INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE INGENIERÍA MECÁNICA Y ELÉCTRICA UNIDAD ZACATENCO

DISEÑO E IMPLEMENTACIÓN DE UN CLUSTER TIPO BEOWULF PARA EL DESARROLLO DE CÓMPUTO CIENTÍFICO AVANZADO

TESIS:

QUE PARA OBTENER EL TÍTULO DE INGENIERO EN COMUNICACIONES Y ELECTRÓNICA

PRESENTAN:

GONZÁLEZ RAMÍREZ EDGAR RUBÉN

RODRÍGUEZ SÁNCHEZ ABIMAEL

DIRECTOR DE TESIS:

DR. MAURO ENCISO AGUILAR

CO-DIRECTOR DE TESIS:

M. en C. ALBERTO MANUEL BENAVIDES CRUZ

MÉXICO, D.F. NOVIEMBRE DE 2008

INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE INGENIERÍA MECÁNICA Y ELECTRICA

UNIDAD PROFESIONAL "ADOLFO LÓPEZ MATEOS"

T E M A D E T E S I S

QUE PARA OBTENER EL TITULO DE INGENIERO EN COMUNICACIONES Y ELECTRONICA

POR LA OPCIÓN DE TITULACIÓN CURRICULAR

DEBERA(N) DESARROLLAR ABIMAEL RODRÍGUEZ SÁNCHEZ
EDGAR RUBÉN GONZÁLEZ RAMÍREZ

"DISEÑO E IMPLEMENTACIÓN DE UN CLUSTER TIPO BEOWULF PARA EL DESARROLLO DE CÓMPUTO CIENTIFICO AVANZADO"

PLANEAR E IMPLEMENTAR UNA PLATAFORMA COMO HERRAMIENTA PARA EL DESARROLLO DE APLICACIONES NUMÉRICAS DE ALTO DESEMPEÑO.

- INTRODUCCIÓN A ARQUITECTURAS DE COMPUTADORAS.
- TOPOLOGÍA FÍSICA DEL CLUSTER.
- TOPOLOGÍA LÓGICA Y CONFIGURACIÓN DE NODOS.
- APLICACIÓN: IMPLEMENTACIÓN DEL CLUSTER
- CONCLUSIONES Y PERSPECTIVAS.

MÉXICO D. F., A 28 DE JULIO DE 2009.

A S E S O R E S

DR. MAURO A. ENCISO AGUILAR

M. EN C. ALBERTO M. BENAVIDES CRUZ

M. EN C. S. RICARDO MENESES GONZÁLEZ
JEFE DEL DEPTO. ACADÉMICO DE
INGENIERÍA EN COMUNICACIONES Y ELECTRÓNICA

AGRADECIMIENTOS

"La esencia de esta tesis es un total y profundo *agradecimiento* a **Jesucristo** por cada una de las bendiciones otorgadas hacia mi persona y a la gente que aprecio."

Asimismo quiero agradecer a mis padres, Arturo Rodríguez y Patricia Sánchez, quienes siempre me han apoyado en todo lo que hago ya que de alguna forma pusieron una educación de muy alto nivel a mi alcance mediante su apoyo incondicional en todos los aspectos, ya que sin su consejo hubiera sido casi imposible realizar esto.

A mi hermana Dina Luz y Arielito, siendo éste el bebe más hermoso, que mediante disciplina y esfuerzo se pueden lograr muchas cosas.

A mis asesores, el Dr. Mauro Enciso Aguilar y al M. en C. Alberto Manuel Benavides Cruz por la confianza brindada, al permitirme trabajar a lado de que ya que provocaron en mi el adoptar a la investigación como un estilo de vida.

A aquellos que tuvieron, al igual que yo, un sueño llamado Ingeniería. Principalmente A cada uno de mis compañeros de la superior, Karla María Castilla, Erica Castro, Laura Méndez, Alejandra Salas, Jazmín Sosa, Jesús Tlapaltotoli y Rubén González, Roberto Koka, Iván Oaxaca y a Ángel Hernández.

A mis mejores amigos Jaime Álvarez, Alejandro Gallardo, pero principalmente a Ismael Rangel por los momentos de investigación y de distracción que vivimos, ya que sin su apoyo hubiera sido imposible haber concluido este sueño.

A cada uno de los profesores de la ESIME Zacatenco que ayudaron en mi formación académica al transmitirme mi conocimiento

A todos aquellos los que me apoyaron y a los que no también ya que al ponerme piedras en el camino me permitieron forjar un carácter y sobre todo, ya al mismo tiempo el poder descubrir de lo que soy capaz de hacer.

A todos gracias.

Abimael Rodríguez Sánchez
Diciembre 2008

TABLA DE CONTENIDO

RELACIÓN DE TABLAS, ECUACIONES Y FIGURAS

TABLAS

FIGURAS

PAG

RELACIÓN DE ECUACIONES

RESUMEN

En este trabajo de Tesis se diseñó e implementó una herramienta computacional llamada cluster tipo Beowulf para realizar cómputo paralelo. Partiendo del uso de memoria compartida y distribuida, se implementó una plataforma pensando en la esencia del paralelismo: la optimización de los recursos de cómputo

El diseño esta montado sobre la distribución de Linux a 64 bits Fedora Core 6 aprovechando todas las ventajas que ofrece el estándar Gigabyt Ethernet. La programación paralela se realizó en lenguaje C tanto para MPI como OPENMP.

Una vez que se definió la plataforma y todas las configuraciones que conlleva, se realizaron algunas pruebas para la comprobación de un verdadero funcionamiento. Para obtener resultados se usó cada uno de los núcleos de los procesadores, obteniendo resultados semejantes a lo calculado.

De acuerdo a los resultados obtenidos, el tiempo de ejecución se decrementa con el manejo de los multicore el cual para este caso particular, contiene un procesador con 4 núcleos lo que es aproximadamente 8 veces más rápida que con un solo procesador haciendo uso de multihilos. Con la implementación del cluster se espera tomar control de algunos proyectos de electromagnetismo.

ABSTRACT

This thesis work has developed and implemented a computational tool, called "Cluster Beowulf ", in order to improve programming parallel. Beginning with shared and distributed memory, the Cluster Beowulf was implemented just thinking on paralelism esence: optimization of computing sources.

Design is built by means of a Linux distribution Fedora Core 6 to 64 bits, using all the advantages that offer Gigabyt Ethernet Standard. The parallel programming was developed in C lenguage both to MPI and OPENMP too.

Once that the base was implemented and all settings were made, it´s necesary make some shoots in order to check the correct use. To get the best results, all the processors shall be used.

According with the results, procesing time is dicreasing with the use of multicore which is improving up to eight times through of multithreading . Cluster´s implementation hope to take over some electromagnetic projects.

INTRODUCCIÓN

Hoy en día el avance de la tecnología y las necesidades del hombre han hecho que la investigación científica, en cualquiera de sus ramas, tenga una importancia extremadamente grande, exigiendo la sociedad científica a las universidades, la implantación de materias de actualidad y la promoción de la investigación para la mejora de algo ya existente, así como el desarrollo de cosas nuevas que permitan reducir las fronteras del conocimiento humano.

En esta tesis se propone la implementación de un cluster, el cual consiste en agrupar varios núcleos de un procesador para ejecutar programas en forma distribuida en cada uno de los núcleos, para el desarrollo de aplicaciones paralelas. Esto traerá grandes beneficios ya que permitirá poder obtener resultados que no se lograrían con un solo procesador.

Los clusters han evolucionado para apoyar actividades en aplicaciones que van desde súper cómputo y software de misiones críticas, servidores web y comercio electrónico así como bases de datos de alto rendimiento. En nuestro caso, el análisis electromagnético se verá plenamente favorecido con la construcción de un cluster debido a que continuamente se están realizando proyectos que necesitan procesamiento de información muy grande

A través de este trabajo, se intenta explicar cómo es que se puede optimizar ese tiempo de ejecución para una aplicación sencilla. Se explica que plataforma es la mejor opción y como se puede implementar una estación de trabajo para desarrollar cómputo paralelo.

El trabajo se encuentra dividido en cinco capítulos y un apéndice.

El Capítulo 1 presenta un estado del arte de la tecnología usada para de esta manera poder introducir el problema y ver de que manera se puede delimitar este mismo. Se introduce a las principales arquitecturas de computadoras, diferentes clasificaciones del procesador, memoria compartida, memoria distribuida y finalmente la clasificación de los clusters.

El capítulo 2 trata acerca de la composición física del cluster beowulf, tocando puntos como el sistema de discos, las tarjetas controladoras, así, como los niveles de seguridad almacenaje.

El capítulo 3 se presenta los sistemas operativos usados para realizar clusters, asimismo se muestra el uso de directivas OPENMP y la configuración de los compiladores que soportan estas directivas de OPENMP así como el paso de mensajes mediante MPI .

El capítulo 4 se presenta las características principales del cluster. Se muestra cada una de las configuraciones realizadas para la implementación de red. Además se muestra el código de los programas realizados en c así como el análisis de los resultados y finalmente se realiza una comparación de forma serial y de forma paralelo con el cálculo del número π.

El capítulo 5 contiene las conclusiones que se obtuvieron a través de toda la investigación así también como el análisis de los resultados obtenidos. Además se hace hincapié en posibles propuestas para trabajos futuros.

El apéndice A presenta una tabla la cual contiene los comandos más utilizados en linux.

JUSTIFICACIÓN

En el departamento de telecomunicaciones de la SEPI ESIME IPN Campus Zacatenco, se desarrollan proyectos de electromagnetismo computacional, utilizando como algoritmo el método de diferencias finitas en el dominio del tiempo.

Debido a que este método es muy demandante en recursos de cómputo, ha surgido la necesidad de explorar el desarrollo de aplicaciones dentro del área del súper cómputo. Se trata con esto de buscar una herramienta para poder dar solución a este método. Cabe señalar que además de este proyecto existen muchos más y por ende se pretende alcanzar resultados importantes.

El cómputo paralelo es una alternativa viable para seguir desarrollando proyectos de investigación debido a que se pueden realizar simulaciones en regiones con dimensiones eléctricas mucho más grandes que las que se han hecho hasta la fecha. Esta alternativa trae grandes beneficios siendo el más valioso entre ellos, el tiempo y la capacidad. Por lo tanto, la implementación de este cluster facilitará el desarrollo de aplicaciones en paralelo en el área de electromagnetismo computacional en esta sección de posgrado.

OBJETIVO GENERAL:

Planear e implementar una plataforma como herramienta para el desarrollo de aplicaciones numéricas de alto desempeño

OBJETIVO PARTICULAR:

Optimizar el uso de memoria compartida en procesadores Quad Core a través del uso de directivas de OPENMP para el desarrollo de aplicaciones paralelas.

HIPÓTESIS

Obtener una plataforma de desarrollo para el cómputo paralelo a partir de un conjunto de servidores Xeon Quad, utilizando software libre.

CAPÍTULO 1:

INTRODUCCIÓN A ARQUITECTURAS DE COMPUTADORAS

1.1 DEMANDA DE VELOCIDAD Y CAPACIDAD DE PROCESAMIENTO .

Hoy en día existe una continua demanda por velocidad de procesamiento. Ciertas áreas requieren de velocidad de procesamiento, dentro de las cuales podemos incluir ciertos modelados numéricos y simulaciones para problemas de carácter científico y de ingeniería. Estos problemas necesitan realizar grandes y repetitivos cálculos en grandes cantidades de datos para ofrecer resultados validos. Cabe señalar que dichos cálculos tienen que ser completados dentro de un razonable periodo de tiempo. En el campo industrial, cálculos de ingeniería y simulación tienen que ser archivados dentro de segundos o minutos si es posible. Una simulación que toma dos semanas para alcanzar una solución, es usualmente inaceptable en un diseño en el que el tiempo necesita ser corto para trabajar eficientemente. Cuando los sistemas llegan a ser muy complejos, se incrementa considerablemente el tiempo para simularlos. Hay algunos problemas que tienen un plazo de tiempo para los cálculos. Tal es el caso de los pronósticos del tiempo en los cuales se necesita tener información actualizada cada cierto tiempo.

Recientes aplicaciones como la realidad virtual, requieren considerable velocidad de procesamiento para obtener resultados en tiempo real. Una manera de incrementar la velocidad de procesamiento, forma que ha sido considerada por muchos años, es utilizando múltiples procesadores trabajando juntos en un mismo problema. El principal problema es dividir en partes la tarea a realizar, y dicha tarea estará desempeñada por un procesador independiente en paralelo. Escribir programas de esta forma es conocido como programación paralela. En el capítulo 3 se abarcará este tema con mucho mayor énfasis.

La plataforma ideal para realizar cómputo paralelo, es un sistema de cómputo el cual contiene múltiples procesadores o muchas computadoras independientes interconectadas de alguna forma.

En 1959 Holland escribió acerca de una computadora capaz de ejecutar un arbitrario número de subprogramas simultáneamente (Holland, 1959). En 1963 Conway describió el diseño de una computadora paralela y su programación (Conway, 1963).

Trabajos con similares títulos continúan siendo escritos 30 años después. El futuro es paralelo.

Programación en paralelo requiere una apropiada plataforma la que nosotros hemos mencionado como una simple computadora con múltiples procesadores internos o múltiples computadoras interconectadas.

1.2 EVOLUCIÓN DE ARQUITECTURAS PARALELAS

Al estudiar el procesamiento a nivel del procesador, se observa que la segmentación consiste en descomponer la ejecución de cada instrucción en varias etapas para poder empezar a procesar una instrucción diferente en cada una de ellas y trabajar con varias instrucciones a la vez, ya que varias instrucciones consecutivas son ejecutadas de forma solapada casi en paralelo.

Sin embargo, todos estos sistemas están basados en la arquitectura Von Neuman, consistente en un procesador y una memoria donde se guardan datos y programas, es decir, una máquina secuencial que procesa datos escalares. Esta arquitectura se ha ido perfeccionando incluyendo el paralelismo de las unidades de control, de cálculo, etc., pero sigue siendo una máquina de ejecución con un único flujo de instrucciones. No hay una frontera definida entre la arquitectura monoprocesador y las masivamente paralelas.

De hecho, las arquitecturas actuales son realmente máquinas paralelas a nivel de instrucción. La evolución de la arquitectura basada en el procesador ha venido ligada con la creación de más y mejores supercomputadoras que tenían que librarse del concepto de monoprocesador para poder hacer frente a las demandas de computación.

El primer paso hacia la paralelización de las arquitecturas de los computadores, se da con la aparición de los procesadores o sistemas vectoriales. El hecho que los procesadores segmentados hayan venido asociados a los supercomputadores paralelos, los pone en la entrada a lo que son los sistemas paralelos, si bien son una extensión del concepto de segmentación.

1.2.1 TIPOS DE ARQUITECTURAS PARALELAS

Con el paso de los años se han propuesto diversos esquemas de clasificación de los ordenadores pero ninguno de ellos ha tenido un éxito completo. El más aceptado ha sido, sin duda el de Flynn[1] aunque es algo rudimentaria y lo lógico sería que hoy con los avances que se han realizado, existiera otro más actual.

La taxonomía de Flynn está basada en la clasificación desde un punto de vista del flujo de datos e instrucciones en un sistema. Un flujo de instrucciones es el conjunto de instrucciones secuénciales que son ejecutadas por un único procesador, y un flujo de

[1] http://telematica.cicese.mx/computo/super/cicese2000/paralelo/part3.html

datos es el flujo secuencial de datos requeridos por el flujo de instrucciones. Con estas consideraciones, Flynn clasifica los sistemas en cuatro categorías:

1.2.2 FLUJO ÚNICO DE INSTRUCCIONES Y FLUJO ÚNICO DE DATOS

De sus siglas en ingles SISD (single instruction stream, single data stream) Este es el concepto de arquitectura serie de Von Neumann[2] donde, en cualquier momento, sólo se está ejecutando una única instrucción. A menudo a los SISD se les conoce como computadoras series. Todas las máquinas SISD poseen un registro simple que se llama contador de programa que asegura la ejecución en serie del programa.

Conforme se van leyendo las instrucciones de la memoria, el contador de programa se actualiza para que apunte a la siguiente instrucción a procesar en serie.

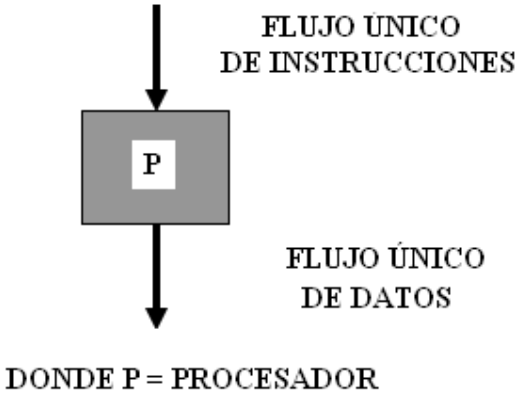


Figura 1.1 Funcionamiento de la arquitectura SISD

1.2.3 FLUJO MÚLTIPLE DE INSTRUCCIONES Y ÚNICO FLUJO DE DATOS

De sus siglas en inglés MISD (multiple instruction stream, single data stream).Esto significa que varias instrucciones actúan sobre el mismo y único sección de datos. La mejor forma de interpretar a los MISD es a partir de la Figura 1.2, en la cual un mismo flujo de datos fluye a través de numerosas unidades procesadoras. Arquitecturas altamente segmentadas, como los arrays sistólicos o los procesadores vectoriales, son clasificadas a menudo bajo este tipo de máquinas. Las arquitecturas segmentadas realizan el procesamiento vectorial a través de una serie de etapas, cada uno ejecutando una función particular produciendo un resultado intermedio. La razón por la cual dichas arquitecturas son clasificadas como MISD es que los elementos de un vector pueden ser considerados como pertenecientes al mismo dato, y todas las etapas del cauce representan múltiples instrucciones que son aplicadas sobre ese vector.

[2] http://ei.cs.vt.edu/~history/VonNeumann.html

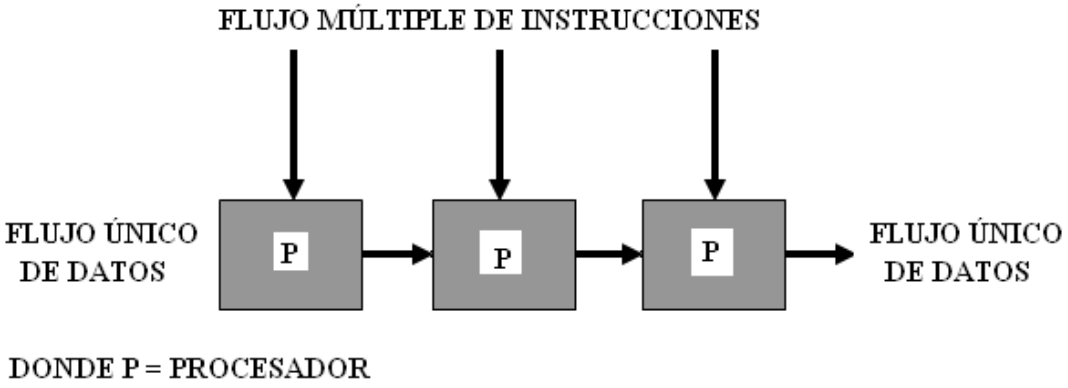


Figura 1.2 Funcionamiento de la arquitectura MISD

1.2.4 FLUJO ÚNICO DE INSTRUCCIÓN Y FLUJO DE DATOS MÚLTIPLE

De sus siglas en inglés SIMD (single instruction stream, multiple data stream).Esto significa que una única instrucción es aplicada sobre diferentes datos al mismo tiempo. En las máquinas de este tipo, varias unidades diferentes de procesamiento son invocadas por una única unidad de control. Al igual que las MISD, las SIMD soportan procesamiento vectorial (matricial) asignando cada elemento del vector a una única unidad funcional diferente para procesamiento concurrente. Por ejemplo, el cálculo de la paga para cada trabajador en una empresa, es repetir la misma operación sencilla para cada trabajador; si se dispone de una arquitectura SIMD esto se puede calcular en paralelo para cada trabajador.

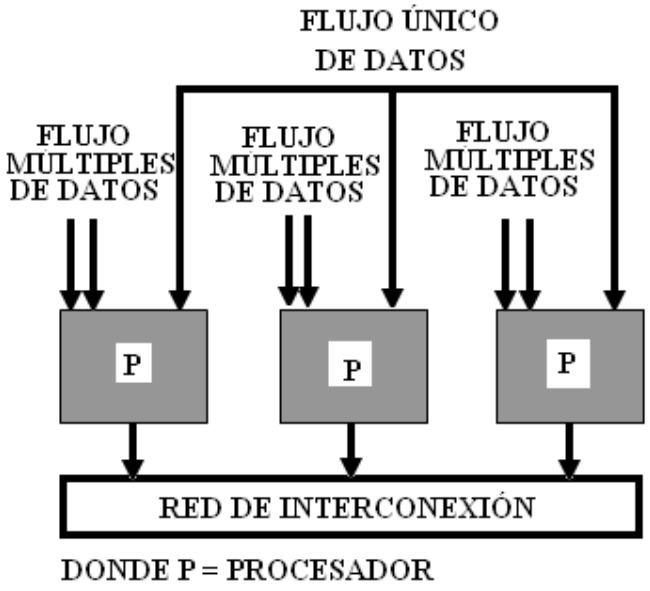


Figura 1.3 Funcionamiento de la arquitectura SIMD

1.2.5 FLUJO DE INSTRUCCIONES MÚLTIPLE Y FLUJO DE DATOS

De sus siglas en inglés MIMD (multiple instruction stream, multiple data stream). Son máquinas que poseen varias unidades de procesamiento en las cuales se pueden realizar múltiples instrucciones sobre diferente flujo de datos de forma simultánea. Las MIMD son las arquitecturas más complejas, pero son también las que potencialmente ofrecen una mayor eficiencia en la ejecución concurrente o paralela. Aquí la concurrencia implica que no solo hay varios procesadores operando simultáneamente, sino que además hay varios programas (procesos) ejecutándose también al mismo tiempo.

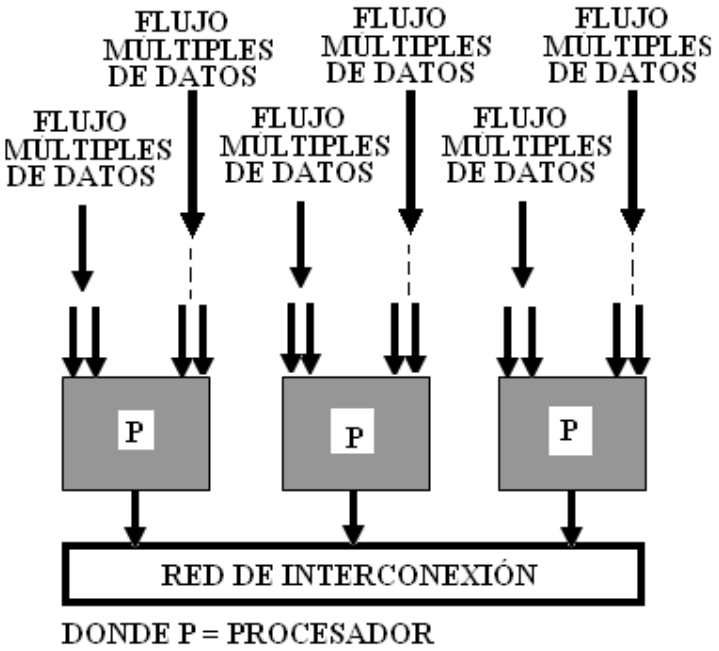


Figura 1.4 Funcionamiento de la arquitectura MIMD

La clasificación de Flynn ha demostrado funcionar bastante bien para la tipificación de sistemas, y se ha venido usando desde décadas por la mayoría de los arquitectos de computadoras. Sin embargo, los avances en tecnología y diferentes topologías, han llevado a sistemas que no son tan fáciles dentro de los 4 tipos de Flynn.

Por ejemplo, los procesadores vectoriales no encajan adecuadamente en esta clasificación, ni tampoco las arquitecturas híbridas. Para solucionar esto se han propuesto otras clasificaciones, donde los tipos SIMD y MIMD de Flynn se suelen conservar, pero que sin duda no han tenido el éxito de la Flynn.

1.3 FLUJO DE DATOS EN SISTEMAS SERIALES Y PARALELOS.

Las características de los sistemas seriales es la de ejecutar un solo trabajo a la vez, los primeros procesadores ocupaban esta tecnología, a los que se conocían como

monousuario o monotarea. Al avanzar la tecnología los procesadores monousuario se fueron haciendo obsoletos y se buscó la forma en que el procesador realizará más de una operación al mismo tiempo o durante el mismo ciclo de reloj, esto fue el principio del paralelismo.

Actualmente, los procesadores utilizan la segmentación de proceso para realizar varias operaciones en un mismo ciclo de reloj, sin embargo, esto no quiere decir que sean paralelos, el usuario percibe que varias aplicaciones se ejecutan al mismo tiempo, esta es una ilusión que provoca el mismo procesador debido a la velocidad de operación que tiene, sin embargo su trabajo es estar pasando del segmento de una aplicación a otro y así aparentar su ejecución simultánea.

Una forma de entender mejor lo anterior es la siguiente: al tener un equipo con un procesador de un 1Ghz y otro con dos procesadores de 500 Mhz, es lógico suponer que ambos tendrán el mismo rendimiento, sin embargo la realidad demuestra todo lo contrario, ya que el equipo con dos procesadores será mucho más rápido que el que tiene un solo procesador, esto se da en la forma de repartición de trabajos, por ejemplo si se tienen dos procesos el equipo con un procesador tendrá que estar pasando de un proceso a otro, esto le quitará rendimiento o provocará tiempos muertos; en el otro caso el equipo de procesador dual repartirá a cada procesador un proceso y esto impedirá tiempos muertos, generando con esto un mayor desempeño.

1.3.1 SISTEMAS SERIALES

Este tipo de sistema es el mayormente utilizado hoy en día por las computadoras de tipo casero y convencional, ya que los sistemas operativos en una instalación estándar realizan las operaciones por medio de un sistema de flujo denominado FIFO (primero en entrar primero en salir), todas las operaciones o instrucciones realizadas por los procesadores son llevadas a cabo en este orden.

A nivel de hardware son marcadas las características que delimitan su peculiaridad ya que todos sus recursos físicos como son procesadores, memorias, discos y sistemas e/s están diseñados para ser utilizados de manera propietaria y secuencial.

En los sistemas de múltiples procesadores aún los recursos como la memoria, los discos y el E/S son compartidos y cada procesador tiene que esperar su turno o recibir un asignamiento de recurso que repercute en el tiempo de respuesta a cada operación.

1.3.2 SISTEMAS PARALELOS

Por el contrario del sistema anterior, todos los recursos de hardware como es memoria, discos, sistema de entrada y salida (E/S) puede ser configurados de tal modo que los procesadores tienen asignado de manera exclusiva cada uno de estos recursos y el número de operaciones que atienden es llevado acabo con mayor velocidad. Además el

sistema operativo es atendido en sus peticiones de manera más ágil por la manera en que los recursos son asignados y ofrecen mas vías de comunicación.

En la tabla 1.1 se muestra un comparativo entre ambos sistemas en las características que más influyen en su desempeño.

COMPARATIVO DE SISTEMAS	
SERIAL	**PARALELO**
• Sist. Simple de 1 a 16 procesadores • Copia simple de S.O. • Asignación de recursos compartida entre uno o más procesadores en un solo sistema • No tolerante a fallas	• Múltiples procesadores mas de 16 • Varias copias de S.O. corriendo simultáneamente • Cada procesador o nodo del sistema tiene sus propios recursos • Tolerante a fallas • Varias aplicaciones corriendo de manera simultánea y de modo redundante.

Tabla 1.1 Comparativo de sistemas

El hardware de sistema de cómputo puede ser diseñado o modelado de acuerdo a la necesidad o necesidades de la aplicación o solución. Esto esta principalmente en función del consumo y tiempo de respuesta que se requiera como resultado del sistema de cómputo; así como tomar en cuenta la disponibilidad, escalabilidad y la rentabilidad que el sistema requiera para estar siempre en línea. Esto puede determinar si se requiere de un sistema serial o paralelo (cluster) de acuerdo a los esquemas que se detallan en la figura 1.5.

El flujo de datos en sistema serial es de tipo secuencial y está determinado por el diseño del hardware y el modo de petición de sistema operativo. Así como el flujo de datos de un sistema paralelo está determinado por el diseño del hardware y el modo en que hace las peticiones el sistema operativo, es decir, para cada modo de procesamiento existe un diseño específico de sistema operativo y diseño en la arquitectura del hardware.

1.4 PROCESADOR

El procesador es un microprocesador el cual es un circuito integrado que contiene todos los elementos necesarios para conformar una "unidad central de procesamiento" UCP, también es conocido como CPU (por sus siglas en inglés: Central Process Unit). En la actualidad este componente electrónico está compuesto por millones de transistores, integrados en una misma placa de silicio.

1.4.1 FUNCIONAMIENTO

Desde el punto de vista lógico y funcional, el microprocesador está compuesto básicamente por: varios registros; una unidad de control, una unidad aritmético-lógica; y dependiendo del procesador, puede contener una unidad de punto flotante.

El microprocesador ejecuta instrucciones almacenadas como números binarios organizados secuencialmente en la memoria principal. La ejecución de las instrucciones se puede realizar en varias fases:

- Pre lectura de la instrucción desde la memoria principal.
- Envío de la instrucción al decodificador.
- Decodificación de la instrucción, es decir, determinar qué instrucción es y por tanto qué se debe hacer.
- Lectura de operándoos (si los hay).
- Ejecución, (lanzamiento de las máquinas de estado que llevan a cabo el procesamiento).
- Escritura de los resultados en la memoria principal o en los registros.

Cada una de estas fases se realiza en uno o varios ciclos de CPU, dependiendo de la estructura del procesador, y concretamente de su grado de segmentación. La duración de estos ciclos viene determinada por la frecuencia de reloj, y nunca podrá ser inferior al tiempo requerido para realizar la tarea individual (realizada en un solo ciclo) de mayor coste temporal. El microprocesador se conecta a un oscilador, normalmente un cristal de cuarzo capaz de generar pulsos a un ritmo constante, de modo que genera varios ciclos (o pulsos) en un segundo. Este reloj, en la actualidad, genera miles de MHz.

1.4.2 CLASIFICACIÓN

1. Por su tamaño de palabra
2. Por su número de núcleos

1.4.2.1 TAMAÑO DE PALABRA

Hoy en día los procesadores que se manejan en el mercado van evolucionando en velocidad de procesamiento. Siendo mucho más populares aquellos procesadores de 32 bits, aunque los procesadores de 64 bits comienzan a tener un gran empuje y esto debido a su gran tamaño de procesamiento por palabra.

- **De 32 bits**

Un procesador de 32 bits es un microprocesador con un tamaño de palabra de 32 bits. La arquitectura de 32 bits ofrece un buen desempeño pero la tecnología esta provocando dar un brinco hacia los procesadores de 64 bits.

- **DE 64 BITS**

Un procesador de 64 bits es un microprocesador con un tamaño de palabra de 64 bits, un requisito para memoria y aplicaciones intensivas de datos como diseño asistido por computadora, administración de base de datos de sistemas, aplicaciones técnicas y científicas, así como servidores de alto desempeño. La arquitectura de una computadora de 64 bits ofrece más alto desempeño que una arquitectura de 32 bits por el procesamiento de dos veces el número de bits en el mismo ciclo de reloj.

El procesador de 64 bits es compatible con viejas aplicaciones y sistemas operativos; detectando si una aplicación o sistema operativo es de 16-bit, 32-bit, o de 64-bit. Esto es esencial para afrontar situaciones donde adquisición de nuevos programas no es viable.

Dentro de las marcas de procesadores de 64 bits que están actualmente en el mercado se encuentran Intel, IBM, Sun Microsystems, Hewlett Packard, y AMD.

1.4.2.2 NUMERO DE NÚCLEOS

Los procesadores pueden ser clasificados de acuerdo al número de procesadores independientes que se ubican en un mismo paquete .A lo anterior se le denomina multicore. Una ventaja de este tipo de procesadores es que permite un funcionamiento paralelo. Un procesador dual-core posee dos núcleos en un mismo circuito integrado. Mientras que un quad-core posee cuatro núcleos en un mismo circuito integrado. Hoy en día Intel es la compañía que esta tratando de pegar fuerte con esta tecnología.

1.3 MULTIPROCESADORES

Un multiprocesador se puede ver como un computador paralelo compuesto por varios procesadores interconectados que pueden compartir un mismo sistema de memoria. Los multiprocesadores se pueden configurar para que ejecute cada uno una parte de un programa o varios programas al mismo tiempo. Un multiprocesador está generalmente formado por “n” procesadores y “m” módulos de memoria. A los procesadores los llamamos $P_1; P_2; ...; P_n$ y a las memorias $M_1; M_2; ...; M_n$. La red de interconexión conecta a cada procesador a un subconjunto de los módulos de memoria.

1.5.1 ACCESO A MEMORIA UNIFORME

De sus siglas en inglés UMA (uniform memory access). En el modelo UMA, la memoria física esta uniformemente compartida por todos los procesadores. Esto quiere decir que todos los procesadores tienen el mismo tiempo de acceso a todas las palabras

de memoria. Cada procesador puede tener su caché privada, y los periféricos son también compartidos de alguna manera. A estos computadores se les suele llamar sistemas fuertemente acoplados dado el alto grado de compartición de los recursos.

Cuando todos los procesadores tienen el mismo acceso a todos los periféricos, el sistema se llama multiprocesador simétrico (SMP). En este caso, todos los procesadores tienen la misma capacidad para ejecutar programas, tal como el kernel o las rutinas de servicio de E/S. En un multiprocesador asimétrico, sólo un subconjunto de los procesadores puede ejecutar programas en un instante de tiempo. El común de los equipos está compuesto en su configuración básica por un solo procesador denominado como maestro y el resto de procesadores se les llama procesadores adheridos (attached processors) estos últimos se integran bajo la demanda que el equipo requiera ya en producción. Es frecuente encontrar arquitecturas de acceso uniforme que además tienen coherencia de caché, a estos sistemas se les suele llamar CC-UMA (caché- coherent uniform memory access).

1.3.1 ACCESO A MEMORIA NO UNIFORME

De sus siglas en inglés NUMA (non uniform memory access).Un multiprocesador de tipo NUMA es un sistema de memoria compartida donde el tiempo de acceso varía según el lugar donde se encuentre localizado el acceso. La figura 1.5 muestra una posible configuración de tipo NUMA, donde toda la memoria es compartida pero de manera local a cada módulo procesador.

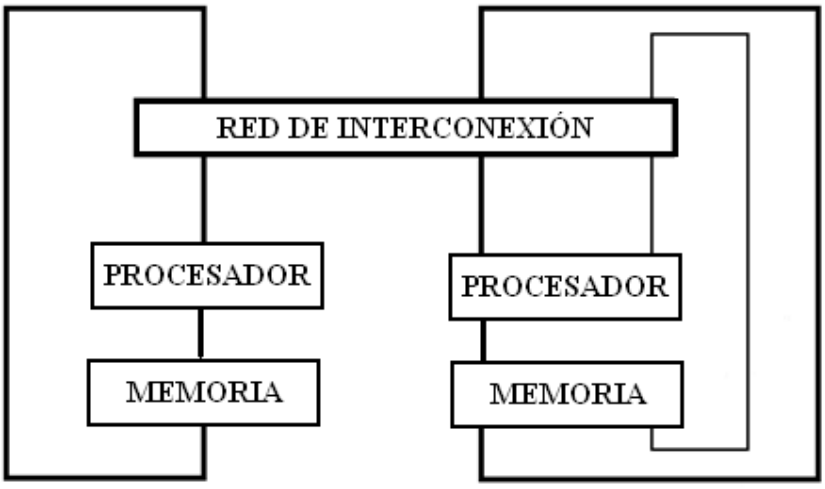


Figura 1.5 Arquitectura NUMA

Otros tipos de configuraciones están basados en agrupaciones de nodos como lo muestra la figura 1.6, que muestra como la comunicación es a través de la red y de éste modo compartirla de manera lógica o virtual. La ventaja de estos sistemas es que el acceso a la memoria local es más rápido que en los UMA aunque un acceso a memoria no local es más lento. Lo que se intenta es que la memoria utilizada por los procesos

que ejecuta cada procesador, se encuentre en la memoria de dicho procesador para que los accesos sean lo más locales posible. Aparte de esto, se puede añadir al sistema una memoria de acceso global.

En este caso se dan tres posibles patrones de acceso. El más rápido es el acceso a memoria local. Le sigue el acceso a memoria global. El más lento es el acceso a la memoria del resto de módulos. Al igual que hay sistemas de tipo CC-UMA, también existe el modelo de acceso a memoria no uniforme con coherencia de caché CC-NUMA (caché –coherent non-uniform memory access) que consiste en memoria compartida distribuida y directorios de caché.

1.5.3 ACCESO A MEMORIA CACHÉ

De sus siglas en inglés COMA (caché only memory access). Un multiprocesador que solo use caché como memoria es considerado de tipo COMA. La figura 1.6 muestra el modelo COMA de multiprocesador.

En realidad, el modelo COMA es un caso especial del NUMA donde las memorias distribuidas se convierten en cachés. No hay jerarquía de memoria en cada módulo procesador. Todas las caché forman un mismo espacio global de direcciones.

El acceso a las caches remotas se realiza a través de los directorios distribuidos de las caches. Dependiendo de la red de interconexión empleada, se pueden utilizar jerarquías en los directorios para ayudar en la localización de copias de bloques de caché. El desplazamiento inicial de datos no es crítico puesto que el dato acabará estando en el lugar que se use más.

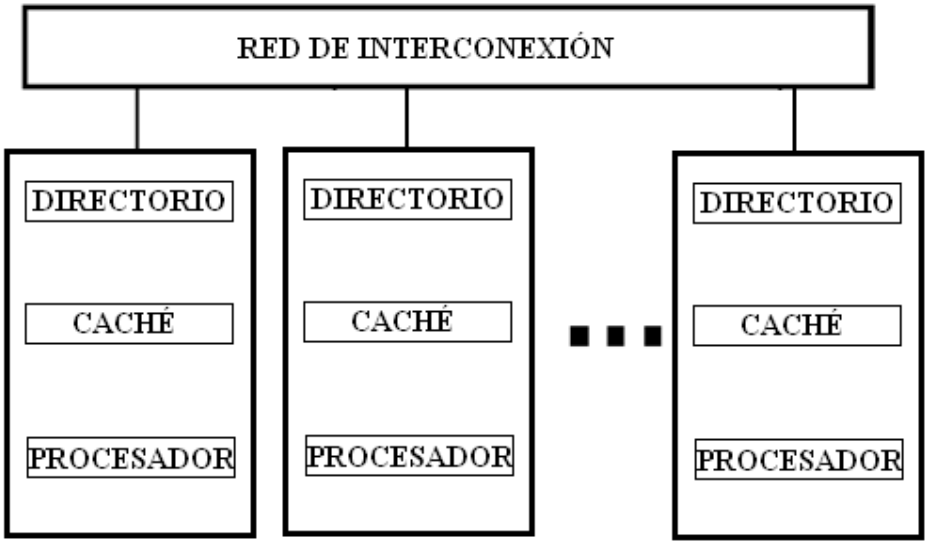


Figura 1.6 Arquitectura COMA

1.6 MEMORIA DISTRIBUIDA

La implementación del multiprocesador de memoria distribuida se da a través de lo que conocemos La arquitectura básica de un sistema multiprocesador de paso de mensaje se muestra en la figura 1.7. Un multiprocesador de paso de mensajes consta de nodos que normalmente están conectados mediante enlaces directos a otros nodos.

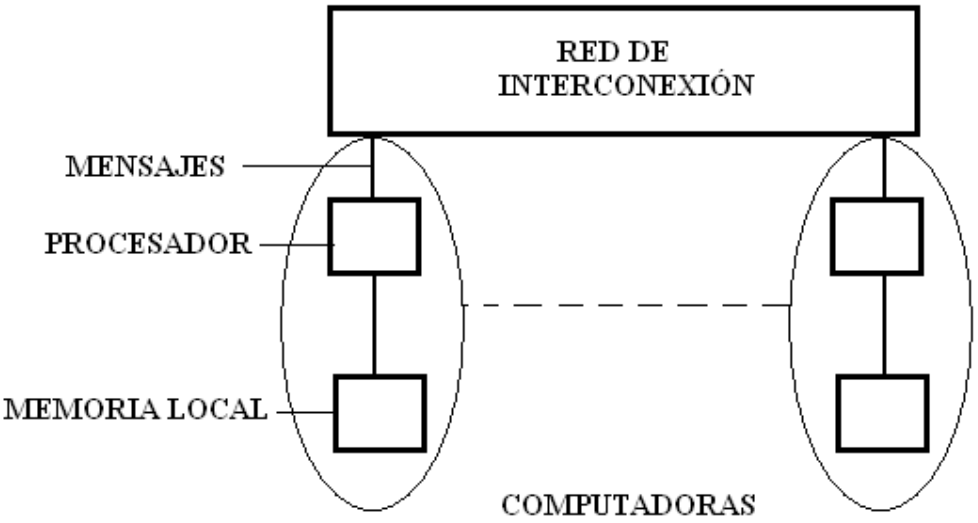


Figura 1.7 Modelo de multiprocesador de paso de mensajes

Cada nodo está compuesto por un procesador junto con una memoria local y canales de comunicación de E/S. No existen localizaciones de memoria global. La memoria local puede usar las mismas direcciones. Dado que cada nodo es un ordenador auto contenido, a los multiprocesadores de paso de mensajes se les suelen denominar multicomputadoras.

El número de nodos puede ser tan pequeño como 16 (o menos), o tan grande como varios millares. Sin embargo, la arquitectura de paso de mensajes muestra sus ventajas sobre los sistemas de memoria compartida cuando el número de procesadores es grande. Para sistemas multiprocesadores pequeños, los sistemas de memoria compartida presentarán probablemente un mejor rendimiento y mayor flexibilidad. El número de canales físicos entre nodos suele oscilar entre cuatro y ocho. La principal ventaja de esta arquitectura es que es directamente escalable y presenta un bajo costo para sistemas grandes.

La transferencia de datos se realiza a través de la red de interconexión que conecta un subconjunto de procesadores con otro subconjunto. La transferencia de operaciones entre procesadores y otros se realiza por múltiples transacciones entre ellos y son transmitidas a través del diseño que se estableció de la red.

Dado que la memoria está distribuida entre los diferentes elementos de proceso, a estos sistemas se les llama distribuidos aunque no hay que olvidar que puede haber

sistemas que tengan la memoria distribuida pero que al mismo tiempo tenga la memoria de forma compartida y por lo tanto no sean multicomputadora. Además, y dado que se explota mucho la localidad, a estos sistemas se les llama débilmente acoplados, ya que los módulos funcionan casi independiente unos de otros.

1.7 MEMORIA COMPARTIDA

El acceso por memoria compartida nos dice que cualquier dirección de memoria es accesible desde cualquier procesador/núcleo: dirección única por posición de memoria.

En la figura 1.8 se ilustra a la arquitectura de memoria compartida.

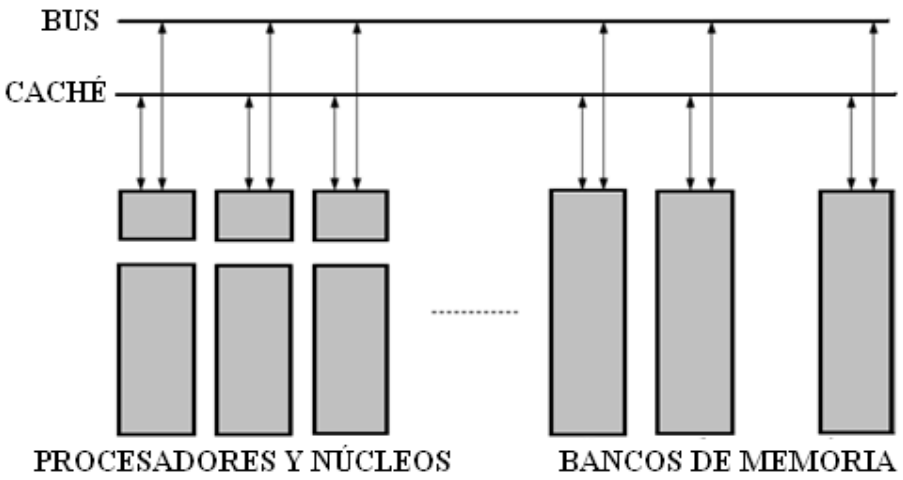


Figura 1.8: Arquitectura de memoria compartida

Los multiprocesadores de memoria compartida presentan algunas desventajas como por ejemplo:

1) Son necesarias técnicas de sincronización para controlar el acceso a las variables compartidas.

2) La contención en la memoria puede reducir significativamente la velocidad del sistema.

3) No son fácilmente ampliables para acomodar un gran número de procesadores.

Un sistema multiprocesador alternativo al sistema de memoria compartida que elimina los problemas arriba indicados es tener únicamente una memoria local por procesador eliminando toda la memoria compartida del sistema. El código para cada procesador se carga en la memoria local al igual que cualquier dato que sea necesario.

Los programas todavía están divididos en diferentes partes, como en el sistema de memoria compartida, y dichas partes todavía se ejecutan concurrentemente por

procesadores individuales. Cuando los procesadores necesitan acceder a la información de otro procesador, o enviar información a otro procesador, se comunican a otro procesador, se comunican enviando mensajes. Los datos no están almacenados globalmente en el sistema; si más de un proceso necesita un dato, éste debe duplicarse y ser enviado a todos los procesadores peticionarios. A estos sistemas se les suele denominar multicomputadoras.

1.8 PROCESO (POTENCIAL PARA EL INCREMENTO DE LA VELOCIDAD DE CÓMPUTO)

No importa que forma de MIMD multiprocesador/multicomputadora usemos, para lograr una mejora en velocidad a través de el uso del paralelismo. Esto es necesario para dividir el cómputo en tareas o procesos que pueden ser ejecutados simultáneamente. El tamaño de un proceso puede ser descrito por su granularidad. La granularidad es definida como la tasa de computación a sincronización. En una ordinaria granularidad, cada proceso contiene un largo número de instrucciones en forma secuencial y toma un substancial tiempo para su ejecución. En fina granularidad, un proceso podría consistir de pocas instrucciones, o quizás aún de una instrucción. Algunas veces la granularidad es definida como el tamaño del proceso entre comunicación o puntos de sincronización.

Generalmente queremos incrementar la granularidad para reducir los costos de creación de comunicación de procesos e ínter procesos, por supuesto que esto reducirá el número de procesos concurrentes y la cantidad de paralelismo.

Para el paso de mensajes, es deseable reducir el encabezado en la comunicación debido al tiempo significante tomado en una comunicación computacional. Como dividimos el problema en diferentes partes, en algún punto del tiempo de comunicación dominara todo el tiempo de ejecución. La velocidad puede ser usada como una granularidad métrica. Esto es muy importante para maximizar la velocidad de cómputo/comunicación mientras se guarda el paralelismo.

La granularidad es relacionada con el número de procesadores que están siendo usadas. Por ejemplo, en la descomposición del dominio, el tamaño de un bloque de datos usados por un procesador que podría ser incrementado al aumentar la granularidad y la velocidad cómputo/comunicación. Con un tamaño fijo del problema, el actual número de procesadores decrementaría. En general nos gustaría diseñar un programa en paralelo en el cual sea fácil variar la granularidad.

1.8.2 HILOS

Los hilos pueden ser definidos como una instancia de subtareas subdivididas para ejecutarse en paralelo, muchas aplicaciones, se dividen en múltiples hilos.

El proceso creado con UNIX bifurcación (fork) es un proceso pesado. El programa está completamente separado con sus propias variables, pila y localidad de

memoria. Un proceso pesado, son particularmente caros para crear en tiempo y en espacio de memoria. Una copia completa del proceso con su propia localidad de memoria, variables, pilas, etc., es crear una ejecución que comience desde la posición de bifurcación. Continuamente una copia del proceso no es requerido. Un mecanismo mucho más eficiente es uno en el que en la rutina concurrente se especifica en compartir el mismo espacio de memoria y las variables globales. Esto puede ser proporcionado por un mecanismo de hilo o un proceso ligero.

En la figura 1.9 se ilustra un proceso.

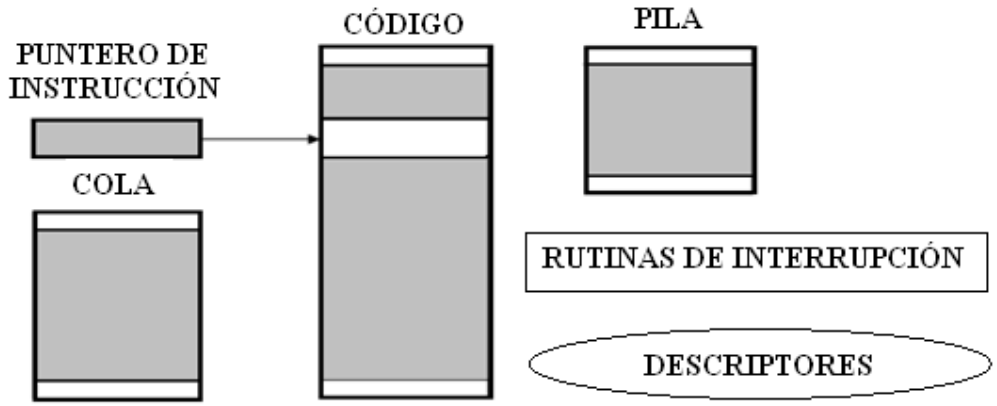


Figura 1.9 Proceso

Un puntero de instrucción (IP) amarra la dirección en la siguiente instrucción del hilo para ser ejecutada. Una pila esta presente por el procedimiento de llamadas así como también un sistema de rutinas y archivos. Como se muestra en la figura 1.10 cada hilo tiene su propio IP apuntando a la siguiente instrucción del hilo que será ejecutada. Cada hilo necesita su propia pila y también una reserva de información en lo que refiere a registros pero comparte el código. Un proceso puede tener más de un hilo.

En la figura 1.10 se ilustra un hilo.

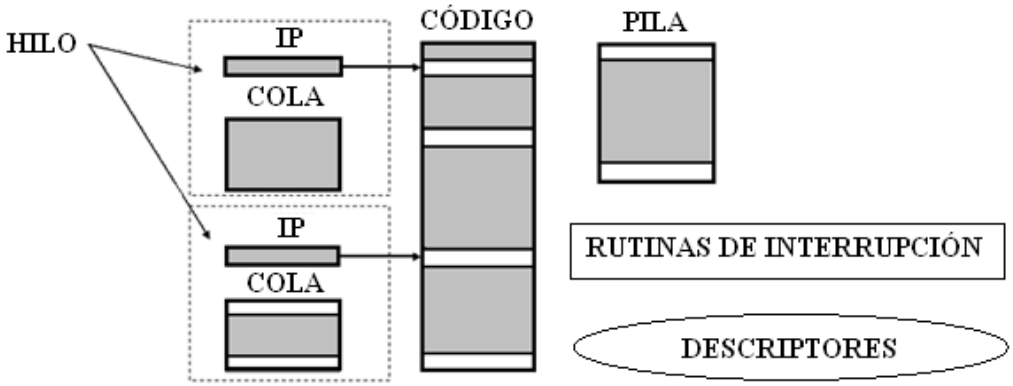


Figura 1.10 hilo

La creación de un hilo puede tomar tres ordenes de magnitud menos tiempo que la creación del proceso. Además, un hilo inmediatamente tendrá acceso a las variables globales compartidas.

1.8.3 FACTOR DE VELOCIDAD

Una medida de desempeño entre un sistema de multiprocesadores y un solo procesador es el factor de velocidad, $S(n)$, definido como:

$$S(n) = \frac{t_s}{t_p} \qquad \text{Ec. (1.1)}$$

Donde:

t_s Es el tiempo de ejecución usando un solo procesador

t_p Es el tiempo utilizando un multiprocesador con n procesadores.

$S(n)$ Da el incremento de velocidad utilizando un multiprocesador.

El factor de velocidad o también conocido como aceleración simple, permite obtener la relación de cuanto se mejora el tiempo de ejecución.

1.8.4 ENCABEZADO

Habrá muchos factores que aparecerán como encabezado en una versión paralela y limitarán la velocidad notablemente.

1) Periodos cuando no todos los procesadores pueden estar desempeñando un buen trabajo y se desperdician. (Esto incluye el tiempo cuando solo un procesador está activo sobre partes seriales de cómputo).

2) Tiempo de comunicación para el envío de mensajes.

1.8.5 LEY DE AMDAHL

Suponiendo que habrá algunas partes que son ejecutados en un procesador, la situación ideal sería para todos los procesadores disponibles para operar simultáneamente para otros tiempos. Si la fracción del cómputo que no puede ser dividido en tareas concurrentes como es f, y no incurren encabezados cuando el cómputo es dividido dentro de partes concurrentes, el tiempo para representar el cómputo con "n" procesadores se muestra en la siguiente ecuación (1.2).

$$T_p = St_s + \frac{(1-S)t_s}{p} \qquad \text{Ec. (1.2)}$$

Donde:

T_P = Tiempo en paralelo

S = es la fracción de trabajo serial intrínsico

t_s = es el tiempo de ejecución usando un solo procesador

p= número de núcleos

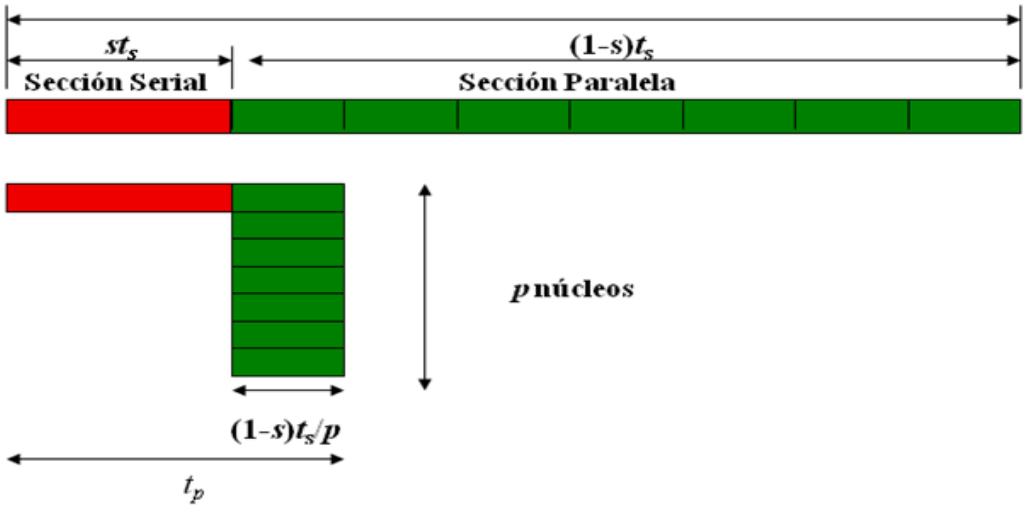


Figura 1.11 Distribución del Tiempo en paralelo

- **EFICIENCIA**

La eficiencia, E, está definida como:

$$E = \frac{t_s}{t_p \times n} \qquad \text{Ec. (1.3)}$$

Donde:

t_s : Tiempo de ejecución usando un procesador

t_p : Tiempo de ejecución usando un multiprocesador.

n: número de procesadores

- **COSTO**

El producto procesador- tiempo ó costo puede ser definido como:

Costo = (tiempo de ejecución) × (número total de procesadores utilizados)

El costo de un cálculo serial sería simplemente el tiempo de ejecución, t_s. El costo de un cálculo paralelo es $t_p \times n$. El tiempo de ejecución en paralelo, t_p, es dado por $\frac{t_s}{S(n)}$. De ahí que el costo de un cálculo en paralelo es:

$$Costo = \frac{t_s n}{S(n)} = \frac{t_s}{E} \qquad \text{Ec. (1.4)}$$

Un óptimo costo en un algoritmo paralelo es uno en el que el costo para resolver un problema en un multiprocesador es proporcional al costo (por ejemplo, tiempo de ejecución) en un solo procesador. Ocasionalmente usaremos esta medida en la comparación de algoritmos.

- **ESCALABILIDAD**

Escalabilidad ha sido un término bastante impreciso. Este es usado para indicar un diseño a nivel hardware que permite al sistema ser incrementado en tamaño y en ejecución para obtener un incremento en el desempeño. Esto podría ser descrito como arquitectura o escalabilidad de hardware. Escalabilidad también es usado para indicar que un algoritmo paralelo puede contener incrementos de número de datos con un bajo y limitado numero de pasos computacionales. Esto podría ser descrito como un algoritmo de escalabilidad.

Por supuesto quisiéramos que todos los sistemas de multiprocesadores sean arquitecturalmente escalables, pero esto dependerá fuertemente sobre el diseño del sistema. Normalmente si agregamos procesadores a un sistema, la interconexión de red tendría que ser expandida.

Quizás una meta oculta de muchos de los diseños de los multiprocesadores es lograr escalabilidad y esto es reflejado en la multitud de interconexión de red que ha sido concebido.

Combinar arquitecturas con algoritmos de escalabilidad sugiere que incrementar el tamaño del problema puede ser acomodado con el incremento del tamaño del sistema para una arquitectura o algoritmo en particular. Considerando que el incremento del tamaño del sistema claramente significa agregar procesadores, incrementar el tamaño del problema requiere su explicación. Intuitivamente pensaríamos que el número de elementos de datos que están siendo procesados en el algoritmo son una medida del

tamaño. Sin embargo, duplicando el tamaño del problema no necesariamente duplicaría el número de pasos. Esto dependería del problema. Una definición alternativa del tamaño del problema, es identificar el tamaño del problema con el número básico de pasos en el mejor algoritmo de forma serial. Por supuesto, aún con esta definición, si incrementamos el número de puntos de datos incrementaremos el tamaño del problema.

1.8.6 TASA DE TRABAJO

La tasa de trabajo se refiere principalmente a cuántos trabajos se pueden ejecutar en un tiempo determinado. Una analogía para ilustrar la tasa de trabajo es haciendo referencia a que nueve señoras pueden tener 9 niños en 9 meses**:**

Tasa de trabajos = 9 niños / 9 meses

Tasa de trabajos = 1 niño por mes

1.9 DEFINICIÓN DE UN CLUSTER DE COMPUTADORAS.

Un cluster se puede definir como el conjunto de unidades de procesamiento interconectados entre si, por diversos tipos de interfases de comunicación, ya sea serial, Ethernet, Fibra y SCSI para llevar a cabo tareas de procesamiento y almacenamiento en conjunto para solucionar una o varias tareas específicas.

El cluster es un tipo más de arquitectura paralela distribuida o compartida, pero con una característica especial: cada computador puede utilizarse de forma independiente.

El caso que nos atañe para este proyecto es un cluster tipo Beowulf y sus características que han desarrollado se detallan a continuación:

El primer Beowulf fue construido con los procesadores DX4 y una red Ethernet a 10 Mbit/s, el sistema fue Linux

Cluster Beowulf no es un paquete de software especial, ni una topología de red, ni un núcleo modificado. Beowulf es una tecnología para agrupar computadores basados en el sistema operativo Linux para formar un supercomputador virtual paralelo.

Beowulf posee una arquitectura basada en multicomputadoras el cual puede ser utilizado para la computación paralela. Este sistema consiste en un nodo maestro y uno o más nodos esclavos conectados a través de una red Ethernet u otra topología de red.

Está construido con componentes de hardware comunes en el mercado, similar a cualquier PC capaz de ejecutar Linux, adaptadores de Ethernet y switches estándares.

Como no contiene elementos especiales, es totalmente reproducible. Una de las diferencias principales entre Beowulf y un cluster de estaciones de trabajo (COW, cluster of workstation) es el hecho de que Beowulf se comporta más como una sola

máquina que como muchas ocasiones de trabajo conectadas. En la mayoría de los casos los nodos esclavos no tienen monitores o teclados y son accedidos solamente vía remota o por terminal serie. El nodo maestro controla el cluster entero y presta servicios de sistemas de archivos a los nodos esclavos. Es también la consola del cluster y la conexión hacia el exterior. Las máquinas grandes de Beowulf pueden tener más de un nodo maestro, y otros nodos dedicados a diversas tareas específicas, como por ejemplo, consolas o estaciones de supervisión. En la mayoría de los casos los nodos esclavos de un sistema Beowulf son estaciones simples. Los nodos son configurados y controlados por el nodo maestro, y hace solamente lo que esté les indique. En una configuración de esclavos sin disco duro, estos incluso no saben su dirección IP hasta que el maestro les dice cual es. En la figura 1.12 se muestra una arquitectura genérica para un cluster tipo Beowulf.

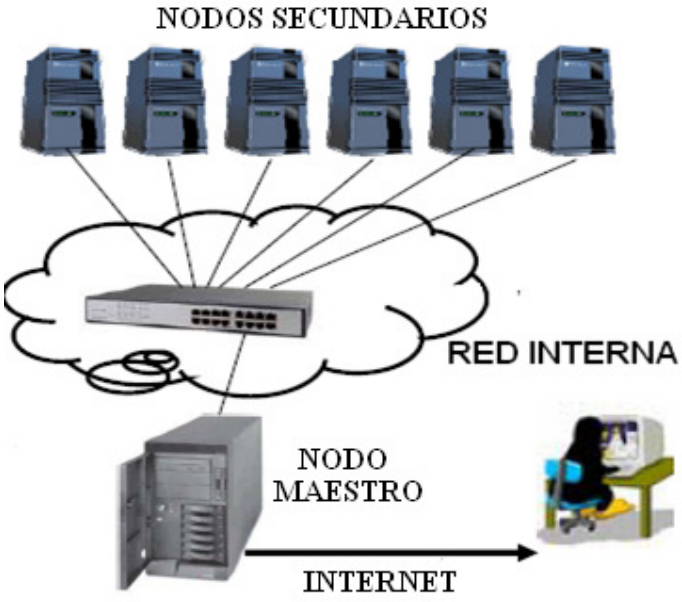


Figura 1.12 Arquitectura genérica de un cluster Beowulf

La topología de red recomendada es un bus, debido a la facilidad para proporcionar escalabilidad a la hora de agregar nuevos nodos al cluster. Protocolos como Ethernet, Fast Ethernet, GigaEthernet, 10/100Mbps Switched Ethernet, etc. son tecnologías apropiadas para ser utilizadas en Beowulf.

Un Beowulf es una clase de computador masivamente paralelo de altas prestaciones principalmente construido a base de un cluster de componentes hardware estándar. Un Beowulf ejecuta un sistema operativo de libre distribución como Linux o FreeBSD, y se interconecta mediante una red privada de gran velocidad

El software puede ejecutarse más rápido en un Beowulf si se dedica algún tiempo a reestructurar los programas. En general es necesario partirlos en tareas

paralelas que se comunican usando alguna librería como MPI o PVM, o combinación MPI -OPENMP

Los clusters permiten aumentar la escalabilidad, disponibilidad y fiabilidad de múltiples niveles de red.

La escalabilidad es la capacidad de un equipo para hacer frente a volúmenes de trabajo cada vez mayores, por ello, deja de prestar un nivel de rendimiento aceptable. Existen dos tipos de escalabilidad:

- Escalabilidad del hardware (también denominada "escalamiento vertical"). Se basa en la utilización de un gran equipo cuya capacidad se aumenta a medida que lo exige la carga de trabajo existente.

- Escalabilidad del software (también denominada "escalamiento horizontal"). Se basa, en la utilización de un cluster compuesto de varios equipos de mediana potencia que funcionan en conjunto de forma muy parecida a como lo hacen las unidades de disco que se implementa en arreglo por medio de un RAID (arreglo redundante de discos de bajo costo). Del mismo modo que se añaden discos RAID para aumentar su rendimiento, se pueden añadir nodos a un cluster para aumentar también su rendimiento.

La disponibilidad y la fiabilidad son dos conceptos que si bien se encuentran íntimamente relacionados, difieren ligeramente. La disponibilidad es la cualidad de estar presente, listo para su uso, a mano, accesible; mientras que la fiabilidad es la probabilidad de un funcionamiento correcto.

Los fabricantes de hardware intentan anticiparse a los fallos aplicando la redundancia en áreas clave como son las unidades de disco, las fuentes de alimentación, las controladoras de red y los ventiladores, pero dicha redundancia no protege a los usuarios de los fallos de las aplicaciones. De poco servirá, por lo tanto, que un servidor sea fiable si el software de base de datos que se ejecuta en dicho servidor falla, ya que el resultado no será otro que la ausencia de disponibilidad. Esa es la razón de que un solo equipo no pueda ofrecer los niveles de escalabilidad, disponibilidad y fiabilidad necesarios que se ofrecen en un cluster.

En resumen, un cluster es un conjunto de computadoras interconectadas con dispositivos de alta velocidad que actúan en conjunto usando el poder del cómputo de varios CPU en combinación para resolver ciertos problemas dados. Se usa un cluster con varios computadores para aproximar a una supercomputadora.

Hoy en día las supercomputadoras son equipos excesivamente costosos que están fuera del alcance de las empresas o instituciones pequeñas. Un cluster, puede ser una solución económica, siendo una combinación de equipos personales (IBM PC Compatibles), y puede ser instalado inclusive por particulares y ofrecer rendimiento muy cercano al de una supercomputadora.

1.9.1 CLASIFICACIÓN DE LOS CLUSTERS

Los clusters se pueden clasificar atendiendo a distintos aspectos:

a) Clusters de alto rendimiento

b) Clusters de balanceo de carga

c) Clusters de alta disponibilidad

1.9.1.1 CLUSTER DE ALTO RENDIMIENTO

Se utiliza normalmente en aplicaciones que realizan cálculos intensivos: simulaciones científicas, modelos meteorológicos, etc. Pretenden sustituir a las grandes y costosas supercomputadoras.

- Utiliza una red privada de alta velocidad. Los nodos están dedicados exclusivamente al cluster. Existe un nodo maestro que se encarga de acceder, compilar y manejar las aplicaciones ejecutar. Normalmente es el único con salida al exterior.

- Sistema operativo libre: GNU/Linux, FreeBSD, NetBSD.

1.9.1.2 CLUSTER DE BALANCEO DE CARGA

Idealmente este tipo de clusters permiten que un conjunto de servidores compartan la carga de trabajo y de tráfico a sus clientes.

Para realizar el balanceo de carga se realiza desde los servidores. En este caso el balanceo queda a nivel de conexión. El balance de carga se puede hacer a dos niveles:

• A nivel de aplicación

En el caso de balanceo de carga a nivel de aplicación consiste en dar prioridades a ciertas aplicaciones. Dichas aplicaciones con mayor prioridad son las que tendrán privilegios para acceder cuando se necesite hacer un balanceo de carga.

• A nivel IP

En el caso de balance de carga a nivel IP, sólo podrán mantener la conexión aquellas IP que sean prioritarias, es decir, solo ciertas IP podrán mantener su conexión y las demás perderán el acceso. De esta manera normaliza la carga y una vez que ya se haya estabilizado podrán conectarse las demás IP al cluster.

1.9.1.3 CLUSTERS DE ALTA DISPONIBILIDAD

Intenta mantener en todo momento la prestación del servicio, encubriendo los fallos que se puedan producir.

Los requisitos para un cluster de alta disponibilidad son:

- Fiabilidad: tiempo durante el cual el sistema puede operar sin pararse
- Disponibilidad: porcentaje del tiempo en el cual el sistema está disponible para el usuario.
- Facilidad de mantenimiento: indica la facilidad de mantener el sistema en condiciones de operación (reparaciones, actualizaciones, etc.) tanto a nivel hardware como software.
- Tolerancia a fallos ó redundancia.
- Redundancia en el hardware. Replicación de componentes.
- Redundancia en el software: administración del hardware redundante para asegurar el correcto funcionamiento en caso de la caída de algún elemento.
- Redundancia temporal. Repetición de la ejecución de un conjunto de instrucciones para asegurar el funcionamiento correcto en caso de que ocurra un fallo.

CAPITULO 2

TOPOLOGÍA FÍSICA DEL CLUSTER BEOWULF

Hoy en día, en la mayoría de los equipos de cómputo el almacenamiento se realiza en discos duros, los discos duros son dispositivos que retienen información, misma que puede ser vista y/ó modificada, por ello es que es un dispositivo de entrada o salida (e/s). Existen varias tecnologías y tipos des discos duros.

La demanda del alto requerimiento de acceso, almacenaje y recuperación de los datos (lectura y escritura) en los discos, hace que las tarjetas controladoras evolucionen y formen parte de un equipo de cómputo, así como lo son las controladoras EIDE y las SCSI. Contemplando que las controladoras SCSI interactúan con unas tarjetas llamadas RAID, estas tarjetas prestan más que almacenamiento seguridad a la información, por la forma en que están agrupadas, de tal forma que todas son vistas como si fueran un solo disco, de ahí que se toman los llamados niveles de RAID.

Una manera de que los usuarios de una red, que acceden todo el tiempo a archivos compartidos, eviten el exceso de trafico en la red, sería implementando un dispositivo NAS, que no es mas que un servidor dedicado a compartir y distribuir los archivos almacenados en los diversos discos duros que lo componen.

El sistema de memoria está conectado al mismo bus que la unidad central de proceso, para que trabajen conjuntamente y estén comunicados directamente uno del otro, así agilizando el acceso y la disponibilidad de información para las diferentes tareas del procesador, ya que en este bloque se encuentra cargados el sistema operativo, aplicaciones, instrucciones de los programas en ejecución, etc.

El sistema de procesamiento es también parte fundamental para el rendimiento y tiempo de respuesta dentro del equipo de cómputo, ya que es aquí donde se ejecutan los diversos tipos de operaciones requeridas por el sistema operativo así como de aplicaciones que se ejecuten dentro del equipo de cómputo. Bifurcando en la velocidad (medida en Mhz) de procesamiento de E/S ya sea de 64 bits ó 32 bits según los ciclos de reloj.

Se hace una pequeña mención de la comparación de un sistema clúster con un sistema SMP.

La interacción entre dispositivos ya sea internos como externos es algo muy importante, los buses tienen un papel muy importante en el aprovechamiento de los recursos del clúster, ya que es la interacción y manipulación de los datos que van a viajar a través de todo el equipo hacia todos los dispositivos.

2.1 SISTEMA DE DISCOS

La mayoría del almacenamiento de datos en las computadoras personales es hecho en discos magnéticos (llamados discos duros). Los Discos Duros, también llamados discos rígidos, son dispositivos capaces de almacenar información digital durante largos períodos de tiempo sin necesidad de recibir energía durante el mismo. Dicha información es grabada en una superficie magnética que posteriormente puede ser leída o rescrita.

2.1.1 SUBSISTEMAS DE DISCOS

El subsistema de discos está constituido básicamente por discos y tarjetas controladas de discos como se ilustra en la figura 2.1.

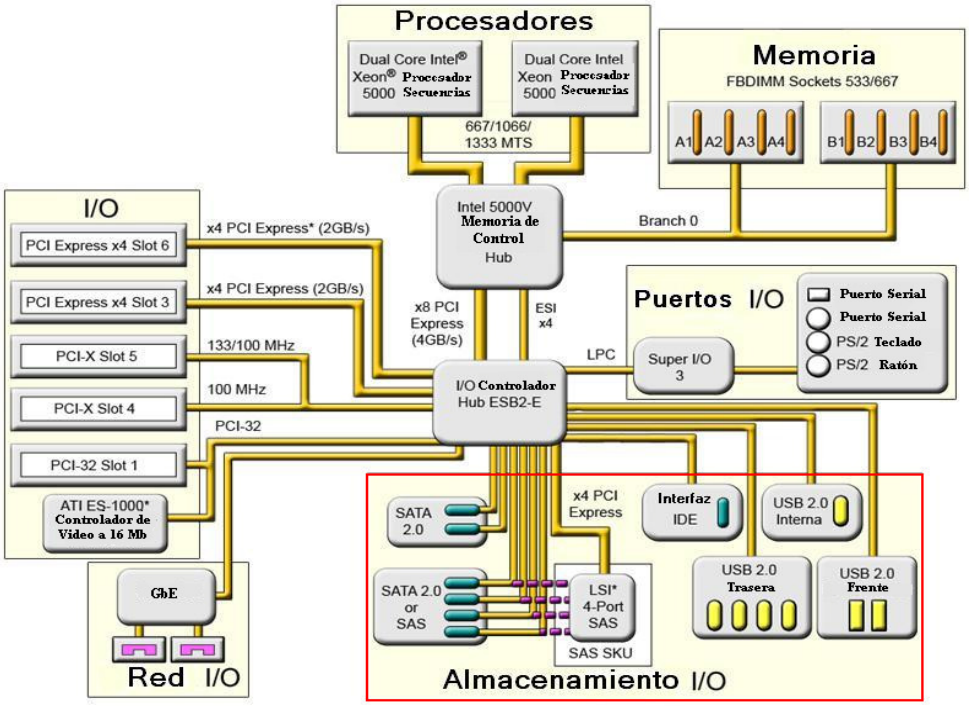


Figura 2.1 Ubicación del subsistema de discos en un diagrama a

bloques del hardware en un equipo de cómputo

A continuación se mencionan los discos duros mas usados hoy en día.

- **SATA**

Los discos duros tipo Serial ATA (SATA) se están convirtiendo en el nuevo estándar de tecnología de discos duros. Actualmente, los fabricantes de tarjetas madre incluyen entradas SATA en sus tarjetas. Debido a que su cable es considerablemente más angosto, las unidades de disco SATA proporcionan mayor flujo de aire y espacio en los sistemas de cómputo que las unidades EIDE. Algunas tarjetas madre más antiguas no

cuentan con puertos SATA, pero puede instalarse una tarjeta controladora PCI SATA a fin de agregar soporte para unidades SATA.

- **EIDE**

Los discos duros EIDE (Siglas en inglés de electrónica mejorada de unidad de disco integrada), llamados también PATA (ATA paralelo), han sido el estándar de la industria de computación durante más de 10 años. Algunas tarjetas madre modernas no cuentan con puertos EIDE, pero puede instalarse una tarjeta controladora PCI EIDE a con el fin de agregar soporte para unidades EIDE. Tanto su sistema operativo como su hardware deben soportar el disco duro que elija:

Sistema operativo	Compatibilidad de discos duros WD SATA	Compatibilidad de discos duros WD EIDE
Windows® Vista™	Sí	Sí
Windows® 2000 o XP	Sí	Sí
Windows 98SE o ME	No	Sí
Mac® OS X	No	Sí
Mac OS 9.x	No	Sí
Linux	Si	Si

Tabla 2.1 Comparación de discos duros WD SATA con WD EIDE

Algunos Otros discos duros se muestran a continuación en la tabla 2.2.

Tecnología	Capacidad Típica	Características
MFM (Modulación por Modificación de Frecuencia)	10 a 40 Mb(Megabytes)	Primer tipo de Disco Duro empleado en los computadores personales, también conocido como norma **ST 506**, corresponde a unidades con interfase tipo IBM
RLL (Corrimiento por limite de Longitud)	40 – 100 Mb(Megabytes)	Muy similar al MFM, pero cuenta con un nuevo tipo de codificación de datos que permite una mayor densidad de información
ESDI (Interfaz de Dispositivo de	80 a 200 Mb(Megabytes)	Desarrollo posterior al MFM, con una ventaja de tener una tarjeta de interfaz más confiable y

Procesamiento Pequeño)		rápida, apta para mayor velocidad de transferencia de datos en procesadores superiores al 8086, se utilizo también en computadores con procesadores 386
AT (AT accesorio o IDE (Inteligencia de manejo electrónico)	40 a 528 Mb (Megabytes)	Comercialmente se conocen como "Discos Inteligentes" porque internamente incluyen una tarjeta acopladora para acceder directamente al bus de datos. Tienen gran acogida por su desempeño y precio bajo
ATA-2 o EIDE (Mejorado IDE)	528 a 8Gb (Gigabytes)	IDE, mejorado con las características, pero diseñado para romper la barrera de los Megabytes
SCSI (Interfaz de Sistema de Computadora Pequeña)	200 Mb a 20Gb ó masAño 2.002, se tiene información que la barrera de capacidad en los Discos Duros ha sido superada a los Terabytes, Información suministrada por MM-SICODIGSA® (Ing. Merchán)	Constituyen un desarrollo dirigido en especial a Sistemas de Alto Desempeño como pueden ser los servidores. Lo mismo que en la tecnología IDE, se consideran "Inteligentes" en la medida en que incorporan directamente la Interfaz

Tabla 2.2 Tipos de Discos Duros (Hard Disk)

2.1.1.1 TARJETAS CONTROLADORAS

Las tarjetas controladas pueden ser de varios tipos: IDE, SCSI y de canal de fibra entre otros. En la figura 2.1 se muestra su ubicación dentro de un diagrama a bloques del hardware en un equipo de cómputo.

Las principales funciones de una tarjeta controladora de disco son:

- Es la encargada de la conversión de direcciones lógicas a físicas (cilindros, pistas y sectores)
- Manejo de las cabezas de los discos
- Lectura y escritura en los platos de los discos
- Almacenaje y recuperación de los datos en el buffer del disco.

La tarjeta controladora EIDE (de sus siglas en ingles enhanced integrated drive electronics), fue la más común durante mucho tiempo, a través de ésta se realizan los accesos a diversos componentes de almacenamiento como son cdrom’s, floppy’s, discos duros etc. Su antecesora fue la controladora IDE (integrated drive electronics), por eso es común que se le conozca también con éste nombre.

Este tipo de controladora tiene como limitante que solo se pueden conectar dos dispositivos por cada cable IDE, aceptando únicamente la tarjeta madre dos cables; la longitud del cable no debe exceder los 18 cm., las velocidades que se pueden alcanzar a través de éstas es de 66 y 160 Mbps, la transferencia y la administración de estas controladoras es llevada a cabo por los chips denominados PIO o DMA. Donde el chipset PIO se considera obsoleto.

Las tarjetas controladoras de interfaz de sistema de computadora SCSI (de sus siglas en ingles Small Computer System Interface) debido a sus características, son las más utilizadas en sistemas de cómputo de grandes y de alto requerimiento de acceso a los datos. Sus principales características son:

- Soporta hasta 15 dispositivos por canal (discos duros, cd - rom, unidades de cinta, quemadores, scanner, etc.)
- Alcanza velocidades desde 5 hasta 360 Mbps de transferencia de datos
- Ha evolucionado a través del tiempo en SCSI 1, SCSI 2 y actualmente ultra-SCSI 360
- Tiene su propio chip set de control, topología y comandos de funcionamiento.

Para la buena operación de este tipo de tarjetas controladoras existen reglas para las direcciones de cada dispositivo a conectar, en las cuales se determina la prioridad que tiene cada uno al realizar el arranque de la máquina como se muestra en la tabla 2.3.

ID de prioridad SCSI
7 Más Alto
6
5
4
3
2
1
0
15
14
13
12
11
10
9
8 Más Bajo

Tabla 2.3 Niveles de prioridad de una tarjeta SCSI

El peso de prioridad de asignación de dispositivos está determinado de la siguiente manera el ID 7 es el de más alto valor y siempre deberá estar asignado a la

controladora, y de manera descendente del 6 al 1 y del 15 al 8 es el peso que tendrá cada dispositivo en la cadena que vaya formando.

De este modo es que una controladora SCSI puede llegar a soportar los 15 dispositivos.

- **TARJETAS RAID**

Existen otras tarjetas que interactúan con las tarjetas SCSI, llamadas tarjetas RAID éste tipo de tarjetas tienen como característica ser las encargadas de agrupar un número determinado de discos duros para que de manera lógica sean vistos por un sistema como un solo disco, otra característica consiste en que al agrupar los discos se les proporciona niveles de seguridad para la información, formando de este modo lo que se le denomina niveles de RAID.

Los niveles de RAID más utilizados son:

a) RAID 0, que consiste en solo agrupar los discos sin ningún nivel de seguridad.

b) RAID 1, que consiste en hacer copias espejos de la información en dos discos a la vez.

c) RAID 5 que consiste en agrupar por lo menos 3 discos y organizar la información de tal modo que se encuentra dispersa y permite la reconstrucción de la información de manera segura y consistente.

Existen diferencias muy marcadas entre la tecnología IDE y la tecnología SCSI detalladas en la tabla 2.4:

IDE	SCSI
• Bus de 16 bit's • Máximo hasta 4 dispositivos por controladora • Buen desempeño en desktop o laptop • Bajo costo	• Bus de 8 y 6 bit's • Máximo de dispositivos conectados 15 por controladora en dispositivos internos y externos • Recomendable para estaciones de trabajo y servidores • Optimo desempeño en discos con velocidades de revoluciones por minuto que van desde 10,000 hasta 15,000

Tabla 2.4 Comparativo entre sistema IDE y SCSI

TIPOS DE ALMACENAMIENTO

En sistemas grandes, donde los usuarios requieren acceder a archivos compartidos, es necesario implementar una forma de almacenamiento en la que los datos se encuentren centralizados en un solo dispositivo y evitar el exceso de tráfico en la red.

- **ALMACENAMIETO ADJUNTO A LA RED.**

Un dispositivo NAS (de sus siglas en ingles networked attached storage) es un servidor que se dedica a compartir archivos. NAS no provee ninguno de los servicios que un sistema servidor típicamente provee como e-mail, autentificación o manejo de archivos. NAS permite más espacio de almacenamiento de discos duros para ser agregados a los ya existentes en la red sin necesidad de apagarlos para mantenimiento o actualizaciones. Con el dispositivo NAS el almacenamiento no es parte integral del servidor. De hecho en este diseño céntrico de almacenamiento, el servidor continúa teniendo todo el procesamiento de datos pero el dispositivo NAS entrega los datos al usuario. Un dispositivo NAS no necesita ser ubicado dentro de un servidor, puede ser ubicado en cualquier parte en la red (LAN) y puede construirse con múltiples dispositivos NAS.

Como ya se menciono un NAS es un dispositivo dedicado al almacenamiento que se conecta en la red. Los clientes mandan las peticiones de archivo directamente al dispositivo NAS pasando por los servidores de propósito general en la red. La mayoría de los NAS tienen un sistema operativo optimizado como el kernel de unix que solo sirve archivos, además de un procesador de E/S y un procesador de archivos. Puede leer archivos de la mayoría de las plataformas (ya que entiende la mayoría de los sistemas de archivos) y tiene la habilidad de compartir datos a múltiples servidores que solo hace una herramienta colaborativa. La figura 2.2 muestra la forma de conexión de una SAN.

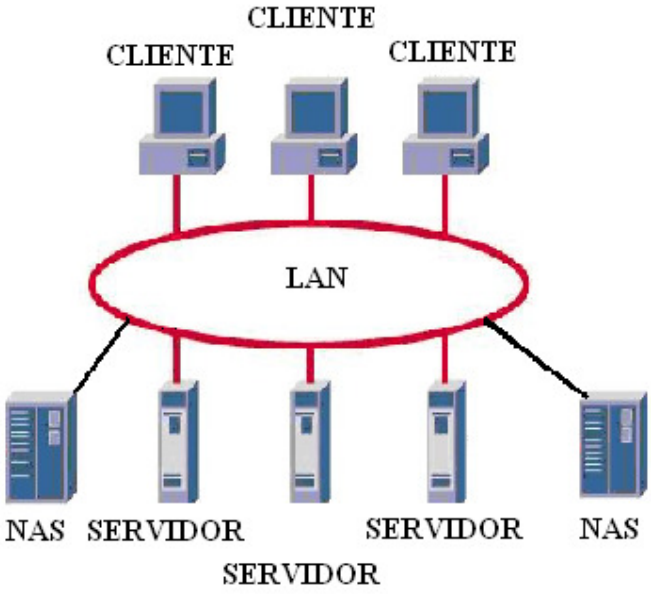


Figura 2.2 Forma de conexión de una SAN

2.2 SISTEMA DE MEMORIA

El sistema de memoria es la unidad del equipo, en donde se almacenan los datos y las instrucciones de los programas en ejecución, que recupera y graba en ella la unidad central de proceso a través de las dos operaciones básicas definidas sobre ella, una de lectura y la otra de escritura. Para su comunicación se conectan directamente al mismo bus ambas unidades, la UCP y la memoria. La memoria principal puede ser central o expandida. La memoria central está dividida en celdas formadas generalmente por un octeto. Cada uno de esos octetos es una unidad direccionable en la memoria. El mapa de memoria se corresponde con el espacio de memoria direccionable. Este espacio viene determinado por el tamaño de las direcciones, es decir, por el tamaño del bus de direcciones. A continuación detallaremos acerca de todo los que comprende esta parte de nuestro equipo.

2.2.1 SUBSISTEMA DE MEMORIA

Dentro de un sistema de cómputo el subsistema de memoria es de los vitales; ya que aquí se carga la información o datos más recurrentes de consulta como son: el Sistema Operativo, las Aplicaciones, etc.

Dentro de un sistema de computo, existen diferentes niveles de caché y memoria L1, L2 y L3 básicamente y éstos tienen por objetivo agilizar la disponibilidad de información para las tareas del procesador, en la figura 2.3 se muestra un esquema de la ubicación de cada uno de estos niveles de caché y memoria

Adicional a estos niveles de caché y memoria existe la memoria convencional que puede ser estática y dinámica (SRAM o DRAM), siendo está última la de mas bajo costo en comparación con las anteriores ya que como referencia, la memoria de acceso mas rápido y cerca físicamente del procesador determina el costo de la misma.

Existe otra terminología para clasificar a los diferentes tipos de memoria que existen:

- Memoria de solo lectura (ROM)
- Memoria de acceso de lectura y escritura temporal (RAM)
- Memoria Virtual.

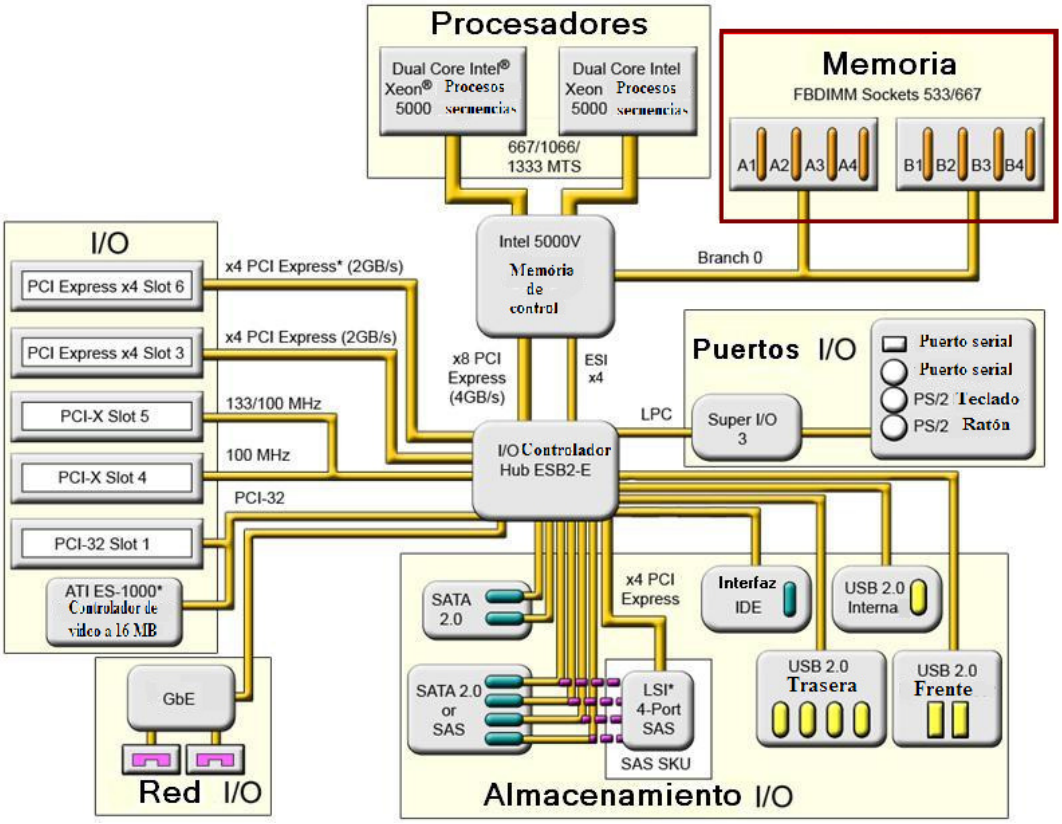


Figura 2.3 Ubicación del subsistema de memoria en un diagrama a bloques del hardware en un equipo de cómputo.

Todas estas memorias tienen diferentes tareas de aplicación. En la tabla 2.5 se presentan algunas memorias y su arquitectura:

SUBSISTEMA	TIPO	ARQUITECTURA
Nivel 2 de caché (L2)	SRAM	Asíncrona o síncrona
Memoria principal	DRAM	SDRAM, DDR-SDRAM, RDRAM
Gráficos	DRAM	SDRAM, DDR- SDRAM, SGRAM, WRAM, RDRAM
Control de disco	DRAM	PM, EDO, SDRAM
Disco	DRAM / Flash	EDO, SDRAM
BIOS	Flash	N/A

Tabla 2.5 Tipos de memorias

Los empaquetamientos que han tenido más aplicaciones son los 2 tipos SIMMs y DIMMs (figura 2.4) teniendo características básicas que las hacen diferentes:

- SIMMs (single in- line memory modules) módulos simples de memoria en línea, este tipo de memoria su empaquetamiento esta constituido por un peine de 72 pines, maneja una velocidad de transferencia de 32 bits, fue utilizado en la mayoría de sistemas de computo en la década de los 80's y buena parte de los 90's.

- DIMMs (dual in – line memory modules) módulos duales de memoria en línea, Su empaquetamiento es diverso en cuanto al número de pines que constituyen el peine que son 72, 88, 168 o 200 pines, maneja velocidades de 32 bits y 64 bits es la mas popular hoy en día al ser utilizada en desktop y servidores; para computadoras de tipo portátil se utilizan los tipo DIMMs llamados SODIMMs.

Figura 2.4 Tipos de empaquetamiento de memorias

El empaquetamiento DIMMs ha tenido gran desarrollo desprendiéndose dos ramas como son DRAM y SRAM (memoria de tipo dinámico y estático respectivamente) siendo a su vez del tipo dinámico la que ha tenido gran diversidad de desarrollo:

- SDRAM (Memoria dinámica síncrona) es una de las de mas uso actualmente, llega a tener accesos en su bus frontal de 100 y 133 Mhz.

- DDR – SRAM (doble flujo de datos en memoria síncrona) tiene dos caminos de acceso de lectura y escritura para llegar al bus frontal y alcanzar velocidades de 166 y 266 Mhz.

- RDRAM este tipo de memoria esta orientada a utilizarse en equipos con altos requerimientos de acceso de lectura y escritura así como seguridad en sus datos de tipo volátil.

Otras características que se han adicionado a las memorias es la corrección de errores por medio de diferentes métodos:

- Corrección de paridad: este método consiste en agregar un dígito binario más al segmento de datos para checar la información y corrige errores de tipo simple.

- No paridad: está orientado principalmente a equipos portátiles y de escritorio

- Memoria tipo ECC (chequeo y corrección de errores) con este método se logra corregir 1 bit y detectar hasta 2 bits de error, esta orientado a servidores.

- Memoria de tipo espejo: este método consiste de tener el doble de bancos de memorias interconectados entre sí, por lo cual, lo que existe en un DIMM de memoria se refleja en el otro, de tal modo que si hay un daño total de un DIMM el espejo pueda tomar el control de la información y no ser perceptible el daño para el usuario.

La memoria Caché: ésta memoria está enfocada a proporcionar información o datos al procesador de manera más rápida debido a que se encuentra a disposición del procesador de manera más directa.

Físicamente el nivel de memoria 1 se encuentra dentro del encapsulado del procesador, el nivel de memoria caché se encuentra entre el procesador y la memoria convencional sirviendo básicamente de buffer en el mapeo de la memoria convencional. Como se muestra en la figura 2.5.

Figura 2.5 Memoria caché de nivel 1 y 2.

En la evolución de la arquitectura de computadoras se ha llegado a la memoria de nivel 3 L3 caché que tiene como tarea principal, el incrementar el desempeño de un procesador en menor tiempo; este tipo de memoria es implementado en procesadores con tecnología que requiere 64 bits es decir éste nivel de memoria es implementado en

los procesadores de Intel denominados TITANIUM y Xeon. Esto se muestra en la figura 2.6.

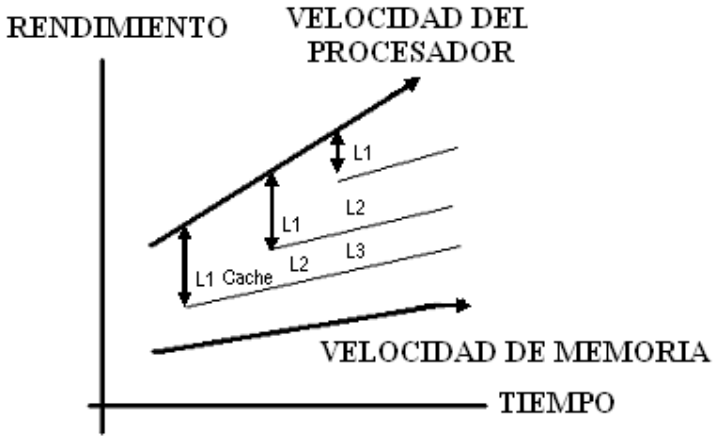


Figura 2.6 Utilización de la memoria caché

2.3 SISTEMA DE PROCESAMIENTO

Todo el trabajo de rendimiento y respuesta de un sistema de cómputo recae en este bloque ya que es el encargado de la ejecución de todos tipos de operaciones requeridas por el sistema operativo, aplicaciones que se ejecutaran en el sistema de cómputo. En la figura 2.7 se muestra la ubicación del sistema de procesamiento en un diagrama a bloques del hardware de un equipo de cómputo.

Los componentes básicos de que está constituido un procesador en su encapsulado son los siguientes:

- Una unidad lógica aritmética
- Una unidad de punto flotante
- Memoria caché
- Una unidad de registro de instrucciones, datos y control de circuitos.

- La velocidad o rendimiento de los procesadores es determinado por sus ciclos de reloj, mismos que son medidos en MHz, todos sus componentes internos son constituidos por capas de silicio y silicón. Otra característica de estos es su velocidad de e/s que puede ser de 64 bits para itanium así como los xeon, y de 32 bits para los denominados pentium I, II, III y IV. En principio todos los procesadores desarrollaban todas las actividades, pero en su evolución se pensó en separar las operaciones matemáticas para agilizar su rendimiento y se integró en la composición de los sistemas de cómputo un coprocesador matemático que tuvo en principio está función.

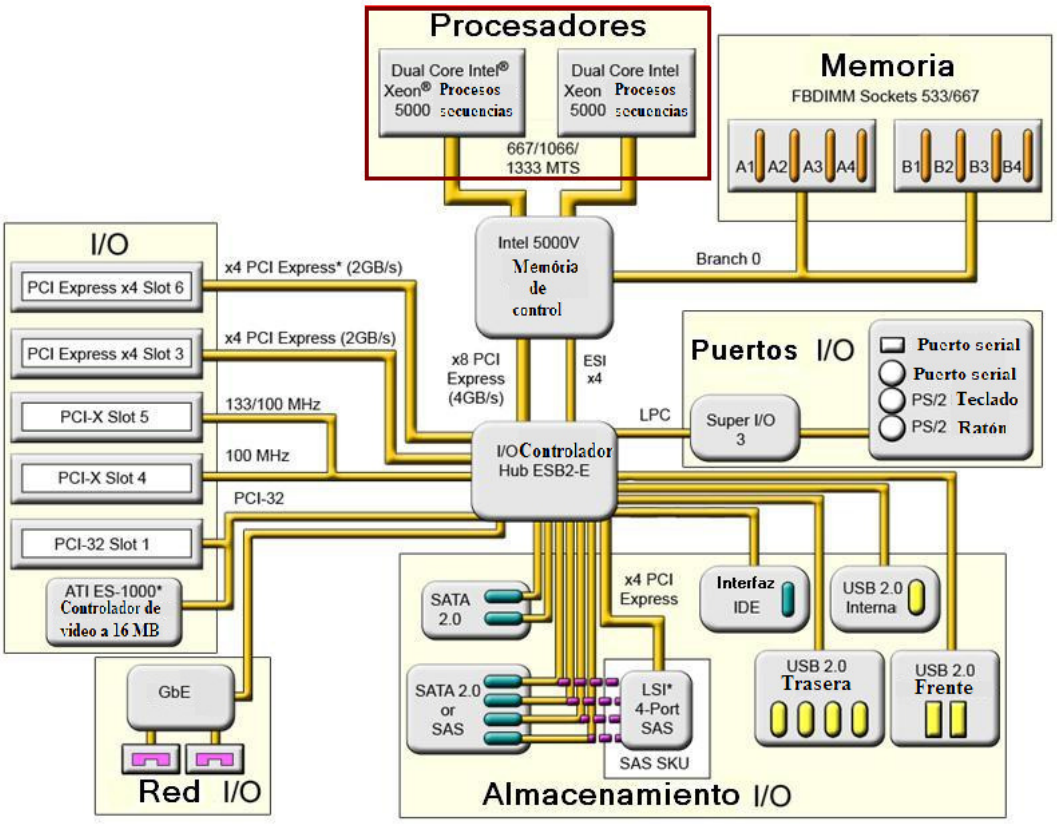


Figura 2.7 Ubicación del sistema de procesamiento en un diagrama a bloques del hardware en un equipo de cómputo.

Todos los procesadores actuales tienen en su composición interna diferentes velocidades de reloj para determinar su velocidad interna de procesamiento y externa para determinar su velocidad de E/S se indican como PII 933/133 esto indica que internamente el procesador tiene una velocidad interna de 933 y una velocidad externa de 133 MHz.

Un concepto importante dentro de los procesadores es el modo de transferencia entre procesador y la L1 caché, L2 caché y la memoria externa que se lleva acabo atreves de bloques de información.

En la arquitectura de procesadores son tres tecnologías las que han determinado el modo de operación de instrucciones que son:

- MMX que lleva acabo 57 instrucciones. Su característica principal consiste en tener mayor rendimiento en datos de tipo grafico y represento a la Familia X86.

- SSE lleva acabo alrededor de 70 instrucciones y fue el sucesor del MMX y constituyo la familia de PII, PIII XEON.

- SSE2 lleva acabo 144 instrucciones y constituye actualmente a los procesadores PIV y XEON que como principal característica tienen mejor rendimiento en actividades de tipo grafico como son imágenes en 3D, video y audio.

Otro diferenciador de la evolución de los procesadores es el tipo de empaquetamiento que han tenido a través de su existencia PGA con 370 y 423 pines que fueron utilizados por los procesadores celeron y PII y PIV respectivamente, también el micro BGA y micro PGA que se utilizo en sistemas portátiles.

En la figura 2.8 se nuestra los 2 tipos de empaquetamiento ya mencionados, y en la tabla 2.6 los sockets donde son colocados los procesadores y sus características.

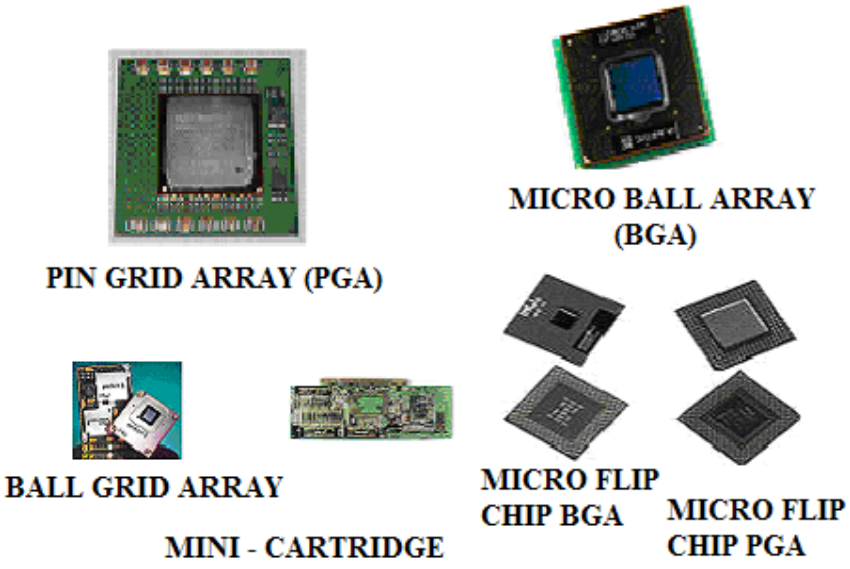


Figura 2.10 Empaquetamiento PGA y BGA

SOCKET	PINES	VOLTAJE	DESCRIPCIÓN
Socket 1	169	5.0	17 x 17 PGA 486 únicamente
Socket 2	238	5.0	19 x 19 PGA 486 y P24T
Socket 3	237	3.3 a 5.0	19 x 19 PGA 486 y P24T
Socket 4	273	5.0	21 x 21 PGA Pentium P5
Socket 5	320	3.	37 x 37 PGA Pentium P54C

		3	
Socket 6	235	3.3	19 x 19 PGA 486 y P24T
Socket 7	321	3.3	Pentium P54CS
Socket 8	387	2.1 a 3.5	Socket asimétrico (para Pentium Pro)
Socket 370 o PGA 370	370	1.5 a 2.1	PPGA Céleron (PGA370)
PGA 423	423	1.1 a 1.85	Pentium IV (Willimatte) con RDRAM
mPGA 478B	478	2.1 a 3.5	Pentium IV (Willimatte o Northwood)
Socket 603	603	1.1 a 1.85	XEON (Foster)
Socket M	418	0.9 a 1.9	Itanium

Tabla 2.6 Tipo de sockets y sus características.

Los empaquetamientos de Slot 1 y 2 fueron de gran uso en computadoras del tipo estaciones de trabajo y servidores debido a su bajo costo, rendimiento y la facilidad del procesamiento simétrico. Este tipo de encapsulamiento fue utilizado de la manera en que se indica en la tabla 2.6.

Slot 1	Slot 2
SECC, SECC2 o SEP	SECC únicamente
Algunos Celeron o Pentium II/III	Pentium II/III Xeon
Caché L2 pequeño; alta o media velocidad	Caché L2 grande, alta velocidad
2 way SMP	4 u 8 way SMP
Físicamente pequeño	Físicamente grande

Tabla 2.7 Características del slot 1 y 2

Dentro del modo de operación de los procesadores, existen los que en su composición de encapsulamiento los constituyen sistemas de múltiples procesadores llevando a su vez dos tipos de trabajo asimétrico y simétrico. Este tipo de arreglos está orientado con la visión de tener mayor rendimiento, ya que proporciona mayor capacidad en el bus de la parte frontal del equipo y capacidad de procesamiento de acuerdo al número de procesadores que estén integrados en un mismo sistema de modo

par. Siendo el modo de operación simétrico o asimétrico el principal diferenciador entre ellos; como lo muestra la figura 2.9.

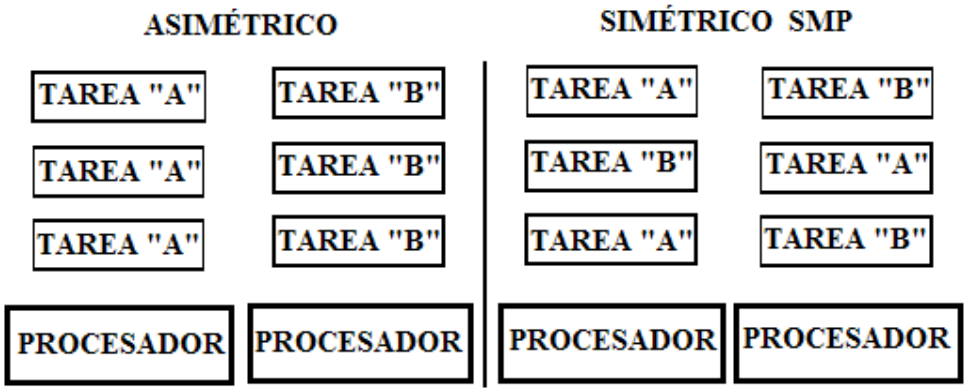


Figura 2.9: Modos de operación asimétrico y simétrico.

En modo simétrico las cargas de trabajo son repartidas de manera que sea parejo el aprovechamiento del poder de procesamiento; mientras que en el modo asíncrono cada procesador tiene una tarea especifica. Cabe mencionar, que éste tipo de sistemas, el almacenamiento de memoria es compartido, como lo muestra la figura 2.10.

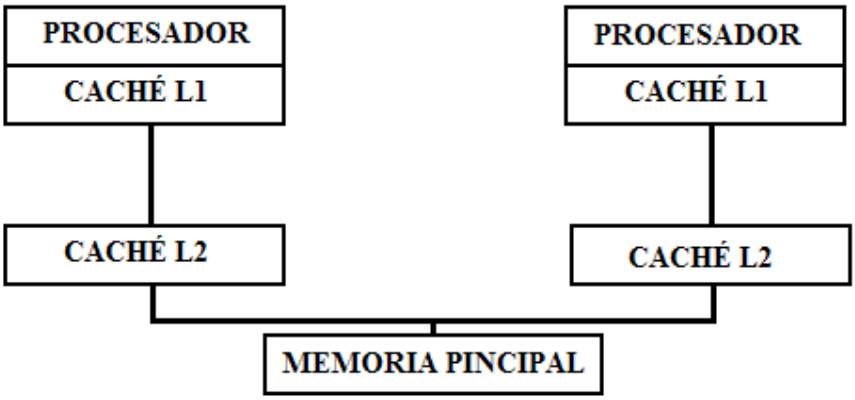


Figura 2.10 Memoria compartida por sistemas multiprocesador.

Los sistemas dedicados xeon quad core, es una nueva tecnología que maximizan el rendimiento, son servidores con 4 núcleos, 8MB de caché, rendimiento optimizado y uso eficaz de la energía. El procesador intel xeon quad core, basado en la potente micro arquitectura core de intel le ayuda a maximizar el rendimiento con menos requisitos de enfriamiento. Con un desempeño hasta un 50% superior al del procesador xeon Dual core y hasta un 150% mayor que otros procesadores.

Tiene una increíble potencia para ejecutar aplicaciones de uno o varios subprocesos y la ventaja empresarial que necesita al aprovechar la caché de 8MB, el bus

frontal de máxima velocidad y la DIMM de búfer completo. Los servidores dedicados de internet xeon basados en el procesador Intel xeon quad core también soportan varias tecnologías avanzadas de servidor que ayudan a las compañías a optimizar sus operaciones, reducir costos, y asegurar la continuidad de un negocio:

- La virtualización de tecnología de intel (virtualization technology; intel VT) ofrece asistencia de hardware para entornos virtuales basados en software para soportar nuevas capacidades, incluyendo sistemas operativos 64 bits y aplicaciones.

- La aceleración de tecnología de los dispositivos de e/s (acceleration technology; intel i/oat) ofrece aceleración e/s que mejora significativamente el rendimiento del flujo de datos.

2.3.1 SISTEMAS SMP vs. CLUSTER

Existe un tipo de arreglo (cluster) de sistemas de cómputo que pueden realizar el mismo trabajo de procesamiento que un sistema SMP. Este arreglo de cluster duplica el poder de cómputo de un sistema simple a uno de mayor capacidad con características físicas que lo llevan a hacer más robusto, con mayor redundancia y disponibilidad que un sistema SMP ya que al contar con por lo menos dos equipos la caída de uno de estos no implica la perdida de la operación ó producción de un sistema de computo. En contraparte con un sistema SMP al tener este todo centralizado en un solo gabinete la falla de este conlleva a la caída del sistema si este no cuenta con dispositivos redundantes.

Un sistema SMP tiene como principal desventaja la de tener que compartir el sistema de almacenamiento de memoria mientras que un clúster tiene independencia en cada uno de sus componentes lo que lo hace más independiente, como se observa en la Fig. 2.11 y 2.12.

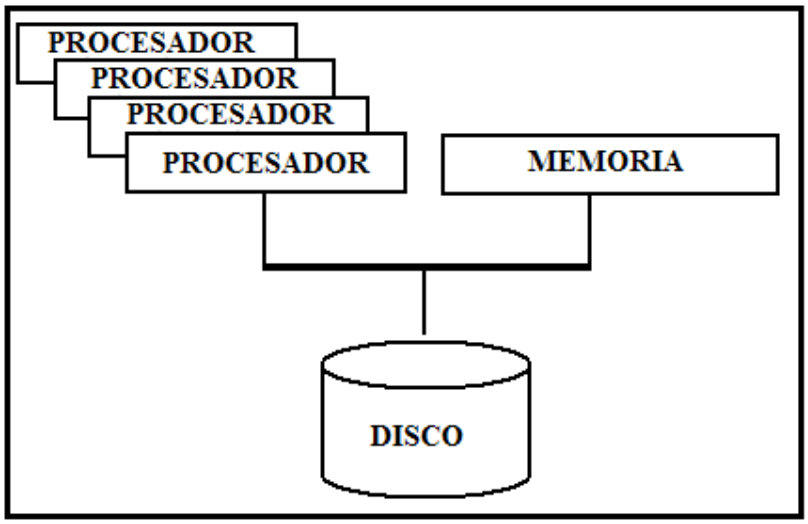


Figura 2.11 Almacenamiento en un sistema SMP

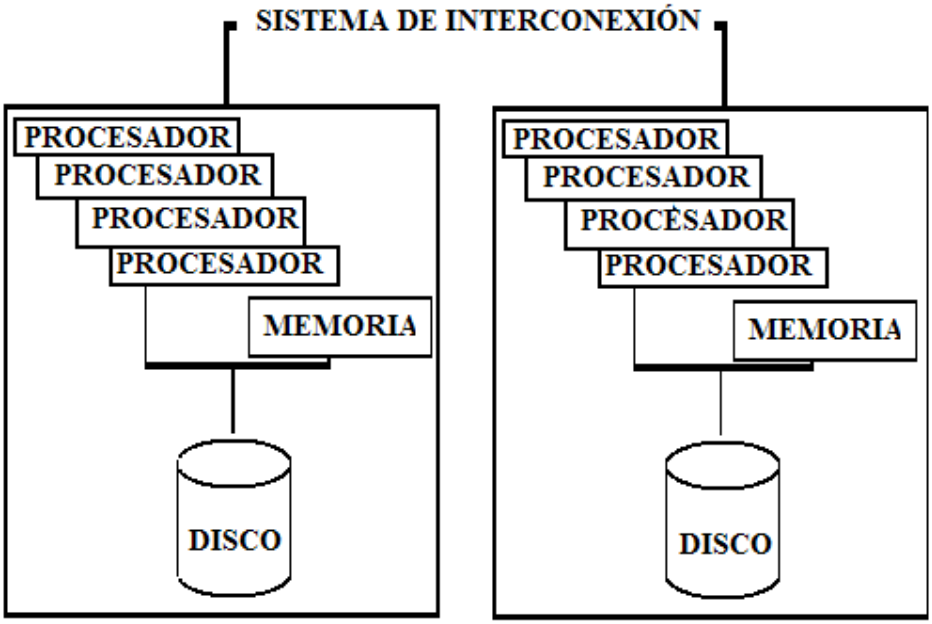


Figura 2.12 Almacenamiento en un clúster

Existen otras ventajas en el sistema de cluster, una muy representativa es el bajo costo y modularidad de crecimiento, ya que su sistema de interconexión puede llegar a ser tan diverso, al poder ser construido a través de tarjetas de fibra, tarjetas Ethernet, SCSI y ATM, no teniendo como limitante la configuración física de cada uno de los nodos que conformen al cluster, por lo que llega a ser un grupo de equipos con diferentes características orientados a representar de manera lógica un solo equipo con grandes capacidades de procesamiento.

De acuerdo al tipo de aplicaciones y modelos de cómputo utilizados, pude llegar a necesitarse un sistema de alta capacidad de procesamiento paralelo, SMP ó de clúster, siendo este último una opción viable para este tipo de necesidades.

En un esquema más a detalle un cluster sería esquematizado como se muestra en la Fig. 2.13.

En el arreglo de cluster existen diversas topologías de acuerdo a la necesidad que se quiera llegar a cubrir y estas son:

a) Solución de respaldo hardware

b) Procesamiento paralelo (Balance de cargas y estabilidad)

c) Base de datos.

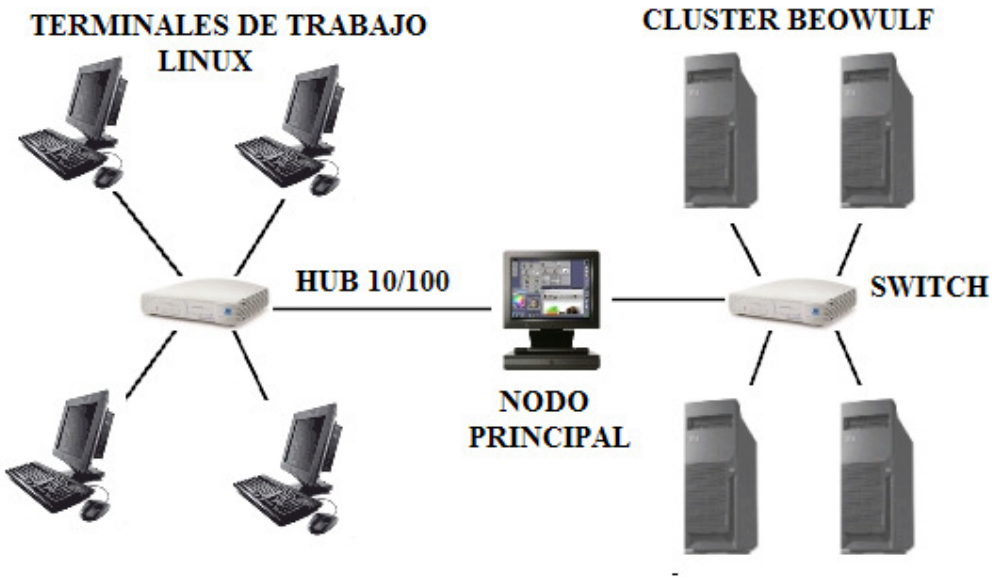


Fig. 2.13 Esquema detallado de un cluster

2.4 ARQUITECTURA DEL BUS

El bloque de composición del bus de buses tiene un valor relevante ya que es el que proporciona el modo de interacción de un sistema de cómputo con dispositivos externos <discos externos, impresoras, scanner, lectoras,> y de comunicaciones (Tarjetas de red, MODEM, fibra, SCSI).

A través de la evolución de los sistemas de cómputo este subsistema ha sido de los que más han cambiado en su composición, velocidades y formatos de tarjetas.

La tarea básica del bus, es la transferencia de datos entre el procesador y las tarjetas que están conectadas a través de el dando a la información un formato de bloque de lectura, escritura.

A la fecha han existido dos versiones de buses PCI 2.1 Y 2.2 teniendo la primera una capacidad de velocidad de transferencia de datos de 0 a 66Mhz y la segunda agregando la capacidad de ser detectada de manera automática por el sistema así como también la posibilidad de administración del encendido y apagado de cada uno de los Slot que componen al sistema.

Con la evolución de los ciclos del reloj en los procesadores desde el año 2001 se encuentra también disponible otra modalidad de PCI que es la PCI-X que proporciona el doble canal de acceso a la tarjeta llegando de este modo hasta los 133Mhz.

Otro bus de gran popularidad actualmente es el universal serial bus ó USB, este tipo de bus tiene la característica de ser de los más veloces y de los que más

posibilidades de conexión de dispositivos periféricos proporciona a un sistema de cómputo siendo 127 los que se podrían conectar a través de este bus, además que no requiere disponibilidad de E/ S.

Otro componente esencial dentro del sistema de buses son los chipset (conjunto de chips), este dispositivo encargado del control de flujo de información entre cada uno de los subsistemas de cómputo. De acuerdo al tipo de tecnología o tarjetas que componen a un sistema de cómputo se puede determinar que tipo de chipset están utilizando.

El bus de administración del sistema tiene como primer microprocesador de Intel en incorporar una interfaz de bus de administración del sistema, el procesador Xeon agrega varias funciones de facilidad de uso a la línea de productos de Intel. Dentro del cartucho, dos nuevos componentes (además del sensor térmico) usan esta interfaz para comunicarse con otro hardware y software de administración del sistema.

CAPÍTULO 3: TOPOLOGÍA LÓGICA Y CONFIGURACIÓN DE NODOS

3.1 INSTALACIÓN DEL SISTEMA OPERATIVO.

Sin el software, una computadora no es más que una masa metálica sin utilidad. Sin embargo, con el software una computadora puede almacenar, procesar y recuperar información. El software para computadoras puede clasificarse en general en dos clases: los programas de sistema, que controlan la operación de la computadora en sí y los programas de aplicación, los cuales resuelven problemas para sus usuarios.

El programa fundamental de cualquier sistema de cómputo es el sistema operativo (SO), quien controla todos los recursos de la computadora y proporciona la base sobre la cual pueden escribirse los programas de aplicación.

El sistema operativo es el encargado de brindar al usuario una forma amigable y sencilla de operar, interpretar, codificar y emitir las órdenes al procesador central, para que realice las tareas necesarias y específicas para completar una orden.

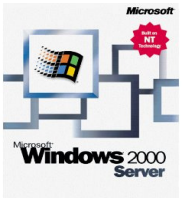
Un sistema operativo se define como un conjunto de procedimientos manuales y automáticos, que permiten a un grupo de usuarios compartir un equipo de cómputo eficazmente.

Existen diversos arreglos de hardware para la construcción y configuración de un cluster basados en sistemas operativos windows, novel, hpux, AIX, solares, linux, etc.

3.1.1 SELECCIÓN DEL SISTEMA OPERATIVO.

En un inicio, el sistema operativo con el cual se crearon los clusters fue linux. En la actualidad, el uso de los clusters se ha extendido considerablemente, por lo cual las demás plataformas han modificado su sistema, a fin de permitir la construcción de los mismos. Esto permite al usuario final y desde luego al creador del cluster tener una mayor flexibilidad al momento de elegir las características deseadas para el cluster, sin depender directamente de un sistema operativo. Desde luego, se tienen que verificar las opciones que cada uno ofrece y así hacer la elección correcta.

A continuación se mencionan algunos de los sistemas operativos utilizados para la creación de clusters:

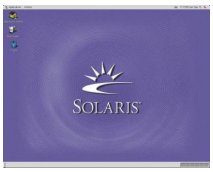


a) Windows 2000 Server

Provee la capacidad para la construcción de un cluster, sin embargo no permite la modificación o manipulación de su núcleo por ser software propietario. Para su utilización se debe pagar una licencia por cada equipo en donde sea instalado. La mayoría de los clusters construidos con este sistema son para generar respaldos o distribuir el trabajo entre los nodos, los cuales tienen cada uno una tarea específica, y no implementa el paralelismo o la división de un mismo trabajo en partes más pequeñas.

Los requisitos de hardware para su instalación son:

- Procesador: intel pentium 133 Mhz hasta intel quad core
- Memoria (RAM): 256 Mb de RAM recomendado (128 Mb mínimo soportado;
- Espacio en disco duro: 1.0 GB
- Monitor: VGA o monitor de alta resolución.



b) Solaris

Sistema unix propietario de SUN, permite el desarrollo del cluster y sus equipos ofrecen un gran desempeño durante largas horas de trabajo sin ocasionar errores o fallas en el sistema. Existe además software de instalación para la construcción de cluster sin la intervención del usuario para su configuración. Su obtención es mediante el pago de una licencia o en la adquisición de hardware SUN.

Los requisitos de hardware son:

- Plataforma x86(32 y 64 bits)
- Espacio en disco: 600 Mb para computadoras de escritorio; En el caso de servidores se debe tener en cuenta 1 Gb
- Memoria: 64 Mb como mínimo.

c) Linux

Sistema operativo de fuente abierta (opensource) basado en unix. En este sistema surgió el primer cluster, por las ventajas que en ese momento ofrecía. Al ser de fuente abierta es permitido modificar o manipular según las necesidades del usuario. Soporta arquitecturas VESA local bus y PCI. También soporta buses MCA (arquitectura propietaria de IBM), y ESDI. linux soporta también multiprocesadores sobre la base de arquitecturas intel MP.

Los requisitos de hardware para su instalación son:

- Procesador: 386, o superior.
- Memoria: mínimo 4Mb (recomendable 8Mb para ambiente gráfico)
- Espacio en disco duro : 500Mb

En linux se puede trabajar en modo texto o en su defecto en modo grafico. Para saber que elegir, se puede considerar lo siguiente:

- Para modo texto se recomienda: 233 MHz G3 o mejor, 128MB RAM.
- Para gráficos: 400 MHz G3 o mejor, 256 MB RAM.

Al revisar las características de estos sistemas operativos y el hardware requerido para su instalación, linux tiene la mejor opción de sistema operativo por las exigencias mínimas en cuestión de hardware, además de ofrecer un gran soporte para el trabajo intensivo durante un largo periodo de tiempo. Además, linux reconoce la memoria que se tiene a diferencia que en Windows solo reconoce hasta cierto límite de memoria.

3.1.1.1 SELECCIÓN DE LA DISTRIBUCIÓN.

Una vez decidido que se utilizará Linux como SO lo siguiente es la elección de la distribución del mismo que será adecuada o brindará las mejores características para la construcción del cluster.

Una distribución de Linux es un conjunto de programas recopilados a lo largo y ancho de sitios en Internet, organizados de tal manera que ofrezcan una solución particular o general hacia él o los usuarios.

No cambia en nada Linux de una distribución a otra, ya que en realidad Linux es tan solo un archivo pequeño llamado kernel el cual se encarga de poner orden y administrar óptimamente la computadora, tanto a la hora del arranque como durante el tiempo que permanece encendida.

Las personas o empresas que integran las distribuciones seleccionan software a su criterio, y lo incorporan dentro de una serie de discos compactos con conjuntos de herramientas y aplicaciones de instalación. Así que a veces una distribución no significa que sea mejor que otra por tener diez discos o porque esté contenida en un solo disco. Depende más que nada, de la aplicación que se quiera desarrollar.

Las principales distribuciones son las siguientes:

DEBIAN….

GNU DEBIAN: Denotada como la mejor de las distribuciones en el mundo, enfocada primordialmente a desarrolladores, programadores, administradores de red, centros de cómputo de alto desempeño.

MANDRIVA: Usuarios de cómputo del hogar, oficina, escuela. Bastante fácil de instalar, amigable y con una cantidad de paquetes suficientes para comenzar a conocer Linux en serio.

RED HAT: Usuarios enfocados a pequeñas redes, estudiantes universitarios, programadores y centros de información de tamaño mediano.

COREL: Usuarios que no tienen mucho espacio en la computadora, interesados en conocer Linux sin comprometer mucho espacio y que solo necesitan paquetes básicos, por lo regular recomendada para estudiantes y entusiastas de la computación.

SUSE: Distribución alemana multipropósito, algunas configuraciones requieren de conocimientos adicionales. Programas e interfaces que facilitan su uso.

SLACKWARE: Distribución para los amantes de la historia de Linux. Bastante completa, recomendada para desarrolladores y programadores.

FEDORA: Es un sistema operativo completo para escritorio y servidor, creado íntegramente con software de código abierto.

3.1.2 INSTALACIÓN EN EL NODO MAESTRO

El programa de instalación de Fedora Core 6 para procesador de 64 bits, ofrece más de una ventana de diálogo para el proceso de instalación. Además de darle la posibilidad de insertar comandos desde el intérprete de comandos de la shell, tiene a su disposición muchos tipos de diferentes mensajes.

La versión de Fedora que se ocupó para este proyecto fue fedora core 6 para 64 bits que se puede descargar desde la siguiente página:

http://fedora.mirror.iweb.ca/core/6/x86_64/iso/

Existen varias opciones para poder llevar a cabo la instalación. El más conveniente es usar el dvd de instalación. Lo primero que necesitamos hacer es configurar el BIOS para que arranque desde el disco. El BIOS contiene configuración que controla el orden de los dispositivos de arranque. Si su PC arranca desde un dispositivo distinto al del medio de arranque de fedora core, hay que verificar la configuración de arranque del BIOS.

Cuando se ejecute el dvd, la primera pantalla que va aparecer es la siguiente:

Figura 3.1 Pantalla de arranque

En seguida presionamos enter para ejecutar en modo por defecto. En el modo por defecto, la instalación usa una interfaz gráfica si es posible. Si el programa de instalación se ejecuta desde el cd o dvd de instalación de fedora core, en el modo por defecto se usa estos discos como la fuente de instalación.

En seguida nos va aparecer la siguiente ventana.

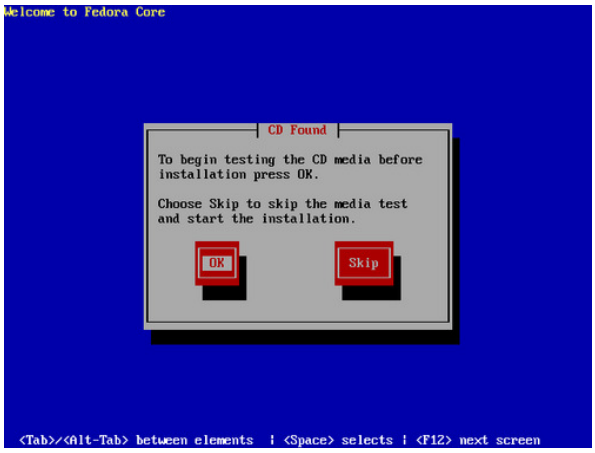


Figura 3.2 Pantalla de prueba de medio

Seleccione “OK” para probar los discos, o seleccione “SKIP” para proceder con la instalación sin probar los discos. Es recomendable seleccionar skip (sino se esta seguro de que los discos estén completamente bien grabados con cada una de las librerías, lo recomendable es realizar una prueba a los discos).

En seguida nos aparece una ventana en la cual nos pregunta en que idioma queremos realizar la instalación. El programa de instalación muestra una lista de idiomas soportados por fedora.

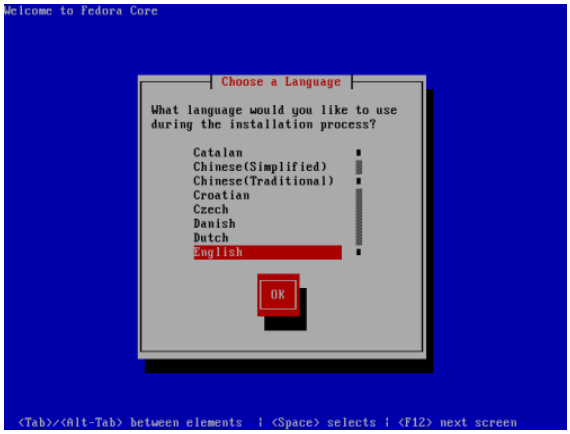


Figura 3.3 Pantalla de selección de idioma

Resalte el idioma correcto en la lista y seleccione“OK” .

3.1.3 PARTICIONAMIENTO DEL DISCO

Nos preguntará que en donde queremos realizar la instalación, si se desea tener más sistemas operativos y por lo tanto si se requiere hacer particiones independientemente de las que ya ocupa por defecto el instalador de la distribución de linux.

Figura 3.4 Pantalla de particionamiento

Seleccionamos la opción que aparece eliminar todas las particiones de los discos seleccionados y realizar una configuración por defecto.

Si los discos seleccionados tienen alguna partición de linux, esta opción las elimina e instala fedora core en el espacio libre resultante. Esta opción no modifica las particiones asignadas a otros sistemas operativos, no linux. Sin embargo, no discrimina entre particiones asignadas a distintas distribuciones de linux, y las eliminará.

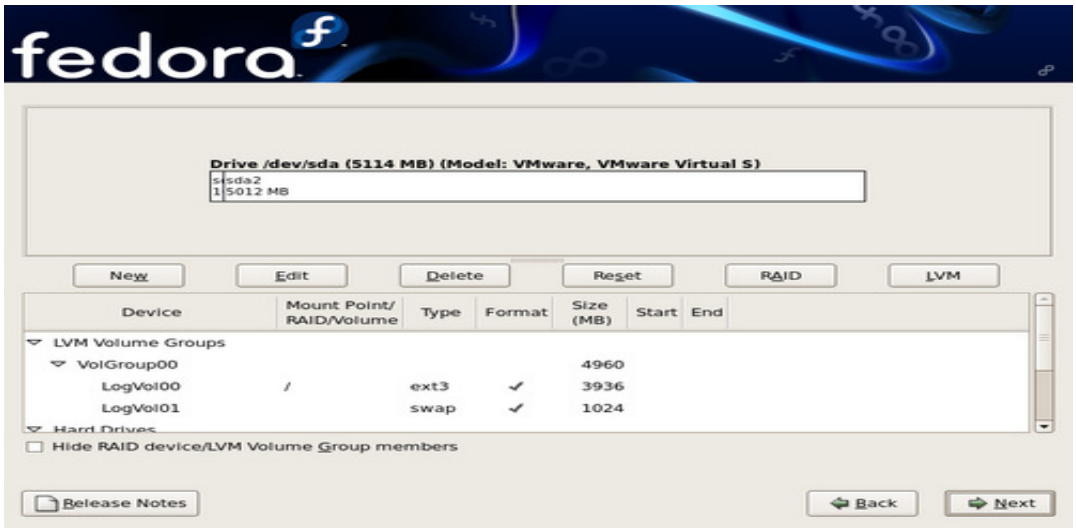


Figura 3.5 Pantalla de configuración de disco

La forma recomendada de particionar el disco para una instalación correcta de linux es la siguiente:

Una partición/swap (de al menos 32 MB) se usa para soportar memoria virtual. En otras palabras, los datos se escriben en la partición swap cuando no hay suficiente RAM para almacenar los datos que su sistema está procesando. Si la computadora tiene 16 MB para almacenar los datos que su sistema está procesando. Incluso si tienes más memoria, se recomienda seguir utilizando una partición swap. El tamaño de la partición swap debe ser igual a la cantidad de memoria RAM que tiene el sistema.

Una partición /boot (32 MB máximo). Esta partición montada sobre /boot contiene el kernel del sistema operativo (permitirá arrancar al sistema), junto a otros ficheros utilizados para el proceso de arranque. Debido a las limitaciones de las mayorías de las BIOS de las PCS, se aconseja la creación de una pequeña partición que contendrá estos ficheros.

Una partición /boot, ocupará el resto del espacio disponible en disco y que será donde se encontrarán los directorios y documentos de los usuarios (/home) y los programas instalados.

3.1.4 CARGADOR DE ARRANQUE

Un cargador de arranque es un pequeño programa que lee e inicia un sistema operativo. fedora core usa el cargador de arranque GRUB por defecto. Si tiene varios sistemas operativos, el cargador de arranque determina con cual iniciar, usualmente ofreciéndolos en un menú.

La siguiente pantalla muestra las opciones de configuración del cargador de arranque.

Figura 3.6 Pantalla de configuración del cargador de arranque

3.2 SELECCIÓN DE ZONA HORARIA

Esta pantalla le permite especificar la zona horaria correcta para la ubicación de su computadora. Especifique la zona horaria si planea usar NTP (protocolo de hora de red), para mantener la precisión del reloj de su sistema.

Para seleccionar una zona horaria usando el mapa, primero ubique el puntero del ratón sobre la región en el mapa. Haga clic una vez para amplificar la región en el mapa. Luego seleccione el punto amarillo que represente la ciudad más cercana a su ubicación. Una vez que seleccione un punto, se vuelve una “X” roja para indicar su selección.

Figura 3.7. Pantalla de selección de la zona horaria

3.4 CLAVE DE ROOT

Fedora utiliza una cuenta especial llamada root (cuenta de superusuario) para la administración del sistema. El administrador es el que se encarga de dar ciertos privilegios a cada uno de los usuarios.

Ingrese la clave de root en el campo “clave de root”. fedora mostrará los caracteres como asteriscos por seguridad. Ingrese la misma clave en el campo “confirme” para asegurar que sea la correcta.

Figura 3.8 Pantalla para poner la clave de root

Después de poner la clave de root, seleccione next para continuar.

3.5 INSTALACIÓN EN EL NODO ESCLAVO.

La instalación para el nodo esclavo se va a realizar de igual forma que se realizó la instalación del sistema operativo y cada uno de los paquetes. La diferencia se realiza cuando se administran los nodos. En el capítulo cuatro hablaremos específicamente de este gran aspecto.

3.6 FORMAS DE HACER CÓMPUTO PARALELO

Existen diferentes formas de realizar cómputo paralelo, dependiendo del tipo de aplicación que queramos ejecutar y según las características que se tengan en cuestión de hardware. Como ya se explicó al principio de este capítulo, linux es la plataforma ideal para realizar cómputo paralelo por las ventajas que ofrece. Pero reiteramos que se depende totalmente del hardware para realizar cómputo paralelo.

Independiente de la distribución de Linux con la que se desee trabajar, existen algunos paquetes para realizar cómputo paralelo. Las tres más importantes son las siguientes:

- OPEN MP (multiprocesamiento de código abierto)
- MPI (interfaz de paso de mensajes),
- PVM (máquina virtual paralela)

A continuación se va realizar una descripción de cada una de los paquetes arriba mencionados.

3.6.1 OPENMP

OPENMP es una interfaz de aplicación de programa (API), que nos permite añadir concurrencia a las aplicaciones mediante paralelismo con memoria compartida. Se basa en la creación de hilos de ejecución paralelos compartiendo las variables del proceso padre que los crea. Esta disponible en múltiples plataformas y lenguajes, desde las derivadas de unix hasta las plataformas windows. Existen extensiones para los lenguajes más conocidos como c, c++, fortran,...

3.6.1.1 MODELO DE PROGRAMACIÓN

- OPENMP está basado sobre la existencia de múltiples hilos en la programación con memoria compartida. Un proceso de memoria compartida consiste de múltiples hilos.

Paralelismo Implícito:

- OPENMP es un explícito modelo de programación (no automático), ofreciendo al programador un completo control sobre la paralelización.

3.6.1.1.1 MODELO DE BIFURCACIÓN - UNIÓN (FORK - JOINJOIN MODEL):

- OpenMP utiliza el modelo de bifurcación de ejecución paralela.

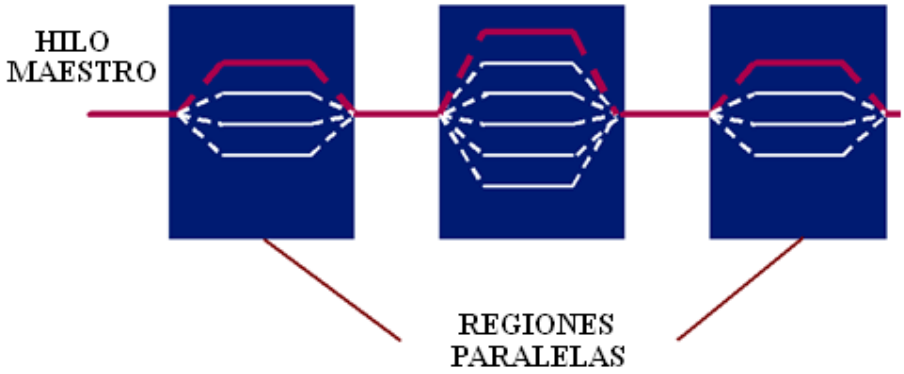


Figura 3.9 Modelo bifurcación-unión

- Todos los programas basados en OPENMP comienzan con un solo proceso: el hilo maestro. El hilo maestro ejecuta secuencialmente hasta la primera región paralela construida que es encontrada.
- Bifurcación: el hilo maestro después crea un equipo de hilos paralelos.
- Las declaraciones en el programa que son encerradas por la región paralela construida que son después ejecutados en paralelo entre varios equipos de hilos.
- Unión: cuando el equipo de hilos completa las declaraciones en la construcción de la región paralela, sincroniza y termina, dejando sólo el hilo maestro.

Compilador basado en la directiva:

- El paralelismo con OPENMP es especificado a través del uso de directivas en el compilador las cuales son insertadas en c/c++ o fortran a través del código.

Una directiva o pragma es la forma de indicarle al compilador alguna instrucción al momento de la compilación.

3.6.1.3 SINTAXIS DEL PRAGMA OPENMP

La mayoría de los bloques de construcción en OPENMP son directivas de compilación o pragmas.

- En C y C++, los pragmas toman la siguiente forma:

#pragma omp construct [clause[clause]…]

3.6.1.4 REGIONES PARALELAS

- Define una región paralela sobre un bloque de código estructurado.
- Los hilos se crean como 'parallel'.
- Los hilos se bloquean al final de la región.
- Los datos se comparten entre hilos al menos que se especifique otra cosa.

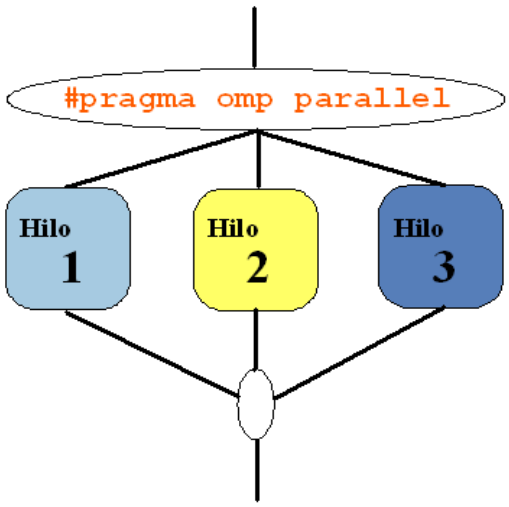


Figura 3.10 Diagrama de regiones paralelas ejecutado por varios hilos

El propósito principal de una región paralela es:

- Una región paralela es un bloque de código que será ejecutado por múltiples hilos. Esta es la principal construcción paralela en OPENMP.

3.6.1.4.1 PRAGMAS

A continuación se muestran los pragmas más comunes en el uso de programación paralela:

- parallel: esta directiva nos indica que la parte de código que la comprende puede ser ejecutada por varios hilos.

- for: Igual que parallel pero optimizado para los bucles **for**. Su formato es:

```
#pragma parallel for [clausula, ... , clausula]
```

- section y sections: indica secciones que pueden ejecutarse en paralelo pero por un único hilo.

- single: la parte de código que define esta directiva, solo se puede ejecutar un único hilo de todos los lanzados, y no tiene que ser obligatoria mente el hilo padre.

- master: la parte de código definida, solo se puede ejecutar por el hilo padre.

- critical: Solo un hilo puede estar en esta sección. Definen secciones críticas o condiciones de carrera.

- atomic: Se utiliza cuando la operación atañe a solo una posición de memoria, y tiene que ser actualiza solo por un hilo simultáneamente. Operaciones tipo **x++** o **--x** son las que usan esta clausula.

- flush: Esta directiva resuelve la consistencia, al exportar a todos los hilos un valor modificado de una variable que ha realizado otro hilo en el procesamiento paralelo.

Enseguida se muestran en la tabla 3. Las directivas y las cláusulas para fortran 90

Directiva	Propósito
!$OMP PARALLEL [clausulas] bloque de sentencias !$OMP END PARALLEL	Define una región paralela. Un bloque de código va a ser ejecutado en paralelo utilizando multihilos
!$OMP DO [clausulas] bloque con sentencia DO !$OMP END DO	Especifica que el ciclo DO que sigue a la directiva sea ejecutado en paralelo. Esta directiva debe estar dentro de una región paralela
!$OMP PARALLEL SECTIONS [clausulas] [!$OMP SECTION] bloque de sentencias !$OMP END PARALLEL	Especifica una región paralela la cual encapsula una sección de código no iterativo para ser ejecutado en multihilos. Cada sección es ejecutada 1 vez por 1 hilo.

SECTIONS	
!$OMP SINGLE [cláusulas] bloque de sentencias !$OMP END SINGLE	Las sentencias que estén dentro de SINGLE serán ejecutadas por 1 hilo de 1 grupo. Los hilos que no se estén utilizando se esperan a que termine el hilo, a menos que se especifique NOWAIT
!$OMP PARALLEL DO [clausulas] bloque de sentencias !$OMP END PARALLEL DO	Especifica una región paralela que contiene un ciclo DO.
!$OMP MASTER bloque de sentencias fortran. !$OMP END MASTER	Las sentencias que se encuentran en este bloque son ejecutadas únicamente por el hilo maestro del grupo.
!$OMP CRITICAL [(name)] bloque de sentencias fortran. !$OMP END CRITICAL [(name)]	Restringe el acceso a las sentencias en este bloque a un hilo a la vez. Un hilo espera fuera del bloque hasta que ningún otro hilo este ejecutando ese bloque.
!$OMP BARRIER	Sincroniza todos los hilos en un grupo. Cada hilo espera hasta que todos los hilos lleguen a este punto.
Cláusulas	Propósito
PRIVATE(*list*)	Hace que las variables en la lista sean privadas para cada hilo
SHARED(*list*)	Todos los hilos comparten las variables de la lista
DEFAULT(*PRIVATE* | *SHARED* | *NONE*)	Especifica el atributo por default para todas las variables en una región paralela DEFAULT(PRIVATE | SHARED | NONE)
REDUCTION(*list*)	Realiza una reducción a las variables que se encuentran en la lista

Tabla 3. Directivas y cláusulas más importantes (Fortran 90)

Funciones más utilizadas

- omp_set_num_threads: Fija el número de hilos simultáneos.

- omp_get_num_threads: Devuelve el número de hilos en ejecución.
- opm_get_max_threads: Devuelve el número máximo de hilos que lanzará nuestro programa en las zonas paralelas. Es muy útil para reservar memoria para cada hilo.
- omp_get_thread_num: Devuelve el número del thread dentro del equipo (valor entre 0 y omp_get_num_threads()-1)
- opm_get_num_procs: Devuelve en número de procesadores de nuestro ordenador o disponibles (para sistemas virtuales).
- opm_set_dinamic: Valor booleano que nos permite expecificar si queremos que el número de hilos crezca y decrezca dinámicamente.

3.6.1.5 BLOQUES DE CONSTRUCCIÓN EN PARALELO

Los hilos se asignan a un conjunto de iteraciones independientes.

Los hilos deben de esperar al final del bloque de construcción de trabajo en paralelo.

#pragma omp parallel

#pragma omp for

for(i = 0; i < 12; i++)

c[i] = a[i] + b[i]

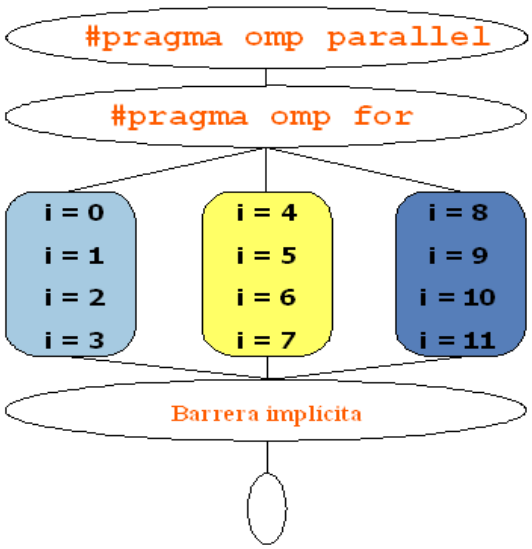


Figura 3.11 Ejemplo de paralelización del bloque de construcción **for**

3.6.1.6 FORMA DE EJECUTAR LAS DIRECTIVAS DE OPENMP

El paralelismo a nivel de OPENMP lo realiza el compilador, esto es, el compilador es quien realiza la paralelización, el programador sólo le indica al compilador mediante directivas, las secciones que desea se ejecuten en paralelo, por tanto no hay una biblioteca que se pueda incluir a cualquier programa para utilizar OPENMP, es el compilador quien debe ser capaz de entender y realizar las tareas de paralelización.

Hoy en día existen algunos compiladores que soportan las directivas de OPENMP. En Linux se ubican dos principalmente, que son:

- Compilador intel c++ para linux a 32/64 bits
- Compilador GCC

3.6.1.7 COMPILADOR INTEL

El compilador de intel para linux es un compilador para crear aplicaciones en y c++ que pueden correr en arquitecturas de 64 bits. Dicho compilador se puede descargar en una versión de evaluación en la siguiente liga. Además se resumen las versiones de compilador y versión de OPENMP soportadas.

http://www.intel.com/support/mt/sp/performancetools/c/linux/sb/cs-007720.htm

3.6.1.8 CONFIGURACIÓN DEL COMPILADOR

El compilador de Intel c++ para linux para procesador de 64 bits, compila códigos fuente de c y c++. En arquitecturas de 64 bits, la ruta de instalación es la siguiente:

/opt/intel/cce/10.1.xxx.

Donde xxx representa un número de tres dígitos asignado a cada compilador. La documentación se encuentra en la carpeta .doc ubicada en donde se encuentre la instalación del compilador.

3.6.1.9 FORMA DE COMPILAR DESDE UNA TERMINAL

Antes de comenzar a compilar cualquier archivo es necesario ubicarse en la siguiente ruta, en donde se encuentra instalado el compilador:

opt/Intel/cce/10.1.018/bin

Una vez que ya estemos ubicados en esa ruta, vamos a publicar el acceso a las librerías así como el uso de los pragmas cada vez que abramos una terminal. La manera de publicar es tecleando como superusuario lo siguiente:

source/opt/intel/cce/10.1xxx/bin/iccvars.sh

Una vez que ya se hizo lo anterior, para compilar un programa que este en lenguaje "c" se hace de la siguiente forma:

icc nombre_del_archivo.c –o nombre_del_archivo.out

Para compilar un archivo con el compilador de c++ se hace de la siguiente forma:

icpc nombre_del_archivo.cpp –o nombre_del_archivo.out

En ambos casos se va a crear un ejecutable. Es necesario correr el ejecutable para dicho ejecutable se podrá correr desde la terminal de Linux como:

./nombre_del_archivo

Este compilador maneja un comando para ayuda, el cual muestra al introducir un comando todas sus características así como la sintaxis correcta para utilizarlo. Dicho comando es:

man icc

3.6.1.10 COMPILADOR GCC

Las siglas GCC significan GNU Compiler Collection (Colección de compiladores GNU). Antes estas siglas de GNU C Compiler (compilador c GNU). Como su nombre indica es una colección de compiladores y admite diversos lenguajes: c,c++, objetive c, chill, fortran, y java.

El compilador se distribuye bajo la licencia GPL (general public license), lo que lo hace de libre distribución: se pueden hacer copias de él y regalarlas o venderlas siempre que se incluya el código fuente (o se indique cómo conseguirlo) y se mantenga la licencia. Existen versiones para prácticamente todos los sistemas operativos. Viene incluido en la mayoría (si no en todas) las distribuciones de GNU/linux. La versión DOS de este compilador es el DJGPP.

3.6.1.11 FORMA DE COMPILAR

Para compilar un programa escrito en c se debe teclear lo siguiente:

gcc nombre_del_archivo -o nombre_del_archivo.out

Para correr el ejecutable, como ya se explicó en el punto anterior se realiza con:

./nombre_del_archivo

3.6.2 LIBRERÍA DE PASO DE MENSAJES (MPI)

El paso de mensajes es una tarea ampliamente usada en ciertas clases de máquinas paralelas, especialmente aquellas que cuentan con memoria distribuida. En los últimos años, se ha logrado un proceso substancial en convertir aplicaciones significativas hacia este tipo de tareas. Más recientemente diferentes sistemas han demostrado que un sistema de paso de mensajes puede ser implementado eficientemente y con un alto grado de portabilidad.

Al diseñarse MPI (Interfaz de paso de mensajes), se tomaron en cuenta las características más atractivas de los sistemas existentes para el paso de mensajes, en vez de seleccionar uno solo de ellos y adoptarlo como el estándar. Resultando así, en una fuerte influencia para MPI los trabajos hechos por IBM, INTEL, Nx/", Express, nCUBE's Vernex, p4 y PARMACS. Otras contribuciones importantes provienen de Zipcode, chimp, PVM, Chameleon y PICL.

La meta de MPI, es desarrollar un estándar práctico, portable, eficiente y flexible (que sea ampliamente usado), para escribir programas que implementen el paso de mensajes. Por lo cual la interfaz intenta establecer, para esto un estándar práctico, portable, eficiente y flexible.

El estándar final para MPI fue presentado en la conferencia de supercomputación en noviembre de 1993, constituyéndose así el foro para el MPI.

En un ambiente de comunicación con memoria distribuida, en la cual las rutinas de paso de mensajes son de bajo nivel, los beneficios de la estandarización son muy notorios.

MPI es un sistema complejo, el cual comprende 129 funciones, de las cuales la mayoría tiene muchos parámetros y variantes.

3.6.2.1 Metas del MPI

- Diseñar una interfaz de programación aplicable (no necesariamente para compiladores o sistemas que implementen una librería).
- Permitir una comunicación eficiente. Evitando el copiar de memoria a memoria y permitiendo, donde sea posible, la sobre posición de computación y comunicación, además de aligerar la comunicación con el procesador.
- Permitir implementaciones que puedan ser utilizadas en un ambiente heterogéneo.
- Permitir enlaces convenientes en c y fortran para la interfaz.

- Asumir un interfaz de comunicación seguro. El usuario no debe lidiar con fallas de comunicación. Tales fallas son controladas por el subsistema de comunicación interior.

- Definir un interfaz que no sea muy diferente a los actuales, tales como PVM, NX, Express, p4, etc., y proveer de extensiones para permitir mayor flexibilidad.

- Definir un interfaz que pueda ser implementado en diferentes plataformas, sin cambios significativos en el software y las funciones internas de comunicación.

- La semántica de la interfaz debe ser independiente del lenguaje.

- La interfaz debe ser diseñada para producir tareas seguras.

3.6.2.2 MODELO DE PROGRAMACIÓN

En el modelo de programación MPI, consta de uno o más procesos comunicados a través de llamadas a rutinas de librerías para mandar (send) y recibir (receive) mensajes hacia y desde otros procesos. En la mayoría de las implementaciones de MPI, se crea un conjunto fijo de procesos al iniciar el programa, y un proceso es creado por cada tarea. Sin embargo, estos procesos pueden ejecutar diferentes programas. De ahí que, el modelo de programación MPI es algunas veces referido como MPMD (múltiple program múltiple data) para distinguirlo del modelo SPMD (single program múltiple data), en el cual cada procesador ejecuta el mismo programa.

Debido a que el número de procesos en un cálculo de MPI es normalmente fijo, se puede enfatizar en el uso de los mecanismos para comunicar datos entre procesos. Los procesos pueden utilizar operaciones de comunicación punto a punto para mandar mensajes de un proceso a otro, estas operaciones pueden ser usadas para implementar comunicaciones locales y no estructuradas. Un grupo de procesos puede llamar colectivamente operaciones de comunicación para realizar tareas globales tales como Broadcast, etc. La habilidad de MPI para probar mensajes da como resultado el soportar comunicaciones asíncronas. Probablemente una de las características más importantes del MPI es el soporte para la programación modular.

Un mecanismo llamado comunicador permite al programador del MPI definir módulos que encapsulan estructuras internas de comunicación (estos módulos pueden ser combinados secuencialmente o paralelamente).

3.6.2.3 INSTALACIÒN

El paquete rpm de LAM/MPI se puede descargar desde el sitio web http://www.lam-mpi.org/download/ en donde se podrá obtener la última versión de dicho software.

Para realizar correctamente la instalación, debemos leer el readme (archivo de lectura) de instalación e ir siguiendo cada uno de los pasos para su instalación.

3.6.2.4 CONFIGURACIÓN

En el directorio /usr/boot se crearan los siguientes archivos con sus correspondientes configuraciones.

1. Archivo conf.lam, se recogerá la topología de red utilizada. lamd $inet_topo

2. archivo bhost.def, se añadirá al final los nombres de todos los nodos que forman parte del cluster.
   localhost pc1 pc2 …

Para testar si la configuración es correcta se deberá de realizar lo siguiente:

$>lamboot

Si la salida muestra algún error entonces es que hay algún problema con la instalación, comunicación entre nodos, etc.

3.6.2.5 COMANDOS BÁSICOS

Aunque MPI es un sistema complejo y multifacético, podemos resolver un amplio rango de problemas usando seis de sus funciones, estas funciones inician y terminan un proceso de cómputo, identifican procesos, además de mandar y recibir mensajes.

- MPI_INT:

Es una rutina de inicialización y debe ser llamada antes de cualquier otra rutina de MPI. Únicamente se debe llamar una vez, ya que cualquier otra llamada subsecuente será errónea. Todos los programas que utilicen MPI deben de contener una llamada a MPI_Init (). Su sintaxis es:

#include “mpi.h”

int MPI_Init (int *argc,char ***argv)

argc puntero al número de argumentos

argv puntero al vector de argumentos

- MPI_FINALIZE:

Se utiliza para terminar el entorno MPI. Al hacer la llamada a int MPI_Finalize (void) ya no se podrá hacer una llamada a cualquier función MPI (ni siquiera MPI_Init ()). El

usuario debe cerciorarse que todas las comunicaciones pendientes que involucran a un proceso estén terminadas antes de que el proceso llame a la rutina MPI_Finalize () Su sintaxis es la siguiente:

#include “mpi.h”

int MPI_Finalize()

Todos los procesos deben llamar esta rutina antes de terminar. Después de haber llamado esta rutina el número de procesos no esta definido. Debe llamarse poco antes de terminar el programa en cada proceso.

- MPI_COMM_SIZE:

Determina el tamaño del grupo asociado con un comunicador. Su sintaxis es la siguiente:

#include “mpi.h”

int MPI_Comm_size (MPI_Comm comm., int *size)

Parámetros

Parámetro	Entrada / Salida	Descripción	Tipo de Dato
Comm	Entrada	Indica el handle de comunicación.	MPI_Comm
Size	Salida	Número de procesos en el grupo de comm	Entero

Tabla 3.1 Parámetros de MPI_COMM_SIZE

- MPI_COMM_RANK:

Determina el rango del proceso actual dentro del comunicador. Su sintaxis es la siguiente:

#include “mpi.h”

int MPI_Comm_rank (MPI_Comm comm, int *rank)

Parámetros

Parámetro	Entrada / Salida	Descripción	Tipo de Dato
Comm	Entrada	Indica el handle de comunicación.	MPI_Comm

Rank	Salida	Indica el grado del proceso llamador en el grupo de comm	Entero

Tabla 3.2 Parámetros de MPI_COMM_RANK

- MPI_SEND:

Envía datos en un mensaje. Su sintaxis es:

#include "mpi.h"

int MPI_Send (void *buf, int count, MPI_Datatype, int dest, int tag, MPI_Comm comm)

Parámetros

Parámetro	Entrada / Salida	Descripción	Tipo de Dato
Buf	Entrada	Dirección inicial del buffer de envío.	Opcional
count	Entrada	Número de elementos en el buffer de envío.	Entero positivo
datatype	Entrada	Tipo de dato de los elementos del buffer de envío.	Handle (Manejador)
Dest	Entrada	Grado del destino.	Entero
Tag	Entrada	Etiqueta del mensaje	Entero
Comm	Entrada	Handle de comunicación	Handle (Manejador)

Tabla 3.3 Parámetros de MPI_SEND:

El tipo de datos de cada elemento puede ser de los siguientes: tipos de datos C:MPI_CHAR (char), MPI_SHORT(short), MPI_INT (int), MPI_LONG (long), MPI_FLOAT (float), MPI_DOUBLE (double), MPI_UNSIGNED_CHAR (unsigned char), MPI_UNSIGNED_SHORT (unsigned short), MPI_UNSIGNED (unsigned int), MPI_UNSIGNED_LONG (unsiiigned log), MPI_UNSIGNED_DOUBLE (unsigned double (no siempre existente)).

- MPI_RECV:

Recibe un mensaje. Su sintaxis es:

#include "mpi.h"

int MPI_Recv(void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Status *status)

Parámetros

Parámetro	Entrada / Salida	Descripción	Tipo de Dato
Buf	Salida	Dirección inicial del buffer recibido.	Opcional
count	Entrada	Número de elementos en el buffer recibido.	Entero positivo
Datatype	Entrada	Tipo de dato de los elementos del buffer recibido.	Handle
Source	Entrada	Grado de la fuente.	Entero
Tag	Entrada	Etiqueta del mensaje	Entero
Comm	Entrada	Handle de comunicación	Handle
Status	Salida	Objeto de status	Status

Tabla 3.4 Parámetros de MPI_RECV

Se utilizan los mismos tipos de datos como en MPI_Send. El parámetro count indica el número máximo de elementos que pueden recibirse, el número de elementos recibidos puede determinarse con MPI_Get_count.

- 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

Todas las funciones (excepto las dos primeras) toman un manejador (handle) "comunicador" como argumento. El comunicador identifica el grupo de procesos y el contexto en el cual la operación se debe realizar. Los comunicadores proveen un mecanismo para identificar subconjuntos de procesos durante el desarrollo de programas modulares y para garantizar que los mensajes provistos con diferentes propósitos no sean confundidos. Por ahora, es suficiente proveer el valor de default MPI_COMM_WORLD, el cual identifica todos los procesos en un cálculo.

Las funciones MPI_INT y MPI_FINALIZE son utilizadas para iniciar y terminar un proceso MPI, respectivamente; MPI_INIT debe ser llamada antes de cualquier otra

función MPI y se llama solamente una vez por proceso. Ninguna función MPI puede ser llamada después de MPI_FINALIZE.

Las funciones MPI_COMM_SIZE y MPI_COMM_BANK determinan el número de procesos en el cálculo actual y el identificador (entero) asignado al proceso actual, respectivamente. Los procesos en un grupo de procesos son identificados con un único y continuo número (entero), empezando en cero.

El estándar del MPI no especifica como se inicia un cálculo paralelo. Pero un mecanismo típico podría ser el especificar desde la línea de comandos, el número de procesos a crear.

3.6.2.6 DETERMINISMO

El paso de mensajes en módulos de programación son por defecto no determinísticos; el orden de llegada de los mensajes enviados desde dos procesos A y B hacía un tercer proceso C, no esta definido. Pero, MPI garantiza que dos mensajes enviados desde un proceso A, hacía otro proceso B, llegaran en el orden en que fueron enviados.

En el modelo de programación Tarea / Canal, el determinismo es garantizado al definir canales separados para diferentes comunicaciones y al asegurar que cada canal tiene un solo escritor y un solo lector. Por lo cual, un proceso C puede distinguir mensajes recibidos de A o B tal y como llegan en tales canales. MPI no soporta canales directos pero provee mecanismos similares; en particular, permite una operación de recibimiento para especificar una fuente, tag y / o contexto.

3.6.2.7 ESPECIFICACIONES EN EL CONTEXTO DEL LENGUAJE C:

- Los nombres de las funciones son tal y como se presentan en la definición del MPI pero solo con el prefijo de MPI y la primera letra de la función en mayúsculas.

- Los valores de los estados son regresados como códigos enteros. El código de regreso para una ejecución exitosa es MPI_SUCESS.

- También esta definido un conjunto de códigos de errores.

- Las constantes están en mayúsculas y son definidas en el archivo mpi.h.

- Los manejadores (handles) son representados por tipos especialmente definidos en mpi.h.

- Los parámetros de las funciones del tipo entrada son pasados por valor, mientras que los parámetros salida y E/S son pasados por referencia (como apuntadores).

- Las variables de estado (status) tienen el tipo MPI_Status y es una estructura con campos status. MPI_SOURCE y status. MPI_TAG son ejemplo de ello.

- Los tipos de datos del MPI están definidos para cada tipo de datos de C: MPI_CHAR, MPI_INT, MPI_LONG_INT, MPI_UNSIGNED_CHAR, etc.

3.6.2.8 OPERACIONES GLOBALES

Como ya se ha explicado, los algoritmos paralelos ejecutan llamadas a operaciones para coordinar la comunicación en múltiples procesos. Por ejemplo, todos los procesos pueden necesitar cooperar para invertir una matriz distribuida o para sumar un conjunto de números distribuidos en cada proceso. Claramente, estas operaciones globales pueden ser implementadas por un programador usando las funciones send y receive por conveniencia y para permitir la optimización, MPI provee un conjunto especializado de funciones colectivas de comunicación que obtienen operaciones de este tipo.

3.6.2.9 COMUNICACIÓN ASÍNCRONA

La necesidad de tener una comunicación asíncrona puede presentarse cuando un cálculo necesita acceder los elementos de un dato estructurado compartido en una manera no estructurada. Una implementación no estructurada es el encapsular los datos estructurados en un conjunto de tareas de datos especializados, en el cual las peticiones de lectura y escritura pueden ser ejecutadas. Este método no es eficiente en MPI debido a su modelo de programación MPMD.

Una implementación alternativa con MPI, es el distribuir las estructuras de datos compartidas entre los procesos existentes, los cuales deben pedir periódicamente las solicitudes pendientes de lectura y escritura. Para esto MPI utiliza tres funciones:

- MPI_IPROBE: Verifica la existencia de mensajes pendientes sin recibirlos, permitiéndonos escribir programas que generan cálculos locales con el procesamiento de mensajes sin previo aviso. El mensaje puede ser recibido usando MPI_RECV.

- MPI_PROBE: Es utilizado para recibir mensajes de los cuales se tiene información incompleta. Primero detecta la llegada del mensaje utilizado MPI_PROBE. Después, determina la fuente del mensaje y utiliza

MPI_GET_COUNT para conocer el tamaño del mensaje. Finalmente direcciona un buffer para recibir el mensaje.

3.6.2.10 MODULARIDAD

Conocemos ya tres formas generales de composición que pueden ser usadas en la construcción modular de programas paralelos a saber, secuencial, paralelo y concurrente.

MPI soporta la programación modular a través de su mecanismo de comunicador (*comm*), el cual provee la información oculta necesaria al construir un programa modular), al permitir la especificación de componentes de un programa, los cuales encapsulan las operaciones internas de comunicación y proveen un espacio para el nombre local de los procesos.

Una operación de comunicación MPI siempre especifica un comunicador. Este identifica el grupo de procesos que están comprometidos en el proceso de comunicación y el contexto en el cual ocurre la comunicación. El grupo de procesos permite a un subconjunto de procesos el comunicarse entre ellos usando identificadores locales de procesos y el ejecutar operaciones de comunicación colectivas sin meter a otros procesos. El contexto forma parte del paquete asociado con el mensaje. Una operación *receive* puede recibir un mensaje solo si este fue enviado en el mismo contexto. Si dos rutinas usan diferentes contextos para su comunicación interna, no puede existir peligro alguno en confundir sus comunicaciones.

A continuación se describen las funciones que permiten a los comunicadores ser usados de manera más flexiblemente.

- MPI_COMM_DUP: Un programa puede crear un nuevo comunicador, conteniendo el mismo grupo de procesos pero con un nuevo contexto para asegurar que las comunicaciones generadas para diferentes propósitos no sean confundidas, este mecanismo soporta la composición paralela.

- MPI_COMM_SPLIT: Un programa puede crear un nuevo comunicador, conteniendo solo un subconjunto de grupo de procesos. Estos procesos pueden comunicarse entre ellos sin riesgo de tener conflictos con otros cómputos concurrentes. Este mecanismo soporta la composición paralela.

- MPI_INTERCOMM_CREATE: Un programa puede construir un intercomunicador, el cual enlaza procesos en dos grupos. Soporta la composición paralela.

- MPI_COMM_FREE: Esta función puede ser utilizada para liberar el comunicador creado a usar funciones anteriores.

3.6.3 PVM

Una PVM (parallel virtual machine) es una herramienta diseñada para solucionar una cantidad de problemas asociados con la programación paralela. Para ello, va a crear una nueva abstracción, que es la máquina paralela virtual, empleando los recursos computacionales libres de todas las maquinas de la red puesta a disposición de la biblioteca. Es decir, se dispone de todas las ventajas económicas asociadas a la programación distribuida, ya que se emplean los recursos hardware de dicho paradigma; pero programando el conjunto de máquinas como si se tratara de una sola maquina paralela, que es mucho mas cómodo.

La maquina paralela virtual es una maquina que no existe, pero un API apropiado permite programarla como si existiese. El modelo abstracto que permite usar el API de la PVM consiste en una maquina multiprocesador completamente escalable (es decir, que se puede aumentar y disminuir el numero de procesadores al instante). Para ello, va a ocultar la red que emplea para conectar las máquinas, y sus características específicas. Este planteamiento tiene numerosas ventajas respecto a emplear una supercomputadora, de las cuales, las más destacadas son:

- Precio. Así como es mucho mas barato una computadora paralela que la computadora tradicional equivalente, un conjunto de ordenadores de mediana o baja potencia es muchísimo mas barato que la computadora paralela de potencia equivalente. Al igual que ocurrirá con el caso de la computadora paralela, van a existir factores (fundamentalmente, la lentitud de la red frente a la velocidad del bus de la computadora paralela) que van a ser, de que sean necesarios, mas ordenadores de pequeña potencia que los teóricos para igualar el rendimiento. Sin embargo, aun teniendo esto en cuanto a la solución es mucho mas barata además, al no ser la PVM una solución que necesite de maquinas dedicadas (el daemon de PVM corre como un proceso mas), se puede emplear en el proceso, los tiempos muertos de los procesadores de todas las maquinas de la red a las que tengamos acceso. Por ello, si ya existe una red montada, el costo de tener un supercomputador paralelo va a ser cero, no abra que comprar nada nuevo, y además la biblioteca pvm es software libre, por lo que no hay que adquirir una licencia para utilizarla.

- Tolerancia a fallos. Si por cualquier razón falla uno de los ordenadores que conforman la PVM y el programa que la usa esta razonablemente bien hecho la aplicación puede seguir funcionando sin problemas. En un caso en el que la aplicación va a estar corriendo durante meses, es crítico que la aplicación sea tolerante a fallas. Siempre hay una razón por lo que alguna maquina pueda fallar, y la aplicación pueda continuar asiendo los cálculos con aquel hardware que continué disponible.

- Heterogeneidad. Se puede crear una maquina paralela virtual a partir de ordenadores de cualquier tipo. La PVM va a abstraer la topología de la red, la tecnología de la red, la cantidad de memoria de cada maquina, el tipo de procesador y la forma de almacenar los datos. Este último punto es de extrema importancia, ya que el principal problema en los sockets era la programación de rutinas de conversión de formatos de datos entre todos los ordenadores de la red, debido a que la codificación tanto de enteros y de flotantes puede ser distinta. Por ultimo, permite incluir en la PVM hasta maquinas paralelas. Una máquina paralela en una PVM se puede comportar tanto como una máquina secuencial (Caso, por ejemplo del soporte SMP de Linux) o, como ocurren en muchas máquinas paralelas, presentarse a la PVM como un conjunto de máquinas secuenciales.

El uso de la PVM tiene una gran desventaja el olvidar el paralelismo fuertemente acoplado. En una red ethernet, la red deja de funcionar para todas las aplicaciones (incluida PVM) por la cantidad de colisiones que se producen al intentar paralelismo fuertemente acoplado. En una red de tecnología más avanzada; es decir, más cara (como ATM) el problema es menor, pero sigue existiendo.

La segunda desventaja es que la abstracción de la máquina virtual, la independencia del hardware y la independencia de la codificación tienen un coste. La PVM no va a ser tan rápida como son los sockets. Sin embargo, si el grado de acoplamiento se mantiene lo suficientemente bajo, no es observable esta diferencia.

La arquitectura de la PVM se compone de dos partes. La primera parte es el daemon, llamado pvmd. En la versión tres de PVM, el nombre es pvmd3. El daemon funcionará en todas las máquinas que vayan a compartir sus recursos computacionales con la máquina paralela virtual. A diferencia de otros usuarios en su directorio particular, de hecho, la instalación por omisión es así. Esto permite hacer superconmutación como usuarios, sin tener que discutir con la administración de la red que programas pueden ejecutarse. Una vez que un usuario (o súper usuario) instaló en un directorio la PVM, todos los usuarios pueden hacer uso de esa instalación con el requisito de que el directorio donde este instalada la PVM sea de lectura al usuario que quiera hacer uso de ella.

El daemon pvmd3 es el responsable de la máquina virtual de por si, es decir, de que se ejecuten los programas para la PVM y de coordinar los mecanismos de comunicación entre máquinas, la conversión automática de datos y de ocultar la red al programador. Por ello, una vez que la PVM este en marcha, el paralelismo es independiente de la arquitectura de la máquina, y solo depende de la arquitectura de la máquina virtual creada por la PVM.

La segunda parte es la biblioteca de desarrollo. Contiene las rutinas para operar con los procesos, transmitir mensajes entre los procesadores y alterar las propiedades de la máquina virtual. Toda aplicación se ha de enlazar a la biblioteca para poderse ejecutar

de una sola vez. Tendremos tres ficheros de la biblioteca, la libpvm3.a (biblioteca básica en c), libgpvm3.a (biblioteca de tratamiento de grupos) y la libfpvm3.a (biblioteca para fortran).

Un programa para PVM va a ser un conjunto de tareas que cooperan entre si. Las tareas se van a intercambiar en información empleando paso de mensajes. La PVM, es de forma transparente al programador, en la que va a ocultar las transformaciones de tipos asociadas al paso de mensajes entre máquinas heterogéneas. Toda tarea de la PVM puede incluir o eliminar máquinas, arrancar o parar otras tareas, mandar datos a otras tareas o sincronizarse con ellas.

Cada tarea en la PVM tiene un número que la identifica particularmente, denominado TID (task identification number). Es el número al que se mandan los mensajes habitualmente. Sin embargo, no es el único método de referenciar una tarea en la PVM. Muchas aplicaciones paralelas necesitan hacer el mismo conjunto de acciones sobre un conjunto de tareas. Por ello, la PVM incluye una abstracción nueva, el grupo. Un grupo es un conjunto de tareas a las que se refiere con el mismo código, el identificador de grupo. Para que una tarea entre o salga de un grupo, basta con avisar de la salida o entrada al mismo. Esto dota de un mecanismo muy cómodo y potente para realizar programas empleando modelos SIMD (single instruction, multiple data), en el que vamos a dividir los datos en muchos datos pequeños que sean fáciles de tratar, y después se codifica la operación simple y replica tantas veces como datos unitarios obtenidos de dividir el problema. Para trabajar con un grupo, además de enlazar la biblioteca de la PVM libpvm3.a, se enlaza también la de grupos libgpvm3.a.

Habitualmente para arrancar un programa para la PVM, se enlazará manualmente desde un ordenador contenido en el conjunto de máquinas, una tarea madre. La tarea se lanzará con el comando spawn desde un monitor de la máquina virtual, que a su vez se activará con el comando PVM. Esta tarea se encargará de iniciar todas las demás tareas, bien desde su función main.

Para evitar la incomodidad de andar realizando transformaciones continuas de datos, la PVM define clases de arquitecturas. Antes de mandar un dato a otra máquina comprueba su clase de arquitectura. Si es la misma, no necesita convertir los datos, con lo que se tiene un gran incremento en el rendimiento. En caso en que sean distintas las clases de arquitectura, se emplea el protocolo XDR (protocolo de presentación de datos) para codificar el mensaje.

Las clases de arquitectura están mapeadas en números de codificación de datos, que son los que realmente se transmiten y, por lo tanto, los que realmente determinan la necesidad de la conversión.

El modelo de paso de mensajes es transparente a la arquitectura para el programador, por la comprobación de las clases de arquitectura y la posterior codificación con XDR de no coincidir las arquitecturas. Los mensajes son etiquetados al

ser enviados con un número entero definido por el usuario, y pueden ser seleccionados por el receptor tanto por dirección de origen como por el valor de la etiqueta.

El envío de mensajes es no bloqueante. Esto quiere decir que el que envía el mensaje no tiene que esperar a que el mensaje llegue, sino que solamente espera a que el mensaje sea puesto en la cola de mensajes. La cola de mensajes asegura que los mensajes de una misma tarea llegaran en orden entre sí. Esto no es trivial, ya que empleando el protocolo UDP (protocolo de datagrama de usuario) pueden enviarse dos mensajes y que lleguen fuera de orden (UDP es un protocolo no orientado a conexión).

La comunicación de las tareas con el daemon se hace empleando TCP. Esto se debe a que, a ser comunicaciones locales, la carga derivada de la apertura y cierre de un canal es muy pequeña. A demás, hay tantas conexiones como en el caso de la conexión entre daemons, ya que las tareas no se conectan entre sí ni con nada fuera del nodo, por lo que solo hablan directamente con su daemon. Esto determina que serán solo conexiones TCP, lo que es una cifra razonable.

La recepción de los mensajes pueden hacerse mediante primitivas bloqueantes, no bloqueantes o con un tiempo máximo de espera. La PVM dotará de primitivas para realizar los tres tipos de recepción. En principio serán más cómodas las bloqueantes, ya que dan un mecanismo de sincronización bastante cómodo. Las de tiempo máximo de espera serán útiles para trabajar con ellas como si fuesen bloqueantes. Estas últimas dan el soporte al hecho de que pueden mandarnos el mensaje si se quedara colgado el proceso. Por último, la recepción de mensajes mediante primitivas no bloqueantes, hace de la sincronización un dolor de cabeza. De cualquier forma, en los tres casos anteriores citados de la misma PVM se encargará de avisar cuando finalizó una tarea. Para informar lo que pasa, emplea un mecanismo de eventos asíncronos.

La PVM puede ser empleada de forma nativa, con funciones en c y en c++, y como procedimientos en fortran. Basta para ello tomar las cabeceras necesarias (si trabajamos en c o en c++) para que los tres puedan enlazar con la biblioteca adecuada, que viene con la distribución estándar. En el caso de c es libpvm3.a y en el de fortran libfpvm3.a.

CAPÍTULO 4

APLICACIÓN

Durante el desarrollo de este trabajo de Tesis se analizaron múltiples casos de cómo implementar el cluster SEPI (CSEPI). En el capítulo 3 se explicó, la importancia de escoger el sistema operativo adecuado en el que Fedora Core 6 a 64 bits puede considerarse la mejor opción por el tipo de aplicación que vamos a desarrollar, ya que esta distribución está orientada a redes y nos facilita el evitar descargar algunas paqueterías por el simple hecho de que ya las trae incluidas al momento de realizar la instalación.

De esta manera se ahorra tiempo. En este capítulo se presentan los planteamientos de los casos más relevantes, partiendo de las configuraciones básicas como son lo referente a la red y los compiladores hasta llegar a la aplicación paralela en el CSEPI. Así mismo se realiza un análisis de los resultados.

Las partes involucradas para el diseño y la implementación del CSEPI se agruparon según la forma en como se fue desarrollando. Lo primero que se consideró en este proyecto, fue las características generales del hardware, lo que se explica a continuación:

4.1 CARACTERÍSTICAS DE CSEPI

El CSEPI fue planeado con equipo de cómputo mucho más robusto del que conocemos hoy en día. Dicho hardware lo proporcionó SEPI y son dos servidores xeon quad core de Intel con las siguientes características:

- Procesador intel xeon E5405@2.00Ghz
- Memoria RAM 7.99 GB
- Disco duro de 480 GB

Una vez que ya se tiene presente las características principales del equipo podemos observar que no tenemos limitaciones en cuanto a memoria ya que cuenta con los requisitos necesarios para la instalación de cualquier sistema operativo.

Figura 4.1: Cluster SEPI (CSEPI)

4.2 IMPLEMENTACIÓN:

Dentro de la implementación debe considerarse

- Configuración de la red
- Pruebas de funcionalidad de la red
- Programación paralela

4.2.1 CONFIGURACIÓN DE LA RED EN FEDORA

Como se puede apreciar los dos equipos son idénticos, por lo que este tipo de cluster será homogéneo a nivel hardware como a nivel software. A nivel software porque cada nodo tiene instalado la misma paquetería incluyendo la distribución del sistema operativo.

Debido a que se piensa en un futuro aumentar el número de nodos, se adquirió un Switch LYNKSYS para la conexión en red. La instalación del software se llevo a cabo como se mostró en el capítulo anterior (capítulo III).

Figura 4.2 Switch Linksys

Es importante remarcar que la instalación debe ser idéntica en cada nodo, no importando que sea esclavo o maestro. Con idéntico debe entenderse que se instalen los mismos paquetes.

Los paquetes instalados en cada nodo son:

- linux fedora core 6 para procesador de 64 bits
- Compilador Fortran 64 bits para linux
- Compilador Intel C++ 64 bits para linux
- MPI lam
- Compilador GCC versión 4.2

Únicamente en el nodo maestro es conveniente el arranque en modo gráfico, los demás nodos arrancarán en modo texto para evitar utilización innecesaria de memoria RAM. Aunque en este caso, no se vería afectado al rendimiento por el hecho de que las características de este equipo soporta este tipo de arranque.

En las distribuciones Fedora tuvimos que configurar las interfaces de red manualmente editando los ficheros de configuración en modo superusuario:

Para la red ethernet, la interfaz eth0, los ficheros que necesitamos modificar a través de la siguiente ruta:

/etc/sysconfig/network-scripts/ifcfg-eth0 y /etc/sysconfig/networking/devices/ifcfg-eth0

Para configurar el interfaz de red y que su contenido sea idéntico a lo siguiente. Cabe señalar que esta configuración es para el nodo I o servidor.

```
DEVICE=eth0
USERCTL=no
ONBOOT=yes
BOOTPROTO=none
NETMASK=255.255.255.0
IPADDR= 148.204.31.193
IPXPRIMARY_802_2="no"
IPXACTIVE_802_2="no"
IPXPRIMARY_802_3="no"
IPXACTIVE_802_3="no"
IPXPRIMARY_ETHERII="no"
IPXACTIVE_ETHERII="no"
IPXPRIMARY_SNAP="no"
IPXACTIVE_SNAP="no"
BROADCAST=148.204.31.255
NETWORK=148.204.31.0
PEERDNS=no
TYPE=Ethernet
GATEWAY= 148.204.31.254
IPV6INIT=no
```

El siguiente fichero **/etc/sysconfig/network** se utiliza para configurar las características generales de la red:

```
NETWORKING=yes
# FORWARD_IPV4 removed; see /etc/sysctl.conf
HOSTNAME=localhost.localdomain
DOMAINNAME=localhost.localdomain
GATEWAY="148.204.31.254"
GATEWAYDEV="eth0"
IPX="no"
IPXINTERNALNETNUM="0"
IPXINTERNALNODENUM="0"
IPXAUTOPRIMARY="off"
IPXAUTOFRAME="off"
FORWARD_IPV4="yes"
```

La configuración de red para el nodo II o cliente se realiza de manera idéntica cambiando la dirección IP, la cual es 148.204.31.194, en los campos que se requiere, el GATEWAY el BROADCAST y la NETMASK se dejan igual que como se mostró anteriormente. El nombre de los nodos y su dirección IP quedo como se muestra en la siguiente tabla 4.

Nodo	Nombre	Dirección IP
I	Servidor	148.204.31.193
II	Cliente	148.204.31.194

Tabla 4.1 Dirección IP de nodos

Para la configuración de manera gráfica se sigue la siguiente ruta:

Sistema/Administración/Red

En donde nos pide la clave de root para poder acceder con los permisos correspondientes, aquí primero vamos a activar el dispositivo ETH0 que con el que vamos a realizar la conexión ethernet. Esta operación se muestra en la siguiente figura:

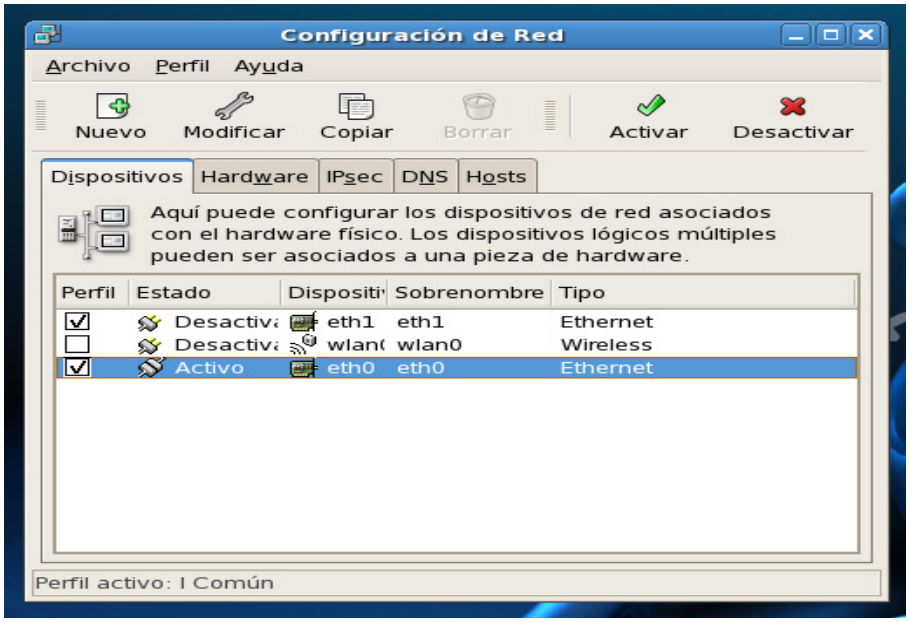


Figura 4.3 Pantalla de configuración de red

Una vez activado el dispositivo, en la opción de modificar le ingresamos los campos correspondientes:

- **Dirección: 148.204.31.193**
- **Máscara: 255.255.255.0**
- **Dirección de la puerta de enlace: 148.204.31.254**

Los datos que se citaron son del nodo I o servidor, estos datos cambiaron para el nodo II o cliente, solo en la dirección la cual es la siguiente:

Dirección: 148.204.31.194

La máscara y la dirección de la puerta de enlace son los mismos. Esta operación se muestra en la siguiente figura:

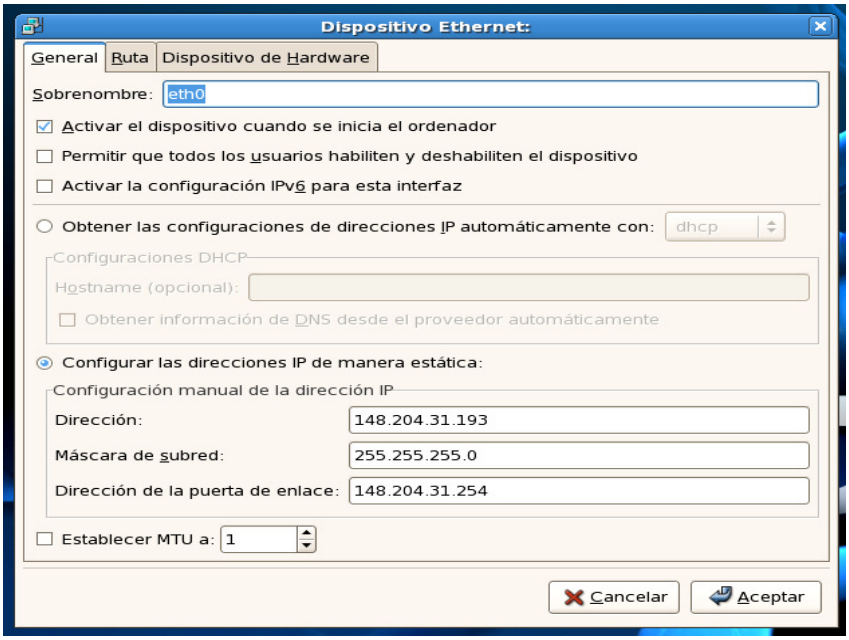


Figura 4.4 Pantalla de dispositivo ethernet

Una vez realizado lo anterior tenemos que configurar la parte DNS, en este fichero podemos modificar el nombre del host, al igual que las direcciones de los servidores de red del cluster en SEPI las cuales son:

- **DNS primario: 148.204.103.2**
- **DNS secundario: 148.204.102.3**

Estas direcciones son las mismas que se utilizan al configurar el nodo II, Esta operación se muestra en la siguiente figura 4.4:

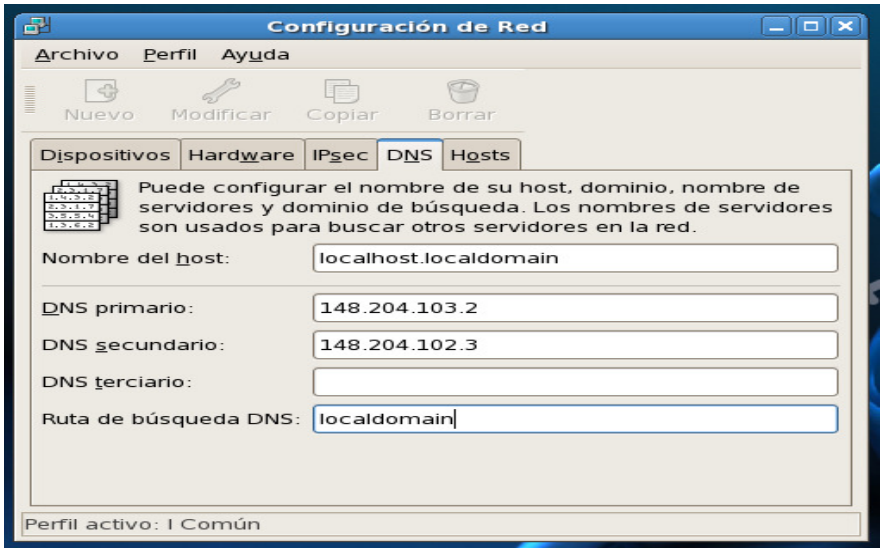


Figura 4.5 Pantalla de configuración del DNS

Enseguida pasamos a configurar la siguiente pestaña que es la del host, en la cual vamos a configurar la dirección IP, el nombre del host y el alias que va llevar el equipo. Aquí solo cambiaría según la tabla de direcciones IP del servidor con respecto a la del nodo II. La siguiente operación se muestra en la siguiente figura 4.6:

Figura 4.6 Configuración del host

4.2.2 PRUEBAS DE FUNCIONALIDAD DE LA RED

Lo primero que se realizó es comprobar que realmente existiera comunicación entre los dos nodos. Fue una prueba de funcionalidad para corroborar que el servidor tuviera acceso al otro nodo. Para ello se tuvo que realizar unas pruebas desde la consola.

1) En el menú principal seguimos la siguiente ruta:

   Aplicaciones/Accesorios/Terminal

   Esta ventana es la Terminal o también conocida como consola, en donde se van a introducir cada uno de los comandos.

2) Una vez que ya tenemos abierta la consola introduciremos un pwd para conocer la ubicación actual y por ende vamos a esta en /root. En el caso que no se esté como administrador, es el directorio home.

3) Vamos a mandar un ping para comenzar con la conexión en el que vamos a teclear lo siguiente: ***ping 148.204.31.194.***

   Si la prueba es correcta, nos va aparecer una pantalla como la que a continuación se muestra:

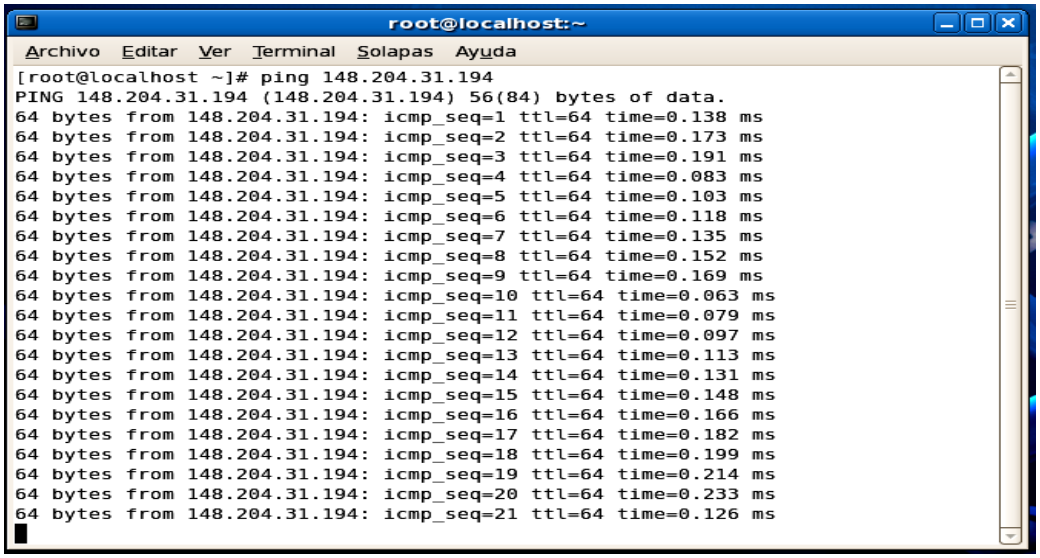


Figura 4.7 Utilización del ping en una terminal

4) Vamos a intentar logearnos al otro nodo. Existen muchas formas de realizarlo, nosotros utilizaremos un ssh (programa para logearse remotamente). Es un programa para logearse con otra máquina vía remota y para ejecutar comandos en una máquina remota. Lo único que necesitamos hacer es teclear lo siguiente:

ssh 148.204.31.194

Nos aparecerá lo siguiente pantalla:

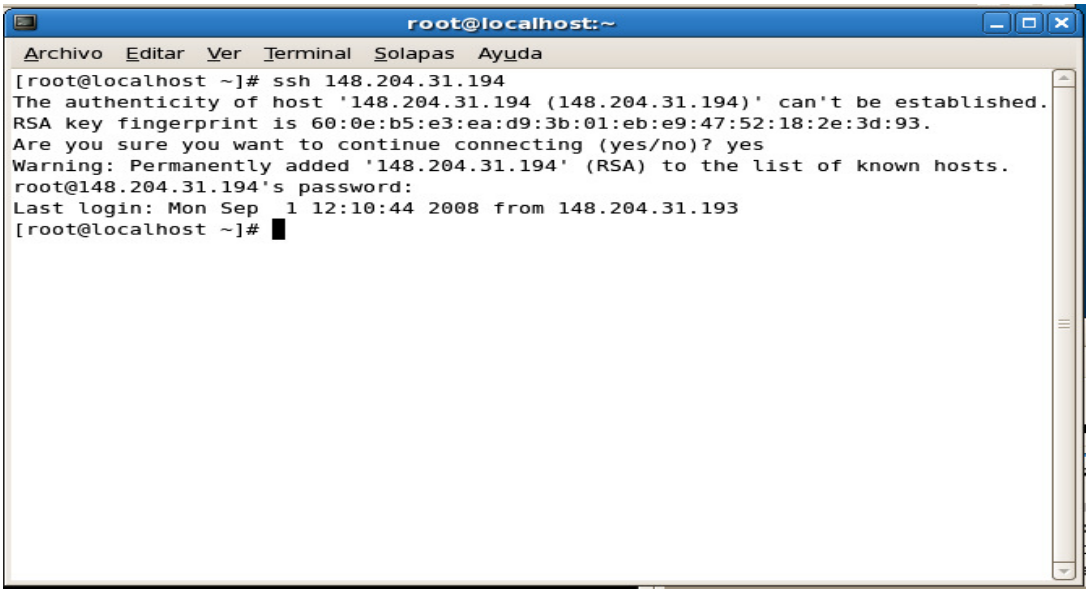


Figura 4.8 Conexión remota mediante el comando ssh

4.2.3 PROGRAMACIÓN PARALELA EN C

4.2.3.1 PROGRAMA HOLA MUNDO SERIAL

La primera prueba que se realizó en el compilador fué el programa “hola mundo.c” de forma serial para asegurarse que los compiladores esten funcionando correctamente. El código de dicho programa se muestra a continuación

```
#include <stdio.h>

#include <omp.h>

int main() {

                int i;

                printf("Hola Mundo\n");

                for(i=0;i<10;i++){

                printf("Iter:%d\n",i);

        printf("%d \n",omp_get_thread_num());  //obtiene el id del thread actual

}

                printf("Adiós Mundo\n");

}
```

Se compila y se ejecuta tecleando lo siguiente en la terminal:

```
[root@localhost bin]# gcc his.c -fopenmp -o his

[root@localhost bin]# ./his
```

Hola mundo

Iter:0

0

Iter:1

0

Iter:2

0

Iter:3

0

Iter:4

0

Iter:5

0

Iter:6

0

Iter:7

0

Iter:8

0

Iter:9

0

Adiós Mundo

Lo primero que tenemos que observar al correr este programa es que es un programa que se va a correr en forma serial pero con un pequeño detalle, que en las librerías ya se incluye a openmp debido a que la función omp_get_thread_num ya es una función de está librería la cual se encarga de obtener el número de hilos. Se considera un número de iteraciones de 10 en el cuál primeramente imprime en pantalla Hola Mundo y comienza un ciclo en donde va ir mostrando las iteraciones y despliega el número de hilo. Como se está ejecutando de forma serial podemos observar que sólo va a aparecer un número de hilo que en este caso es cero debido a que de forma serial se considera que un procesador toma por defecto un hilo.

4.2.3.2 HOLA MUNDO PARALELO

Para paralelizar este programa es necesario tener en mente dos cosas. Primeramente en la estructura de un programa paralelo con OpenMP se debe considerar la librería omp.h. Segundo, no se debe confundir una función propia de OpenMP con una directiva o pragma. La función es propia de la librería. La directiva no requiere librería. El siguiente codigo va a mostrar el número de hilos que se tiene durante proceso. Recordar

que un proceso puede contener muchos hilos. Dichos hilos para este caso, serán el número de procesadores con que se cuenta.

```
#include <stdio.h>

#include <unistd.h>

#include <omp.h>

main()

{

#pragma omp parallel

  {

  {

  char name[128];

  int nthreads = omp_get_num_threads();

  printf("Número de hilo: %d\n",nthreads);

  int mythread = omp_get_thread_num();  //obtiene el id del thread actual

  int proc = kmp_get_process_num();  //Rn the process number of the current p.

  int  nprocs = kmp_get_num_processes();  //Rn the number of ps involved in the
computation.

  int  procthread = kmp_get_process_thread_num();//Rn the thread number of the
current thread with respect to the current p.

          //gethostname(name, sizeof(name));

          printf("Hola de hilo %d en el host.\n",

             mythread);

          printf("\tproceso #: %d, hilo del proceso #: %d, # proceso: %d\n",

                   proc, procthread, nprocs);

    }

  }
```

}

Este programa calcula el número de hilos en un proceso. El análisis del siguiente programa lo explicaremos a través de algunas figuras. La figura 4.9 nos ayudará a comprender lo que pasa antes y después del main:

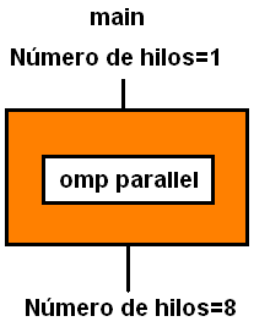


Figura 4.9 Descripción del Pragma OMP para el cálculo de PI

En la figura podemos observar que enseguida de la función principal (main), tenemos 1 sólo hilo, debido a que hasta ese punto no se ha paralelizado nada. Una vez que entra la directiva omp parallel se observa claramente que ha paralelizado, buscando el número de hilos y mostrando 8.

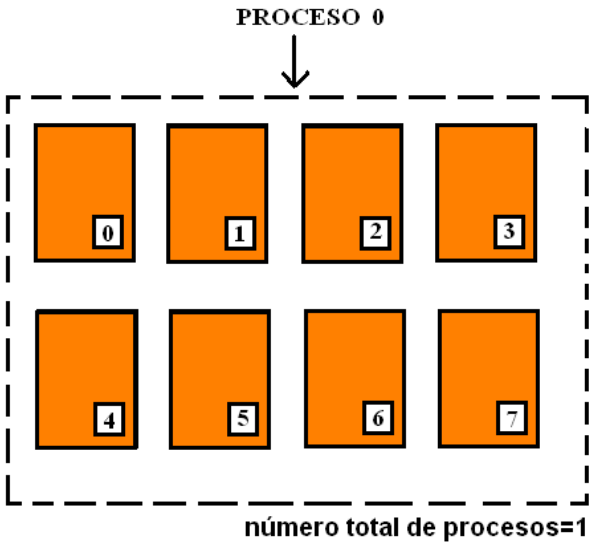


Figura 4.10 Número de hilos dentro de un proceso

El grupo de 8 hilos va a pertenecer a un mismo proceso denominado proceso 0. Aquí es donde se va presentar el primer problema. En el capítulo número tres partiendo del modelo del bifurcación-unión, cuando es ejecutada la directiva en los 8 hilos se

habilitan, y de acuerdo al modelo al finalizar cada uno se une. ¿Podríamos preguntarnos que hilo acabará primero? ¿Cómo puedo mantener un orden?

Dependiendo de la tarea de cada hilo, estos se pueden adelantar como se muestra el resultado y además puede observarse que en las líneas que están con negritas y con letra más grande, los hilos intentan adelantarse, de tal manera que no permiten que se despliegue toda la información de cada hilo.

[root@localhost bin]# icc hello.c -openmp -o hello

hello.c(11): (col. 1) remark: OpenMP DEFINED REGION WAS PARALLELIZED.

[root@localhost bin]# ./hello

Número de hilos: 8

Hola del hilo 0 en el host.

proceso #: 0,hilo del proceso #: 0, # procesos: 1

Número de hilos: 8

Hola del hilo 1 en el host.

proceso #: 0, hilo del proceso #: 1, # procesos: 1

Número de hilos: 8

Hola del hilo 4 en el host.

process #: 0, process thread #: 4, # procesos: 1

Número de hilos: 8

Hola del hilo 3 en el host.

proceso: 0, hilo del proceso #: 3, # procesos: 1

Número de hilos: 8

Número de hilos: 8

Hola del hilo 6 en el host.

proceso #: 0, hilo del proceso #: 6, # procesos: 1

Número de hilos: 8

Hola del hilo 5 en el host.

process #: 0, process thread #: 5, # procesos: 1

Hola del hilo 7 en el host.

proceso #: 0, hilo del proceso #: 7, # procesos: 1

Número de hilos: 8

Hola del hilo 2 on host.

proceso #: 0, hilo del proceso #: 2, # procesos: 1

Para que no pase esto, es necesario utilizar una directiva más, llamada:

#pragma omp critical // Define una región crítica en un bloque estructurado

Esta directiva tiene como función principal que los hilos esperen su turno –en un momento, solo uno llama a proceso de tal manera que no permite que el otro hilo entre antes de que el hilo anterior termine de mostrar toda la información.

De esta forma aseguramos que ningún hilo se adelante.

4.2.3.3 OBTENCIÓN DE π

Con el cálculo de π, se pretende entender la aceleración o factor de velocidad. Que tanto puede mejorar el tiempo de ejecución a medida que se incrementa el número de hilos, considerando que cada núcleo contiene 2 hilos. Para ello partimos del código para el cálculo de π. Como lo muestra el siguiente código de forma serial:

```
#include <stdio.h> // Declaración de

#include <time.h> // Librerias

long long num_steps = 1555555555; //Declaración de forma global de num_steps

double step; //Declaración de forma global de step

int main(int argc, char* argv) //Función principal (main) con argc entero

                                //y argv puntero de tipo caracter

{

time_t t1,t2; // tipo aritmético capaz de representar el tiempo en t1 y t2

time(&t1);// retorna la mejor aproximación por la implementación del tiempo actual en
```

```
                // formato condensado al puntero t1
printf("Timestamp de inicio de ejecucion:  %s\n",ctime(&t1));//impresión del tiempo t1
                                                        //en un formato separado
                                                        //como argumento.
double x=0.0, pi=0.0, sum=0.0; //Declaración e inicialización de las variables x, pi,
                                        //sum de tipo double
int i;//Declaracion de I de tipo entero
step = 1.0/(double)num_steps; // el inverso del contenido de num_steps asignado
                                        // a la variable step
for (i=0; i<num_steps; i++)// ciclo for(inicialización de i; mientras que i sea menor
                                        //a num_steps; incremento de i)
        {
x = (i + .5)*step; //Asignación a x de la multiplicación de (la suma de i + .5)
                        // con la variable step
sum = sum + 4.0/(1.+ x*x);//Acumulación de la división de 4 / (1+x2)
        }
pi = sum*step; //Se le asigna la aproximacion del verdadero valor de PI
time(&t2);// retorna la mejor aproximación por la implementación del tiempo actual
                //en formato condensado al puntero t2
   printf("Timestamp de final de ejecucion:  %s\n",ctime(&t2));// impresión del tiempo
                                                        //t2 en un formato
                                                //separado como argumento.
printf("El valor Calculado de PI es = %15.12f\n",pi);//impresión de la aproximación
                                                        //de pi
return 0;//regresa el valor de cero.
```

}

Al ejecutar el programa muestra los siguientes resultados de forma serial:

Timestamp de inicio de ejecución: Thu Oct 30 **18:29:33** 2008
Timestamp de final de ejecución: Thu Oct 30 **18:30:12** 2008
El valor de PI es de **3.141592653589**

Tabla 4.2 Resultados del análisis del número π

Claramente se puede observar el tiempo de inicio de ejecución y el final del mismo. Lo que hace el siguiente programa va a ser el cálculo de pi mostrando el tiempo en el que inicia la ejecución y el tiempo en el que lo termina. Como mencionamos en la Ley de Amhdal, en el capítulo 1, en un programa secuencial solo hay cierta parte del programa que se puede paralelizar debido a que solo ciertas partes de programa en las que se puede realizar. Además observar claramente que como vimos en el capítulo 3, se deben tomar en cuenta que para paralelizar el código de integración numérica usando OpenMP se deben considerar que directivas serían convenientes utilizar, así como algunas cláusulas que me van a permitir tener un mejor control sobre los hilos.

Saltarían inmediatamente las siguientes tres preguntas: ¿Qué variables se pueden compartir? ¿Qué variables deben ser privadas?

La solución está en la siguiente cláusula:

REDUCTION (list) la cual como vimos en el capítulo número 3 realiza una reducción a las variables que se encuentran en la lista. Las variables que se pueden reducir son aquellas que están dentro de funciones. Todas aquellas que sean variables globales no se pueden paralelizar debido a que son partes del main.

La parte de código a paralelizar, utilizando la directiva de OPENMP, es el ciclo for del código anterior. Incluyendo una directiva más una cláusula permitirá distribuir toda la operación en los 8 hilos.

```
#pragma omp parallel for private(x) reduction(+:sum)//Directiva de OpenMp para
//realizarla en varios hilos (paralelizar FOR)

for (i=0; i<num_steps; i++)

        {

x = (i + .5)*step; //Asignación a x de la multiplicación de (la suma de i + .5) con la
//variable step

sum = sum + 4.0/(1.+ x*x); //Acumulación de la división de 4 / (1+x2)
```

El cálculo de pi se realiza para 1555555555 iteraciones y se obtiene lo siguiente:

Timestamp de inicio de ejecución: Thu Sep 30 **18:31:55** 2008
Timestamp de final de ejecución: Thu Sep 30 **18:32:00** 2008
El valor calculado de PI es = **3.141592653589**

Tabla 4.3 Resultados del análisis de π a 1555555555 iteraciones

El inconveniente de este programa es que el tiempo nos no lo puede dar en milisegundos. Debido a que existe un problema con la función de tiempo. Nos manda un error, en el sentido que imprime mal el tiempo. El programa serial utilizando esta función que permite mostrar el tiempo por segundo es de la siguiente forma:

```
//Este programa calcula pi en forma serial

#include <stdio.h>

#include <time.h>

long long num_steps = 1555555555;

double step;

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

{

        clock_t start, stop;

        double x, pi, sum=0.0;

        int i;

        step = 1./(double)num_steps;

        start = clock();

        for (i=0; i<num_steps; i++)

        {

                x = (i + .5)*step;

                sum = sum + 4.0/(1.+ x*x);

        }
```

```
        pi = sum*step;

        stop = clock();

    printf("start: %f\n",(double)start);

    printf("stop:  %f\n",(double)stop);

        printf("El valor de PI es %15.12f\n",pi);

        printf("El tiempo para el cálculo de  PI fue de %f seconds\n",(double)(stop -
start)/CLOCKS_PER_SEC);

        return 0;

}
```

De esta manera de compilar y ejecutar los programas, se obtiene los siguientes resultados:

stop: 39630000.000000
El valor de PI es **3.141592653590**
El tiempo para el cálculo de PI fue de **39.630000 seconds**

Tabla 4.4 Resultados de la ejecución de π en forma serial.

Como podemos apreciar, en este caso, el tiempo nos lo da de forma exacta en milisegundos sin necesidad de realizar una comparación. Este criterio no puede ser el mismo para el caso del programa en paralelo debido a que no se tiene control sobre los hilos. Para tener control sobre los hilos se necesitan particionar procesos cosa que no es el propósito de este tema.

4.3 TIEMPO SERIAL vs. TIEMPO PARALELO

Una vez realizado el análisis del programa en paralelo así como sus resultados, es conveniente realizar una comparación entre los tiempos de ejecución de ambos códigos, es decir, observar mediante el monitor del sistema de fedora core 6 el tiempo en el que se está ejecutando. En la siguiente figura se ejecuta primero de forma paralela y enseguida de forma paralela. La idea es comparar los tiempos tanto en la ejecución paralela como en la ejecución de forma serial.

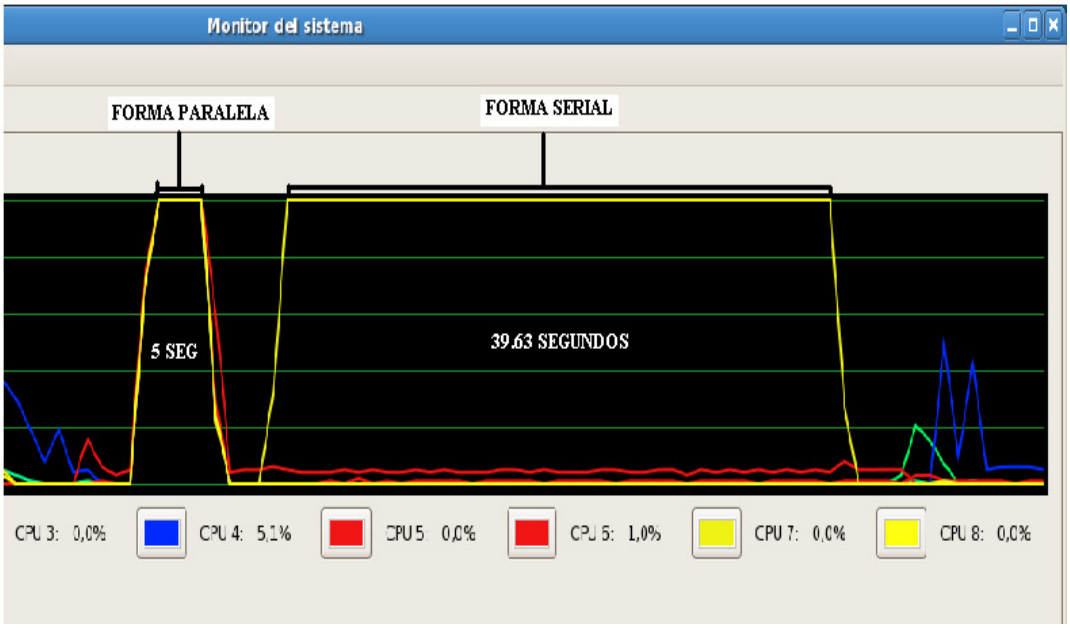


Figura: 4.11 Tiempo serial vs. Tiempo paralelo

En el capítulo 1 se habló de El Factor de Velocidad o también conocido como aceleración simple. La aceleración nos permite conocer en que cantidad se mejora el procesamiento de información al aumentar el número de núcleos. Obteniendo la aceleración simple para los tiempos de la ejecución del programa pi tanto paralelo como serial, obtenemos lo siguiente.

$$S(n) = \frac{t_s}{t_p} = \frac{39.63}{5} = 7.926X$$

Con esto se observa que la velocidad con que se realiza el cálculo de π es 8 veces más rápido si se hace con 8 hilos que si se hiciera con uno solo. En otras palabras

$$7.926 \approx 8 veces\_más\_rápido$$

Además se puede observar claramente que inmediatamente que se ejecuta el programa en forma paralelo cada uno de los procesadores se enfocan en un mismo proceso a diferencia de la forma serial que están efectuando cada procesador cierta tarea.

4.4 EJECUCIÓN DEL PROGRAMA π VÍA REMOTA

Por último, lo que se pretende, a través de la red, es ejecutar los mismos programas pero vía remota, es decir, vamos a ejecutar el mismo programa desde el otro nodo de manera que se pueda ejecutar:

Lo realizamos como vimos al realizar la comprobación de la conexión de la red entre los dos nodos. Esto se hace de la siguiente manera:

Abrimos una terminal y vamos a ejecutar lo siguiente, que ya hemos visto en ambos nodos.

ping 148.204.31.194.

ssh 148.204.31.194

Una vez que se pudo logearse entre ambos nodos, vamos a ejecutar desde el nodo II el programa π en forma paralela. Los resultados deben ser casi los mismos, con una pequeña diferencia justificada por el paso de mensajes. Lo interesante de poder ejecutar remotamente este programa, es que vamos a utilizar los recursos del nodo I. Para comprobar esto, se puede abrir en ambos nodos, el monitor del sistema en donde se puede visualizar los procesos. Se observará que en el caso del nodo II no se tendrá registro de procesos importantes, mientras que en el nodo I se observará el proceso del programa que calcula el número Π.

CAPÍTULO 5

CONCLUSIONES Y PERSPECTIVAS

5.1 CONCLUSIONES

Se realizaron las instalaciones de paquetes necesarios, así como las pruebas para el diseño de una plataforma en base a la consideración de algunos diseños que ya existen en ciertas arquitecturas, pero los cuales debieron ser optimizados a la arquitectura de procesador con el que contábamos. Esto es muy diferente a lo que ya se tiene hecho debido a que los procesadores de 64 bits apenas están siendo introducidos en el mercado. Es por ello que la determinación del sistema operativo más viable a usar, instalación de paquetes, tipo de red a utilizar y aplicación a 64 bits se necesito evaluar para este proyecto.

Podemos decir básicamente que las arquitecturas de computadoras que hoy en día existen dejan mucho que desear, optando por realizar algunas modificaciones para poder obtener un mejor desempeño.

La primera conclusión importante en el uso de procesadores de 64 bits, es que no quiere decir que sea mucho más rápido un procesador de 64 bits o el doble de rápido que tener un procesador de 32 bits. La diferencia principal que se puede observar, es que sí se tiene un aumento de velocidad. Pero, la principal ventaja se obtiene en el hecho de que por tener un procesamiento de palabra de 64 bits cuando los procesos son muy grandes en número y en capacidad, se puede apreciar que los procesadores de 64 bits pueden seguir trabajando con muchas aplicaciones o con una aplicación muy demandante; mientras que el procesamiento de palabra de 32 bits simplemente no pueden realizar la aplicación. Aunque un mismo proceso pequeño podría ser ejecutado en ambos procesadores en el mismo tiempo.

Muchas veces como usuarios no sabemos que es lo que compramos. Y nos referimos a que a través del uso de multicore pudimos observar que no nos sirve de mucho tener 1 procesador con 4 núcleos, si las aplicaciones que usualmente utilizamos son ejecutadas de forma serial. Son muy pocas las aplicaciones que utilizan el paralelismo. Ejemplo de estas podrían ser algunos videojuegos y cualquier tipo de aplicación virtual por lo que sería innecesario tener varios núcleos. Lo único que estaríamos haciendo en el caso de que no se tengan aplicaciones paralelas es pagar por una tecnología que no estamos usando.

Se muestra en este proyecto el objetivo de una solución que resuelve el paradigma del cómputo paralelo por medio de hardware con características especiales, que al agruparse cada uno de los núcleos, se puede maximizar su rendimiento considerablemente, ofreciendo capacidades de procesamiento de alto rendimiento que no podría brindar de manera individual debido a sus limitaciones desde su diseño, ya que son equipos de cómputo que no están diseñado propiamente al procesamiento de alto rendimiento.

Otra característica que se resalta en esta solución es su modularidad, es decir se pueden agregar nodos o equipos de diferentes arquitecturas sin que esto decremente su rendimiento, sino por el contrario la hace mucho más poderosa en su rendimiento y capacidades de procesamiento; y no lo limita o la hace exclusivo en cuanto a la configuración del hardware a agregar; de este modo, es un arreglo flexible y económico que aprovecha tecnologías tanto actuales como de generaciones anteriores en su composición física.

Al no contarse con una aplicación específica para el clúster las pruebas a realizarse se limitaron únicamente a verificar su funcionamiento y comunicación entre los nodos que lo conforman.

Estas pruebas arrojaron resultados positivos por lo cual la hipótesis de crear una plataforma dedicada funcional para el cómputo paralelo a partir de equipo con capacidad de procesamiento de alto desempeño (Quad Core), utilizando software libre es correcta debido a que se obtuvieron mejoras en el tiempo de ejecución de algunos programas paralelos. Mejoras que son muy considerables si trata de cálculos muy

grandes, en los que se requiera quizás cálculos de matrices muy grandes. No cabe duda que si

Esto permite entregar un equipo que sirva como herramienta a la línea de investigación "Cómputo paralelo aplicado al análisis electromagnético utilizando el método de diferencias finitas en el dominio del tiempo"

5.2 CONSIDERACIONES DE LA IMPLEMENTACIÓN DEL CLUSTER

Para realizar particiones e instalación de programas se necesitan diferentes métodos para realizarlo. Principalmente la instalación de paquetes, en Fedora Core 6 son muy complicados por el hecho de los procesadores de 64 bits son muy pocos usados y únicamente en los foros se puede tener acceso a ese tipo de problemas para poderle dar solución. Es un poco complicado por el hecho de no estar tan familiarizado como es el caso de Windows en el que las instalaciones son de forma manual, en Linux las instalaciones se hacen desde una terminal por comandos lo que complica si no se tiene experiencia en Linux y con un procesamiento de palabra de 64 bits.

Linux tiene dos características importantes en comparación con Windows:

- La primera de ellas y la que más trabajo nos costó entender, es que porque si tenemos un procesador con 4 núcleos en el códec aparecían 8. Y la respuesta es que Linux permite trabajar de dos modos a los procesadores que tienen varios núcleos: como SMP en donde podríamos optimizar al máximo cada uno de los núcleos o en su defecto como procesador normal para aplicaciones seriales.

- La segunda es que Linux reconoce los 8GB de memoria y los 480 GB de espacio en disco a diferencia que en Windows XP sólo reconoce 2GB en RAM.

- La tercera es que en Linux sele puede dar prioridad a los procesos de tal manera que siempre se ejecute primero el que se desea por medio del administrador de procesos.

5.3 PERSPECTIVAS

El cluster puede ser utilizado en una gran cantidad de aplicaciones, tanto en la programación de métodos (por ejemplo el método de diferencias finitas en el dominio del tiempo), como en aplicaciones de alta disponibilidad (que el cluster esté al servicio de todos los que trabajan en SEPI para el apoyo a tareas y proyectos).

Algunas implicaciones del paralelismo serian las siguientes

- Diseño de Computadoras Paralelas
- Diseño de Algoritmos Eficientes
- Evaluación de Algoritmos Paralelos
- Desarrollo de lenguajes de programación Paralelos
- Desarrollo de Utilerías de Programación Paralela
- Portabilidad de aplicaciones paralelas

Además podemos ver claramente las siguientes observaciones acerca del paralelismo:

- El paralelismo solo era usado para aplicaciones de alto rendimiento (No más)
- El cómputo paralelo no se va a detener
- Desarrollar aplicaciones paralelas no es simple
- Evaluar el rendimiento en paralelo es complejo

5.4 PROPUESTAS PARA TRABAJOS FUTUROS

Durante la realización de este proyecto se encontraron varios puntos de mejora, así como algunos temas alternativos, los cuales convienen para futuros proyectos de investigación:

-Utilización de licencias para el mejor desempeño del clúster.

-Para poder realizar experimentos con áreas de cálculo mucho más grandes se propone incrementar el número de nodos.

-Uso de partición de procesos para la implementación de multihilos

APÉNDICE "A":

COMANDOS BÁSICOS DE LINUX 

Comandos	Descripción
ls	Lista los ficheros de un directorio concreto
ls -l	Lista también las propiedades y atributos
ls -la	Lista ficheros incluidos los ocultos de sistema
cd nom_directorio	Cambia de directorio
more nom_fichero	Muestra el contenido de un fichero de forma paginada
ls -la | more	Lista las ficheros de forma paginada
mv [ruta1]fichero1 [ruta2]fichero2	Mueve y/o renombra un fichero.
rm archivo o directorio	Elimina archivos o directorios
rm -R directorio	Borra un directorio recursivamente
cp archivo1 archivo2	Realiza una copia de un fichero
tail nom_archivo	Muestra las últimas líneas de un archivo de forma estática
tail -f nom_archivo	Muestra las últimas líneas del fichero de manera dinámica
head -numero nom_archivo	Muestra las primeras (número) líneas de un fichero
mkdir nom_directorio	Crea un directorio
rmdir nom_directorio	Elimina un directorio
chmod xxx nom_fichero	Cambia los permisos de acceso de un fichero
chown usuario fichero/directorio	Cambia el propietario de un fichero o directorio
chgrp grupo fichero/directorio	Cambia el grupo (-R para realizar el cambio recursivo)
ps aux	Muestra una lista de los procesos activos
kill -x(de 0 a 9) PID	Elimina un proceso via nº PID que le identifica
mount	Vemos el listado de dispositivos montados
mount /dev/fd0 /mnt/floppy	Montaje de la disquetera
mount /mnt/cdrom	Punto de montaje del CD-ROM
umount	Desmonta los puntos de montaje anteriores
hostname	Visualiza el nombre de la máquina
fsck	Escanea el disco

init 0	Apaga la máquina de manera correcta
init 6	Reinicia la máquina
compress/uncompress	Comandos de compresión de archivos " .Z"
gunzip nom_archivo.gz	Descompresión de archivos " .gz"
tar xvf fichero.tar	Descomprime archivos " .tar"
top	Lista procesos de CPU
find / -name nom -print	Encuentra ficheros según patrón
pwd	Visualiza el directorio actual
grep 'cadena' archivo	Muestra las líneas del archivo que contienen la cadena
date	Da la hora y la fecha del sistema
cal	Muestra el calendario del mes actual
clear	Borra la pantalla
who	Informa de los usuarios conectados
whoami	Más información sobre nuestra máquina
finger	Información más completa que who
su	Entrar a la sesión como root (necesario passwd)
su nom_usuari	Estando como root entramos como otro usuario
fdisk	Gestión de particiones ('m' = menú de comandos)
setup	Configuración gráfica de dispositivos
rpm -i nombre_paquete	Instalación de paquetes Fedora Core
startx	Arranca el entorno gráfico
Ctrl+Alt+F2	Salir del entorno gráfico a un terminal
Alt+F'x'	Conmutar entre terminales
make config	Configuración del Kernel
make xconfig	Entorno gráfico de configuración del Kernel
ifconfig -a	Información de las propiedades de red, equivale a ifconfig /all en Windows
dump/restore	Copias de seguridad y restauración
./comando	Ejecuta un comando si no tenemos el path del directorio

GLOSARIO

API

Siglas de Aplication Program Interface. Interfaz de Aplicación del Programa. Es el conjunto de rutinas del sistema que se pueden usar en un programa para la gestión de Entrada / Salida, gestión de ficheros etc.

ARRAYS SISTÓLICOS

Una arquitectura y un paradigma aplicables a la computación en paralelo. Los algoritmos paralelos sistólicos exhiben un flujo direccional por el cual los datos se dividen y quedan en cola para pasar por iguales secuencias de procesamiento al moverse de un proceso al siguiente. La metáfora del flujo de sangre ha servido para caracterizar el flujo de datos a lo largo del proceso.

BIOS

Sistema de E/S básica (Basic Imput Ouput System). Suele tratarse de uno o varios chips de memoria ROM (habitualmente EPROMs) que contienen las rutinas básicas de entrada y salida, los primeros pasos que debe dar un ordenador al encenderse, la configuración básica del sistema, etc.

BUS

Es como una especie de carretera por donde circulan los datos dentro de loa computadora comunicando a los diferentes componentes de la placa madre. Hay de varios tamaños, de 16, 32 o 64 bits, que se corresponden con menor o mayor capacidad de transferencia de información y por tanto mayores prestaciones de la máquina.

CANAL DE FIBRA

(Fiber Channel). Tipo de camino de transmisión usado como un canal interno de computador, así como un medio de redes. Funciona con interfaces existentes, como IPI, SCSI y HiPPI. En

una LAN, puede utilizarse como una espina dorsal de alta velocidad. Las velocidades van hasta 100 Mbytes/seg. Utilizando fibra óptica.

CICLO DE RELOJ

La velocidad a la que el procesador puede ejecutar instrucciones se conoce como ciclo de reloj o velocidad de reloj. Esta velocidad se expresa en gigahertz (Ghz), siendo 1 Ghz el equivalente a mil millones de ciclos por segundo. Mientras más rápido sea el reloj, más instrucciones podrá ejecutar la CPU por segundo.

CHIP

Anglicismo por el cual se conocen los circuitos integrados desarrollados sobre obleas de silicio. Generalmente es una parte rectangular de una oblea de silicio o de otro material semiconductor como arseniuro de galio, en el cual, por sofisticadas técnicas microlito gráficas se han implantado o desarrollador complejos circuitos electrónicos.

CHIPSET

El chipset controla al sistema y sus capacidades. Todos los componentes se comunican con el procesador a través del chipset, es el centro de toda la transferencia de datos. El chipset usa el controlador DMA y el controlador de bus para organizar el flujo estable de datos que él controla. El chipset es una serie de chips acoplados directamente a la tarjeta madre y normalmente es el segundo en tamaño después del procesador. Los chipsets están integrados (soldados en la tarjeta madre) y no pueden ser actualizados sin una tarjeta madre nueva.

CLÁUSULA

Una cláusula es una parte ó condición muy especifica muy específica dentro de una directiva.

COSTO

El costo seria definido básicamente por la relación, Costo = (tiempo de ejecución) × (número total de procesadores utilizados). El costo de un cálculo sería simplemente el tiempo de ejecución.

DHCP

Es un protocolo con arquitectura cliente / servidor basado en UDP sobre IP, que ofrece un mecanismo automático para obtener los parámetros de una interfaz de red a través de un servidor, que ofrece sus servicios en una red de área local.

DIMM

Siglas de Dual In Line Memory Module, un tipo de encapsulado, consiste en una pequeña placa de circuito impreso, que almacena chips de memoria, que se inserta en un zócalo DIMM en la placa madre y usa generalmente un conector de 168 contactos. Memoria RAM de 64 bits pensada para Pentium II, y que es bastante sencillo suponer, de una mayor velocidad de transferencia de datos.

DIRECTIVA #PRAGMA

La directiva #pragma, permite a los compiladores C++ definir sus directivas particulares (que no corresponden a nada establecido en el estándar ANSI) sin interferir con otros compiladores que soporten este tipo de directivas. Si el compilador no reconoce la etiqueta nombre-de-directiva, la directiva es ignorada sin ningún tipo de mensaje o advertencia.

DMA

Acceso directo a memoria, un proceso que permite a un dispositivo (externo o interno) transferir datos a la memoria del ordenador a alta velocidad, sin que estos datos pasen por el procesador (Direct Memory Access).

ESDI

Siglas de Enhanced Small Device Interface, Interfaz Resaltada de dispositivos pequeños. Interfaz de unidad de disco que transfiere datos en el intervalo de uno a tres Mbytes/seg. ESDI era la interfaz de alta velocidad para computadores pequeños, pero ha sido remplazada por los controladores IDE y SCSI.

ETHERNET

Sistema de red de área local de 10 Mbps. Se ha convertido en un estándar de red corporativa.

ESPEJEO DE DISCO

Es el almacenamiento de una imagen, duplicada de su información en otro disco; así, si cae el primer disco, el segundo guarda una copia exacta de la información. A pesar de la gran seguridad que ofrece esta tecnología, su costo es muy elevado.

FBGA

Siglas de FINE – PITCH BALL GRID ARRAY. Matriz de rejilla de bolas de pequeño tamaño FBGA es una opción de carcasa para chips de memoria flash de AMD

FIREWALL

En español, "cortafuegos" Sistema de seguridad, encargado de proteger una red de área local de accesos no autorizados desde una WAN a la que esté conectada.

INTERFAZ

Se denomina así, a todo aquel medio físico que conecta un dispositivo periférico con la computadora; también se le conoce así a todo el software que comunica al usuario con la misma.

IRQ

Solicitud de interrupción (interrupt ReQuest). Línea de interrupción que los dispositivos emplean para avisar al procesador de que necesitan su atención. Según el procesador, suele haber varias líneas (por ejemplo, 16), y cada dispositivo debería emplear una distinta, para evitar conflictos con otros dispositivos.

MAINFRAME

así se le llama a las grandes computadoras, capaces de atender a miles de usuarios y miles de programas "al mismo tiempo" asignándoles un periodo muy pequeño a la atención de cada programa. Su capacidad de trabajo es muy alta, por lo que normalmente se encuentran en empresas de gran tamaño. Sus programas están compuestos por cientos de miles o millones de líneas de código.

MCA

Micro Channel Arquitecture. Arquitectura introducida por IBM en sus ordenadores personales PS12. La Arquitectura Microcanal se refiere a las especificaciones que hacen posible la conexión de cualquier periférico con un bus MCA.

MIDDLEWARE

Recibe este nombre el conjunto de servicios o facilidades a las que es posible acudir en el ámbito de una arquitectura, por ejemplo, del tipo SAA.

SEGMENTACIÓN

Sistema de gestión de memoria, en el que tanto la memoria física como la memoria virtual se dividen en bloques (en general de distintos tamaños, llamados segmentos), que son las que se vuelcan a disco a medida que se va llenando la memoria física, o se recuperan de disco cuando vuelven a ser necesarias.

SIMM

Single Inline Memory Module, Mòdulo sencillo de memoria en línea. Pequeña tarjeta que contiene chips de RAM y que es insertada en la tarjeta madre para aumentar la capacidad de memoria de la computadora.

SIP

Signal Internet Provider. Proveedor de Señal para las ISP. (Single In-Line Packages). Los módulos SIP tienen una hilera de 30 patillas pequeñas insertadas en la banda de toma correspondiente. No se popularizaron tanto como los SIMMs por que (por tener patitas en vez de contactos eran más delicadas). Para muchos usuarios este tipo de memoria pasó desapercibida. En realidad son lo mismo que un SIMM de 30 pines pero con patitas.

SO DIMMs

Un tipo de memoria que se utiliza comúnmente en las computadoras portátiles se llama SO DIMM o DIMM de delineado pequeño. La principal diferencia entre un SO DIMM y un DIMM

es que el SO DIMM, debido a que su uso es para computadoras portátiles, es significativamente más chico que el DIMM estándar. Los SO DIMMs de 72 pines tienen 32 bits y los de 144 tienen 64 bits de ancho.

SOJ (GUÍA "J" DE DELINEADO PEQUEÑO)

Los empaques SOJ obtuvieron un nombre debido a los pines que salen del chip, tienen forma de la letra "J". Los SOJ son componentes que se montan en superficie, es decir, se montan directamente en la superficie del PCB.

TIEMPOS MUERTOS

De un procesador, los intervalos de tiempo en que éste se encuentra en espera de un dato.

TSOP

Thin Small Outline Package. Tipo de paquete DRAM que utiliza los conductores en forma de "gaviota" en los dos lados. La DRAM TSOP se instala directamente en la superficie de la placa del circuito impreso. La ventaja del paquete TSOP consiste en que tiene un tercio del grosor de un paquete SOJ. Los componentes TSOP se utilizan comúnmente en los SODIMM (de perfil pequeño) y en las aplicaciones de memoria "tarjeta de crédito".

VESA

Video Electronics Standard Association. Asociación encargada de establecer los estándares de video y multimedia de las PC's.

WIDGET

Una representación estandarizada de un control que podría ser manipulado por los usuarios. Barras de desplazamiento, pulsadores, y casillas de texto son ejemplos de widgets.

REFERENCIAS

[1] **Parallel programming: Techniques and aplications using networked workstations and parallel computers**

Barry Wilkinson, Michael Allen

Prentice Hall

[2] **Programación sobre sistemas de memoria compartida: programación con OPENMP**

María J. Martín

Universidad de A. Coruña

[3] **Linux network administrator´s guide**

Olaf Kirch & Ferry Dawson

O Reilly, 2nd Edition

[4] **High performance TCP/IP networking**

Raj Jain

Prentice Hall; United States edition (November 16, 2003)

[5] **Parallel programming in C with MPI and OPENMP**

Michael J. Quinn

McGraw-Hill, ©2004.

[6] **High-level parallel programming models and supportive environments**

Frank Mueller

Springer

[7] **Computer Communications and Networks**

Sammes,

A.J.

[8] **Linux Bible 2008: Boot up to Ubuntu, Fedora, KNOPPIX, Debian, openSUSE, and 11 Other Distributions**

Vicki Stanfield

[9] **Linux System Administration, Second Edition (Craig Hunt Linux Library)**
Roderick W. Smith

Sybex

[10] Modern Operating Systems
Andrew Tanenbaum
Prentice Hall (2nd Edition)

Sitios en Internet:

OPENMP

http://openmp.org/wp/

LAM/MPI

http://www.lam-mpi.org/

Foro Fedora

http://fedoraforum.org/