UNIVERSIDAD DE LAS CIENCIAS INFORMATICAS
FACULTAD # 6

Trabajo de Diploma

Diseño de algoritmo paralelo para el análisis de selección positiva

Trabajo de Diploma para optar por el título de
Ingeniero en Ciencias Informáticas

Autor(es): Yudelkis Abad Fuentes

Yurima Ibañez Alfonso

Tutor(es): Lic. Dannier Trinchet Almaguer

Lic. Orlando Martínez Pérez

Cotutor: Lic. Sandy Henríquez Villafruela

Ciudad de la Habana, Junio 2007

DECLARACIÓN DE AUTORÍA

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

Yurima Ibañez Alfonso

Yudelkis Abad Fuentes

Lic. Dannier Trinchet Almaguer

Lic. Orlando Martínez Pérez

_____ Firma del Autor

_____ Firma del Tutor

_____ Firma del Autor

_____ Firma del Tutor

Datos de Contacto

Tutores:

Lic. Dannier Trinchet Almaguer

Universidad de las Ciencias Informáticas

Email: trinchet@uci.cu

Lic. Orlando Martínez Pérez

Universidad de las Ciencias Informáticas

Email: orlandomp@uci.cu

Cotutor:

Lic. Sandy Henríquez Villafruela

Universidad de las Ciencias Informáticas

Email: sandyh@uci.cu

“El futuro pertenece a quienes creen en la belleza de sus sueños”

Eleanor Roosevelt

AGRADECIMIENTOS

A nuestros padres.

A nuestros tutores Orlando, Trinchet y Sandy, por su ayuda.

A la Revolución y a Fidel por haber creado este proyecto tan lindo.

A Blanco por su apoyo y colaboración.

A todas aquellas personas que tuvieron que ver con nuestra formación

A nuestros compañeros de grupo.

A nuestros amigos que nos brindaron su apoyo en todo momento.

A los que en algún momento nos preguntaron por la tesis

A todos muchas gracias

DEDICATORIA

A Quienes me enseñaron a nadar contra la corriente, y a luchar para poder alcanzar mis metas: Mis padres, Noris y Ortedis. ¡Mi triunfo es el de ustedes!

A mi hermano por confiar en mí y siempre darme todo el apoyo que necesité. A mi hermana por siempre haber alegrado mi corazón.

A mi tía Idelgare y Abel, por siempre estar pendientes de mí…

A toda mi familia que me han entregado su amor y apoyo siempre.

A Mailén por todas las penas y alegrías vividas juntas, gracias por tu amistad.

A Alexis, Aidita y el Migue por ser mi familia en esta etapa de mi vida.

A todos mis amigos viejos y nuevos por estar conmigo en todo este tiempo, donde he vivido momentos felices y tristes.

A mis amistades de la universidad y en especial a los que han estado conmigo en todo momento: Yudith, Marisley, Sianny, Yurima, Heydi y Joel.

A nuestros tutores Orlando y Trinchet por tener la paciencia necesaria y apoyarnos en todos los momentos difíciles. ¡Nunca los olvidaré!

A todas las personas que han creído en mí…

A todos gracias por siempre haber estado para mí en algún momento de mi vida.

Yudelkis Abad Fuentes

DEDICATORIA

A mis padres, por su cariño, apoyo incondicional comprensión y dedicación,

A mi hermana por sus consejos y por ayudarme en todo momento.

A mi abuelita, por su amor y dedicación

A mi familia, por su apoyo y preocupación por mí.

A nuestros tutores Orlando y Trinchet por su ayuda y por la paciencia que tuvieron con nosotras, nuestro más sincero agradecimiento

A mis eternos compañeros de estudio, que junto a ellos pasé cinco años maravillosos.

A mis amigos que estuvieron conmigo en momentos de felicidad y tristeza.

A Yirenia y Kisleisis por ser mis mejores amigas.

A Lázaro, por su amistad sincera y estar ahí para mí en cualquier momento

A Liyanis, Yudy, Dayi, Lisbeth y Dunia que estuvieron conmigo en estos últimos momentos y que me apoyaron tanto.

A todos, mi cariño infinito

Yurima Ibañez Alfonso

RESUMEN

El resultado de esta investigación presenta, el diseño de un algoritmo paralelo para el análisis de selección positiva a partir de un algoritmo secuencial existente que realiza el análisis de secuencias de ADN y proteínas, sometidas a mutaciones (cambios) favorables, a lo largo del proceso evolutivo a partir de las técnicas y métodos tradicionales. Esta investigación surge como una necesidad ante los principales problemas y retos que presenta este algoritmo secuencial, dado fundamentalmente por los tiempos de respuestas. El diseño presentado, se describe a partir de su pseudocódigo, y se espera que sea capaz de mejorar considerablemente los tiempos de respuestas, así como, garantizar la calidad de procesamiento necesario en este tipo de análisis

Palabras claves:

Análisis de Selección Positiva

Ambiente paralelo

ADN

Proteínas

ÍNDICE

Agradecimientos V
Dedicatoria VI
Dedicatoria VII
Resumen VIII
Introducción 1
Capítulo 1: Análisis de selección positiva. Paralelismo 4
1.1. Definiciones básicas 4
1.3 Aplicaciones de software para el análisis de secuencias 7
1.4 Paralelismo 8
1.4.1 Arquitecturas paralelas 9
1.4.1.1 Taxonomía Basada en el acceso a la Memoria 10
1.4.1.2 Paralelismo de datos y paralelismo de control 13
1.4.2 Principios para el diseño de algoritmos paralelos 14
1.4.2.1 Descomposición, tareas, y gráficos de dependencias 15
1.4.2.2 Técnicas de descomposición de tareas. 15
1.4.2.3 Técnicas de distribución para equilibrar la carga. 17
1.4.3 Paradigmas de la programación paralela 18
1.4.3.1 Paso de mensajes 19
Conclusiones: 21
Capítulo 2: algoritmo Paralelo. 22
2.1. Algoritmo secuencial por pasos 22
2.1.1 Método utilizado para estimar dn y ds 23
2.1.2 Pruebas al algoritmo secuencial 26
2.2. Construcción paralela del algoritmo. 27
2.2.2 algoritmo paralelo 35
2.2.3 Pseudocódigo del algoritmo paralelo 38
2.3 Librería de definiciones y funciones (MPI) 41
2.4.Métricas de evaluación del desempeño de programas paralelos 43
2.4.1. Velocidad computacional 43

2.4.1.1 Tiempo de ejecución:..... 43
2.4.1.2 Granularidad de los procesos: 43
2.4.1.3 Factor de aceleración (speedup)..... 44
2.4.1.4 Eficiencia (E)..... 45
2.4.1.5 Coste (C)..... 45
Conclusiones..... 46
Recomendaciones: 47
Referencias Bibliográficas 48
Bibliografía 49
Anexos 54
Glosario de Términos..... 55

INTRODUCCIÓN

Los sistemas biológicos evolucionan, estos cambios, mutaciones, deleciones, e inserciones se producen en el ADN de un modo aleatorio y se manifiestan a distintos niveles: proteínas, células, organismos. La interacción con el medio determina cuáles de estos cambios son aceptados y cuáles no. Si un cambio tiene un efecto negativo, existirá una presión selectiva para no aceptarlo (selección negativa). Si este cambio proporciona alguna ventaja, la presión será positiva (selección positiva) y si el cambio no tiene un efecto importante, hablamos de evolución neutral.

Las secuencias de los genes existentes hoy en día son un espejo del pasado, analizando estas secuencias podemos hacer "arqueología biológica" y tratar de reconstruir la historia del gen, determinando qué cambios se han producido, cuándo se han producido y por qué. La información resultante puede ayudarnos a comprender la función de una proteína (Biología Molecular), a determinar por qué una población es resistente a una determinada enfermedad (Epidemiología), o a resolver un caso legal (Medicina Forense). Este tipo de estudios también sirve, para conocer las relaciones de parentesco que hay entre los seres vivos.

Con los avances de la Bioinformática se han propuesto varios métodos para realizar estas inferencias filogenéticas, siendo en la actualidad los desarrollados a partir de técnicas de biología molecular los más utilizados y confiables.

Las herramientas informáticas más usadas se han desarrollado como una respuesta a las necesidades de obtener más información (Estructura, función) sobre las secuencias de ADN y proteínas aprovechando el conocimiento previo almacenado en bases de datos. Estos datos almacenados son utilizados para encontrar secuencias similares a la secuencia problema, o secuencias que tengan alguna propiedad común, así como, para obtener reglas que permitan posteriormente predecir, con mayor o menor éxito, propiedades en secuencias nuevas.

Para el análisis de las inferencias filogenéticas existen aplicaciones informáticas, que contienen una extensa incorporación de métodos estadísticos y permiten a los científicos, determinar el grado de variabilidad que existe en la expresión hereditaria de los organismos, incluso, de diferentes especies.

Los estudios de Selección positiva se realizan a partir del análisis de grandes cantidades de secuencias de ADN o proteínas, basándose fundamentalmente, en las inferencias filogenéticas. En la

actualidad las aplicaciones informáticas disponibles, han sido implementadas secuencialmente lo que provoca un gran consumo de tiempo en ejecución y recursos computacionales, por lo que esto constituye una limitante para llevar a cabo este importante proceso.

De ahí que nos planteamos el siguiente **Problema Científico**: ¿Cómo contribuir de forma eficiente al análisis de selección positiva?

Para lograr la calidad de este proceso, es innegable la utilización e interés en el procesamiento paralelo, por un gran número de razones: crecimiento de la potencia de cálculo dada por la evolución tecnológica, transformación y creación de algoritmos que explotan la concurrencia para obtener mejores tiempos de respuesta, la necesidad de tratar sistemas que le brinden datos en tiempo real, entre otros. El límite físico de los algoritmos secuenciales que existen hoy en día para el análisis de selección positiva convierte como una salida factible: la solución paralela.

Por lo que el **objeto de investigación es:** Ambientes de cómputos paralelos

El **Campo de Acción**: Los sistemas paralelos en el análisis de selección positiva.

El **objetivo fundamental de esta investigación:** Diseñar un algoritmo paralelo para el análisis de selección positiva.

Para dar cumplimiento al objetivo general se trazaron los siguientes objetivos específicos:

1. Delimitar las dependencias que existe entre los datos para una mejor visión de paralelización.
2. Aplicar las técnicas de descomposición, sincronización de procesadores, balanceo de carga y mapeo al algoritmo.
3. Mostrar los detalles de la implementación de la propuesta de algoritmo paralelo, a través del pseudcódigo de sus componentes.

Las tareas a realizar en este trabajo para cumplir estos objetivos son

1. Estudio sobre el proceso biológico de análisis de selección positiva.
2. Estudio de los algoritmos secuenciales para el análisis de selección positiva
3. Estudio de las técnicas para el diseño de algoritmos paralelos
4. Identificación de las partes paralelizables en el proceso de análisis de selección positiva.

5. Distribuir entre los procesadores las diferentes tareas.
6. Paralelización del algoritmo secuencial de selección positiva.
7. Describir los elementos básicos del algoritmo y su esquema en general.
8. Estudio y análisis de las principales métricas de evaluación de desempeño de los algoritmos paralelos.

El trabajo de diploma está estructurado en 2 capítulos:

Capítulo 1: Acercamiento al proceso biológico de análisis de selección positiva. Análisis de los principios básicos del diseño de los algoritmos paralelos.

Capítulo 2: Descripción del algoritmo secuencial, la estrategia de paralelización así como el seudocódigo del algoritmo paralelo. Se presentan además las métricas de estimación del tiempo de respuesta y de evaluación del desempeño de los algoritmos paralelos para su uso en la implementación del algoritmo propuesto.

Finalmente son presentadas las Conclusiones y Recomendaciones para investigaciones futuras.

CAPÍTULO 1: ANÁLISIS DE SELECCIÓN POSITIVA. PARALELISMO

En este capítulo se introducirán algunos aspectos básicos relativos al análisis de selección positiva, incluyendo sus características esenciales. Así como los principios básicos para el diseño de algoritmos paralelos.

Introducción al análisis de selección positiva

En el transcurso de la evolución, la información hereditaria ha ido sufriendo diferentes tipos de alteraciones (o mutaciones), originando una gran diversidad de especies cuya complejidad es también extremadamente diversa.

De todos los tipos de mutaciones que introducen cambios en el material hereditario, las sustituciones de residuos -ya sean nucleótidos o aminoácidos- son las más importantes en el proceso de análisis de selección positiva y de nuestro interés.

Como estos términos pueden resultar algo complejos en el desarrollo de este capítulo se explicarán algunos términos biológicos relacionados con el proceso en estudio para un mejor entendimiento del mismo.

1.1. DEFINICIONES BÁSICAS

Filogenia molecular

La filogenia es una hipótesis (porque nunca estaremos seguros de qué ha ocurrido exactamente) acerca de la historia. El objetivo de los métodos de reconstrucción filogenética es utilizar del mejor modo posible la información actual para hacer inferencias acerca del pasado, es decir, obtener hipótesis que sean lo más fiables posible. Las hipótesis resultantes normalmente se expresan en forma de un árbol, en el que la topología nos habla de las relaciones de parentesco entre las distintas entidades (p.e. genes o especies) y la longitud de las ramas del árbol, de la distancia que hay entre ellas. Hay que señalar que en algunas situaciones (cuando existe recombinación o transferencia horizontal de genes) la historia evolutiva no se puede representar como un árbol.

La filogenia molecular permite el estudio detallado de las características funcionales o estructurales de las secuencias, permitiendo asignar a grupos y a subgrupos propiedades particulares. Junto con la información proveniente de los alineamientos y de las estructuras constituyen el procedimiento más importante para la caracterización molecular de las propiedades (estructura, función) de las secuencias. [1]

Alineamiento de secuencias

Un alineamiento de secuencias en Bioinformática es una forma de mostrar ADN, ARN, o estructuras primarias proteicas para resaltar las zonas de similitud, que podrían indicar relaciones funcionales o evolutivas entre los genes o proteínas consultados. [2]

Si dos secuencias en un alineamiento comparten un ancestro común, las no coincidencias pueden interpretarse como puntos de mutación, y los huecos como indels (mutaciones de inserción o deleción) introducidas en uno o ambos linajes en el tiempo que pasa desde que divergieron. En el alineamiento de secuencias proteicas, el grado de similitud entre los aminoácidos que ocupan una posición concreta en la secuencia puede interpretarse como una medida aproximada de conservación en una región particular o motivos de secuencia entre linajes.

Árboles filogenéticos

Diagrama que utiliza el evolucionismo para representar sus resultados y que representa el camino que ha seguido la evolución para producir la gran diversidad de formas biológicas. En dicho diagrama las formas extintas son intermedias entre otras formas más primitivas y formas más nuevas, es decir, en los puntos terminales del árbol filogenético siempre estarán ocupados por formas actuales y los nodos representan eventos de especiación. Teóricamente existe la posibilidad de convertir un cladogramo a un árbol filogenético. [3]

Con el análisis a los árboles filogenéticos, se obtiene una clasificación de las secuencias en familias y subfamilias, indicando para cada una cuál es, aparentemente, la más cercana. Con este análisis se pueden ver que zonas se conservan y cuales no, y así revelar muchos aspectos funcionales de las proteínas

1.2 Análisis de selección positiva

La selección positiva es asumida o manejada principalmente en el nivel de secuencia de aminoácido, y solo ligeramente en el nivel de secuencia del nucleótido. Esto es porque la mayor parte de las funciones biológicas importantes en los organismos son realizadas principalmente por proteínas en lugar de ácido desoxirribonucleico (ADN).

Las sustituciones de nucleótidos van a ser producidas por alteraciones en el material hereditario y son determinadas por las fuerzas evolutivas fundamentalmente, por la deriva genética y la selección natural. La sustitución de nucleótidos que no cambian la codificación del aminoácido es clasificada como una sustitución sinónima de lo contrario, es no sinónimas.

Como producto de estas sustituciones, si la capacidad relativa de adaptación de un mutante particular (W_m) es similar a la capacidad de adaptación de la media poblacional ($\bar{W}$), la tasa de sustituciones de nucleótido (R_N; tasa en la cual las sustituciones de nucleótido son fijadas en la población) van a ser muy cercanas a la tasa de mutación neutra (R_M; la tasa en la cual la nueva mutación surge). Sin embargo, si W_m es mayor que $\bar{W}$, R_N va a ser mayor que R_M. Este mecanismo evolutivo es llamado *selección positiva*.

Las sustituciones sinónimas (R_S) pueden ser aproximadamente similares a R_M mientras que la parte de sustituciones no sinónimas (R_N) puede variar según el tipo y la fuerza de selección natural. Si la selección positiva ocurre, R_N debe ser más rápido que R_S; Por lo tanto, la selección natural que funciona en el nivel de secuencia de aminoácido puede ser descubierta comparando R_N con R_S, donde $R_N > R_S$ es un indicador de selección positiva.

Tanto R_S como R_N pueden ser estimados mediante la comparación de dos secuencias de nucleótido. Primero, el número de sustituciones sinónimas para todos los sitios sinónimos (d_S) y sustituciones no sinónimas por sitios no sinónimos (d_N) entre dos secuencias es calculado, luego d_S y d_N es dividido por dos veces su tiempo de divergencia (2t); es decir $R_S = d_s/2t$ y $R_N = d_N/2t$. Sin embargo, t normalmente no se conoce en las secuencias que están en consideración. Incluso en tal situación, la selección natural puede ser detectada más fácil comparando d_S y d_N que R_S y R_N. [4]

1.2.1 Métodos de estimación de d_S y d_N

Muchos métodos han sido propuestos para estimar d_S y d_N mediante el uso de secuencias de nucleótidos, los que se clasifican según el modo de analizar las secuencias en:

- ✓ Análisis por pares de secuencias.
- ✓ Análisis por regiones (Sitios) dentro de la secuencia.

De estas categorías, la usada en el algoritmo a paralelizar es la de *estimación por sitios dentro de las secuencias,* ya que es el único método que hace una comparación por codones para las secuencias, emitiendo que sitios se encuentran sometidos bajo la acción de la selección positiva.

Análisis por sitios dentro de la secuencia

Este análisis basa su cálculo en el método de Yang y Nielsen (1998) [4]; Susuki y Gojobory (1999) [4]. El primero es una versión del modelo de análisis por codones y el segundo se basa en el análisis de sitios específicos de codones. Estos métodos utilizan el árbol filogenético para relacionar las secuencias en investigación, y así estimar d_S y d_N.

De estos métodos el usado en el algoritmo es el de Yang y Nielsen (1998), el cual realiza la estimación por el método estadístico de máxima verosimilitud. [5]

1.3 APLICACIONES DE SOFTWARE PARA EL ANÁLISIS DE SECUENCIAS

En la bibliografía consultada encontramos varias aplicaciones de software que utilizan los métodos descritos anteriormente para estimar d_S y d_N, Estas aplicaciones, tienen como desventaja, el gran consumo de tiempo y recursos computacionales en el análisis de selección positiva para grandes cantidades de secuencias de ADN o proteínas. Una de estas aplicaciones es el PALM. Software que implementa el algoritmo secuencial a paralelizar en este trabajo y del cual se darán a conocer algunas de sus características específicas.

Análisis filogenético de Máxima Verosimilitud (PAML)

El PALM es un conjunto de programas para el análisis filogenético de secuencias de ADN o de proteínas, mediante el método de máxima verosimilitud. Este programa se basa en el método YN98 de Yang y Nielsen e incorpora el modelo general de codones de Markov [4], ya que incluye la posibilidad de radios de transición/transversión, la probabilidad de transiciones de codón a codón, y la posibilidad de una fuerza selectiva.

Además estima proporciones sinónimas y no sinónimas, realiza pruebas de hipótesis concernientes a los radios de proporciones d_S/d_N , y a varios análisis de probabilidad de aminoácidos, realiza la reconstrucción de la secuencia ancestral.

1.4 PARALELISMO

Se entiende por arquitectura en paralelo a un conjunto de procesadores interconectados capaces de cooperar en la solución de un problema. El procesamiento paralelo funciona bajo el principio de división del trabajo en subtareas y asignación de estas a distintos elementos computacionales para lograr su ejecución simultánea. Estos deben comunicarse entre si cuando es necesario llevar a cabo ciertas subtareas. [6]

La palabra procesadores, no se refiere solo a los procesadores en su significado más estricto, dígase un procesador SPARC o Pentium IV, sino que puede acoger también a computadoras completas o redes de computadoras, por ejemplos una SUN Workstation o una red de SUN Workstations. El número de procesadores indica (en parte) la capacidad de procesamiento de una máquina paralela. Los procesadores se identifican con números naturales consecutivos comenzando por 0. El conjunto de estos números será denotado por IND, siendo la cardinalidad de este conjunto |IND| la cantidad de procesadores.

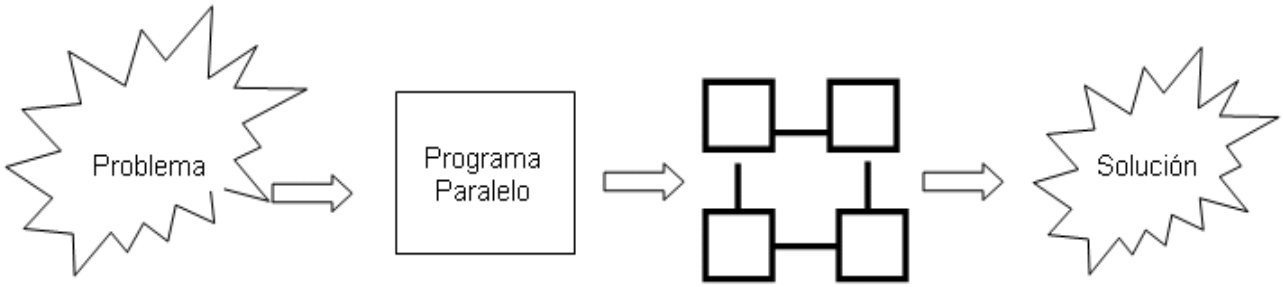


Figura 1.4: Paralelismo

La comunicación paralela surge como una necesidad de lograr un mayor poder computacional, a partir de los grandes avances de la tecnología y en la que no se busca paralelizar como meta en sí, el objetivo fundamental es obtener mayores rendimientos que permitan resolver problemas cada vez más grandes, en tiempos de computación aceptables. La computación paralela logra acelerar la ejecución de un programa mediante su descomposición en fragmentos que pueden ejecutarse simultáneamente, cada uno en su propia unidad de procesamiento.

Idealmente al paralelizar un problema secuencial utilizando n procesadores, se multiplica por n la velocidad computacional de su ejecución en un solo procesador, sin embargo existen otros factores de índole computacional y físico que hacen que nunca se obtenga un escalado en el rendimiento que sea igual o superior al número de procesadores, estos son las comunicaciones entre los procesadores que son lentas en comparación con la velocidad de transmisión de datos y cómputo dentro de un procesador y en la práctica los problemas no pueden dividirse perfectamente en partes totalmente independientes y se necesita alguna interacción entre ellas lo que ocasiona una disminución de la velocidad. Sin embargo, el incremento de la velocidad se aproxima al número de procesadores que se integran en el sistema en determinados algoritmos con una paralelización intrínseca muy fuerte.

1.4.1 ARQUITECTURAS PARALELAS

Las principales arquitecturas en las que se pueden clasificar actualmente a los sistemas paralelos son:

- ✓ Arreglos de Procesadores, también conocidas como máquinas MPP (Massively Parallel Procesing, + de 1000 CPU).
- ✓ Multiprocesadores (SMP).
- ✓ Multicomputadoras de Memoria Distribuida. (cantidad de CPU < = 1000).
- ✓ Máquinas de Memoria híbrida (compartida distribuida).
- ✓ Clúster (a partir del 2000 aparece el término de Constellation).

El procesamiento paralelo se puede definir, en esencia, como la conjunción de dos o más procesadores que computan, en forma concurrente, subtareas correspondientes a un problema más grande.

1.4.1.1 Taxonomía Basada en el acceso a la Memoria

Dentro del procesamiento paralelo se pueden identificar modelos principales, donde la diferencia entre uno y otro reside en la forma en la que cada procesador accede a su memoria.

Máquinas de Memoria Compartida

En el primero, llamado modelo de memoria compartida, o conocido también como sistemas multiprocesador, los procesadores que conforman el sistema acceden a una única memoria común a todos, como si fuese un espacio de dirección global. Y la comunicación entre las tareas se hace gracias a operaciones de escritura/lectura sobre esta memoria Los cambios efectuados en una localización de memoria por un procesador son visibles para el resto de los procesadores. El espacio de dirección global proporciona una programación de uso fácil desde el punto de vista de la memoria.

En el modelo de memoria compartida los procesadores se comunican con la memoria a través de un bus o sistema de switches (llaves) de alta velocidad. Esto les permite lograr un mejor desempeño en comparación con los sistemas de memoria distribuida. Otra ventaja que presenta este modelo es su uso más eficiente de la memoria, dado que no hay necesidad de replicación de datos. No obstante, este tipo de arquitecturas presentan dos importantes dificultades: el alto costo actual de este tipo de hardware y la escasa portabilidad para migrar un programa codificado en un sistema multicomputador hacia otra plataforma de memoria compartida.

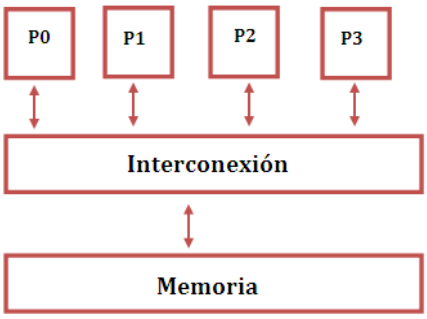


Figura 1.4.1.1(a) Modelo de Memoria Compartida

Máquinas de Memoria Distribuida

En contraste, en el modelo de memoria distribuida, también denominado como sistemas multicomputador o clúster de computadoras, cada procesador posee su propia memoria local, lo que quiere decir, que no existe concepto alguno de un espacio de dirección global entre todos los procesadores. Como cada procesador tiene su propia memoria local, los cambios que hace a su memoria local no tienen ningún efecto en la memoria de otros procesadores. Por tal motivo, si un procesador desea acceder a la memoria local de otro, se requiere de un intercambio de mensajes entre ambos a través de una red que los comunica. Los sistemas distribuidos poseen varias ventajas, en primer lugar permite desarrollar software más portables debido a los estándares existentes en los protocolos de comunicación, además de que cada procesador puede acceder a su propia memoria rápidamente sin interferencia.

Por otra parte los clúster poseen bajo costo y buena escalabilidad dado que usualmente están integrados por computadoras interpersonales conectadas por una red Ethernet Por estas razones la tendencia actual en procesamiento paralelo apunta hacia el empleo de este tipo de sistemas multiprocesador. Estos sistemas poseen además algunas desventajas como son: que el programador es responsable de muchos de los detalles asociados con la comunicación de los datos entre los procesadores. Es más difícil de distribuir la estructura de datos existente, a la hora de compartir toda la memoria del sistema

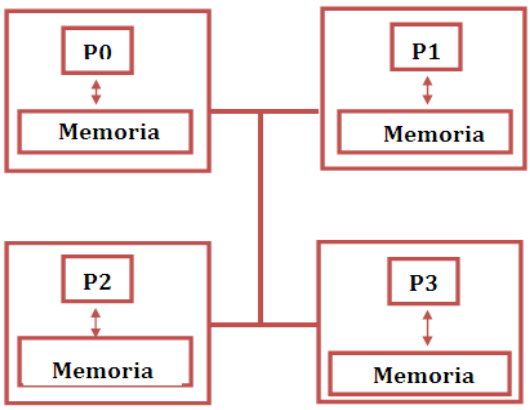


Figura 1.4.1.1 (b): Modelo de Memoria ditribuida

Clúster

Un clúster es un conjunto de computadoras personales (PC) o estaciones de trabajo conectadas entre sí por una red de computadoras, que actúan en conjunto usando el poder de cómputo de varios CPU en combinación para resolver ciertos problemas dados. Presenta combinaciones de los siguientes servicios:

- ✓ Alto rendimiento
- ✓ Alta disponibilidad
- ✓ Equilibrio de carga
- ✓ Escalabilidad

Ventajas:

- ✓ Los clústers se construyen con un esfuerzo relativamente moderado
- ✓ Bajo coste
- ✓ Usan hardware convencional y accesible al mercado
- ✓ Generalmente usan un sistema de comunicación basado en una red de área local rápida.
- ✓ Utilizan software de libre distribución.
- ✓ Son ampliables
- ✓ Cada nodo del clúster puede se un sistema completo utilizable para otros propósitos.

El éxito de los clústers se debe más a los avances en el rendimiento de las redes de comunicación que a la mejora de los procesadores teniendo en cuenta que lo primero es lo que más afecta en el rendimiento global para no alcanzar aún velocidades comparables a las que logran los procesadores. Un clúster se puede clasificar en dependencia de si cada computador que lo forma esta exclusivamente dedicado a él (Beowulf), o no lo esta (Redes de estaciones de trabajo).

Los clústers Beowulf son lo que tienen más auge en la actualidad, los cuales presentan diversas capacidades para el cómputo paralelo con un relativo alto rendimiento, convirtiéndose en una buena opción para cubrir las necesidades de cómputo intensivo y de desarrollo.

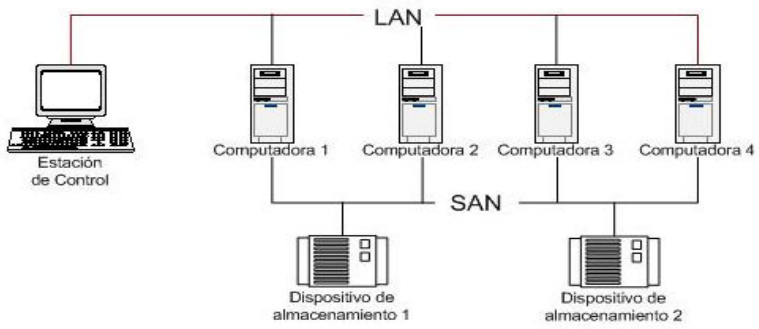


Figura 1.4.1.1(c): Representación de un clúster Beowulf

1.4.1.2 Paralelismo de datos y paralelismo de control

El paralelismo ideal es aquel en el cual un problema puede ser dividido en partes completamente independientes que pueden ser ejecutadas simultáneamente. Existen diferentes formas de lograr el paralelismo, a través de: Procesamiento concurrente de operaciones de entrada y salida en diferentes procesadores, acceso concurrente a memoria usando sistemas de almacenamiento multipuertos y sistemas de intercalado de memoria. Además de la ejecución concurrente de instrucciones usando múltiples unidades funcionales y la transmisión concurrente de datos entre los dispositivos usando buses.

Al intentar dividir un problema, se identifican dos tipos de paralelismo: el paralelismo de datos y el paralelismo de control o funcional.

El paralelismo de datos se obtiene al asignar elementos de datos a varios procesadores, cada uno de los cuales realiza la misma operación simultáneamente sobre sus datos. Este tipo de paralelismo consiste en una secuencia simple de instrucciones o flujo de instrucciones cada una de las cuales se aplica a diferentes elementos de datos.

El paralelismo de control se logra realizando diferentes operaciones sobre varios conjuntos de datos simultáneamente, o sea, se refiere a la ejecución simultánea de diferentes flujos de instrucciones.

1.4.2 PRINCIPIOS PARA EL DISEÑO DE ALGORITMOS PARALELOS

El desarrollo de algoritmos es un componente fundamental en la solución de problemas mediante el uso de computadoras. Un algoritmo secuencial es una secuencia de pasos básicos para solucionar un problema dado, usando una computadora. Semejantemente, un algoritmo paralelo permite solucionar un problema dado usando procesadores múltiples. Sin embargo, especificar un algoritmo paralelo implica más que una simple especificación de pasos. Un algoritmo paralelo tiene la dimensión agregada de la concurrencia y el diseñador del algoritmo debe especificar la entrada de pasos que pueden ser ejecutados simultáneamente. Esto es esencial para obtener cualquier ventaja del uso de una computadora paralela. En la práctica, especificar un algoritmo paralelo puede incluir alguno o todos de los siguientes puntos:

- ✓ Identificar las partes o porciones del trabajo que pueden ser ejecutadas (realizadas) de manera concurrente.
- ✓ Mapear la piezas concurrentes sobre múltiples procesos que corren en paralelo.
- ✓ Distribuir la entrada, salida, y datos intermedios asociados al programa
- ✓ Manejar el acceso a datos compartidos por múltiples procesadores
- ✓ Sincronizar los procesadores en las diferentes fases de ejecución del programa paralelo.

Dividiendo un cómputo en cómputos más pequeños y asignándolo a diferentes procesadores para su ejecución paralela son los dos pasos clave en el diseño de algoritmos paralelos. [7]

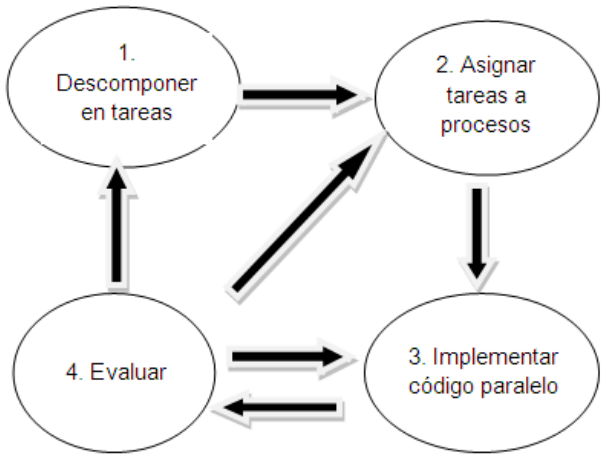


Figura 1.4.2: Pasos para implementar un programa Paralelo

1.4.2.1 Descomposición, tareas, y gráficos de dependencias

En el desarrollo de algoritmos paralelos es importante tener en cuenta tres elementos fundamentales, primero, el proceso de dividir el cómputo en partes más pequeñas, algunas de las cuales o todas puedes ser ejecutadas en paralelo, es a lo que le llamamos descomposición, en segundo lugar es importante definir las principales tareas o sea las unidades de cálculos definidas por el programador en la cual el cómputo principal es subdividido por medio de la descomposición.

La ejecución simultánea de tareas múltiples es la clave para reducir el tiempo requerido para resolver el problema entero. Las tareas pueden ser de tamaño arbitrario, pero una vez que han sido definidas, se consideran unidades indivisibles de cómputo. Las tareas en que un problema se descompone no tienen por qué ser todas de un mismo tamaño. [7]

Por último el grafo de dependencia no es más que la abstracción usada para representar las dependencias entre tareas y su orden relativo de ejecución.

1.4.2.2 Técnicas de descomposición de tareas.

Las técnicas de descomposición son clasificadas como descomposición recursiva, descomposición de datos, descomposición exploratoria, y la descomposición especulativa. [7]

Descomposición recursiva

En esta técnica, el problema es resuelto dividiéndolo primeramente en una variedad de subproblemas independientes. Donde cada uno de esos subproblemas es resuelto de manera recursiva, aplicando una división similar en subproblemas más pequeños seguidos por una combinación de sus resultados.

Descomposición de los datos:

Es uno de los métodos más usados para derivar concurrencia en algoritmos que operan con grandes estructuras de datos. En este método, se hace la descomposición de cómputos en dos pasos. En el primer paso, los datos con los que se realiza el cómputo son divididos, y en el segundo paso, estos datos divididos se usan para inducir una partición del cálculo en tareas. Las operaciones en las que estas tareas se ejecutan en diferentes particiones de datos son normalmente similares. Se debe explorar y evaluar todas las posibles maneras de particionar los datos y determinar cuál rinde una natural y eficaz descomposición computacional.

Descomposición exploratoria

Es usada para descomponer problemas cuyos cómputos fundamentales corresponden a una búsqueda de espacio para las soluciones. En la descomposición exploratoria, se divide el espacio buscado en partes más pequeñas, e investiga cada uno de estas partes concurrentemente, hasta que sean encontradas las soluciones deseadas.

Descomposición especulativa

Es usada cuando un programa puede tomar una de varias ramas computacionalmente significativas que dependen de la salida de los cómputos que la preceden. En esta situación, mientras una tarea ejecuta el cálculo, del cual su salida es usada en la decisión del próximo cálculo, otras tareas pueden empezar concurrentemente los cálculos de la próxima fase. Este escenario es más o menos similar a la evaluación de una o más ramas mediante el uso de un interruptor que garantiza la ejecución de las tareas de manera concurrente hasta que ésta sea totalmente realizada (semáforo).

Descomposición híbrida

Esta técnica de descomposición no es exclusiva, y puede ser combinada con otras. Como el cálculo es estructurado en múltiples fases, se hace necesario aplicar diferentes tipos de descomposición en las diferentes fases. Por ejemplo, cuando se va a hallar el mínimo dentro de una cadena de n longitud una descomposición puramente recursiva puede resultar en muchas más tareas que el número de procesadores, p, disponible. Una descomposición eficiente separaría la entrada en P partes aproximadamente iguales y pondría a cada tarea a calcular el mínimo de la cadena asignada a él. El resultado final puede obtenerse encontrando el mínimo de los resultados de los P intermedios usando la descomposición recursiva.

Las distintas técnicas de descomposición descritas permite identificar si existe o no concurrencia, y como descomponer un problema en tareas que se ejecutarán en paralelo. El siguiente paso en el proceso de diseño de un algoritmo en paralelo es tomar esas tareas y asignarlas a los procesadores disponibles. Mientras se supervisa un plan de distribución para construir un buen algoritmo, se debe tomar algo clave en la descomposición. La naturaleza de las tareas y las interacciones entre las mismas tienen gran significado en la distribución. [7]

1.4.2.3 Técnicas de distribución para equilibrar la carga

Una vez que el cómputo haya sido descompuesto en tareas, estas son mapeadas en los procesadores con el objetivo de que todas sean ejecutadas en la menor cantidad de tiempo posible. En orden de lograr un menor tiempo de ejecución, la sobrecarga de ejecutar las tareas en paralelo debe ser minimizada. Para lograrlo, existen dos claves fundamentales. El tiempo que se gasta en la interacción entre procesadores y el tiempo que pueden ser gastados en algunos procesadores cuando estén ociosos.

Algunos procesadores pueden estar ociosos inclusive antes de que acabe de ejecutar el cálculo, por diferentes razones: una mala distribución que puede causar que varios procesadores terminen más tempranos que otros, entonces, todas las tareas mapeadas en procesadores que no hayan terminado, deben esperar por tareas mapeadas en otros procesadores que van a ir terminando en el orden que fueron satisfechas las restricciones impuestas por el grafo de dependencia. Ambas (interacciones y ocio) son funciones de la distribución. Por tanto, una buena distribución de tareas en procesadores debe esforzarse por lograr los objetivos paralelos mediante la reducción de la cantidad de tiempo que gastan los procesadores cuando interactúan recíprocamente entre sí, y la reducción de la cantidad total de tiempo de algunos procesadores que se encuentran ociosos mientras los otros están abrumados con ejecuciones de otras tareas.

Estos dos objetivos siempre se encuentran en conflictos. Por ejemplo, el objetivo de minimizar las interacciones puede ser fácilmente logrado mediante la asignación de tareas que necesitan interactuar con cualquier otra en el mismo procesador. En muchos casos, como la distribución puede resultar desbalanceada en la carga de trabajo entre los procesadores, los procesadores con menos carga pueden quedarse ociosos cuando otros con sobrecarga tratan de terminar con sus tareas. Igualmente, el balance de carga entre procesadores debe se necesario para asignar tareas interactivamente pesadas a diferentes procesadores.

Como la ejecución de un algoritmo paralelo termina en una fase dada, es posible que aunque todos los procesadores tengan que realizar la misma cantidad de trabajo, en momentos diferentes, sólo un fragmento de los procesadores está activo mientras el resto contiene tareas que deben esperar por otras tareas para terminar. Semejantemente, una sincronización pobre entre las tareas relacionadas puede llevar al ocioso si una de ellas tiene que esperar enviar o recibir datos de otra. Una buena distribución debe asegurar que los cálculos e interacciones entre los procesadores en cada fase de ejecución del algoritmo paralelo sean bien equilibrados.

Las técnicas de distribución usadas en los algoritmos paralelos son ampliamente clasificadas en dos categorías: Estáticas y dinámicas.

Distribución estática: Las técnicas de distribución estática distribuyen las tareas entre los procesadores antes de la ejecución del algoritmo. La opción de una buena distribución en este caso depende de varios factores, incluyendo el conocimiento del tamaño de las tareas, el tamaño de los datos asociados a cada una, las características de las interacciones entre tareas, e incluso el paradigma de programación paralelo. Los algoritmos que usan un mapeo estático, generalmente son muy fáciles de diseñar y programar.

Distribución dinámica: Las técnicas de distribución dinámica distribuyen el trabajo entre los procesos durante la ejecución del algoritmo. Si las tareas son generadas dinámicamente, entonces ellas pueden ser distribuidas también dinámicamente. Si el tamaño de las tareas son desconocidos, entonces una distribución estática puede llevar a una sobrecarga seria y una distribución dinámica es, generalmente en este sentido, más efectiva. Si la cantidad de datos asociados a la tarea es relativamente larga para el cálculo, entonces una distribución dinámica, lo que hace es distribuir esos datos entre los procesos.

Los algoritmos que requieren una distribución dinámica generalmente son más complicados, particularmente en el paradigma de paso de mensaje.

1.4.3 PARADIGMAS DE LA PROGRAMACIÓN PARALELA

Existen muchos métodos de programación en cómputos paralelos, estos son:

- ✓ *Paralelismo de datos*: El paralelismo lo determina el particionamiento de los datos.
- ✓ *Paso de mensajes*: El usuario hace llamadas a bibliotecas para explícitamente compartir información entre los procesadores.
- ✓ *Memoria compartida:* Múltiples procesos comparten un espacio de memoria común
- ✓ *Operación remota a memoria:* Un proceso puede acceder a la memoria de otro proceso sin su participación.
- ✓ *Threads (hebras, hilos):* Un proceso teniendo múltiples (concurrentes) trayectorias de ejecución
- ✓ *Modelo combinado*: compuesto de dos o más de los mencionados anteriormente.

De estos modelos los más usados son el de paso de mensajes y el paralelismo de datos.

1.4.3.1 Paso de mensajes

En este modelo un conjunto de procesadores usando solo memoria local, se comunican enviando y recibiendo mensajes La transferencia de datos requiere operaciones cooperativas para ser ejecutada por cada procesador (una operación de envío debe tener una operación de recepción)

Es el modelo propuesto en la estrategia de paralelización del los algoritmo presentado en esta investigación, a partir del uso de la librería portable y estándar de paso de mensajes (MPI). Este fue el modelo seleccionado ya que esta disponible para diferentes máquinas paralelas, por lo que hace posible que el algoritmo pueda ser ejecutado sobre varias arquitecturas paralelas. La comunicación entre tareas es por pase de mensajes. Con MPI todo el paralelismo es explícito: el programador es el responsable del paralelismo del programa y la implantación de las directivas MPI. Usa un enfoque SPMD (Single Program Multiple Data), simple programa múltiples datos.

Al arrancar una aplicación se lanzan en paralelo N copias del mismo programa (procesos). Estos procesos no avanzan sincronizados instrucción a instrucción sino que la sincronización, cuando sea necesaria, tiene que ser explícita. Los procesos tienen un espacio de memoria completamente separado. El intercambio de información, así como la sincronización, se hacen mediante paso de mensajes.

MPI dispone de funciones de comunicación punto a punto (que involucran sólo a dos procesadores), y de funciones u operaciones colectivas (que involucran a múltiples procesadores). Los procesadores pueden agruparse y formar comunicadores, lo que permite una definición del ámbito de las operaciones colectivas, así como un diseño modular.

Modelos y modos de comunicación con MPI

MPI define dos modelos de comunicación: bloqueante (blocking) y no bloqueante (nonblocking). El modelo de comunicación tiene que ver con el tiempo que un proceso pasa bloqueado tras llamar a una función de comunicación, sea ésta de envío o de recepción. Una función bloqueante mantiene a un proceso bloqueado hasta que la operación solicitada finalice. [8]

Una no bloqueante supone simplemente “encargar” al sistema la realización de una operación, recuperando el control inmediatamente. El proceso tiene que preocuparse, más adelante, de averiguar si la operación ha finalizado o no.

Queda una duda, sin embargo: ¿cuándo damos una operación por finalizada? En el caso de la recepción está claro: cuando tengamos un mensaje nuevo, completo, en el buffer asignado al efecto. En el caso de la emisión el proceso es más complejo: se puede entender que la emisión ha terminado cuando el mensaje ha sido recibido en destino, o se puede ser menos restrictivo y dar por terminada la operación en cuanto se ha hecho una copia del mensaje en un buffer del sistema en el lado emisor. MPI define un envío como finalizado cuando el emisor puede reutilizar, sin problemas de causar interferencias, el buffer de emisión que tenía el mensaje. Dicho esto, podemos entender que tras hacer un send (envío) bloqueante podemos reutilizar el buffer asociado sin problemas, pero tras hacer un send no bloqueante tenemos que ser muy cuidadosos con las manipulaciones que se realizan sobre el buffer, bajo el riesgo de alterar inadvertidamente la información que se está enviando.

Al margen de si la función invocada es bloqueante o no, el programador puede tener un cierto control sobre la forma en la que se realiza y completa un envío. MPI define, en relación a este aspecto, cuatro modos de envío: básico (basic), con buffer (buffered), síncrono (synchronous) y listo (ready).

Cuando se hace un envío **con buffer** se guarda inmediatamente, en un buffer al efecto en el emisor, una copia del mensaje. La operación se da por completa en cuanto se ha efectuado esta copia. Si no hay espacio en el buffer, el envío fracasa.

Si se hace un envío **síncrono**, la operación se da por terminada sólo cuando el mensaje ha sido recibido en destino. En función de la implementación elegida, puede exigir menos copias de la información conforme ésta circula del buffer del emisor al buffer del receptor.

El modo de envío **básico** no especifica la forma en la que se completa la operación: es algo dependiente de la implementación. Normalmente equivale a un envío con buffer para mensajes cortos y a un envío síncrono para mensajes largos. Se intenta así agilizar el envío de mensajes cortos a la vez que se procura no perder demasiado tiempo realizando copias de la información.

En cuanto al envío en modo **listo**, sólo se puede hacer si antes el otro extremo está preparado para una recepción inmediata. No hay copias adicionales del mensaje (como en el caso del modo con buffer), y tampoco podemos confiar en bloquearnos hasta que el receptor esté preparado.

CONCLUSIONES:

En este capitulo se dieron algunas definiciones básicas relativas al análisis de selección positiva y a la programación paralela. Se propusieron algunas de las técnicas y métodos utilizadas para el análisis de selección positiva, así como los principios básicos para la paralelización de algoritmos. Se explica además las herramientas utilizadas para paralelización.

CAPÍTULO 2: ALGORITMO PARALELO.

En este capítulo se describe el algoritmo secuencial, se presenta la estrategia de paralelización así como el seudocódigo del algoritmo paralelo. Además de las métricas de estimación del tiempo de respuesta y de evaluación del desempeño del algoritmo paralelo propuesto.

2.1. ALGORITMO SECUENCIAL POR PASOS

Para la descripción precisa del algoritmo secuencial, debemos partir de que este tiene como datos de entrada un conjunto de secuencias a analizar, alineadas, a partir de un alineamiento múltiple realizado por alguna aplicación de software existente o manualmente, además del árbol filogenético correspondiente a estas.

1. El algoritmo usa como primera medida, la especificación del modelo a utilizar, teniendo en cuenta las distintas suposiciones que incluye cada uno de ellos para el valor de omega (ω)
2. Analiza por codones (Triplete de bases nitrogenadas) las sustituciones sinónimas (d_S) y las no sinónimas (d_N) por sitios (Ver sección 1.1), esto da la medida de cambios que han ocurrido en la información de las secuencias de ADN o Proteínas en estudio.
3. A partir de estos valores (d_S **y** d_N) estima el valor de ω.
4. El algoritmo, utiliza el método estadístico de máxima verosimilitud, estima los parámetros especificados para cada uno de los modelos, mediante la maximización de la función de probabilidad: $L(\theta, X) = f(\theta|X)$

   Según el principio de probabilidad, la función de probabilidad contiene toda la información en los datos acerca de los parámetros θ. El mejor punto de estimación de θ va a estar dado por el θ que maximice la probabilidad L o el registro de probabilidad $\ell(\theta, X) = \log\{(\theta, X)\}$. Esta función permite seleccionar el modelo que con mayor probabilidad represente los datos en estudio.

5. A partir de los métodos empíricos selecciona los sitios que mayor probabilidad (entre 95 y 99%) tengan de estar sometidos a selección positiva.

2.1.1 MÉTODO UTILIZADO PARA ESTIMAR DN Y DS

El análisis de selección positiva se realiza por sitios, un sitio para nuestro algoritmo es un codón (triplete de nucleótido). Para el análisis de selección positiva se parte del árbol filogenético, estableciendo en primer lugar los ancestros correspondientes a cada una de las secuencias. Se estiman las sustituciones sinónimas por sitios sinónimos del codón r (desde r=1 hasta el número de codones de la secuencia) a partir de cada una de las ramas del árbol. Las hojas del árbol (nodos que no tienen hijos) están dadas por el número de secuencias a analizar. Destacar que cada una de las secuencias a analizar tiene la misma longitud, por tanto todas tienen el mismo número de sitios (codones).

Es necesario el cálculo para cada uno de los sitios r:

C_{S_r} → Diferencias sinónimas

C_{N_r} → Diferencias no sinónimas

S_{S_r} → Sitios sinónimos

S_{N_r} → Sitios no sinónimos

El árbol filogenético de n-1 ramas (ver figura 2.1.2) es usado para buscar:

Desde j=1 hasta el número de ramas

$b_{S_j}^{(r)}$ ← Número promedio de sitios sinónimos por la rama j en el sitio r

$b_{N_j}^{(r)}$ ← Número promedio de sitios no sinónimos por cada rama

C_{S_j} ← Suma de las diferencias sinónimas para el codón r (C_{S_r}) en cada una de las ramas j

C_{N_j} ← Suma de las diferencias no sinónimas para el codón r (C_{N_r}) en cada una de las ramas j

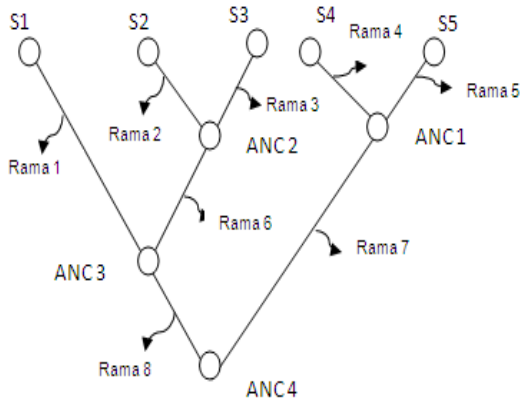


Figura 2.1.1: árbol filogenético de 8 ramas

Método explicado por pasos

1. Sean s y k los últimos codones de la rama j. Calcular los sustituciones sinónimos $\left(S_{S_S'}\right.$ y $\left.S_{S_K'}\right)$ y las no sinónimas $\left(S_{N_S'}\right.$ y $\left.S_{N_K'}\right)$ para los codones s y k.

- Si los codones j y k difieren en una sola posición los valores de $b_{S_j}^{(r)}$ y $b_{N_j}^{(r)}$ (promedio de sustituciones sinónimas y no sinónimas) respectivamente, para cada una de las ramas, se estiman como

$$b_{S\,j} = \left(S_{S_j} + S_{S_K}\right)/2 \qquad b_{N_j} = \left(S_{N_j} + S_{N_K}\right)/2$$

- Si difieren en más de una posición el análisis se hace a partir de los posibles cambios intermedios que pueden haber sucedido. Se supone que en una primera mutación haya cambiado en un nucleótido y en una segunda se haya realizado el siguiente cambio. Por ejemplo suponiendo ACG (Thr) como el codón j y AAT (Asn) como el codón k, estos dos codones difieren en 2 nucleótidos. Los cambios pueden haber ocurrido de la siguiente forma.

Camino1: ACT (Thr) → AAG (Lys) → AAT (Asn)

Camino2: ACG (Thr)→ ACT (Thr)→ AAT (Asn)

Para estos casos b_{S_j} se calcula a partir del promedio de las sustituciones sinónimas para cada uno de los caminos o estados por los que haya tenido que pasar para lograr cambiar en más de una posición el codón.

$$b_{S_j} = \left[\left(S_S(ACT) + S_S(AAG) + S_S(AAT)\right)/3\right]/2$$

En el caso de b_{N_j} se estima de igual forma que para cuando j y k difieren en una posición.

2. Calcular

- $S_S(r) = \sum b_{S_j} * l_j / l_t$
- $S_N(r) = \sum b_{N_j} * l_j / l_t$

Siendo l_j: Longitud de la rama j l_t: Longitud total del árbol

3. Calcular

$$C_s(r) = \sum_{j=1}^{Ctdad\_Ramas} C_{s'_j} \quad \text{donde} \quad C_{s'_j} = \sum_{k'=1}^{Ctdad\_Ancestros} C_{s''_{jk}}$$

De la misma forma se calcula $C_N(r)$ para cada uno de los codones

4. Finalmente el método calcula $d_N(r)$ y $d_S(r)$ (para cada sitio r) a partir de

$$d_S(r) = C_S(r)/S_S(r) \qquad d_N(r) = C_N(r)/S_N(r)$$

2.1.2 PRUEBAS AL ALGORITMO SECUENCIAL

Como medida de la eficiencia de los algoritmos se suelen estudiar los recursos (memoria y tiempo de ejecución) que consume el algoritmo. Hemos desarrollado un análisis que especifique la evolución del tiempo de ejecución en función del tamaño de los valores de entrada para el algoritmo secuencial descrito en la sección previa.

Para este análisis se tomaron varios juegos de datos de entrada, obtenidos a partir del software auxiliar: Clustalx (Ver anexo 1).

Secuencias	Tiempo (min)
10	00:00.39
15	00:03:09
20	00:04:42
25	00:07:03
30	00:12:44
35	00:26:15

Secuencias	Tiempo (min)
10	00:02.26
15	00:02:34
20	00:05:56
25	00:13:32
30	00:15:56
35	00:25:03

Figura 2.1.2 (a): Corrida para diferentes entradas de juegos de datos.

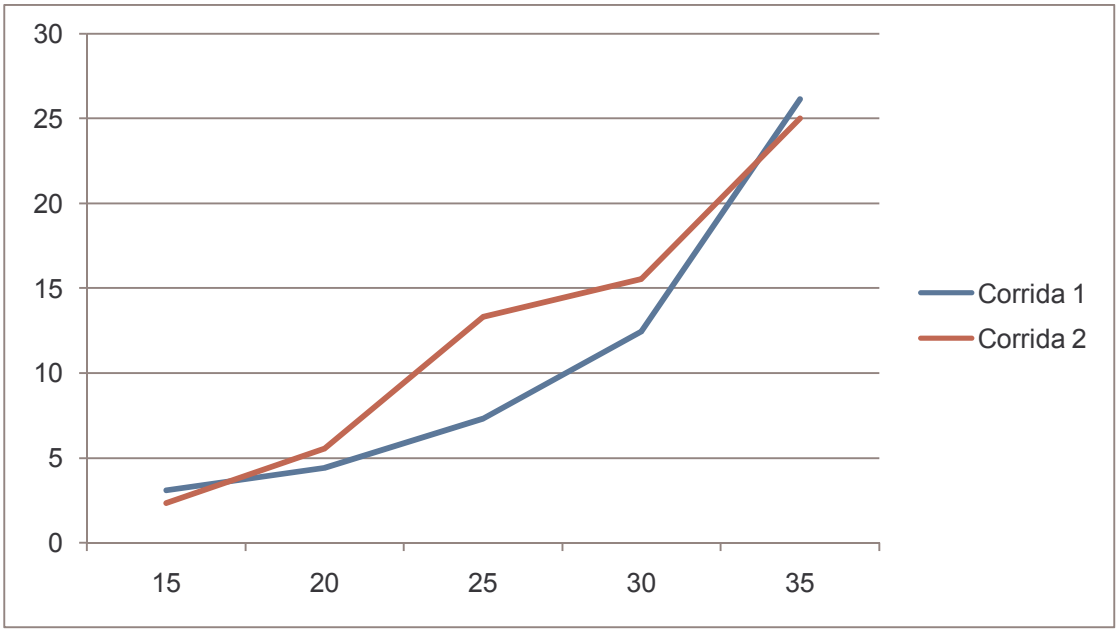


Figura 2.1.2 (b): Tiempo de ejecución para los juegos de datos.

2.2. CONSTRUCCIÓN PARALELA DEL ALGORITMO

El tiempo de ejecución del algoritmo secuencial podría llegar a ser notable para el análisis de grandes cantidades de secuencias, en aras de obtener una solución en menos tiempo, diseñamos un algoritmo paralelo.

Un recurso de mucha utilidad para paralelizar un determinado algoritmo es el grafo de dependencias de datos, el cual se puede construir partiendo del algoritmo secuencial, pues éste expresa en cierta medida el orden de las operaciones y los cálculos. Siempre que se desee diseñar un algoritmo paralelo, primeramente se debe pensar en cómo los datos iniciales del problema serían distribuidos por los distintos procesadores, así como los datos intermedios y finales que se irían calculando en los diferentes procesos. El grafo de dependencia de datos nos sirve como punto de partida para encontrar una distribución adecuada.

Las dependencias entre los procesos[1] pueden adquirir diferentes formas:

- **Dependencia de datos**: Se refieren a situaciones en que el orden en que se obtienen los datos marca el orden de ejecución de los procesos. Existen diferentes tipos de dependencia de datos, estas son:

  - ✓ **Dependencias de flujo:** Un proceso P_2 es dependiente por flujo de otro proceso P_1, si P_2 sigue en el orden de programa a P_1, y una salida del proceso P_1 se utiliza como entrada en el proceso P_2.

  - ✓ **Antidependencias:** Un proceso P_2 no es dependiente de otro proceso $P_{1,}$ si P_2 sigue en el programa a P_1 y una salida de P_2 se utiliza como entrada de $P_{1.}$

  - ✓ **Dependencia de Salidas**: Dos procesos son dependientes por la salida si escriben sobre recursos compartidos entre ellos.

  - ✓ **Dependencias de entrada/salida**: Se produce cuando dos procesos acceden a un mismo fichero o dispositivo.

  - ✓ **Dependencias desconocidas**: Existen situaciones en que hay peligro de dependencias de datos, pero no es posible conocerlas antes de la ejecución. Un ejemplo de este tipo de casos son las instrucciones que afectan a variables cuyos subíndices están a su vez, subindexados.

  Otros tipos de dependencias que pueden existir entre los procesos de un sistema paralelo son:

- **Dependencias de control**: Estas se producen cuando el orden de ejecución de los procesos no se puede conocer hasta en momento de la ejecución. Esto afecta a la ejecución de instrucciones o procedimientos afectados por instrucciones condicionales.

- **Dependencias de recursos**: Estas se deben a conflictos en el uso de recursos compartidos. Estos recursos pueden ser recursos de almacenamiento, tales como áreas de memoria compartida, recursos de ejecución, unidades aritméticas.

[1] Proceso: Debe interpretarse como cualquier fragmento de un programa en ejecución

Condiciones de Bernstein: Bernstein (1966) descubrió una serie de condiciones para que dos procesos pudieran ejecutarse en paralelo. Antes de analizar estas condiciones veamos una serie de definiciones previas.

Conjunto de entrada, conjunto de lectura o dominio de un proceso P_i como el conjunto de todas las variables de entrada necesarias para la ejecución de ese proceso. A este conjunto lo denotaremos por I_i.

Conjunto de salida, de escritura o rango de un proceso P_i es el conjunto formado por todas las variables modificadas después de ejecución de P_i. A este conjunto lo denominaremos como O_i.

Estos dos conjuntos están formados por operandos o resultados que pueden residir en registros del procesador, en memoria o incluso en ficheros.

Bernstein plantea que dado dos procesos P_1 y P_2 cuyos conjuntos de entrada son I_1 e I_2 y cuyos conjuntos de salida son O_1 y O_2 respectivamente. Estos procesos se podrán ejecutar en paralelo, y se denotará por $P_1 \mid\mid P_2$ si se verifican las siguientes condiciones.

$$O_1 \cap I_2 = \phi$$

$$I_1 \cap O_2 = \phi$$

$$O_1 \cap O_2 = \phi$$

Según el algoritmo secuencial a paralelizar, en un primer paso se calcula l_j (longitud de cada una de la ramas del árbol) a partir de las diferencias entre las secuencias padre e hijo de la rama j. Luego es calculado l_t (longitud del árbol) en función de los l_j. Como segundo paso se calculan los $b_S^{(r)}$ y $b_N^{(r)}$ promedio de sitios sinónimos para el codón r a partir de los datos de $b_{S_j}^{(r)}$ y $b_{N_j}^{(r)}$ que son el promedio de sitios sinónimos y no sinónimos para el codón r en cada una de las ramas j respectivamente. Los valores de $C_S^{(r)}$ y $C_N^{(r)}$ son calculados a partir de los valores $C_{S_j}^{(r)}$ y $C_{N_j}^{(r)}$. El problema queda reducido a los cálculos intermedios por cada uno de los sitios de las secuencias, se procede entonces al cálculo

final y objetivo principal de este algoritmo que es estimar el $d_S^{(r)}$ y $d_N^{(r)}$ (sustituciones sinónimas y no sinónimas por sitios)

Un grafo de dependencia que ilustra lo descrito anteriormente, es el que se muestra en la figura a continuación, en el mismo, los vértices constituyen los procesos y los arcos la comunicación entre ellos.

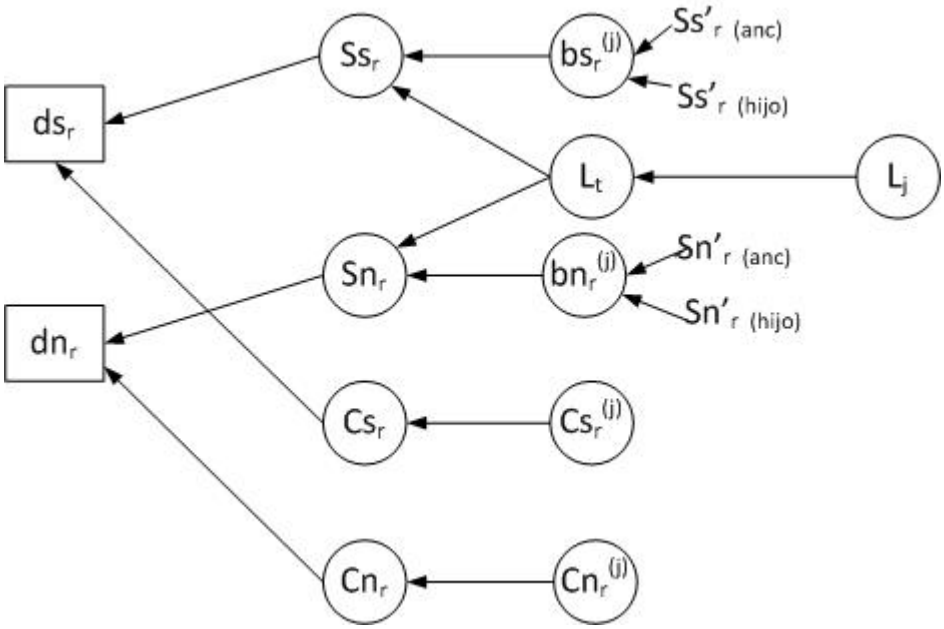


Figura 2.2 (a): Grafo de dependencia de datos

El proceso de cálculo de un ds_r, está en función de cada uno de los Ss_r y Cs_r, que a su vez, estos dependen de otros datos. En el algoritmo secuencial estos cálculos se realizan de forma iterativa y dependiente, un valor C_{S_j} se calcula después de haber calculado el de $C_{S_{j-1}}$, punto significativo, pues el cálculo de C_{S_j} no depende en lo absoluto del valor calculado de $C_{S_{j-1}}$, de forma análoga ocurre con los valores de b_{S_j}, b_{N_j} y C_{N_j}.

Es por ello que el algoritmo paralelo que solucione el análisis de selección positiva, debe ser capaz de calcular estos valores de forma concurrente.

Para la paralelización del algoritmo secuencial descrito, el dominio de datos a procesar se divide en varios “trozos”, y se asigna cada “trozo” a un procesador distinto para que le aplique un conjunto de pasos.

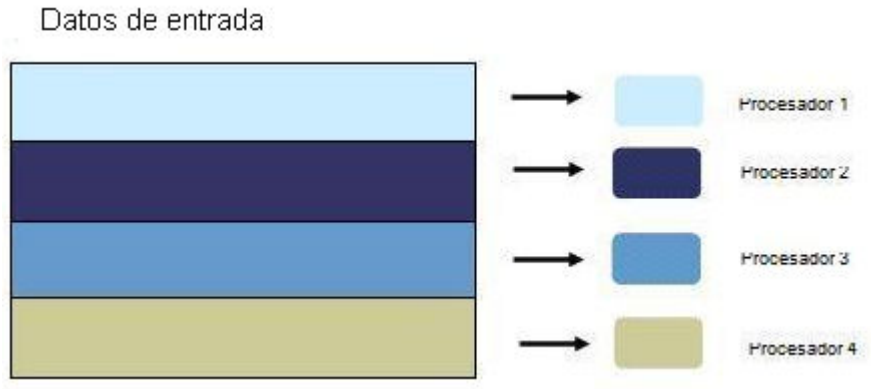


Figura 2.2 (b): Descomposición de datos de entrada

En nuestra paralelización del algoritmo de análisis de selección positiva el dominio de datos lo constituyen las secuencias de ADN o proteínas. A cada proceso se le asigna un subconjunto de los datos de entrada, para determinar si están sometidas a selección positiva.

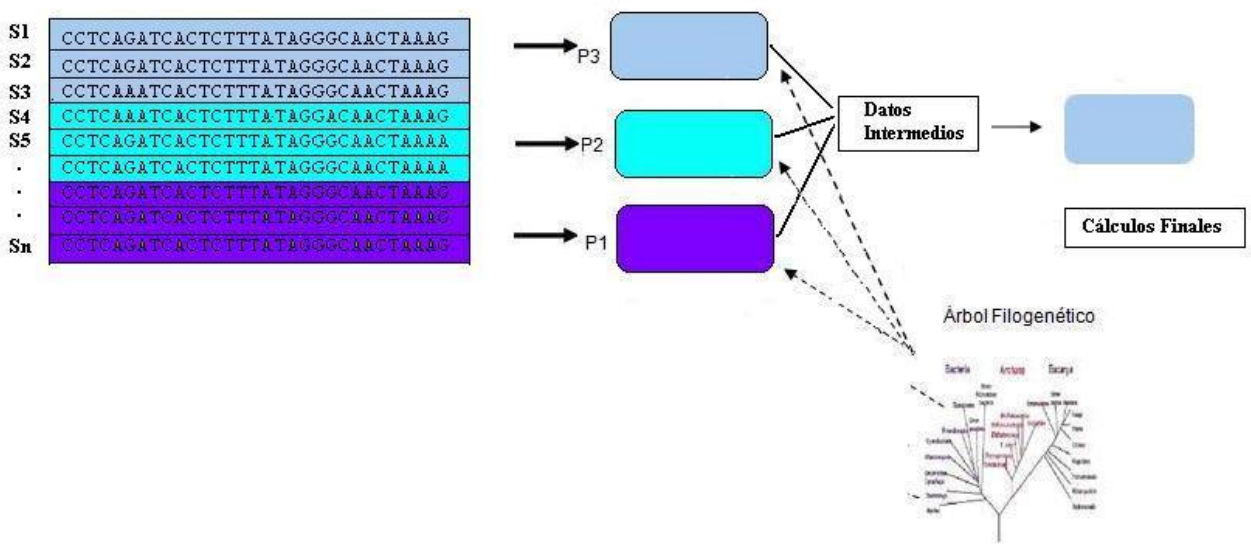


Figura 2.2(c): Esquema General de la propuesta del Algoritmo Paralelo

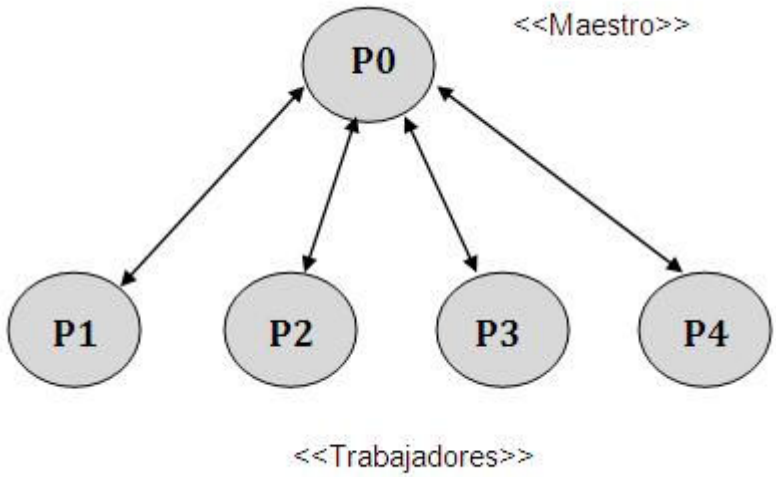
2.2.1 Estrategia de paralelización del algoritmo

Para la paralelización del algoritmo seguimos una serie de pasos o fases, importantes para el diseño de algoritmos paralelos. Estos son fundamentalmente los que se presentan a continuación.

- ✓ ***Descomposición funcional***: Identificar las funciones que debe realizar el algoritmo y las relaciones entre ellas.
- ✓ ***Partición:*** Distribución de las funciones en procesos y esquema de comunicación entre procesos. Objetivos: maximizar el grado de paralelismo y minimizar la comunicación entre procesos.
- ✓ ***Localización:*** *A*signación de los procesos a cada uno de los procesadores del sistema.
- ✓ Objetivo: ajustar los patrones de comunicación a la topología del sistema.
- ✓ ***Escalado:*** Determina el tamaño óptimo del sistema en función de algún parámetro de entrada.

Para la paralelización aplicamos dos tipos fundamentales de descomposición: ***La descomposición por datos*** para aplicar simultáneamente parte del algoritmo sobre diferentes porciones de datos independientes. Esta estrategia permite un buen balanceo de carga con poca comunicación, pero tiene el inconveniente que el conjunto de trabajo aumenta en cada uno de los procesadores y realizan cada uno funciones similares. Además de la descomposición funcional para aprovechar las fases independientes en el cálculo de las sustituciones sinónimas y no sinónimas por sitios para realizarlas en paralelo.

Se ha utilizado una estrategia maestro / trabajadores para descomponer el trabajo. El maestro reparte el trabajo entre una colección de trabajadores. La comunicación y sincronización entre estos elementos se propone que se realice mediante funciones de la librería de Paso de Mensaje (MPI).



.

Figura 2.2.1(a): Topología empleada en la paralelización.

El P0 indicará qué se ejecuta en cada procesador y se pondrá a esperar un mensaje de cada uno de ellos, que le informa los datos obtenidos y realiza los cálculos finales del proceso.

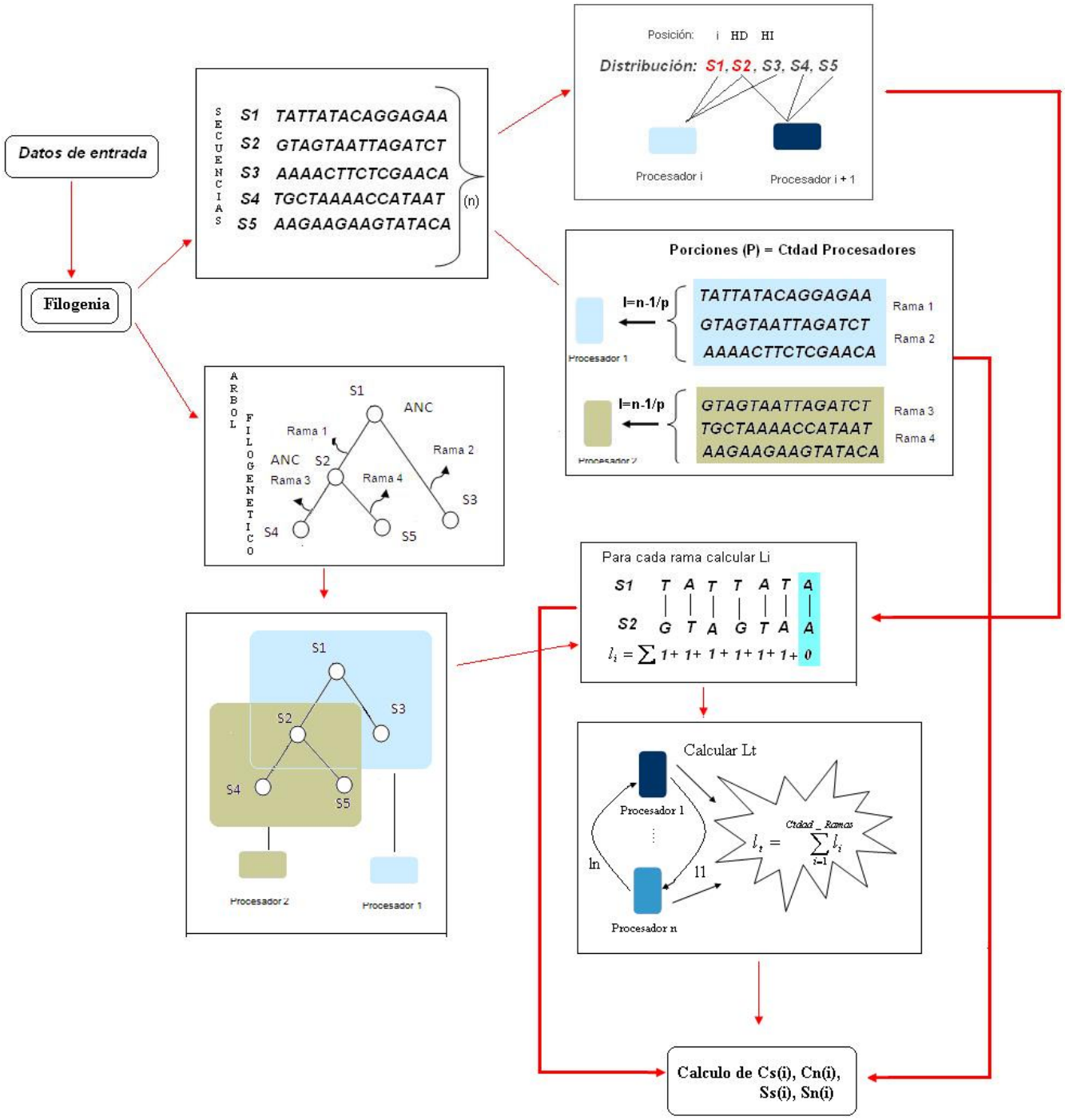


Figura 1.2.1 (b) Representación de las tareas a paralelizar

2.2.2 ALGORITMO PARALELO

Para distribuir los datos de las secuencias y el árbol entre los distintos procesadores se tuvo la disyuntiva de hacer la distribución por ramas (a cada procesador se le envía un conjunto de ramas de las secuencias) o por secuencias (a cada procesador se le envía una porción de cada secuencia). Para este problema, pensamos que la distribución por ramas permite una mejor prestación del algoritmo que la distribución por porciones de secuencias, pues el costo de comunicación es menor. Esta afirmación está dada porque la mayoría de los análisis son realizados sobre cada uno de los sitios de las ramas, o sea se hacen a partir de las secuencias ancestros con cada uno de sus secuencias hijas, y entonces para un procesador encargado de calcular los datos intermedios es mucho mejor tener almacenado en su memoria un conjunto de ramas "emparentadas" del árbol filogenético.

Distribuir las secuencias sin tener en cuenta a la rama a la que pertenece significaría más flujo de datos y por tanto una mayor comunicación, dado que se calculan los valores por porciones, cada procesador debe esperar por los datos del resto de los procesadores para terminar sus respectivos análisis.

Algoritmo

1. Se realiza la distribución de las secuencias (porciones de secuencias emparentadas) a cada procesador trabajador *(**Ver sección 2.2.3)***
2. Se calculan los datos intermedios **(Ver sección 2.2.3)**
    a. En cada procesador p':

        Para cada una de las ramas asignadas se calcula l_j y se envía al procesador central para cálculos posteriores.

        Luego se realiza un análisis por sitios, de cada una de las rama j asignadas al procesador p' y se calculan los valores de $b_{S_j}^{(r)}$ y $b_{N_j}^{(r)}$.

        Se multiplica cada uno de los $b_{S_j}^{(r)}$ y $b_{N_j}^{(r)}$ por el l_j correspondiente a la rama j.

        Se calculan además $C_{S_j}^{(r)}$ y $C_{N_j}^{(r)}$.

Estos valores obtenidos en cada uno de los procesadores p' son enviados al procesador central para realizar los cálculos finales

3. Se procede a ejecutar los cálculos finales (Ver sección 2.2.3).

a. En el procesador central

Se reciben los datos enviados por todos los procesadores p'

Se calcula l_t a partir de los l_j recibidos de cada procesador p'

Se obtienen además los C_N y C_S para cada uno de los sitios, a partir de los datos obtenidos de cada uno de los procesadores; y por último se calcula el d_S y el d_N por sitios (Objetivo final).

Distribución de datos a procesadores:

La distribución consiste en asignarle a cada procesador un conjunto de ramas emparentadas (i, HI, HD). Siendo n la cantidad de secuencias, n-1 la cantidad de ramas y P el número de procesadores, a cada procesador P' se le asignan $\left(n-1/|P|\right)$ ramas. De esta forma, cada procesador se encargará de calcular los datos intermedios correspondientes a las ramas que les fueron asignadas. (Paso 1)

En la siguiente figura se representan la distribución de los datos.

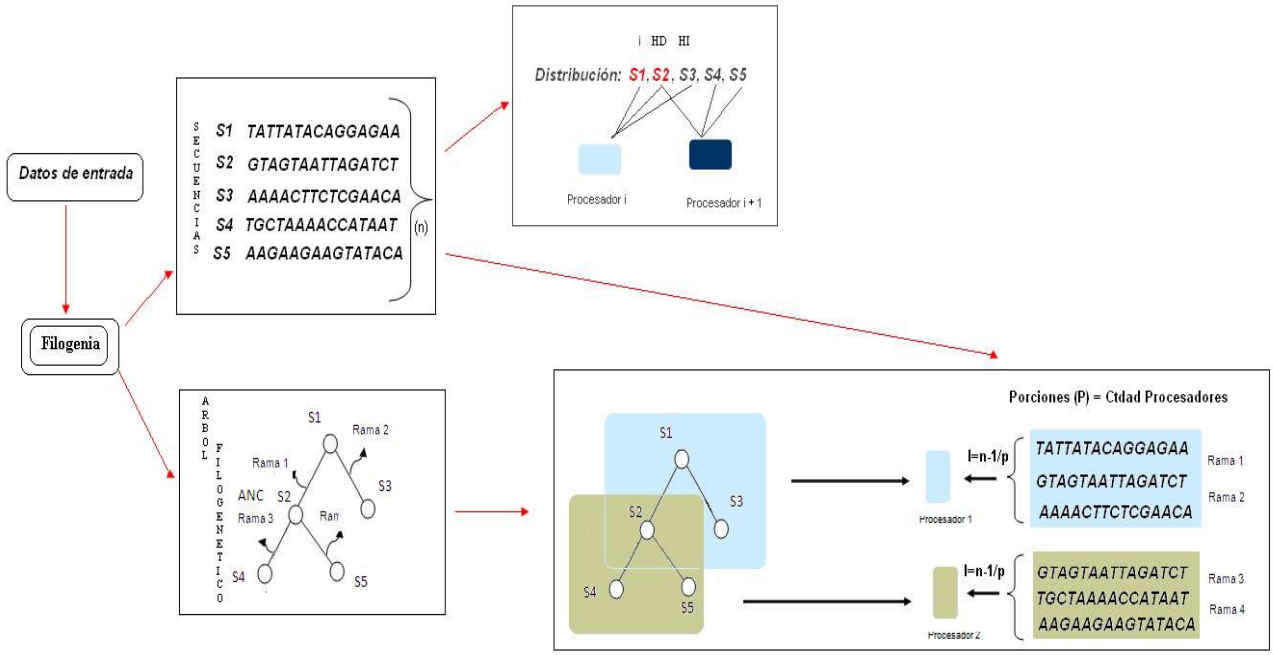


Figura 2.2.2 (a): Representación de la distribución de los datos.

Nótese que si el árbol es binario completo y sus nodos tienen un orden relativo al de las secuencias, podemos afirmar que se puede asignar a cada procesador p', las secuencias i, $2i$ y $2i+1$, o sea el nodo padre con las secuencias hijas correspondientes (HI, HD).

Todo procesador P' debe mandarle al procesador central P_c, los valores de las variables calculadas correspondientes a las ramas que le fueron asignadas (Fase de distribución). Luego el procesador maestro realiza los cálculos finales. (Fase de combinación).

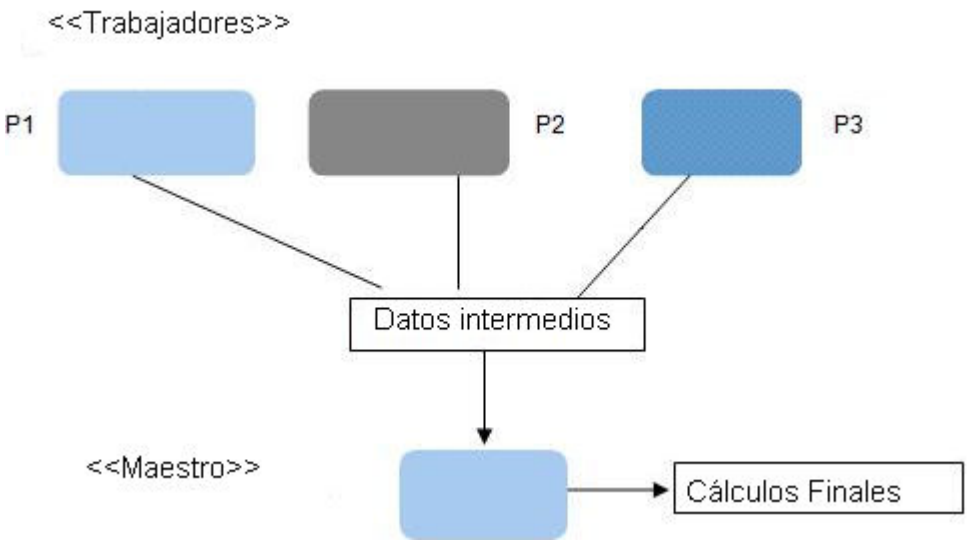


Figura 2.2.2 (b): Comunicación entre procesadores.

2.2.3 PSEUDOCÓDIGO DEL ALGORITMO PARALELO

/****Fase de entrada****/

Repartir todas las ramas entre los p procesadores

$P_i \leftarrow i, i+1, ..., i+\frac{n-1}{|P|}-1$ // Ramas emparentadas (Padre, HI, HD)

/****Fase de distribución****/

Trabajadores:

Para cada p' hacer:

Para j =i to $j+\frac{n-1}{|p|}-1$ // Para cada Rama

Hacer:

{

$l_j \leftarrow$ *Diferencia entre las* sec *uencias que componen* (j) // Rama j

Enviar (l_j, P_C) // enviar todos los l_j al procesador central.

Para r=0 to n-1 //Analizar cada sitio de la rama j

{

$b_{s_j}^{(r)} = \left(s_{s_k}^{(r)} + s_{s_h}^{(r)}\right)/2$ // Sustituciones sinónimas entre las secuencias // k y h para el sitio r.

$b_{n_j}^{(r)} = \left(s_{n_k}^{(r)} + s_{n_h}^{(r)}\right)/2$ // Sustituciones no sinónimas entre las // Secuencias k y h para el sitio r.

$b_{s_j}^{*} = \left(b_{s_j}^{(r)} * l_j\right)$ //Guardar en una matriz la multiplicación del // $b_{s_j}^{(r)}$ por el l_j correspondiente a la rama j

$b_{n_j}^{*} = \left(b_{n_j}^{(r)} * l_j\right)$ //Matriz que contiene por sitios la multiplicación del // $b_{n_j}^{(r)}$ por el l_j correspondiente a la rama j

Calcular $C_{s_j}^{r}$ // Diferencias de las sustituciones sinónimas entre //los sitios que formas las secuencias de la rama

Calcular $C_{n_j}^{r}$ // Diferencias de las sustituciones no sinónimas entre // los sitios que formas las secuencias de la rama

}

}

Enviar $\left(b_{S_j}^{*}, P_C\right)$ //Enviar la matriz al procesador central para calcular S_S

Enviar $\left(b_{N_j}^{*}, P_C\right)$ //Enviar la matriz al procesador central para calcular S_N

Enviar $\left(C_{S_j}^{r}, P_C\right)$ //Enviar las diferencias sinónimas al procesador central (d_s)

Enviar $\left(C_{N_j}^{r}, P_C\right)$ //Enviar las diferencias no sinónimas al procesador central (d_n)

Fin del trabajo

/*Fase de combinación***/**

Procesador Maestro //Realiza los cálculos finales a partir de los datos recibidos del resto de los procesadores

Inicio del trabajo:

Si P=0

l_t = o; //Inicializar l_t

Para p=1 to P-1 //Recibir por procesadores

Para j= p hasta $\frac{n-1}{|P|}-1$ //Recibir por ramas

{

Recibir (l_{j_p}, p)

$l_t += l_{j_p}$

Recibir $(M\_b^*_{s_j}, p)$ // M_b va a ser una matriz que contiene todos los
// b_S por sitios (r) y ramas (j)

Recibir $(M\_b^*_{N_j}, p)$ // $M\_b^*_{N_j}$ va a ser una matriz que contiene todos los
// b_N por sitios (r) y ramas (j)

Recibir $(M\_C^r_{s_j}, p)$ // $M\_C^r_{s_j}$ va a ser una matriz que contiene todos los
// C_S por sitios (r) y ramas (j)

Recibir $(M\_C^r_{N_j}, p)$ // $M\_C^r_{N_j}$ va a ser una matriz que contiene todos los
// C_N por sitios (r) y ramas (j)

}

Desde l=0 to cantidad de sitios //Para reducir a un arreglo

{

$b_S^*[l] += b_S^{(r)}[l]$

$S_S^{(r)}[l] = \sum b_S^*[l] / l_t$

$S_N^{(r)}[l] = \sum b_N^*[l] / l_t$

$$d_S[l] = \frac{C_S[l]}{S_S[l]}$$

$$d_N[l] = \frac{C_N[l]}{S_N[l]}$$

}

Fin del trabajo

2.3 LIBRERÍA DE DEFINICIONES Y FUNCIONES (MPI)

Para la comunicación entre los procesadores, se propone el uso de la librería de paso de mensajes MPI.

Un programa MPI tiene la siguiente forma:

```
# include "mpi.h"

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

MPI_Init (argc, char* argv [ ])  /*Ninguna función de MPI debe ser llamada antes que esta*/

MPI_Finalize (); /*Ninguna función de MPI puede ser llamada después que esta*/
}
```

MPI tiene implementada un conjunto de funciones que permite el paso de mensajes, entre los diferentes nodos (procesadores) que conforman el cómputo paralelo. Para la paralelización de este algoritmo, teniendo en cuenta la estrategia planteada en la sección anterior, es importante la utilización de un grupo de estas funciones. Estas funciones inician y terminan un cómputo, identifican procesos, además de enviar y recibir mensajes.

- *MPI_INT*: Inicia un cómputo.
  MPI_INT (int *argc, char ***argv), argc, argv son requeridos solo por el contexto del lenguaje C, en el cual son los argumentos del programa principal.
- *MPI_FINALIZE*: Termina un cómputo. MPI_FINALIZE ().

- *MPI_COMM_SIZE*: Determina el número de procesos en un cómputo. *MPI_COMM_SIZE* (comm, size), IN comm comunicador (manejador [handle]), OUT size número de procesos en el grupo del comunicador (entero).
- *MPI_COMM_RANK*: Determina el identificador del proceso actual ``mi proceso''. *MPI_COMM_RANK* (comm, pid), IN comm comunicador (manejador [handle]), OUT pid identificador del proceso en el grupo del comunicador (entero).
- *MPI_SEND*: Manda un mensaje. MPI_SEND (buf, count, datatype, dest, tag, comm). IN buf dirección del buffer a enviar (tipo x), IN count número de elementos a enviar del buffer (entero>=0), IN datatype tipo de datos del buffer a enviar (handle), IN dest identificador del proceso destino (entero), IN tag message tag (entero), IN comm comunicador (handle)
- *MPI_RECV*: Recive un mensaje. MPI_RECV (buf, count, datatype, dest, tag, comm). OUT buf dirección del buffer a recibir (tipo x), IN count número de elementos a recibir del buffer (entero>=0), IN datatype tipo de datos del buffer a recibir (handle), IN source identificador del proceso fuente, o MPI_ANY_SOURCE(entero), IN tag message tag, o MPI_ANY_TAG(entero), IN comm comunicador (handle), OUT status estado del objeto (estado)
- *IN*: Significa que la función usa pero no modifica el parámetro.
- *OUT*: Significa que la función no usa pero puede modificar el parámetro.
- *INOUT:* Significa que la función usa y modifica el parámetro.

La unidad básica en el modelo de paso de mensajes implementado en este algoritmo son los mensajes, y específicamente las funciones de envío (MPI_Send) y recepción de estos (MPI_Recev), para el logro de la sincronización entre los procesos. Al enviarse un mensaje desde un procesador i hacia un procesador j, utilizando MPI_Send, debe ser recibido utilizando MPI_Recv en el procesador j, indicando que el mensaje viene desde el procesador i. El no establecer este sincronismo, provocará que uno o ambos procesadores se queden bloqueados. Esto se debe a que algunas funciones de envió y recepción de mensajes de MPI son bloqueadoras, es decir que se bloquean hasta que no se termina la operación u ocurre un error. Un mensaje está compuesto por los datos a enviar (Datos) y por la información que define hacia donde va el mensaje (**Sobre**).

El algoritmo envía a cada destinatario (procesador) varios tipos de mensajes y el destinatario da tratos distintos a cada uno de ellos, esa función se realiza adjuntando al *sobre* un *tag*, el cual puede ser un número que permita clasificar los mensajes por tipos.

2.4. MÉTRICAS DE EVALUACIÓN DEL DESEMPEÑO DE PROGRAMAS PARALELOS

Para evaluar el desempeño de los algoritmos paralelos, existen un conjunto de métricas. Entre las más conocidas se encuentran el tiempo de ejecución paralelo, el speedup (ganancia efectiva en velocidad de cómputo usando más de un procesador) y eficiencia (que denota el uso efectivo de los recursos de cómputo). Pero existen otras medidas que pueden ser útiles tales como costo, overhead paralelo, grado de concurrencia, escalabilidad, isoeficiencia, etc. A continuación daremos a conocer las principales métricas a tener en cuenta para evaluar el desempeño de los algoritmos paralelos. Para una futura evaluación de algoritmo propuesto, luego de su implementación.(Ver recomendaciones)

2.4.1. VELOCIDAD COMPUTACIONAL

Hay varios factores que influyen directamente en la velocidad computacional de un sistema paralelo. Entre los más usados están:

2.4.1.1 Tiempo de ejecución:

El tiempo de ejecución de un programa secuencial es el tiempo que pasó entre el principio y el fin de su ejecución en una computadora secuencial. Mientras que para un sistema paralelo es el tiempo que pasa desde la salida del primer procesador hasta la del último que termina su ejecución. Nosotros denotamos el tiempo de ejecución del algoritmo secuencial como TS y del paralelo por TP.

2.4.1.2 Granularidad de los procesos:

Para intentar lograr una mejora de la velocidad computacional en una aplicación se divide el cálculo en procesos que se puedan ejecutar de forma simultánea. El tamaño de un proceso se describe por su granularidad.

Un proceso con granularidad gruesa es aquel que esta compuesto por un gran número de instrucciones secuenciales mientras que un proceso es de granularidad fina cuando dispone de pocas instrucciones secuenciales. Normalmente se desea incrementar la granularidad para reducir los costos de la creación de procesos; pero eso reduce el número de procesos concurrentes y la cantidad de paralelismo por lo que hay que encontrar un compromiso entre una cosa y la otra.

En el modelo de paso de mensaje es necesario reducir la sobrecarga en la comunicación, sobre todo en los clúster donde la latencia de comunicación puede llegar a ser importante y dominar sobre el tiempo total de ejecución.

Esta razón que aparece a continuación se usa como medida de granularidad y se busca maximizarla.

$$G = \frac{Tiempo\ de\ Computación}{Tiempo\ de\ Comunicación} \qquad (1)$$

Cuando el cociente es pequeño se aconseja usar el tipo de computadores en cuestión en algoritmos que necesiten comunicaciones frecuentes y en caso del cociente grande serán adecuados para algoritmos que no necesiten muchas comunicaciones

2.4.1.3 Factor de aceleración (speedup)

Este factor mide el rendimiento relativo entre un sistema paralelo (SP) y un sistema con un único procesador (TS) y se define así:

$$S(N) = \frac{t(1)}{t(N)} \qquad (2)$$

Donde $t(1)$ es el tiempo empleado para ejecutar el proceso en un solo procesador y $t(N)$ es el tiempo empleado para ejecutarlo en el sistema paralelo con N procesadores, en condiciones ideales $t(N) = 1/N$ con lo que en esas condiciones, la ganancia de velocidad (Speedup) dada por la ecuación 2 será.

$$S(N)ideal = \frac{t(1)}{t(N)} = \frac{1}{1/N} = N \qquad (3)$$

Hay varios factores que pueden sobrecargar los programas paralelos y que limitan este factor, entre ellos están:

- ✓ Los períodos en los que no todos los procesadores están realizando un trabajo útil.
- ✓ Los cálculos adicionales que aparecen en el programa paralelo y no están en el secuencial.
- ✓ El tiempo de comunicación para enviar mensajes.

2.4.1.4 Eficiencia (E)

Una forma de medir el rendimiento de un sistema paralelo es comprobando la ganancia de velocidad del sistema con la ganancia de velocidad ideal dada por (3), a esta medida se le denomina eficiencia de un sistema paralelo se obtiene al normalizar la aceleración de un programa paralelo dividiéndolo entre el número de procesadores:

$$E = \frac{S(N)}{S(N)\,Ideal} = \frac{S(N)}{N} = \frac{t(1)/t(N)}{N} = \frac{t(1)}{N\,t(N)} \qquad (4)$$

Esto da la fracción de tiempo que utilizan los procesadores durante la computación, o sea, una media de cuan eficiente se han utilizado los procesadores.

2.4.1.5 Coste (C)

El coste o trabajo se define como:

$$C = t(N) * N \qquad (5)$$

Así, el coste óptimo de un algoritmo paralelo es aquel proporcional al coste que tiene en un sistema con un único procesador.

Estas métricas son utilizadas para evaluar el performance de los algoritmos paralelos y tener una idea de la velocidad computacional que se puede lograr con el paralelismo

Conclusiones:

En este capitulo se presentó una propuesta de algoritmo paralelo para el análisis de selección positiva, obtenida a partir de la aplicación de técnicas de paralelización. Se describe este algoritmo a partir de su pseudocódigo. Además se presentaron las principales métricas de evaluación del desempeño de los algoritmos paralelos, para una vez que nuestra propuesta haya sido implementada, pueda ser evaluada.

CONCLUSIONES

En este trabajo se presentó el diseño de un algoritmo para el análisis de selección positiva basado en el diseño de cómputos paralelos, teniendo en cuenta las técnicas de descomposición, comunicación y sincronización entre los procesos.

La idea básica utilizada fue la de paralelización de un algoritmo secuencial existente, de análisis de selección positiva, cuya función es: analizar secuencias de ADN y proteínas para conocer que sitios tienen mayores probabilidades de mutar satisfactoriamente. El diseño de la estrategia estuvo basado en los principios para el diseño y cálculo de algoritmos paralelos.

Como paradigma de programación paralela se usó la biblioteca de dominio público MPI (Interfaz de paso de Mensajes), que brinda un ambiente de programación concurrente, distribuido y de propósito general.

Se mostraron las métricas fundamentales (speedup, eficiencia, escalabilidad, costo) para representar el desempeño del algoritmo paralelo presentado, estimando una mejora en el tiempo de ejecución en comparación con el secuencial existente, lo que justifica el posible costo que produce la implementación paralela. Concluyendo que: la estrategia paralela planteada para el proceso de análisis de selección positiva, se estima que sea más eficientes que la secuencial existente.

Los aportes principales del trabajo han sido:

- Proponer un seudocódigo de un algoritmo paralelo, para el proceso de análisis de selección positiva
- Mejorar el estudio del proceso biológico de análisis de selección positiva, en los campos de la Biología Molecular y la Epidemiología en los centros del Polo Científico de Cuba.

RECOMENDACIONES:

- Continuar esta investigación, para el logro de la implementación del algoritmo propuesto.
- Implementar este algoritmo utilizando otros modelos de programación paralela.
- Cálculo de las métricas de desempeño para programas paralelos.
- La utilización de este algoritmo por los centros del polo científico de nuestro país, para mejorar el proceso de análisis de selección positiva.

REFERENCIAS BIBLIOGRÁFICAS

[1] Joaquín Dopazo y Alfonso Valencia. "Bioinformática y genómica". 2001. pág. 70. http://bioinfo.cipf.es/docus/courses/filogenias/BioinfoGenomica.pdf.

[2] Hilda Medrano Castañeda. *"Arbol de la vida"*. Sitio entre nosotros. 2000.

[3] Oliva, José de Jesús Rivero; Daniel Milián Lorenzo;. *"Alineamiento de Secuencias".*

[4] Salemi, Marco; Anne Mieke Bañadme;. *"The Phylogenetic Handbook".*

[5] Daniel Peña. "Fundamentos de estadística". Madrid. España : Alianza Editorial, 2001.

[6] Dr Lee Margetts. *"Supercomputación y cálculo en paralelo".* Reino Unido : s.n., 2006.

[7] Grama, Ananth; Anshul Gupta; George Karypis; Vipin Kumar;. *"Introduction to Parallel Computing".* 2003.

[8] José Miguel Alonso. "Programación de aplicaciones paralelas con MPI (Message Passing Interface)". España : s.n., 1997.

BIBLIOGRAFÍA

1. Clúster de computadoras. nº Disponible en: http://es.tldp.org/Manuales-LuCAS/doc-cluster-computadoras/doc-cluster-computadoras-html/node8.html

2. Computación paralela. nº Disponible en: http://www.llnl.gov/computing/tutorials/parallel_comp/

3. The MPI Standard. . nº Disponible en: www.mpi-forum.org.

4 ¿Qué es la filogenia? nº Disponible en:

http://ar.answers.yahoo.com/question/index?qid=20060930112936AAK9dU6.

5. *Árbol de la Vida.* Disponible en: http://www.sciencemag.org/feature/data/tol.

6. *MPI: A Message-Passing Interface Standard.* 1994, nº p. 94-230.

7. Part 2. *Detecting selection – likelihood methods – PAML (*Phylogenetic Analysis by Maximum Likelihood), 2005, nº Disponible en: http://ocw.mit.edu/NR/rdonlyres/Electrical-Engineering-and-Computer-Science

8 ABASCAL, F. *Introducción a la Filogenia Molecular*. nº Disponible en:

http://www.cnb.uam.es/~fabascal/Filogenia_molecular/Filogenia-resumen.pdf.

9. ALONSO, J. M. *Programación de aplicaciones paralelas con MPI* Disponible en: http://www.sc.ehu.es/acwmialj/edumat/mpi.pdf.

10. ALTMAN, D. *De un sistema paralelo a uno compensatorio*

11. BARNEY, B. *Introduction to Parallel Computing* Disponible en: http://www.llnl.gov/computing/tutorials/parallel_comp/.

12. BENTON, M. Exactitud de los Fósiles y de sus Métodos de Medición. 2001, nº Disponible en: http://www.actionbioscience.org/esp/evolution/benton.html.

13. CARRASCO, J. M. G. *Herramientas para programación paralela*. nº

14. CASTAÑEDA, H. M. Árbol de la vida. *Sitio entre nosotros*, 2000, nº Disponible en: http://www.correodelmaestro.com/anteriores/2000/mayo/nosotros48.htm.

15. CINTRA, M. y LLANOS, D. R. Toward efficient and robust software speculative parallelization on multiprocessors. En *SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP). 2003*.

16. CORONA, C. A. C. *Estrategias coordinadas paralelas basadas en Soft-Computing para la solución de problemas de optimización.* Doctoral, Departamento de Ciencias de la Computación e Inteligencia Artificial. Universidad de Granada, 2005.

17. DÍAZ, G.; HOEGER, H.*, et al. Clústers de PCs*

18. DÍAZ, S. V. *Sistemas Multiprocesadores.*

19. DOPAZO, J. y VALENCIA, A. *Bioinformática y Genómica*. 2001, nº p. 2.

20. DRAKOS, N. y MOORE, R. *Construcción y evaluación de desempeño de un clúster tipo Beowulf para cómputo de alto rendimiento.* nº Disponible en:

http://hdgd.com.mx/~roadmr/tesis/escrito-1-split/node1.html.

21. EUROPEA, C. *El árbol de la vida: recuperar nuestras raíces*. 2007, nº Disponible en: http://ec.europa.eu/environment/news/efe/25bis/article_4255_es.htm

22. FOSTER, I. *Designing and building parallel programs. Concepts and Tools for Parallel Software Engineering.* 1995.

23. GARCÍA, J. M. *Un Entorno Avanzado para la Simulación de Multiprocesadores.* [Dpto. de Informática Universidad de Castilla-La Mancha 02071-ALBACETE (Spain)].

24. GOLDMAN, N. y YANGF, Z. *A Codon-based Model of Nucleotide Substitution for Protein-coding DNA Sequences*. 1994, nº Disponible en:

http://mbe.oxfordjournals.org/cgi/reprint/11/5/725.pdf?ck=nck

25. GRACIA, A. S. *Evolución molecular de los genes del sistema olfatorio OS-E y OS-F en diferentes especies de Drosophila*. nº p. 205-206.

26. GRAMA, A.; GUPTA, A., *et al. Introduction to Parallel Computing.* Segunda ed. Reading, Mass.: Addison-Wesley, ©1995. , 2003. 856 p. ISBN 0-201-64865-2.

27. GROPP, W.; LUSK, E., *et al.* Using MPI: *Portable Parallel Programming with the Message-Passing Interface*. 1994, nº

28. GUPTA, M. y NIM, *R. Techniques for run-time parallelization of loops. Supercomputing*. 1998, nº

29. HARO, J. J. D. *Sistemática Filogenética*. 2005, nº Disponible en: http://perso.wanadoo.es/jjdeharo/sistematica/curso/index.html

30. HIDROBO, F. y HOEGER, H. *Introducción a MPI.* Centro Nacional de Cálculo Científico - CeCalCULA - Universidad de los Ángeles,

31. KRIEGER, M. J. B. y ROSS, K. G. *Molecular evolutionary analyses of the odorant-binding protein gene Gp-9 in fire ants and other Solenopsis species*. nº Disponible en: http://mbe.oxfordjournals.org/cgi/reprint/msi203v1.pdf.

32. MARGETTS, D. L. *Supercomputación y cálculo en paralelo*. nº

33. MENEZES, R. *A New Approach to Scalable LindaSystems Based on Swarms*. nº Disponible en: http://cs.fit.edu/~rmenezes/research/publications/SwarmLinda/cs-2003-04.pdf

34. MENTON, D. N. *La ontogenia recapitula la filogenia* nº Disponible en: http://www.herbario.com.br/cie/universi/teoria/1031onfe.htm

35. MIGUEL ANGEL PÉREZ. *Arquitecturas paralelas II*. 2001., nº

36. MURILLO, J. J.; RUFAS, D. C., *et al. Metodología de codiseño hardware-software para la computación paralela distribuida dentro de un chip.* 2006, nº p. 2.

37. MYCO-UAL, G. *Evolución, filogenia, cladística*. 2007, nº Disponible en: http://www.ual.es/GruposInv/myco-ual/clados.htm

38. NESMACHNOW, S. *Una Versión Paralela del Algoritmo Evolutivo para Optimización Multiobjetivo NSGA-II y su Aplicación al Diseño de Redes de Comunicaciones Confiables.*

Diploma, Centro de Cálculo, Instituto de Computación Facultad de Ingeniería. Universidad de la República,

39. NIETO, E. J. P. *Clúster Heterogéneo De Computadoras* nº Disponible en: http://www.wikilearning.com/lam_mpi-wkccp-9705-13.htm.

40. OLIVA, J. D. J. R. y LORENZO, D. M. *Alineamiento de Secuencias.* Maestría,

41. ORTEGA, J. L. *Paralelismo en Clúster.* 2003,

42. PACHECO, P. S. *Parallel Programming with MPI.* Morgan Kaufmann 1997.

43. PALANCAR, J. H. *Paralelismo- Estado actual y proyección futura.*

44. PÉREZ, J. J. P. *Desarrollo de un clúster computacional para la compilación de algoritmos en paralelo en el Observatorio Astronómico.*

45. PÉREZ, V. B. *Análisis, Predicción y Visualización del Rendimiento de Métodos Iterativos en HPF y MPI*. 2002, nº p. 36-38.

46. PLATA, O. *Modelos y Arquitecturas Escalables.* Dept. Arquitectura de Computadores, Universidad de Málaga. 2002

47. PUERTAS, P. G. *Alineamiento de secuencias*. nº Disponible en: http://novacripta.cbm.uam.es/bioweb/courses/CBMSO2005/alignment/Alignments.pdf.

48. PYBUS, O. *Viral Evolution and Bioinformatics*. nº Disponible en: http://evolve.zoo.ox.ac.uk.

49. RAUCHWERGER, L. y PADUA, D. A. *The LRPD test: Speculative run-time parallelization of loops with privatization and reduction parallelization.* 10 ed. 1999, vol. 2, 160–180 p.

50. RÍO, F. M. D.; RUIZ, A. J. R.*, et al. Construcción en paralelo de la representación de polígonos mediante capas.* Departamento de informática (Universidad de Jaén)

51. SALCEDO, J. L. C. *Arquitectura de Sistema de Base de Datos visión general de las técnicas de análisis de datos.* 2006, nº Disponible en: http://elticus.com/

52. SALEMI, M. y BAÑADME, A. M. *The Phylogenetic Handbook.* 283-311 p.

53. SÁNCHEZ, L. P. *Programación Paralela basada en la Programación Lógica con Restricciones.* Trabajo de Diploma presentado como parte de los requisitos para obtener el título de Licenciado en Ciencias de la Computación, Facultad de Matemática y Computación. Universidad de la Habana, 2003.

54. SUNDERAM, V. S. *PVM: A framework for parallel distributed computing. Concurrency: Practice and Experience.* 4 ed. 1990, vol. 2, 315-339 p. Disponible en: www.csm.ornl.gov/pvm/pvmhome.html.

55. VALÍN, F. Á. *Evolución Molecular: Neutralismo y Seleccionismo*. nº p. 253-257.

56. VIDAL, A. M. *Presente y futuro de la Computación Paralela*. 2000, nº

57. YANG, Z. PAML: *Phylogenetic Analysis by Maximum Likelihood*. 2005, nº p. 32-51.

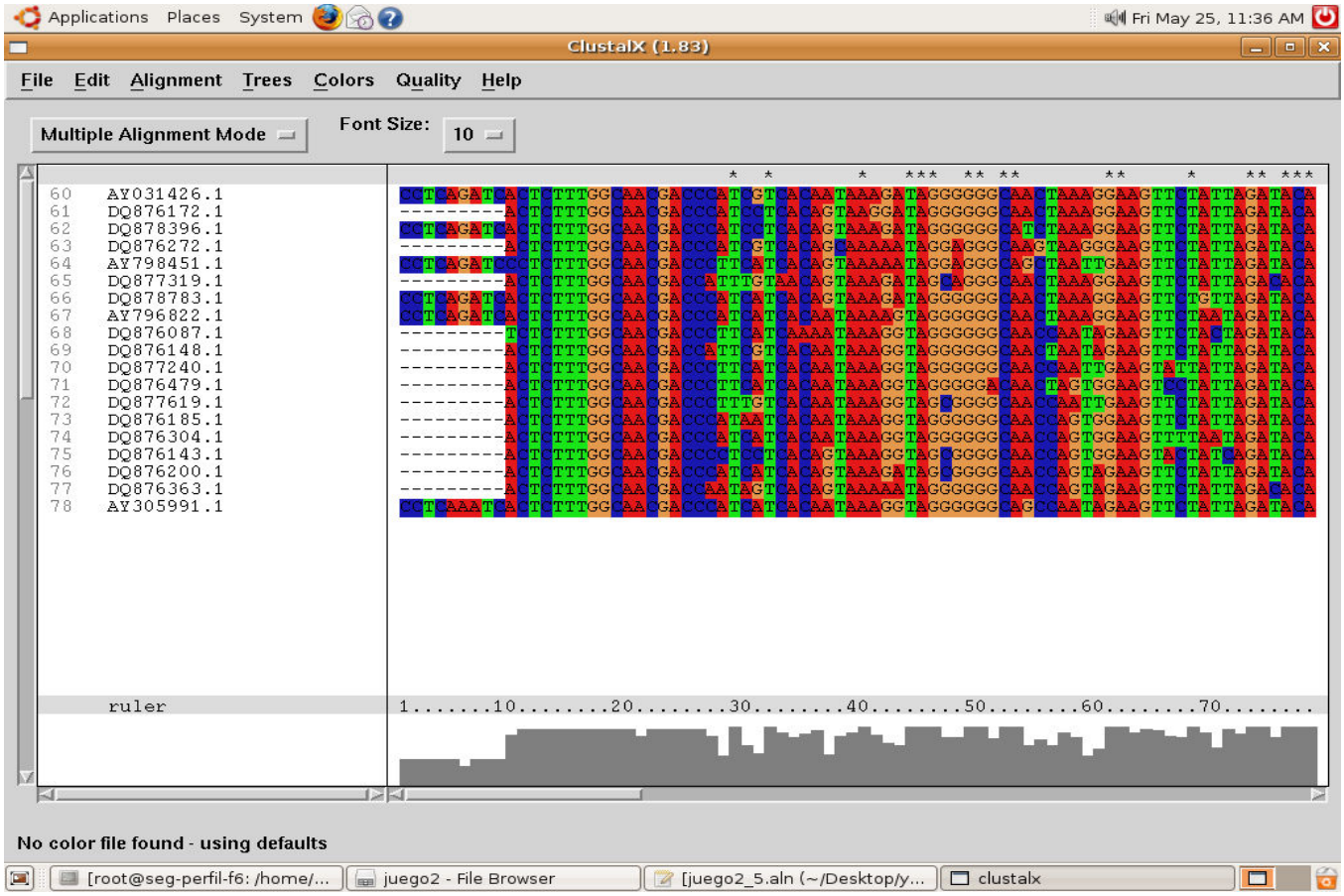
58. ---. *PAML: Phylogenetic Analysis by Maximum Likelihood.* 3.15 ed. England: 2005, Disponible en: http://abacus.gene.ucl.ac.uk/software/palm.html

59. YANG, Z.; N. GOLDMAN*, et al. Comparison of models for nucleotide substitution used in maximum likelihood phylogenetic estimation.* 11 ed. 1994, vol. 3, 16-324 p. Disponible en: http://mbe.oxfordjournals.org/cgi/reprint/11/2/316.pdf

60. YANG, Z. y NIELSEN, R. *Estimating Synonymous and Nonsynonymous Substitution Rates Under Realistic Evolutionary Models.* 17 ed. 200, vol. 1, 32-43 p.

ANEXOS

Anexo 1: Programa Auxiliar ClustalX



GLOSARIO DE TÉRMINOS

ADN: Ácido desoxirribonucleico, material genético formado por dos cadenas complementarias de nucleótidos trenzadas en forma de doble hélice. Estructuralmente la molécula de ADN se presente en forma de dos cadenas helicoidales arrolladas alrededor de un mismo eje (imaginario); las cadenas están unidas entre sí por las bases que la hacen en pares. Los apareamientos son siempre adenina-timina y citosina-guanina. El ADN es la base de la herencia.

Aminoácido: Compuesto que tiene un grupo ácido del tipo -COOH y un grupo amino del tipo -NH2. Moléculas con diversas funciones. Veinte de ellos constituyen elementos básicos para la formación de proteínas (se combinan para formar las proteínas)

Árbol filogenético: árbol que muestra las relaciones de evolución entre varias especies o otras entidades que se cree que tuvieron una descendencia común.

ARN: Cadena de nucleótidos que difiere del ADN por tener el azúcar ribosa en lugar de desoxirribosa, así como la base uracilo en vez de la base timina. Formado por una sola cadena. El ARN participa en la traducción de las instrucciones codificadas en el ADN para la producción de proteínas. Según sus funciones, el ARN puede ser "de transferencia" (ARNt), "mensajero" (ARNm) o ribosómico (ARNr).

Código genético: Conjunto de tripletas o codones que codifican los distintos aminoácidos que forman las proteínas. Existen 64 tripletas o codones para codificar los veinte aminoácidos, por lo que un aminoácido está codificado en la mayoría de los casos por varias tripletas o codones, por eso decimos que el código genético es un código degenerado

Codón: Cada uno de los tripletes de nucleótidos de ARN mensajero, cuyas bases codifican un aminoácido para la síntesis de proteínas.

Costo: Representa el tiempo (el trabajo) realizado por todo el sistema en la resolución del problema. Es el tiempo de ejecución por el número de procesadores usados.

Deriva genética: Cambios en la constitución genética a través del tiempo (ej Cambio evolutivo). Esta es generada por mutaciones y recombinaciones.

Divergencia: Sinónimo de evolución y supone que a partir de una sola forma animal A se puede observar que en las sucesivas generaciones los descendientes van difiriendo algo hasta que resultan en las formas B y C, de forma muy diferente.

Eficiencia: Denota el uso efectivo de los recursos de cómputo. Da idea de la porción de tiempo que los procesadores se dedican a trabajo útil, o sea, una media de cuan eficiente se han utilizado los procesadores.

Escalabilidad: Medida de la capacidad de un sistema de incrementar el speedup en proporción al número de procesadores. Esta refleja la capacidad de una arquitectura o algoritmo de usar de forma eficaz un número creciente del mismo (procesadores). El tema de la escalabilidad, y su relación con la función de isoeficiencia, es de importancia dado que permiten capturar las características de un algoritmo paralelo así como de la arquitectura en la que se le implementa.

Expresión génica: Transformación de la secuencia de nucleótidos del ADN en la secuencia de nucleótidos del ARN para posteriormente traducirse en una secuencia de aminoácidos, es decir se forma una proteína.

Fijación: Proceso mediante el cual una mutación se establece en la población.

Grado de concurrencia: Número máximo de tareas que pueden ejecutarse simultáneamente en un programa paralelo en cualquier momento dado. El grado de concurrencia generalmente depende de la forma del grafico de dependencia y de la granularidad. En general, para los grafos de dependencia de tareas, el grado máximo de concurrencia es siempre igual al número de hojas en el árbol.

Isoeficiencia: Este método propone que un sistema es escalable si es posible mantener la eficiencia del sistema a un valor fijo al aumentar el número de procesadores y el tamaño del problema convenientemente. Esto, da idea de cómo debe crecer el tamaño del problema en función del número de procesadores para mantener la eficiencia constante. Generalmente basta comparar el tiempo secuencial con la función overhead del paralelo

Isovelocidad (isospeed): Este método propone que el sistema es escalable si, al aumentar el número de procesadores, puede mantenerse constante la velocidad de proceso por procesador (número de operaciones por unidad de tiempo y por procesador), aumentando si es necesario el tamaño del

problema. En este caso también, el aumento del tamaño del problema necesario para mantener la velocidad constante es un indicador de la escalabilidad del sistema.

Métricas: Evaluación a través de los requerimientos de tiempo y espacio del programa. Están ligadas tanto al algoritmo como a la arquitectura paralela de destino.

Nucleótido: Molécula formada por una base nitrogenada, una pentosa y una molécula de ácido fosfórico. Constituye la base de los ácidos nucléicos, ADN y ARN. El nucleótido se considera un monómero, mientras que los ácidos nucléicos serían los polímeros.

Overhead paralelo (función overhead): Nos dices qué costo nos está tomando las operaciones en paralelo. Esta es calculada mediante la resta del tiempo de ejecución paralelo y el secuencial. (Aunque hay otra idea de overhead y es difícilmente medible)

Proteínas: Molécula de naturaleza polipeptídica que desempeña una función característica en los seres vivos, en los que puede haber miles de ellas diferentes. Las proteínas son compuestos orgánicos complejos, cuya estructura básica es una cadena de aminoácidos.

Recombinación: Proceso mediante el cual se intercambia material genético, trayendo como resultado la formación de un descendiente de acido nucleico (compuesto de material de 2 o mas ácidos nucleicos parentales).

Sistemas biológicos: Los sistemas biológicos, como las proteínas, son sistemas de gran tamaño constituidos por un número muy grande de átomos. En consecuencia es imposible a nivel computacional tratar estos sistemas en el marco mecanocuántico descrito anteriormente.

Sistemas paralelos: Combinación de algoritmo y arquitectura paralela.

Sustitución: Es la mutación fijada en una población

Sustituciones no sinónimas: Son aquellos cambios nucleotídicos que producen cambios de aminoácidos, es decir, modifican la secuencia de aminoácidos de la proteína

Sustituciones sinónimas: Son sustituciones silenciosas; no repercuten en la proteína, pues codifican para el mismo aminoácido

Sustituciones transicionales: Sustitución de un nucleótido púrico o uno pirimidínico por el otro de su misma clase. Por razón de probabilidad, son mucho más frecuentes las transversiones que las transiciones, ya que en el primer caso cada nucleótido se puede cambiar por dos mientras en el segundo la sustitución se hace solamente por otro.

Sustituciones transversionales: Cambio que se origina en el ADN por sustitución de un nucleótido por otro. Se sustituye un nucleótido de purina por cualquiera de las dos de pirimidina, o la inversa.

Velocidad (Speedup): Ganancia efectiva en velocidad de cómputo usando más de un procesador. Mide la ganancia de velocidad que se obtiene con un programa paralelo. Es el cociente entre el tiempo secuencial y el paralelo.