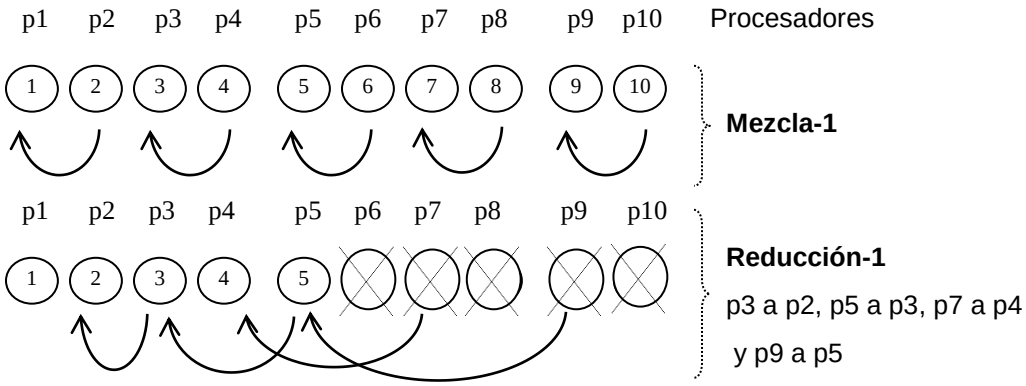
Pe:= P-1:=10 nodos esclavos

Número de mezclas (nm):= $\log_2(Pe) \approx 4$

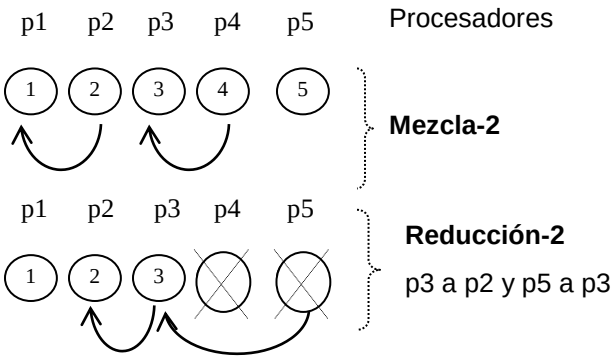
Pd: Número de procesadores descartados en cada etapa de mezcla

- Si el número de Pe es par, Pd:= Pe/2
- Si el número de Pe es impar, Pd:= Pe/2 + 1

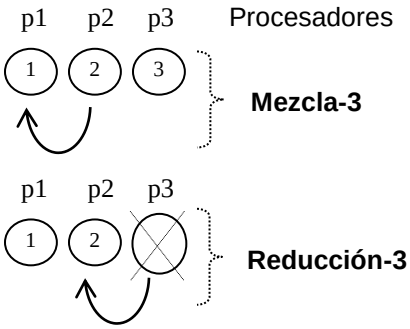
 :Procesador descartado



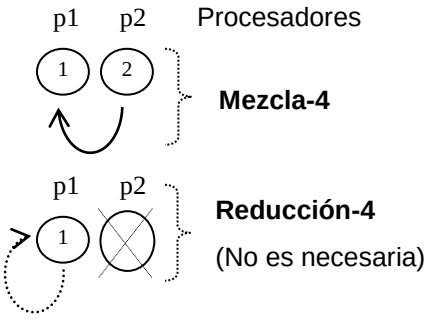
a) Primera etapa de Mezcla y Reducción



b) Segunda etapa de Mezcla y Reducción



c) Tercera etapa de Mezcla y Reducción



d) Cuarta etapa de Mezcla y Reducción

Anexo-10: Algoritmo Paralelo "B"

Etapas de Mezcla y Reducción (para 10 nodos esclavos)

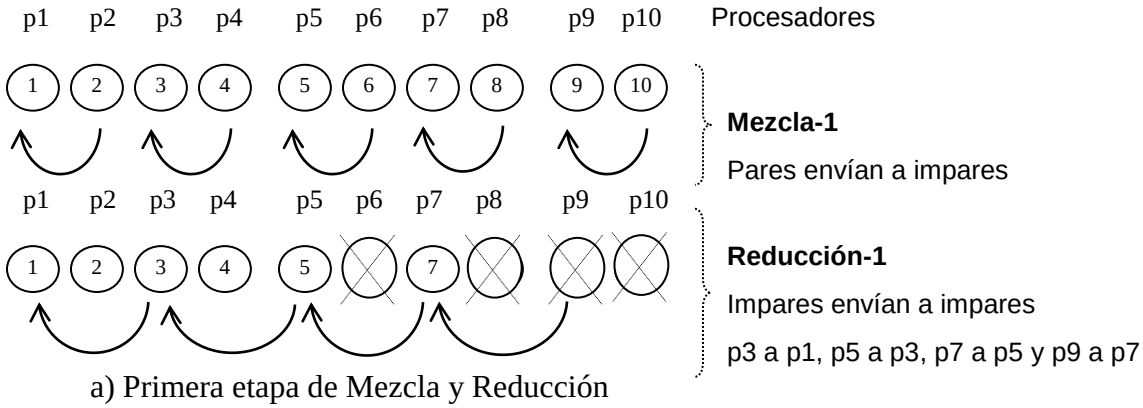
Pe:= P-1:=10 nodos esclavos

Número de mezclas (nm):= $\log_2(Pe)-1 \approx 3$

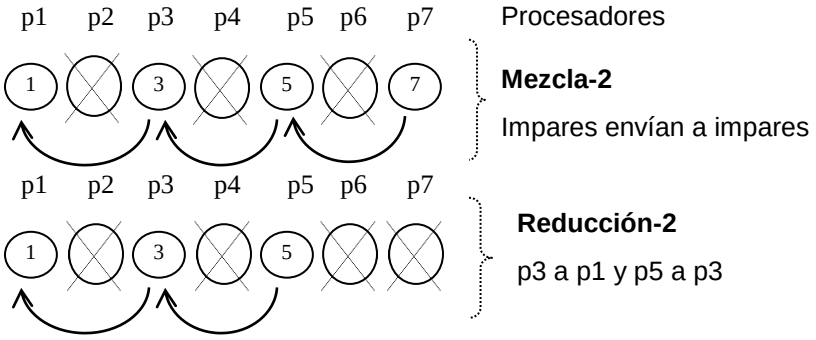
Pd: Número de procesadores descartados en cada etapa de mezcla

Pd:= Pe-2

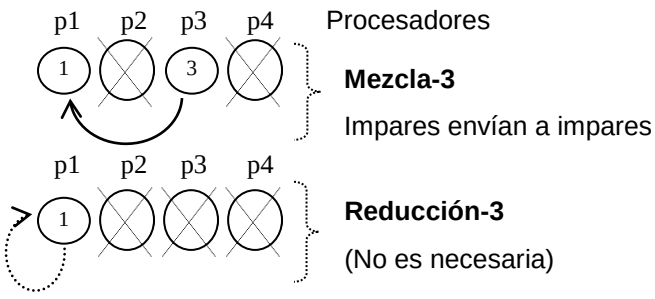
 :Procesador descartado



a) Primera etapa de Mezcla y Reducción



b) Segunda etapa de Mezcla y Reducción



c) Tercera etapa de Mezcla y Reducción

Anexo-11: Algoritmo Paralelo "B"

Etapa 2.1: Mezcla de las CE hacia el procesador p1

{Inicio de la etapa de Mezcla}

1. Pe:= número de procesadores esclavos empleados;
2. nm:= $\log_2$(Pe)-1; *{Determina el número de mezclas necesarias}*
3. **Si** el procesador (p) es par *{p%2==0}*
4. **enviar** sus *CE* al procesador impar precedente; *{p-1}*
5. **sino** {procesadores impares}
6. **recibir** las CE del procesador siguiente (CE_r); *{p+1}*
7. mezclar las CE_r con sus CE; {Función mezc_CE)
8. **Para** cada valor de *nm*
9. **Si** el identificador del procesador (idp) es menor o igual al número de procesadores esclavos (Pe)
   {procesadores impares con idp menor o igual a Pe}

{Inicio de la etapa de Reducción}

10. **Si** el procesador (p) es impar*{p%2≠0}*
11. **enviar** sus *CE* al procesador cuyo idp sea igual a (p-2); *{impar precente}*
12. **recibir** las *CE* (***CE_r***)del procesador impar cuyo idp sea igual a p+2;
13. **mezclar** las CE_r con sus CE; *{Función mezc_CE)*

{Fin de la etapa de Reducción}

14. Pe:= Pe-2; *{Reduciendo los nodos participantes}*

{Fin de la etapa de Mezcla, el nodo 1 tiene todas las CE posibles a construir de acuerdo al criterio seleccionado}

Anexo-12: Algoritmo Paralelo "D"

Etapa D1: Procesamiento de la información y construcción de las CE

```
1. program rset_parD     {Algoritmo paralelo D}
2. array_class := Φ;   {Inicializa el arreglo de clases}
3. array_CE := Φ;  {Inicializa el arreglo de clases equivalentes}
4. Si p==0 {procesador 0 o Nodo Maestro}
5.   enviar la cantidad de objetos en el SInf (m);
                                  {One_to_All Broadcast}
6.   Para cada objeto en el Sinf (m)
7.     enviar el objeto n al resto de los procesadores;
                                   {One_to_All Broadcast}
8.     procesar la información de clase de n;
                                  {Función info_clase}
9.     determinar si existe la CE del objeto n;
                                  {All_to_One Reduce (C)}
10.    b:= identificador del procesador al que le corresponde
           construir la CE de n según el criterio;{n %(P-1)+1}
11.    notificar C al procesador con idp igual a b;
{Fin del trabajo del procesador p0}

12. sino      {Procesadores esclavos}
13.   recibir la cantidad de objetos en el SInf (m);
                                     {One_to_All Broadcast}
14.   Para cada objeto en el Sinf (m)
15.     recibir el objeto n;   {One_to_All Broadcast}
16.     b:= identificador del procesador al que le corresponde
            construir la CE de n según el criterio;{n%(P-1)+1}
17.     C:= valor que indica la presencia o no de la CE a la que
                  pertenece n;       {Función proc_n_CE}
18.     notificar al procesador p0 la existencia (C) de su CE;
                                     {All_to_One Reduce (C)}
19.      Si el idp del procesador es igual a b
20.        recibir la notificación C de p0;
21.        Si no existe la CE a la que pertenece n; {C=0}
22.           construir la CE para n;     {Función const_ceq}
{Fin del trabajo de los procesadores esclavos}
```

Anexo-13: Algoritmo Paralelo "D"

Etapa D2: Construcción de las Aproximaciones

```
23. Si p==0 {procesador 0 o Nodo Maestro}
24. t:= 0; {número de objetos que del SInf están en la
           aproximación inferior}
25. enviar el total de clases K a todos los procesadores
           esclavos;              {One_to_All Broadcast}
26. Para cada clase k distinta en el Sinf
27.   enviar la clase k a aproximar en cada procesador;
                                  {One_to_All Broadcast}
28.   recibir el número de objetos en la aproximación inferior y
              superior para la clase k;
              {All_to_One Reduce (aprox_inf* y aprox_sup*)}
29.    t:= t + aprox_inf*;
30.    determinar la precisión y calidad para la clase k;
                           {Función calc_k}
31. calidad_SInf:= t/m; {Calidad de la clasificación del
                        criterio establecido sobre el SInf}

{Fin del trabajo del procesador Maestro}

32. sino          {Procesadores esclavos}
33.  recibir el total de clases K; {One_to_All Broadcast}
34.  Para cada clase k distinta en el Sinf
35.   recibir la clase k a aproximar en cada procesador;
                                  {One_to_All Broadcast}
36.   construir la aproximación de la clase k;
                                  {Función c_aprox_d}
37.   enviar el número de objetos en la aproximación inferior y
             superior para la clase k;
             {All_to_One Reduce (aprox_inf_all y
             aprox_sup_all)}

{Fin del trabajo de los procesadores Esclavos}
```

Anexo-14: Algoritmo Paralelo "D"

Funciones auxiliares

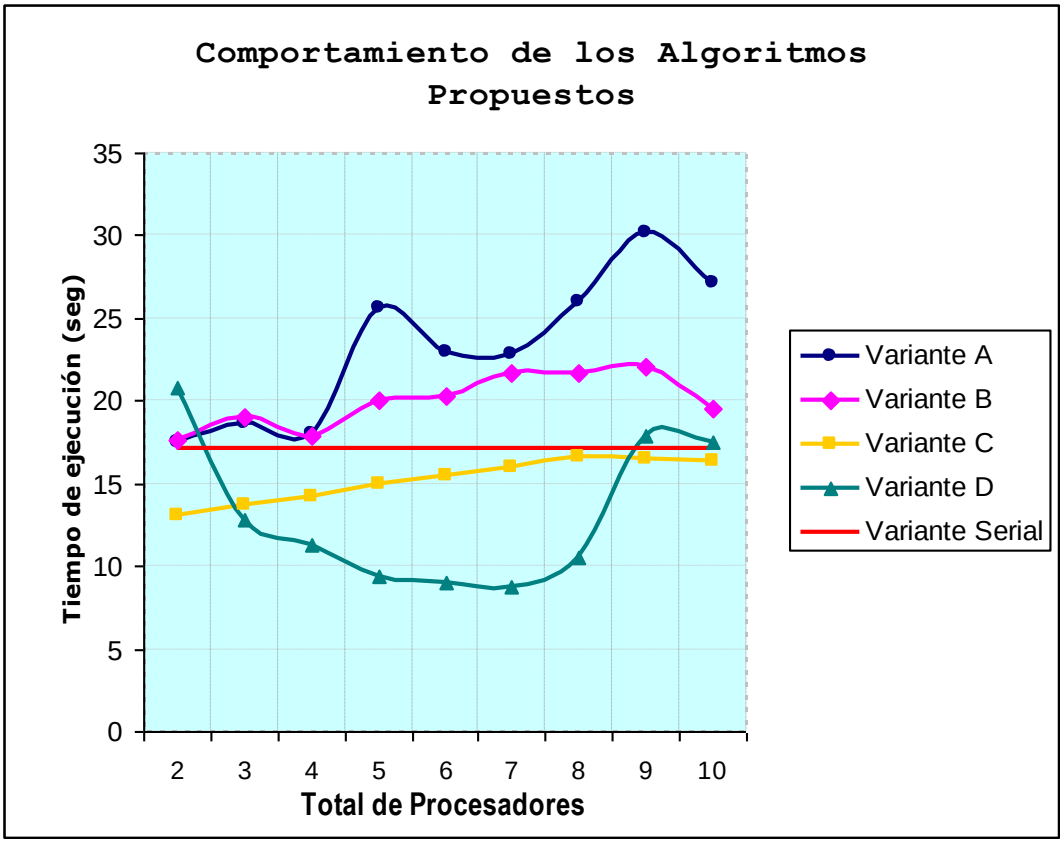
```
{Función que determina y notifica la existencia de una CE a partir
de n y el conjunto de rasgos indicados de n. Si existe añade n a
la CE correspondiente}
function proc_n_CE
1. C := 0;
2. Para cada CE en array_CE y mientras C == 0
3.      Si los valores del criterio de equivalencia de n coinciden
           con los valores de la CE actual
        entonces
4.         adicionar n a la CE actual;
5.         C:= 1;     {Existe la CE a la que pertenece n}
6.  return C;         {Terminar y notificar su existencia}
7. end_function;

function c_aprox_d  {Función que determina el número de objetos en
                     las aproximaciones de la clase k}
1. apx_inf:=0;
2. apx_sup:=0;
3. Para cada clase de equivalencia (CE)
4.   r:= número de objetos en la CE con clase igual a k;
5.   Si r ≠ 0
6.   Si r == cantidad de objetos en la CE (tobj_CE)
7.   apx_inf:= aprox_inf+tobj_CE;
8.   sino apx_sup:= aprox_sup+tobj_CE;
9. end_function;

function calc_k {Función que determina las medidas de precisión  y
                 calidad para la clase k}
1.     precisión:= apx_inf_all/apx_sup_all;
2.     calidad:= apx_inf_all/Nk;
3.  end_function;
```

Anexo-15: Resultados de los Algoritmos paralelos A, B, C y D sobre el cluster del CEI

a) Tiempo de ejecución desde 2 hasta 10 nodos

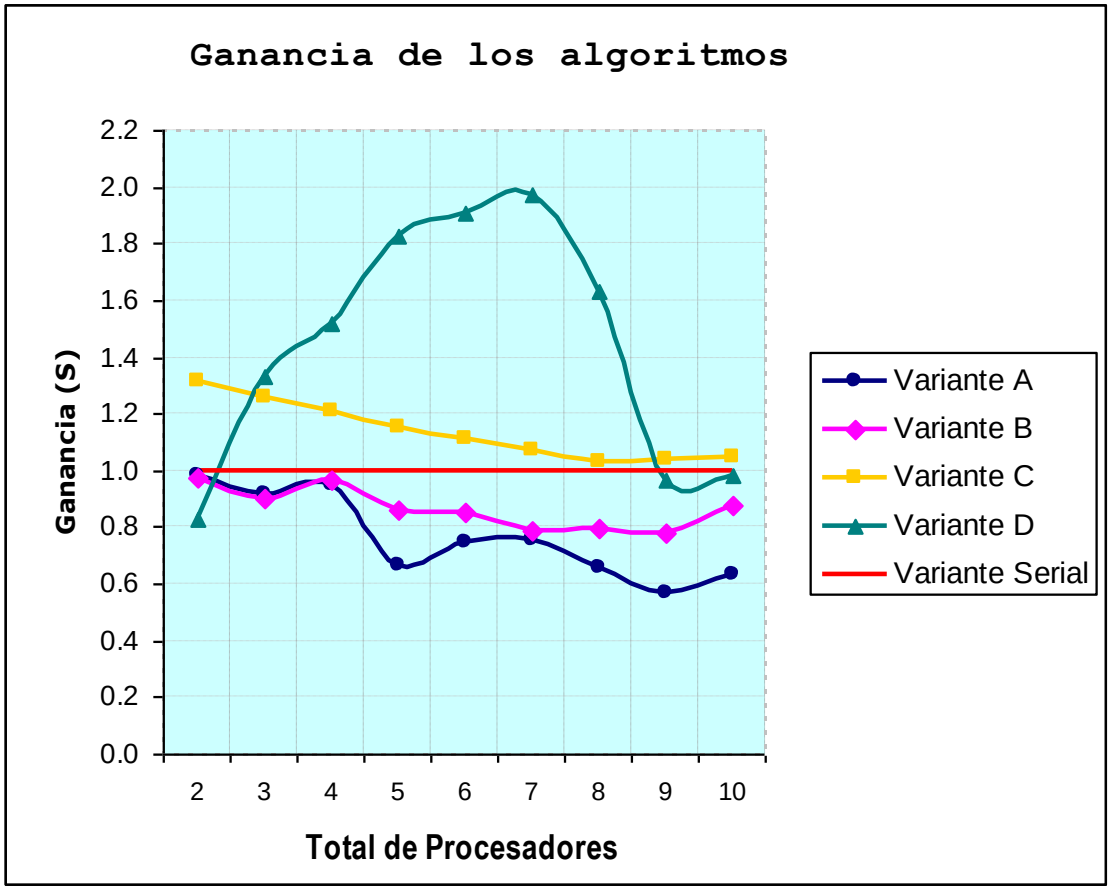
Configuración	time exec (seg)								
Cluster CEI, UCLV	2 (n0-1)	3 (n0-2)	4 (n0-3)	5 (n0-4)	6 (n0-5)	7 (n0-6)	8 (n0-7)	9 (n0-8)	10 (n0-9)
Variante A	17.438	18.660	18.028	25.657	23.006	22.834	26.037	30.209	27.152
Variante B	17.620	19.001	17.830	19.984	20.228	21.722	21.623	22.121	19.523
Variante C	13.079	13.680	14.162	14.927	15.438	15.959	16.629	16.503	16.406
Variante D	20.766	12.863	11.282	9.381	9.013	**8.701**	10.534	17.837	17.442

b) Comportamiento de los algoritmos de acuerdo al número de procesadores empleados.



Anexo-16: Ganancia (S) de los Algoritmos paralelos A, B, C y D sobre el cluster del CEI

Configuración	Ganancia (S)								
Cluster CEI, UCLV	2 (n0-1)	3 (n0-2)	4 (n0-3)	5 (n0-4)	6 (n0-5)	7 (n0-6)	8 (n0-7)	9 (n0-8)	10 (n0-9)
Variante A	0.984	0.920	0.952	0.669	0.746	0.752	0.659	0.568	0.632
Variante B	0.974	0.903	0.963	0.859	0.848	0.790	0.794	0.776	0.879
Variante C	1.312	1.255	1.212	1.150	1.112	1.075	1.032	1.040	1.046
Variante D	0.826	1.334	1.521	1.829	1.904	**1.972**	1.629	0.962	0.984



Anexo-17: Eficiencia (E) de los Algoritmos paralelos A, B, C y D sobre el cluster del CEI

Configuración	Eficiencia (E)								
Cluster CEI, UCLV	2 (n0-1)	3 (n0-2)	4 (n0-3)	5 (n0-4)	6 (n0-5)	7 (n0-6)	8 (n0-7)	9 (n0-8)	10 (n0-9)
Variante A	0.492	0.307	0.238	0.134	0.124	0.107	0.082	0.063	0.063
Variante B	0.487	0.301	0.241	0.172	0.141	0.113	0.099	0.086	0.088
Variante C	0.656	0.418	0.303	0.230	0.185	0.154	0.129	0.116	0.105
Variante D	0.413	0.445	0.380	0.366	0.317	**0.282**	0.204	0.107	0.098