# Universidad Nacional Autónoma de México

## Facultad de Ingeniería

## Análisis del procesamiento en paralelo en las bases de datos

**TESIS DE LICENCIATURA**

QUE PARA OPTAR POR EL GRADO DE

**INGENIERO EN COMPUTACIÓN**

PRESENTA

**Erick Manuel Betanzo Vásquez**

Director de Tesis:

**Ingeniera Luciralia Hernández Hernández**

Ciudad Universitaria

México, Distrito Federal Abril 2013.

# Agradecimientos

A la Universidad Nacional Autónoma de México por concederme la oportunidad de realizar mis estudios desde temprana edad en Iniciación Universitaria y culminarlos en la Facultad de Ingeniería.

A la Ingeniera Luciralia Hernández Hernández, por su apoyo, consejos y paciencia en la realización de este trabajo.

A mis sinodales por dedicarle tiempo a la supervisión de este trabajo.

# Dedicatoria

A mi madre Ana Vázquez Sánchez y a mi padre Manuel Betanzo Velasco, por su amor, cariño, comprensión, paciencia, consejos y sobre todo por ese apoyo incondicional que me han brindado durante cada etapa de mi vida.

A Yarisma Betanzo y Antonio Guzmán por demostrarme su cariño y apoyo en todo momento.

A Braulio y Emiliano por representar la inspiración, además de ser un motor de energía cada día y por el gran amor que les tengo.

A mi familia por su motivación, empuje y compañía en cada instante.

A mis entrañables amigos, Antonio, Arturo, Daniel, Guillermo Jonathan, Leonardo, Mario, Mónica, Ulises y Yusura por no ser invisibles en momentos de dicha y aflicciones.

*"La vida es todo lo agradable que se lo permitas" (Charles Bukowski).*

# Índice

## *Capítulo 1 Conceptos básicos*



## *Capítulo 2 Aspectos de la programación en paralelo*



## *Capítulo 3 El Paralelismo*

# Capítulo 1. Conceptos Básicos


## EXTRACTO

En este primer capítulo se explicarán algunos conceptos básicos, los cuales nos ayudarán a poder comprender conceptos más profundos que se abordarán a lo largo de los capítulos posteriores. Se abordarán y se explicarán conceptos básicos como: qué es un algoritmo, sus principales características, su desarrollo, los tipos de algoritmos que existen, etc. De igual forma se tratarán conceptos fundamentales acerca de las bases de datos, así como los fundamentos de la programación concurrente.

# 1 Introducción

La creación de sistemas de bases de datos nace a partir de la necesidad de almacenar grandes cantidades de datos pertenecientes a un mismo contexto, los cuales se encuentran almacenados sistemáticamente para su consulta posterior.

El avance tecnológico en las diversas áreas de la computación ha evolucionado de forma exponencial. De igual forma, la rama de las bases de datos ha tenido un desarrollo tecnológico importante, el cual se ha dado en gran medida debido al incremento en el volumen de información que éstas manejan; a las nuevas ofertas de hardware que hay en el mercado y a la eficiencia de los nuevos sistemas manejadores de bases de datos.

La importancia de poder almacenar y extraer información en un menor lapso de tiempo resulta de vital importancia, por tal motivo el procesamiento en paralelo en las bases de datos representa una alternativa para poder tener un mejor desempeño en las bases de datos; dependiendo del ambiente en el que éstas se encuentren funcionando.

Conocer los aspectos que conlleva implementar una base de datos con procesamiento en paralelo, así como el funcionamiento que ésta presenta, saber cómo es que trabaja de forma interna, conocer cuándo es posible o no poder paralelizar un proceso y, las ventajas y desventajas resultan de gran utilidad para poder decidir qué tipo de base de datos se va a implantar en la empresa u organización en donde nos encontremos. Todo esto, con el fin de ofrecer el mejor rendimiento.

Procesar grandes cantidades de información en una base de datos con el fin de obtener datos específicos, impacta en el rendimiento de la base de datos, por tal motivo el tiempo es un factor sumamente importante y una forma de poder mejorar el rendimiento de las consultas es mediante el uso del procesamiento en paralelo. En este trabajo de tesis se describe mediante el uso del procesamiento en paralelo como éste sirve a la mejora de consultas realizadas. Aunado a los conceptos que se explicarán a lo largo de los capítulos que conforman este escrito se presenta una parte práctica la cuál se muestra el funcionamiento del procesamiento en paralelo en las bases de datos.

Se abordarán conceptos de algunos tipos de paralelismo que existen en las bases de datos, así como algunas arquitecturas para la implementación de este tipo de bases de datos, de igual forma se detallarán las ventajas y desventajas que representa implementar este tipo de base de datos en cada una de las arquitecturas.

Es el tiempo, el factor que juega el papel más importante, principalmente en este tipo de ambientes de desarrollo y manipulación de información, en donde se desea obtener periodos de tiempo menores a la hora de disponer de los datos.

La importancia que representa el análisis y el conocimiento del procesamiento en paralelo en este tipo de sistemas, resulta ser de gran apoyo para poder realizar una comparativa respecto a otros tipos de procesamiento de datos existentes, ya que nos proporciona una posibilidad más al momento de procesar la información.

Resulta imprescindible aprovechar al máximo los recursos de hardware que hoy día se tienen; en conjunto con el buen diseño de software, éstos ayudan a tener un mejor desempeño en los sistemas

que se manejan. En la actualidad, el avance tecnológico en cuanto a hardware es impactante, pero en un futuro no muy lejano tendrá sus limitantes de desarrollo. Es entonces cuando alternativas de implementación software resultan ser muy valiosas cuando son diseñadas de forma adecuada, tal como lo es una excelente implementación del procesamiento en paralelo.

## 1.1 Definición de Algoritmo

A continuación se dan algunas definiciones de la palabra algoritmo.

a. Definición del Diccionario de la Real Academia de la Lengua Española (Española, 2012)

**algoritmo.** (Quizá del lat. tardío **algobarismus*, y este abrev. del ár. clás. *ḥisābu lḡubār* 'cálculo mediante cifras arábigas').1. m. Conjunto ordenado y finito de operaciones que permite hallar la solución de un problema.

b. Definiciones desde el punto de vista computacional

b.1. Secuencia finita de instrucciones, reglas o pasos que describen de forma precisa las operaciones que un ordenador debe realizar para llevar a cabo una tarea en un tiempo más finito. (Donald E. Knuth, 1968).

b.2. Descripción de un esquema de comportamiento expresado mediante un repertorio finito de acciones y de informaciones elementales, identificadas, bien comprendidas y realizables *a priori*. Este repertorio se denomina léxico (Pierre Scholl, 1988).

b.3. Un algoritmo es un conjunto finito de pasos definidos, estructurados en el tiempo y formulados con base a un conjunto finito de reglas no ambiguas, que proveen un procedimiento para dar la solución o indicar la falta de esta a un problema en un tiempo determinado. (Rodolfo Quispe-Otazu, 2004).

c. Definición del autor

Un algoritmo es un conjunto ordenado y finito de instrucciones concretas y detalladas, que permiten obtener un resultado determinado en un tiempo establecido, a partir de ciertas reglas previamente definidas.

## 1.2 Características de un algoritmo

Las características fundamentales que debe cumplir todo algoritmo son las siguientes:

a. Ser definido: sin ambigüedad, cada paso del algoritmo debe indicar la acción a realizar sin criterios de interpretación.

b. Ser finito: un número específico y numerable de pasos debe componer al algoritmo, el cual deberá finalizar al completarlos.

c. Tener cero o más entradas: los datos son proporcionados a un algoritmo como insumo (o estos son generados de alguna forma), para llevar a cabo las operaciones que comprende.
d. Tener una o más salidas: siempre debe devolver un resultado, de nada sirve un algoritmo que hace algo y nunca sabemos qué fue. El devolver un resultado no debe ser considerado únicamente como

verlos de forma impresa o en pantalla, como ocurre con las computadoras. Existen muchos otros mecanismos susceptibles de programación que no cuentan con una salida de resultados de esta forma. Por salida de resultados debe entenderse todo medio o canal por el cual es posible apreciar los efectos de las acciones del algoritmo.

e. Efectividad: el tiempo y esfuerzo por cada paso realizado debe ser preciso, sin usar nada más ni nada menos, sólo aquello que se requiera para y en su ejecución.

## 1.3 Desarrollo de algoritmos

Aunque, paradójicamente, no exista un algoritmo para el desarrollo de algoritmos, es posible enumerar una serie de pasos que pueden considerarse en conjunto como una guía genérica en la correcta creación de un algoritmo. Los pasos son los siguientes:

1. Declaración del problema.
2. Desarrollo de un modelo.
3. Diseño del procedimiento de solución.
4. Validación.
5. Implementación.
6. Análisis y complejidad.
7. Pruebas.
8. Documentación.

## 1.4 Medios de expresión de un algoritmo

Los algoritmos pueden ser expresados de distintas formas, con el lenguaje natural, pseudocódigo, diagramas de flujo y lenguajes de programación, entre otros.

Las descripciones en lenguaje natural tienden a ser ambiguas y extensas. El usar un pseudocódigo y diagramas de flujo evita muchas ambigüedades del lenguaje natural. Dichas expresiones son formas más estructuradas para representar algoritmos; no obstante, se mantienen independientes de un lenguaje de programación específico.

La descripción de un algoritmo usualmente se hace en tres niveles:

a) Descripción de alto nivel. Se establece el problema, se selecciona un modelo matemático y se explica el algoritmo de manera verbal, posiblemente con ilustraciones y se omiten detalles.
b) Descripción formal. Se usa pseudocódigo para describir la secuencia de pasos que encuentran la solución.
c) Implementación. Se muestra el algoritmo expresado en un lenguaje de programación específico o algún objeto capaz de llevar a cabo instrucciones.

### *Diagrama de flujo*

Los diagramas de flujo (Farina, 1971) son descripciones gráficas de algoritmos, usan símbolos conectados con flechas para indicar la secuencia de instrucciones y están regidos por ISO (International

Organization for Standardization- Organización Internacional de normalización).

Los diagramas de flujo son usados para representar algoritmos pequeños, ya que abarcan mucho espacio y su construcción es laboriosa. Por su facilidad de lectura, son usados como introducción a los algoritmos, descripción de un lenguaje y descripción de procesos a personas ajenas a la computación.

### *Pseudocódigo*

El pseudocódigo es la descripción de un algoritmo que asemeja a un lenguaje de programación pero con algunas convenciones del lenguaje natural (de ahí, que tenga el prefijo pseudo, que significa falso). Tiene varias ventajas con respecto a los diagramas de flujo, entre las que se destaca el poco espacio que se requiere para representar instrucciones complejas. El pseudocódigo no está regido por ningún estándar.

### *Sistemas formales.*

La teoría de autómatas y la teoría de funciones recursivas proveen modelos matemáticos que formalizan el concepto de algoritmo. Los modelos más comunes son la máquina de Turing, máquina de registro y funciones recursivas. Estos modelos son tan precisos como un lenguaje máquina, careciendo de expresiones coloquiales o ambigüedad, sin embargo, se mantienen independientes de cualquier computadora y de cualquier implementación.

*Implementación*

Muchos algoritmos son ideados para implementarse en un programa. Sin embargo, los algoritmos pueden ser implementados en otros medios, como una red neuronal, un circuito eléctrico o un aparato mecánico y eléctrico.

## 1.5 Análisis de algoritmos

Como medida de eficiencia de un algoritmo, se suelen estudiar los recursos, memoria y tiempo, que éste consume. El análisis de algoritmos se ha desarrollado para obtener valores que, de alguna forma, indiquen o especifiquen la evolución del gasto de tiempo y memoria, en función del tamaño de los valores de entrada.

El análisis y estudio de los algoritmos es una disciplina de las ciencias de la computación y, en la mayoría de los casos, su estudio es completamente abstracto sin usar ningún tipo de lenguaje de programación ni cualquier otra implementación; por eso, en ese sentido, comparte las características de las disciplinas matemáticas. Así, el análisis de los algoritmos se centra en los principios básicos del algoritmo, no en los de la implementación particular. Una forma de plasmar o, algunas veces "codificar", un algoritmo es escribirlo en pseudocódigo o utilizar un lenguaje muy simple tal como Léxico, cuyos códigos pueden estar en el idioma del programador.

## 1.6 Algoritmo secuencial y paralelo

*Algoritmo Secuencial*

Es un algoritmo en el que un paso requiere que el paso anterior haya concluido. De esta forma, normalmente una etapa de este tipo de algoritmos normalmente requerirá datos de la etapa anterior. (wikitionary, 2012)

*Algoritmo Paralelo*

Es un algoritmo que puede ser ejecutado por partes en el mismo instante de tiempo por varias unidades de procesamiento para, finalmente, unir todas las partes y obtener el resultado correcto. (wikitionary, 2012)

Los algoritmos paralelos son importantes porque es más rápido realizar grandes tareas de computación mediante la paralelización que mediante técnicas secuenciales. Esta es la forma en que se trabaja en el desarrollo de los procesadores modernos, ya que es más difícil incrementar la capacidad de procesamiento con un único procesador que aumentar su capacidad de cómputo mediante la inclusión de unidades en paralelo, logrando así, la ejecución de varios flujos de instrucciones dentro del procesador.

## 1.7 Fundamentos de Bases de Datos

Una base de datos (DB) es un conjunto de datos organizados y estructurados, los cuales pertenecen a un mismo contexto y son almacenados sistemáticamente para ser usados posteriormente. Las

bases de datos son diseñadas con un propósito específico y deben ser organizadas con una lógica coherente; los datos que se encuentran en una base de datos podrán ser compartidos por distintos usuarios y aplicaciones. (Korth & Silberschatz, 1993)

Un gestor de base de datos es un módulo de programa que proporciona una interfaz entre los datos de bajo nivel almacenados en la base de datos y los programas de aplicación y consultas realizadas al sistema. El DBM es responsable de las siguientes tareas:

- Interacción con el gestor de archivos (el gestor de la base de datos traduce las distintas sentencias DML a comandos del sistema de archivos de bajo nivel; así, el gestor de base de datos es responsable del verdadero almacenamiento, recuperación y actualización de los datos en la base de datos).
- Implantación de la integridad (los valores de los datos que se almacenan en la base de datos deben satisfacer ciertos tipos de restricciones de consistencia).
- Implantación de la seguridad (hacer que se cumplan estos requisitos de seguridad).
- Copia de seguridad y recuperación (detectar fallos y restaurar la base de datos al estado que existía antes de ocurrir el fallo).
- Control de concurrencia (controlar la interacción entre los usuarios concurrentes).

Un sistema de gestión de base de datos (DBMS) es el conjunto de aplicaciones con las cuales podemos manejar nuestra base de datos, la cual consiste en una colección de datos interrelacionados y un conjunto de programas para acceder a esos datos; tiene como objetivo proporcionar de forma eficiente la información que tiene almacenada. Las dos funciones principales que tiene un sistema de gestión de base de datos son definición de las estructuras para

almacenar los datos y la manipulación de los datos. (Connolly & Begg, 2005)

El objetivo primordial de un DBMS es proporcionar un entorno que sea conveniente y eficiente a la vez, para ser utilizado al extraer y almacenar información de la base de datos.

Los sistemas de base de datos están diseñados para gestionar grandes bloques de información, además de mantener la seguridad de la información almacenada, pese a caídas del sistema o intentos de acceso no autorizados.

El lenguaje de definición de datos (DDL) es un lenguaje proporcionado por el sistema de gestión de base de datos que permite a los usuarios, de la misma, llevar a cabo las tareas de definición de las estructuras que almacenarán los datos, así como de los procedimientos o funciones que permitan consultarlos. (Connolly & Begg, 2005)

El lenguaje de manipulación de datos (DML) es un lenguaje proporcionado por el sistema de gestión de base de datos que capacita a los usuarios a acceder o manipular datos según estén organizados por el modelo de datos adecuado. Existen básicamente dos tipos:

a) Procedimentales (con los DML se requiere que el usuario especifique qué datos se necesitan y cómo obtenerlos).
b) No procedimentales (con los DML se requiere que el usuario especifique qué datos se necesitan sin especificar cómo obtenerlos).

El administrador de la base de datos (DBA) es la persona que tiene el control central sobre el sistema, y algunas de las funciones que debe cumplir son las siguientes:

- Definición de esquema.
- Definición de la estructura de almacenamiento y del método de acceso.
- Modificación del esquema y de la organización física.
- Concesión de autorización para el acceso a los datos.
- Especificación de las restricciones de integridad.

Estructuras del sistema global. Un sistema de base de datos se divide en módulos que tratan cada una de las responsabilidades del sistema general. Así, el diseño de una sistema de base de datos debe incluir la consideración de la interfaz entre el sistema de base de datos y el sistema operativo.
Los componentes funcionales de un sistema de base de datos incluyen:

- Gestor de archivos: gestiona la asignación de espacio en la memoria del disco y de las estructuras de datos usadas para representar información almacenada en el disco.
- Gestor de base de datos: proporciona la interfaz entre los datos de bajo nivel almacenados en la base de datos y los programas de aplicación y las consultas que se hacen al sistema.
- Procesador de consultas: traduce sentencias en un lenguaje de consultas a instrucciones de bajo nivel que entiende el gestor de la base de datos.
- Pre-compilador DML: convierte las sentencias en DML incorporadas en un programa de aplicación en llamadas normales a procedimiento en el lenguaje principal.

- Compilador DDL: convierte las sentencias en DDL en un conjunto de tablas que contienen metadatos.

*Metadatos: son datos que describen otros datos.
*Archivos de datos: son los archivos que almacenan la base de datos.
*Diccionario de datos: almacena metadatos sobre la estructura de la base de datos.
*Índices: es una estructura de datos que proporciona acceso rápido a los elementos de datos que contiene valores determinados.

## 1.8 Arquitectura de un sistema de base de datos

La arquitectura de un sistema de base de datos se divide en tres niveles (Connolly & Begg, 2005):

- Nivel físico o lógico. En este nivel se describe la estructura física de la base de datos mediante un esquema interno. Este esquema se especifica mediante un modelo físico y describe todos los detalles para el almacenamiento de la base de datos, así como los métodos de acceso.

- Nivel conceptual o interno. En este nivel se describe la estructura de toda la base de datos para una comunidad de usuarios (todos los de una empresa u organización), mediante un esquema conceptual. Este esquema oculta los detalles de las estructuras de almacenamiento y se concentra en describir entidades, atributos, relaciones, operaciones de los usuarios y restricciones. En este nivel se puede utilizar un modelo conceptual o un modelo lógico para especificar el esquema.

- Nivel de visión o externo. En este nivel se describen varios esquemas externos o vistas de usuario. Cada esquema externo describe la parte de la base de datos que interesa a un grupo de usuarios determinado y oculta a ese grupo el resto de la base de datos. En este nivel se puede utilizar un modelo conceptual o un modelo lógico para especificar los esquemas.

Los tres esquemas o niveles no son más que descripciones de los mismos datos pero con distintos niveles de abstracción. Los únicos datos que existen realmente están a nivel físico, almacenados en un dispositivo como puede ser un disco. En un DMBS, basado en la arquitectura de tres niveles; cada grupo de usuarios hace referencia exclusivamente a su propio esquema externo.

La arquitectura de tres niveles es de gran utilidad para comprender el concepto de independencia de datos, la cual es la capacidad para modificar el esquema en un nivel del sistema, sin tener que modificar el esquema del nivel inmediato superior. Se pueden definir dos niveles de independencia de datos:

a) La independencia lógica, que es la capacidad de modificar el esquema conceptual sin tener que alterar los esquemas externos ni los programas de aplicación. Se puede modificar el esquema conceptual para ampliar la base de datos o para reducirla.

b)La independencia física, que es la capacidad de modificar el esquema físico sin tener que alterar el esquema conceptual (o los externos).

## 1.9 Modelos de datos

Un modelo de datos es una colección de herramientas conceptuales para describir datos, relaciones entre ellos, semántica asociada a los datos y restricciones de consistencia. Los modelos de datos contienen también un conjunto de operaciones básicas para la realización de consultas (lecturas) y actualizaciones de datos. De esta forma, los modelos de datos son el instrumento principal para ofrecer la abstracción de los datos (Korth & Silberschatz, 1993).

Los modelos de datos se dividen en tres grupos:

a) Modelos lógicos basados en objetos.
b) Modelos lógicos basados en registros.
c) Modelos físicos de datos.

### Modelos lógicos basados en objetos.

Los modelos lógicos, basados en objetos se usan para describir datos en los niveles conceptual y de visón, este tipo de modelos se caracteriza por el hecho de proporcionar capacidad de estructuración sumamente flexible y permite especificar restricciones de datos explícitamente. Algunos de este tipo de modelos son:

- Modelo entidad-relación.
- Modelo orientado a objetos.
- Modelo binario.
- Modelo semántico de datos.

El modelo entidad-relación (E-R) se basa en una percepción de un mundo real que consiste en una colección entre objetos básicos, llamados entidades y relaciones. Una entidad es un objeto que es

distinguible de otros objetos por medio de un conjunto específico de atributos. Una relación es una asociación entre varias entidades. El conjunto de todas las entidades del mismo tipo y relaciones del mismo tipo, se denomina conjunto de entidades y conjunto de relaciones, respectivamente. Este modelo representa ciertas restricciones a las que deben ajustarse los contenidos de una base de datos, siendo la cardinalidad de asignación una restricción importante, ya que expresa el número de entidades a las que puede asociarse otra entidad mediante un conjunto de relación.

La estructura lógica global de una base de datos puede expresarse gráficamente por medio de un diagrama E-R, el cual consta de los siguientes componentes:

- Rectángulos (representan conjuntos de entidades).
- Elipses (representan atributos).
- Rombos (Representan relaciones entre conjuntos de entidades).
- Líneas (conectan atributos a conjuntos de entidades y conjuntos de entidades a relaciones).

El modelo orientado a objetos, se basa en una colección de objetos al igual que el modelo E-R. Un objeto contiene valores almacenados en variables de instancia dentro del objeto. A diferencia de los modelos orientados a registros, estos valores son objetos por sí mismos. Un objeto también contiene partes de código que operan sobre sí mismo, esas partes son los métodos.

Los objetos que contienen los mismo tipos de valores y los mismos métodos se agrupan en clases; una clase puede ser vista como una definición de tipo para objetos. La única forma en la que un objeto puede acceder a los datos de otro objeto es invocando a un método de éste.

La diferencia entre el modelo E-R y el modelo orientado a objetos radica en que cada objeto tiene su propia identidad única independiente de los valores que contiene; así, dos objetos que contienen los mismos valores son, sin embargo, distintos.

*Modelos lógicos basados en registros*

Los modelos lógicos basados en registros se utilizan para describir datos en los modelos conceptual y físico y, a diferencia de los modelos de datos orientados a objetos, estos se usan para especificar la estructura lógica global de la base de datos y para proporcionar una descripción a nivel más alto de la implementación. Este tipo de modelos se llaman así porque la base de datos está estructurada en registros de formato fijo de varios tipos. Cada tipo de registro define un número fijo de campos o atributos y cada campo normalmente es de una longitud fija. Los tres modelos de datos mas utilizados son:

- Modelo Relacional
- Modelo de Red
- Modelo Jerárquico

El modelo relacional representa los datos y las relaciones entre los datos, mediante una colección de tablas, cada una de las cuales tiene un número de columnas con nombres únicos.

El modelo de red representa los datos mediante colecciones de registros y, las relaciones entre los datos se representan mediante enlaces, los cuales pueden verse como punteros. Los registros en la base de datos se organizan como colecciones de grafos arbitrarios.

El modelo jerárquico es similar al modelo de red, en el sentido que los datos y las relaciones entre los datos se representan mediante registros y enlaces, respectivamente. Se diferencia del modelo de red, en que los registros están organizados como colecciones de árboles en vez de grafos arbitrarios.

La diferencia entre estos tres tipos de modelos radica en que los modelos relacionales no usan punteros o enlaces como los modelos de red y jerárquicos, pues el modelo relaciones conecta los registros mediante valores que éstos contienen.

*Modelos físicos de datos*

Los modelos físicos de datos se usan para describir datos del nivel más bajo, pero son muy poco utilizados. Los dos modelos más conocidos son:

- Modelo Unificador
- Memoria de Elementos

Los objetivos del modelo de datos son por un lado formalizar y definir las estructuras permitidas para representar los datos y, por otro, diseñar la base de datos.

En el diseño de una base de datos hay que tener en cuenta distintas consideraciones, entre las que destacan las siguientes:

- La velocidad de acceso.
- El tamaño de la información.
- El tipo de información.
- La facilidad de acceso a la información.
- La facilidad para extraer la información requerida.

- El comportamiento del sistema de gestión de base de datos (DBMS) con cada tipo de información.

## 1.10 Fundamentos de Programación Concurrente

*Proceso*

Un proceso es básicamente un programa en ejecución, en el cual intervienen los valores corrientes del contador del programa, registros y variables. El proceso va acompañado de recursos como memoria, archivos, etc. (Pratt & Zelkowitz, 1998)

*Concurrencia*

La concurrencia es la ejecución simultánea de múltiples procesos creados por un único programa y que, potencialmente pueden interactuar entre sí.

Los procesos concurrentes pueden ser ejecutados de forma simultánea, siempre y cuando cada uno sea ejecutado en diferentes procesadores. En cambio, la concurrencia es simulada si existe sólo un procesador encargado de ejecutar los procesos concurrentes, simulando la concurrencia; ocupándose de forma alternada en uno y otro proceso en pequeñísimos intervalos de tiempo. De esta manera simula que se están ejecutando a la vez. La concurrencia puede presentarse en tres contextos diferentes:

- Múltiples aplicaciones: la multiprogramación se creó para permitir que el tiempo de procesador de la máquina fuese compartido dinámicamente entre varias aplicaciones activas.

- Aplicaciones estructuradas: como ampliación de los principios del diseño modular y la programación estructurada; algunas aplicaciones pueden implementarse eficazmente como un conjunto de procesos concurrentes.
- Estructura del sistema operativo: las mismas ventajas de estructuración son aplicables a los programadores de sistemas y se ha comprobado que algunos sistemas operativos están implementados como un conjunto de procesos o hilos.

*Comunicación entre procesos.*

Los procesos con frecuencia necesitan comunicarse con otros procesos. Cuando un proceso del usuario desea leer el contenido de un archivo, éste debe señalar al proceso del archivo lo que desea, después el proceso del archivo tiene que indicar al proceso del disco que lea el bloque que se pide. (Tanenbaum, 1988)

*Análisis de la comunicación entre procesos*

Todos los programas concurrentes implican interacción o comunicación entre hilos, esto ocurre por las siguientes razones:

- Los hilos, incluso los procesos, compiten por un acceso exclusivo a los recursos compartidos, como los archivos físicos (archivos o datos).
- Los hilos se comunican para intercambiar datos.

En ambos casos es necesario que los hilos sincronicen su ejecución para evitar conflictos cuando adquieren los recursos o para hacer contacto cuando intercambian datos.

### *Competencia entre procesos por los recursos*

Los procesos concurrentes entran en conflicto cuando compiten por el uso del mismo recurso. Dos o más procesos necesitan acceder a un recurso durante su ejecución. Cada proceso debe dejar tal y como esté el estado del recurso que utilice. (Tanenbaum, 1988)

La ejecución de un proceso puede influir en el comportamiento de los procesos que compiten. Por ejemplo, si dos procesos desean acceder a un recurso, el sistema operativo le asignará el recurso a uno y el otro tendrá que esperar.

### *Condiciones de Carrera (race conditon)*

Las condiciones de carrera son un problema que emerge del uso concurrente de recursos en el que nuestros procesos pueden corromper o perder información.

La existencia de una condición de carrera no garantiza la pérdida de información en un conjunto de intentos, por lo que se necesitan métodos formales para detectarla y corregirla.

Las condiciones de carrera sólo se pueden presentar cuando dos o más procesos, o hilos de ejecución, tratan de usar concurrentemente un recurso *non-preemtive*.

Los recursos pueden ser:

- *Preemtive:* puede atender de manera concurrente peticiones variadas sin corromper información.

- *Non-Preemtive:* son los que no pueden atender peticiones inconexas sino hasta que cada operación termina y deja el dispositivo en un estado especial.

Para poder resolver las condiciones de carrera se modelan mediante regiones críticas. Una región crítica es la parte del código o algoritmo en la que se emplea un recurso *non-preemtive* y donde se puede presentar la condición de carrera. (Pratt & Zelkowitz, 1998)

Para resolver las condiciones de carrera, basta con satisfacer las siguientes cuatro condiciones:

a) Dos procesos no deben estar simultáneamente en sus regiones críticas correspondientes.
b) No pueden hacerse suposiciones respecto al número o velocidad de los procesadores.
c) Ningún proceso que no esté en su región crítica puede bloquear a otros para que ingresen a su región crítica.
d) Ningún proceso debe esperar indefinidamente para ingresar a la región crítica.

La exclusión mutua es una manera de asegurarse de que, si un proceso está utilizando una variable o archivo compartido, los otros procesos no pueden hacer lo mismo. (Tanenbaum, 1988)

### *Exclusión mutua con espera ocupada (busy waiting)*

- Desactivación de interrupciones: para lograr la solución de la condición de carrera se puede desactivar el mecanismo de interrupciones al entrar a la condición de carrera y así, asegurar que termine los pasos en ella antes de que otro proceso se

active. Al salir de la condición de carrera rehabilitamos las interrupciones para reanudar el multiproceso.

- Variables de cerradura o de control: los procesos comparten una variable (*lock*) que indica que uno se encuentra en la condición de carrera. Para entrar a la condición de carrera el proceso prueba el valor de la variable y, de encontrarla libre, cambia su valor e ingresa a la condición de carrera. Si se encuentra la variable ocupada, entonces se mantiene en un bucle (*loop)* hasta que se libere; al salir de la condición de carrera cambia de nueva cuenta el valor de la variable para liberarla.

*Semáforos*

Un semáforo es una variable contador que controla la entrada a la región critica, las operaciones WAIT (cuando un proceso desea acceder a su sección critica) y SIGNAL (libera el acceso) controlan, respectivamente, la entrada y salida de la región crítica. Mientras se realice cualquiera de las operaciones (WAIT O SIGNAL), ningún otro proceso puede acceder al semáforo. El uso de semáforos en lenguajes de programación de alto nivel tiene ciertas desventajas, como:

a) Una tarea sólo puede esperar un semáforo a la vez, pero suele ser deseable permitir que una tarea espere cualquiera de varias señales.
b) Si una tarea no se señala en el punto apropiado (por ejemplo, a causa de un error de codificación), el sistema completo de tareas se puede trabar; esto es, que las tareas puedan estar esperando, cada una, en una cola de semáforos a que alguna

otra tarea dé la señal, de manera que no quedan tareas en ejecución.

c) Los programas en los que intervienen varias tareas y semáforos se vuelven cada vez más difíciles de entender, depurar y verificar.
d) Las semánticas de *signal* y *wait* implican que todas las tareas que tienen acceso al semáforo compartan memoria.

*Monitores*

Un monitor es una construcción de concurrencia que contiene los datos y procedimientos necesarios para poder realizar la asignación de un recurso compartido determinado o, de un grupo de recursos compartidos, para cumplir con la función de asignación de recurso, un procedimiento debe llamar a determinada entrada al monitor. Muchos de los procesos pueden tratar de entrar al monitor en diversos momentos, pero la exclusión mutua es aplicada rígidamente en los límites del monitor. Sólo se permite la entrada a un proceso a la vez y, los procesos que deseen entrar cuando el monitor se encuentra en uno, deben esperar. Esta espera es administrada de forma automática por el monitor; como la exclusión mutua está garantizada, se evitan problemas de concurrencia. (Tanenbaum, 1988)

Los datos contenidos en el monitor pueden ser globales (para todos los procedimientos dentro del monitor), o locales para un procedimiento específico. Los procesos pueden llamar a los procedimientos del monitor cuando lo deseen, para realizar las operaciones sobre los datos compartidos, pero no pueden acceder directamente a las estructuras de datos internas del monitor.

*Mensajes*

Para la comunicación se emplean las primitivas SEND (para enviar un mensaje) y RECEIVE (para poner al proceso en la espera de un mensaje). La comunicación puede ser sincrónica o asíncronica. (Pratt & Zelkowitz, 1998)

*Características de los procesos concurrentes*

- Interacción de procesos. Los programas concurrentes implican interacción entre los distintos procesos, están compuestos por los procesos que comparten recursos y compiten por el acceso a los recursos, y por los procesos que se comunican entre sí para intercambiar datos.

- Indeterminismo. Las acciones que se especifican en un programa concurrente tienen un orden parcial, dado que existe una incertidumbre sobre el orden exacto de concurrencia de ciertos sucesos, por lo que existe un indeterminismo en la ejecución. Siendo así, si se ejecuta un programa concurrente varias veces, puede producir resultados diferentes partiendo de los mismos datos.

- Gestión de recursos. Un proceso que desea utilizar un recurso compartido debe solicitar dicho recurso, debe esperar a adquirirlo, utilizarlo y, finalmente, liberar el recurso. Si el proceso solicita el recurso y no puede adquirirlo en ese momento, entonces es suspendido hasta que, dicho recurso, se encuentre disponible.

- Comunicación. La comunicación entre los procesos puede ser sincrónica, cuando los procesos requieren de una sincronización para intercambiar datos; o asincrónica, cuando un proceso que

suministra los datos no requiere esperar a que el proceso receptor los recoja, dado que los deja en un buffer de comunicación temporal.

- Violación de la exclusión mutua. En ocasiones, ciertas acciones que se realizan en un programa concurrente no proporcionan los resultados deseados, esto se debe a que existe una parte del programa en donde se realizan estas acciones que constituye una región crítica, es decir, es una parte del programa en la que se debe garantizar que si un proceso accede a la misma, ningún otro proceso podrá acceder; es necesario garantizar la exclusión mutua.

- Bloqueo mutuo o *deadlock*. Un proceso se encuentra en estado de *deadlock* cuando espera por un proceso que jamás ocurrirá. Se puede producir la comunicación de procesos y, más frecuentemente, en la gestión de recursos, existen cuatro condiciones necesarias para que se pueda producir un *deadlock:*
    a. Los procesos necesitan acceso exclusivo a los recursos.
    b. Los procesos necesitan mantener ciertos recursos exclusivos mientras esperan por otros.
    c. Los recursos no se pueden obtener de procesos que se encuentran esperando.
    d. Existe una cadena de procesos en la cual cada proceso posee uno o más de los recursos que necesita el siguiente proceso de la cadena.

# Capítulo 2. Programación en Paralelo


## EXTRACTO

En este segundo capítulo se abordarán conceptos que engloban todo aquello que se relacione con el paralelismo desde su concepto, los diferentes niveles y tipos de paralelismo que existen, las leyes que se encuentran relacionadas al paralelismo, el procesamiento de consultas, así como las arquitecturas de hardware en paralelo.

# 2 Aspectos básicos de la programación en paralelo

Los sistemas de computadora capaces de ejecutar varios programas en forma simultánea son ahora bastante comunes. Un sistema de multiprocesadores tiene varias unidades centrales de procesamiento (CPU) que comparten una memoria en común. Un sistema de computadoras distribuidas o en paralelo tiene varias computadoras, cada una con su propia memoria y CPU, conectadas con vínculos de comunicación en una red en la cual cada una se puede comunicar con las otras. En esta clase de sistemas, se ejecutar muchas tareas de manera simultánea.

Incluso en una sola computadora, suele ser útil proyectar un programa de manera que se componga de muchas tareas independientes que se ejecutan en forma concurrente en la computadora virtual, no obstante, en la computadora real sólo se puede estar ejecutando una, en un momento dado. La ilusión de ejecución concurrente en un solo procesador se obtiene intercalando la ejecución de las tareas individuales, de manera que, cada una ejecute una porción de su código, y así sucesivamente. Los sistemas operativos que manejan multiprogramación y tiempo compartido proporcionan este tipo de ejecución concurrente para programas de usuarios individuales. Lo que nos interesa es la ejecución concurrente de tareas dentro de un solo programa.

## 2.1 Principios de lenguajes de programación en paralelo

Las construcciones de programación en paralelo aumentan la complejidad del diseño del lenguaje, puesto que, varios procesadores pueden estar teniendo acceso a los mismos datos de manera

simultánea. Para el estudio del paralelismo resulta de suma importancia conocer los siguientes cuatro conceptos:

a) Definiciones de variables. Las variables pueden ser mutables o de definición. Las variables mutables son las variables comunes que se declaran en todos los lenguajes secuenciales. Se pueden asignar valores a las variables y cambiarse durante la ejecución del programa. A una variable de definición se le puede asignar un valor una sola vez. La ventaja de esta clase de variable es que no existen problemas de sincronización. Una vez asignado un valor cualquier tarea puede tener acceso a la variable y obtener el valor correcto.

b) Composición en paralelo. La ejecución avanza de un enunciado al siguiente. Además de los enunciados secuenciales y condicionales de los lenguajes de programación secuenciales es necesario agregar un enunciado en paralelo.

c) Estructura del programa. Los programas en paralelo se apegan en general a uno de los dos modelos de ejecución siguientes:

- Pueden ser transformativos cuando la meta es transformar los datos de entrada en un valor de salida apropiado. El paralelismo se aplica para acelerar el proceso; por ejemplo, en efectuar rápidamente la multiplicación de una matriz multiplicando varias secciones de la misma en paralelo.
- Pueden ser reactivos cuando el programa reacciona ante estímulos externos, llamados sucesos. Los sistemas en tiempo real y, de comando y control, son ejemplos de sistemas reactivos. Un sistema operativo y un sistema de procesamiento de transacciones, como un sistema de reservaciones, son ejemplos representativos de esta clase de sistemas reactivos. Se caracterizan por tener en general un comportamiento no

determinista, puesto que nunca es explícito el momento exacto en que va a ocurrir un suceso.

d) Comunicación. Los programas en paralelo se deben comunicar unos con otros. Esta comunicación tiene lugar típicamente a través de memoria compartida con objetos de datos comunes a los que tienen acceso todos los programas en paralelo o, por medio de mensajes, donde cada programa en paralelo tiene su propia copia del objeto de datos y pasa valores de datos entre los programas en paralelo.

### 2.1.1 Tareas

Cada tarea se considera como una dependiente de la tarea que inició. Cuando una tarea está lista para terminar, debe esperar hasta que todas sus dependientes hayan concluido para que pueda finalizar. Así, la división en múltiples secuencias de ejecución se invierte conforme las tareas terminan; fundiéndose cada vez en menos secuencias hasta que, finalmente, sólo queda en una secuencia. En circunstancias normales cada una de estas tareas de máximo nivel controla una parte importante del sistema (que actualmente suele ser un sistema de computadoras distribuidas) y, una vez iniciada, se espera que se ejecute para siempre.

### 2.1.2 Gestión de tareas

La definición de una tarea en un programa difiere de la definición de subprograma ordinario, excepto para definir cómo se sincroniza la tarea y de qué manera se comunica con otros programas. La mayor

parte del cuerpo de una definición de tarea contiene declaraciones y enunciados ordinarios que se ocupan del procedimiento que la tarea lleva a cado mientras trabaja en forma independiente de otras tareas.

Una vez que se inicia una tarea, los enunciados de su cuerpo se ejecutan en orden, igual que para un subprograma ordinario. Cuando una tarea termina, no regresa el control. Su secuencia individual de ejecución en paralelo simplemente concluye. Sin embargo, una tarea no puede terminar hasta que sus dependientes hayan terminado y, cuando termina, cualquier tarea de la cual sea dependiente deberá ser notificada respecto a ello para que esa tarea también pueda terminar. Una tarea finaliza cuando completa la ejecución de los enunciados de su cuerpo; una tarea que nunca finaliza se escribe de modo que contenga una iteración infinita que avanza en ciclos continuamente (hasta que ocurre un error).

### 2.1.3 Sincronización de tareas

Cuando se ejecutan de forma concurrente varias tareas, cada una de éstas procede de forma asíncrona respecto a las otras; es decir, que cada tarea se ejecuta a su propia velocidad, en forma independiente de las demás. Por ejemplo, se tiene una tarea A, la cual ha ejecutado 10 enunciados, y se tiene una tarea B, la cual se inició al mismo tiempo, puede haber ejecutado o no algún enunciado, o puede haberse ejecutado ya hasta su conclusión y estar terminada.

Para que dos tareas que se ejecutan de forma asíncrona coordinen sus actividades, el lenguaje debe suministrar un medio de sincronización, de modo que una tarea puede avisar a otra cuando completa la ejecución de una sección particular de su código. Por

ejemplo, una tarea puede estar controlando un dispositivo de entrada y la segunda, estar procesando cada lote de datos conforme es introducido desde el dispositivo. La primera tarea lee en un lote de datos, indica a la segunda que ha llegado el lote y luego inicia la preparación para introducir el lote siguiente de datos. La segunda tarea espera la señal de parte de la primera tarea, luego procesa los datos, después indica a la primera que ha completado el procesamiento y, entonces, espera otra vez la señal de que ha llegado otro lote. Las señales que se envían entre las tareas permite que éstas sincronicen sus actividades para que la segunda no comience a procesar datos antes de que la primera haya terminado de leerlos e introducirlos y, para que la primera no sobrescriba datos que la segunda todavía está procesando.

Las tareas que están sincronizando sus actividades en esta forma son un poco similares a corrutinas, pues las señales sirven para decir a cada tarea cuándo debe esperar y cuándo seguir adelante; un poco, como el uso de llamadas de reanudación entre corrutinas para indicar a una corrutina que debe proceder. Sin embargo, con las corrutinas sólo hay un orden de ejecución, mientras que aquí, puede haber varios. (Tanenbaum, 1988)

*Interrupciones*

La interrupción es una señal recibida por el procesador de un ordenador, indicando que debe "interrumpir" el curso de ejecución actual y pasar a ejecutar un código específico para tratar esta situación.

Una interrupción supone la ejecución temporal de un programa, para pasar a ejecutar una "subrutina de servicio de interrupción", que pertenece al BIOS (*Basic Input Output System*).

La sincronización de tareas concurrentes, a través del uso de interrupciones, es un mecanismo común que se encuentra en hardware de computadoras.

*Semáforos*

Un semáforo es un objeto de datos que se usa para sincronizar dos o más tareas, de modo que, su ejecución se realice de forma ordenada y sin conflictos entre las tareas. Los semáforos llevan una cuenta de las señales que se envían los procesos. Se compone de dos partes:

- Un contador de enteros, cuyo valor es siempre positivo o cero (1 ó 0); éste se utiliza para contar el número de señales enviadas, pero que aún no han sido recibidas.
- Una cola de tareas que está esperando el envío de señales.

Un semáforo binario es un indicador de condición (S) que registra si un recurso está disponible o no, por lo tanto, en un semáforo binario, el contador sólo puede tener los valores de cero y uno. Si, para un semáforo binario, S=1, entonces el recurso está disponible y la tarea lo puede utilizar; si, S=0, el recurso no está disponible y el proceso debe esperar.

En un semáforo general, el contador puede adoptar cualquier valor positivo entero. El semáforo binario resulta adecuado cuando hay que proteger un recurso que pueden compartir varios procesos, pero cuando lo que hay que proteger es un conjunto de recursos similares, se puede usar una versión más general del concepto de semáforo, que lleve la cuenta del número de recursos disponibles. En este caso,

el semáforo se inicia con el número total de recursos disponibles (n) y, las operaciones de espera y señal, se diseñan de modo que se impida el acceso al recurso protegido por el semáforo cuando el valor de éste es menor o igual que cero.

Cada vez que se solicita y obtiene un recurso, el semáforo se decrementa y se incrementa cuando se libera uno de ellos. Si la operación de espera se ejecuta cuando el semáforo tiene un valor menor que uno, el proceso debe quedar en espera de que la ejecución de una operación señal, libere alguno de los recursos.

Al igual que en los semáforos binarios, la ejecución de las operaciones son indivisibles, esto es, una vez que se ha empezado la ejecución de uno de estos procedimientos, se continuará hasta que la operación se haya completado.

Se definen dos operaciones primitivas para un objeto de datos semáforo P

- *Signal* (P). Cuando es ejecutada por una tarea A, esta operación prueba el valor del contador en P; si es cero, entonces la primera tarea de la cola de tareas se quita de la fila y se reanuda su ejecución; si no es cero o si la cola está vacía, entonces el contador se incremente en uno (lo que indica que se ha enviado una señal, pero todavía no se ha recibido). En cualquier caso, la ejecución de la tarea A continúa después de que se ha completado la operación *signal*.

- *Wait* (P). Cuando es ejecutada por una tarea B, esta operación prueba el valor del contador en P; si es diferente de cero, entonces el valor del contador se reduce en uno (lo cual indica que B ha recibido una señal).

Tanto *signal* como *wait* tienen una semántica simple que requiere el principio de atomicidad; cada operación completa su ejecución antes de que cualquier otra operación concurrente pueda tener acceso a sus datos. La atomicidad impide que ocurran ciertas clases de sucesos no deterministas indeseables.

*Mensajes*

Un mensaje es una transferencia de información de una tarea a otra. Proporciona un medio para que cada tarea sincronice sus acciones con otra tarea y, sin embargo, la tarea permanece libre para continuar ejecutándose cuando no necesita estar sincronizada. El concepto básico es similar a un tubo; un comando *send* (enviar) coloca un mensaje en el tubo, en tanto que, una tarea que espera un mensaje emite un comando *receive* (recibir) y acepta un mensaje proveniente del otro extremo del tubo. La tarea que envía está libre para continuar ejecutándose, enviando más mensajes y llenando la cola de mensajes, mientras que, la tarea receptora continuará ejecutándose en tanto existan mensajes pendientes en espera de ser procesados.

*Regiones críticas*

Una región crítica es una serie de enunciados de un programa dentro de una tarea, donde la tarea está operando sobre cierto objeto de datos que comparte con otras tareas.

Las regiones críticas se pueden implementar en las tareas asociando un semáforo con cada objeto de datos compartido. Los objetos de datos compartidos se hacen parte ordinariamente de un ambiente común explícito que es accesible para cada tarea.

*Monitores*

Un monitor es un objeto de datos compartido, junto con el conjunto de operaciones capaces de manipularlo; un monitor es similar a un objeto de datos definido por un tipo de datos abstracto. Una tarea sólo puede manipular el objeto de datos compartido usando las operaciones definidas, de modo que, el objeto de datos esté encapsulado. Otro enfoque para exclusión mutua es mediante el uso de un monitor; para hacer valer la exclusión mutua sólo es necesario exigir que, cuando mucho, una de las operaciones definidas para el objeto de datos pueda estar en ejecución en cualquier momento dado.

El requisito de exclusión mutua y encapsulamiento en un monitor hace que resulte natural representar al monitor mismo como una tarea. El objeto de datos compartido se convierte en un objeto de datos local dentro de la tarea y, las operaciones se definen como subprogramas locales dentro de la tarea.

*Paso de Mensajes*

Otra solución al problema de datos compartidos entre tareas, consiste en prohibir objetos de datos compartidos y proporcionar sólo la compartición de valores de datos mediante el paso de valores como mensajes.

El uso del paso de mensajes como base para compartir datos asegura la exclusión mutua sin mecanismos especiales, puesto que, cada objeto de datos es propiedad exactamente de una sola tarea, y ninguna otra tarea puede tener acceso directo al objeto de datos.

## 2.2 Programación en Paralelo

La programación en paralelo es una técnica de programación basada en la ejecución simultánea, ya sea en un mismo ordenador (con uno o varios procesadores) o en un clúster de ordenadores. A diferencia de la programación concurrente, en la programación en paralelo se enfatiza la verdadera simultaneidad en el tiempo de la ejecución de las tareas.

El uso de varios procesadores al trabajar en conjunto para resolver una tarea en común, cada procesador trabaja una porción del problema, así, los procesos pueden intercambiar datos a través de la memoria o por una red de interconexión.

Los sistemas multiprocesador consiguen un aumento del rendimiento si se utilizan estas técnicas.

En los sistemas de monoprocesador el beneficio en rendimiento no es tan evidente, pues el CPU es compartido por múltiples procesos en el tiempo.

Una gran problemática de este tipo de programación es la complejidad al sincronizar unas tareas con otras, ya sea mediante secciones críticas, semáforos o paso de mensajes, para garantizar la exclusión mutua en las zonas del código que sea necesario.

*Necesidad de la programación en paralelo.*

La implementación de la programación en paralelo de sistemas sumamente grandes es de gran ayuda para dar solución a diversas problemáticas, tales como:

- Resolver problemas que no caben en un CPU.

- Resolver problemas que no se pueden concretar en un tiempo razonable.
- Poder ejecutar problemas mayores y de forma más rápida.
- Decrementa la complejidad de un algoritmo al usar varios procesadores.

## 2.3 Aspectos de la programación en paralelo

- Diseño de computadoras paralelas, teniendo en cuenta la escalabilidad y comunicaciones.
- Diseño de algoritmos eficientes, no hay ganancia si los algoritmos no se diseñan bien.
- Métodos para evaluar los algoritmos en paralelo: qué tan rápido dan solución a un problema y la eficiencia con la que se usan los procesadores.
- Los programas paralelos deben de ser portables y los compiladores paralelizables.

### 2.3.1 Evaluación de Algoritmos en paralelo

La evaluación de los algoritmos paralelos requiere de métricas de prestaciones que consideren el tamaño del problema a resolver (m) y el número de procesadores empleados para ello (Prc). Las métricas que se utilizan típicamente son:

- **Tiempo de Ejecución**, que se denota con la función:

$$T(m,Prc)= TA(m,Prc)tf +TC(m,Prc)$$

Donde:

*TA(m,Prc)* es el tiempo empleado por el algoritmo para realizar operaciones aritméticas.

*tf* es el tiempo de ejecución de una operación en punto flotante.

*TC(m,Prc)* es el tiempo empleado por el algoritmo para realizar comunicaciones entre procesadores.

En el modelo de paso de mensajes, el tiempo de comunicación de un mensaje de longitud m, de un nodo a otro, en una red de procesadores se expresa como:

$$TC(m,Prc) = tm + mtv$$

Donde:

*tm es el* tiempo de establecimiento de comunicación entre los nodos (latencia).

*tv* es el tiempo de transferencia de un mensaje de longitud 1 (inverso del ancho de banda).

- **Aceleración de ejecución** (*Speedup*), mide la ganancia de velocidad de ejecución del algoritmo paralelo con respecto al mejor algoritmo secuencial que resuelve el mismo problema. El *Speedup* está dado por:

$$S(m,Prc) = T(m,1)/T(m,Prc)$$

Idealmente se espera que el *Speedup* sea igual al número de procesadores empleados.

- **Eficiencia**, mide el porcentaje del tiempo de ejecución que los procesadores mantienen útilmente empleados. La Eficiencia se define como:

$$E(m,Prc) = S(m,Prc)/Prc$$

En donde la eficiencia ideal es igual a 1.

## 2.4 Arquitecturas de computadoras (Taxonomía de Flynn

La taxonomía de Flynn (Flynn, 1966) es una clasificación de arquitecturas de computadoras, la cual distingue entre instrucciones y datos, los cuales pueden ser simples o múltiples; fue propuesta por Michael J. Flynn en 1972. Las cuatro clasificaciones definidas por Flynn se basan en el número de instrucciones concurrentes (control) y en los flujos de datos disponibles en la arquitectura. En la siguiente tabla se muestra dicha clasificación.

| | ***Una Instrucción*** | ***Múltiples Instrucciones*** |
|---|---|---|
| ***Un Dato*** | SISD | MISD |
| ***Múltiples Datos*** | SIMD | MIMD |

**Tabla 2.1** Taxonomía de Flynn

### 2.4.1 Modelo SISD (*Single Instruction Single Data*)

Una unidad de procesamiento recibe una sola secuencia o flujo de instrucciones que operan en una secuencia de datos en una única memoria. Este tipo de sistemas son los monoprocesadores convencionales con arquitectura de von Neumann. La única posibilidad de efectuar paralelismo en este tipo de máquinas sería el paralelismo implícito.

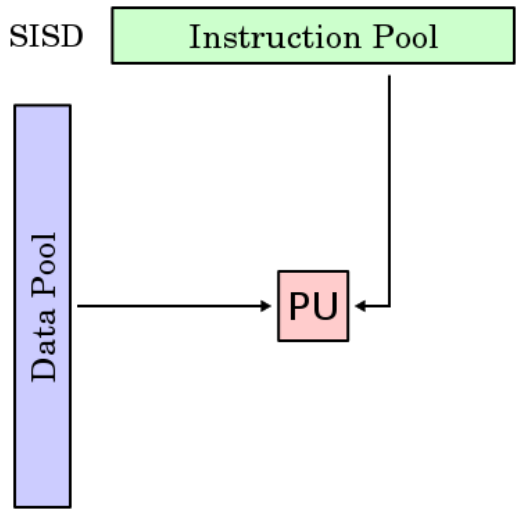



**Figura 2.1** Esquema de la arquitectura SISD

### 2.4.2 Modelo MISD (Multiple Instruction Single Data)

Existencia de N procesadores con un simple flujo de datos, ejecutando instrucciones diferentes en cada procesador (no se usa en la industria).

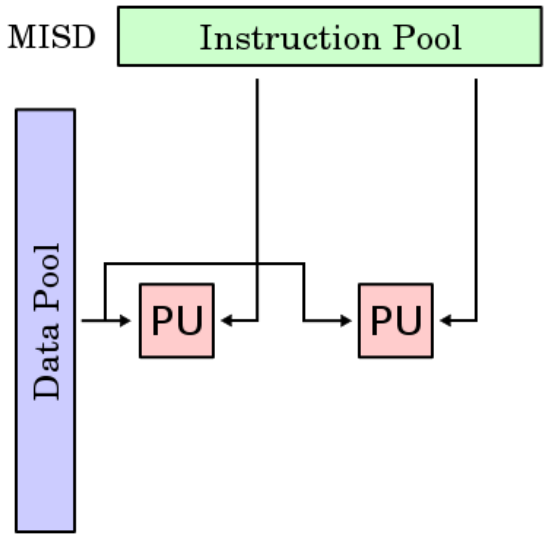



**Figura 2.2** Esquema de la arquitectura MISD

### 2.4.3 Modelo SIMD (*Single Instruction Multiple Data*)

Múltiples procesadores que, de forma sincronizada ejecutan la misma secuencia de instrucciones, pero en diferentes datos.

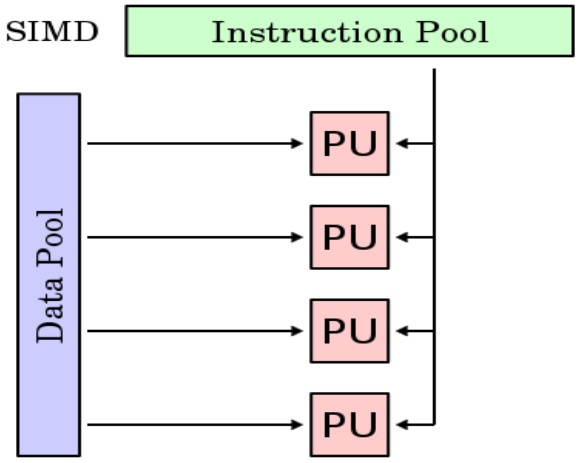



**Figura 2.3** Esquema de la arquitectura SIMD

### 2.4.4 Modelo MIMD (*Multiple Instruction Multiple Data*)

Cada procesador puede ejecutar su propia secuencia de instrucciones y tener sus propios datos; es asincrónico.

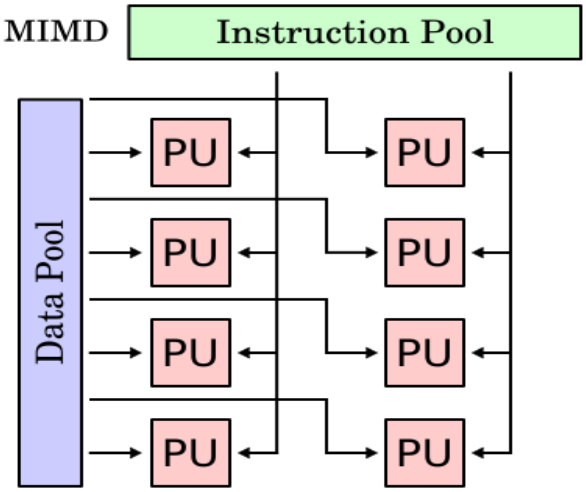



**Figura 2.4** Esquema de la arquitectura MIMD

## 2.5 Métodos para conectar los procesadores en paralelo

Existen diferentes métodos para poder conectar procesadores en paralelo, se abordarán los siguientes:

- Malla.

Arreglo de nodos en rejilla q-dimensional
Un nodo se comunica con 2q nodos.
No hay jerarquía.

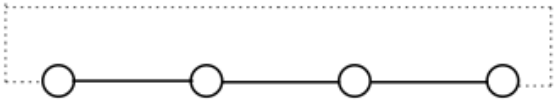


**Figura 2.5** Malla

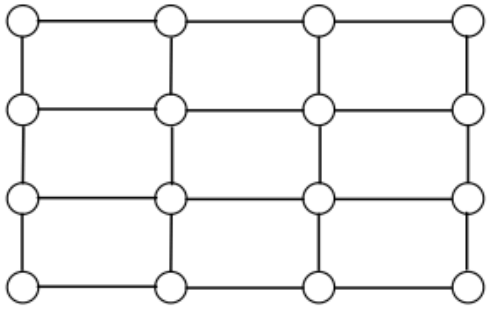


**Figura 2.6** Malla

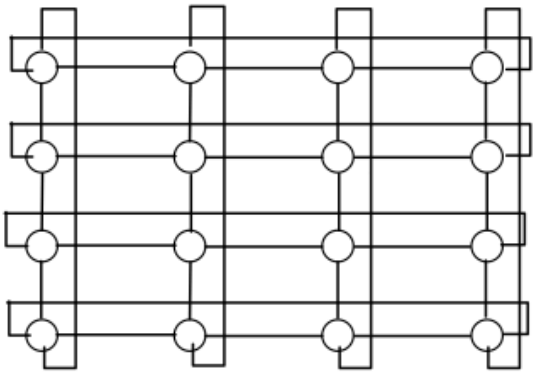


**Figura 2.7** Malla

- Árbol binario.

Existen $2^k$-1 nodos.

Con profundidad de k-1.

Un nodo tiene más de 3 ligas.

En el interior, un nodo se puede comunicar a lo más, con 2 nodos hijos y con el nodo padre.

Diámetro pequeño.

AB pobre.

Maestro-Esclavo.

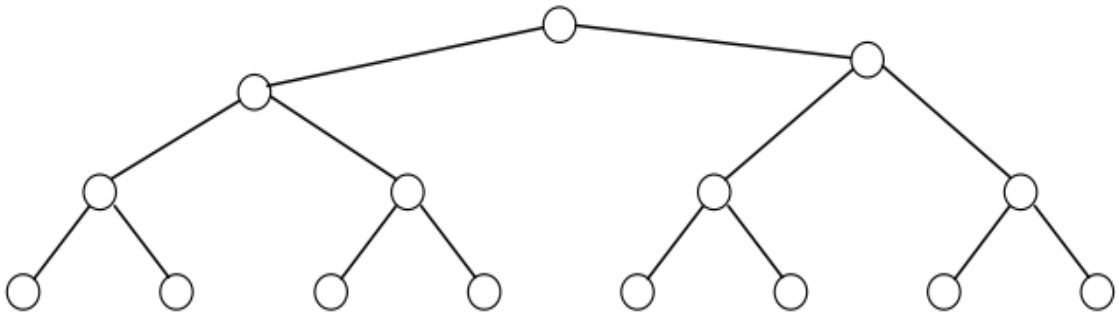


**Figura 2.8** Árbol binario

- Hiper-árbol

Unión de un árbol binario invertido y un árbol con k hijos.

Objetivo: mejorar el AB.

- Piramidal

Combinación de una malla y un árbol.

Reduce el diámetro con respecto a la malla.

El máximo número de ligas por nodo no es mayor a 9.

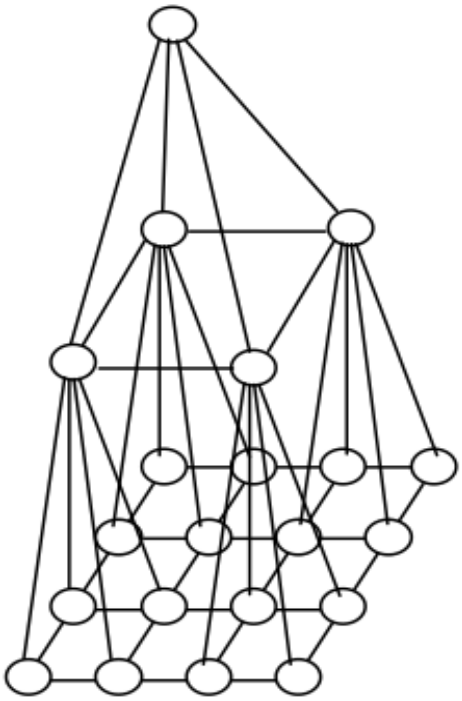


**Figura 2. 9** Piramidal

- Mariposa

Consiste de $(k+1)2^k$ nodos dividido en (k+1) renglones.

K es el rango cuyo valor está entre 0 y n.

Los nodos internos tienen 4 conexiones y los extremos sólo 2.

Las ligas forman un patrón de mariposa.

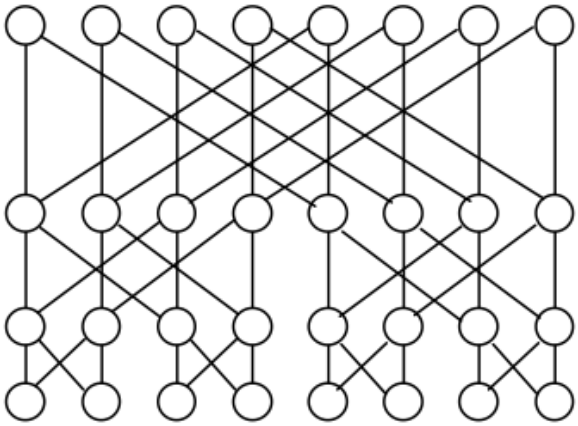


**Figura 2.10** Mariposa

- Hipercubo

Consta de $2^k$ nodos formando un hipercubo k-dimensional.

Los nodos se etiquetan de 0 a $2^k$-1.

Dos nodos son adyacentes si en la etiqueta varía un bit en una posición

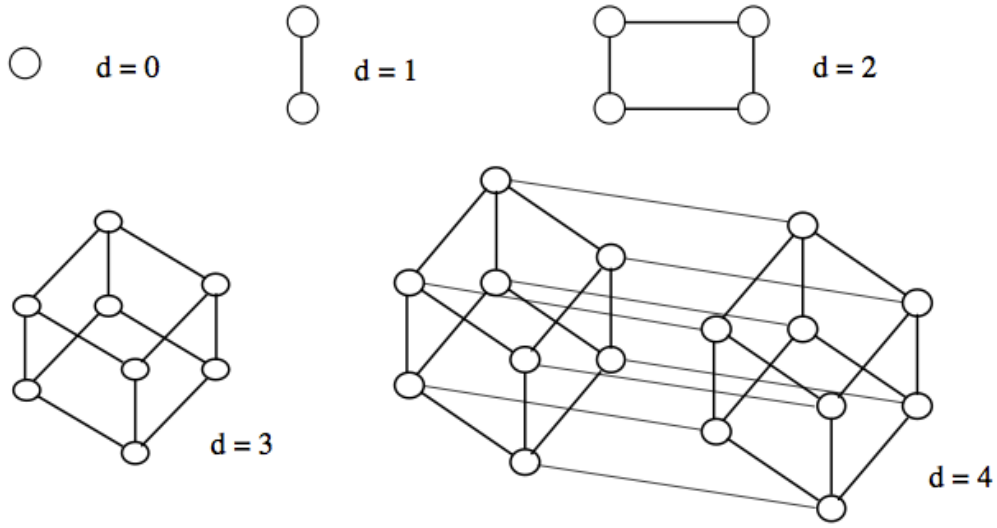



**Figura 2.11** Hipercubo

- Cubo cíclico

Es un hipercubo adimensional.

Existen k nodos por vértice.

Diámetro 2 veces mayor al hipercubo.

Ancho de bisección es menor que al hipercubo.

Criterios de organización.

a) Diámetro: máxima distancia entre dos nodos.

b) Ancho de bisección: es el número mínimo de bordes que deben ser removidos a fin de dividir la red en 2 mitades.

c) Número de bordes por nodo: si éste es un valor constante e independiente al arreglo, permite escalar fácilmente el sistema con un mayor número de nodos

d) Máximo de longitud de borde: por razones de escalabilidad es mejor que el valor de contornos y nodos puedan ser constantes.

## 2.6 Clasificación de computadoras (RISC y CISC)

De acuerdo al número de instrucciones que maneja el CPU de una computadora, podemos clasificar las computadoras en dos tipos, RISC y CISC.

RISC (*Reduced Instruction Set Computer*).

En la arquitectura computacional, RISC (Computadoras con un conjunto de instrucciones reducido) es un tipo de microprocesador con las siguientes características fundamentales:

1. Instrucciones de tamaño fijo y presentadas en un reducido número de formatos.
2. Sólo las instrucciones de carga y almacenamiento acceden a la memoria por datos.

Además, estos procesadores suelen disponer de muchos registros de propósito general.

El objetivo de diseñar máquinas con esta arquitectura es posibilitar la segmentación y el paralelismo en la ejecución de instrucciones y reducir los accesos a memoria.

RISC es una filosofía de diseño de CPU para computadora que está a favor de conjuntos de instrucciones pequeñas y simples que toman menor tiempo para ejecutarse. El tipo de procesador más utilizado comúnmente en equipos de escritorio, el x86, está basado en CISC en lugar de RISC, aunque las versiones más nuevas traducen instrucciones basadas en CISC x86 a instrucciones más simples basadas en RISC para uso interno antes de su ejecución.

La idea fue inspirada por el hecho de que muchas de las características que eran incluidas en los diseños tradicionales de CPU para aumentar la velocidad, estaban siendo ignoradas por los programas que eran ejecutados en ellas. Además, la velocidad del procesador en relación con la memoria de la computadora que accedía era cada vez más alta. Esto provocó la aparición de numerosas técnicas para reducir el procesamiento dentro del CPU, así como el hecho de reducir el número total de accesos a memoria.

CISC (*Complex Instruction Set Computer*).

En la arquitectura computacional, CISC (Computadoras con un conjunto de instrucciones complejo) es un modelo de arquitectura de computadora. Los microprocesadores CISC tienen un conjunto de instrucciones que se caracteriza por ser muy amplio y permitir operaciones complejas entre operandos, situados en la memoria o en los registros internos, en contraposición a la arquitectura RISC.

Este tipo de arquitectura dificulta el paralelismo entre instrucciones, por lo que, en la actualidad, la mayoría de los sistemas CISC de alto rendimiento implementan un sistema que convierte

dichas instrucciones complejas en varias instrucciones simples del tipo RISC, llamadas generalmente, microinstrucciones.

Los CISC pertenecen a la primera corriente de construcción de procesadores, antes del desarrollo de los RISC.

# Capitulo 3 Paralelismo.

## EXTRACTO

En este segundo capítulo se abordarán conceptos que engloban todo aquello que se relacione con el paralelismo desde su concepto, los diferentes niveles y tipos de paralelismo que existen, las leyes que se encuentran relacionadas al paralelismo, el procesamiento de consultas, así como las arquitecturas de hardware en paralelo.

# 3 Cómputo en paralelo

El cómputo en paralelo es definido como el uso simultáneo de más de un procesador para resolver un problema, explotando la programación en paralelo al acelerar el tiempo de ejecución.

En estos días existe una creciente demanda para obtener computadoras que sean mucho más rápidas que las existentes, esto es, para dar una solución en un menor tiempo a todas aquellas peticiones que son requeridas en distintos entornos de aplicación de tipo científico, médico, social, empresarial, etc., pues es necesario depurar grandes volúmenes de cálculo a velocidades mayores y con cálculos precisos. Por tal motivo, las prestaciones de las computadoras de la actualidad mejoran de forma notable y continua, haciendo que la velocidad de las computadoras vaya en aumento conforme a las necesidades del hombre y, con el paso del tiempo se tenga una evolución tecnológica. Como es de esperarse, a pesar del gran avance tecnológico que se tiene, siempre existirán limitantes físicos en el aumento de la velocidad de las computadoras, tales como, la velocidad de la luz, la disipación de calor en los circuitos integrados, etc. Es por eso que todas estas limitantes físicas hacen que haya una búsqueda exhaustiva en obtener otras alternativas para aumentar las prestaciones que las máquinas ofrecen y, no depender solamente del aspecto físico de las mismas, obteniendo así, resultados favorables al encontrar alternativas como lo es el cómputo en paralelo.

## 3.1 El Paralelismo

El paralelismo es un aspecto de la computación en el cual un problema puede ser realizado de forma simultánea, esto es, basado en el principio de poder dividir un problema grande y así, obtener varios problemas pequeños, los cuales son solucionados de forma paralela posteriormente.

Existen diferentes niveles de paralelismo:

a) A nivel de bit : es una forma del cómputo en paralelo basado en el incremento del tamaño del procesador de palabra. El aumento del tamaño de la palabra reduce el número de instrucciones que el procesador debe ejecutar para realizar una operación en variables, cuyo tamaño es mayor que la longitud de la palabra.

b) A nivel de instrucción: es la medida de cuántas operaciones, en un programa de computadora, pueden ser realizadas de manera simultánea.

c) A nivel de datos: consiste en subdividir el conjunto de datos de entrada a un programa, de tal forma que a cada procesador le corresponda un subconjunto de esos datos. Cada procesador realizará la misma secuencia de operaciones que los otros procesadores sobre su subconjunto de datos asignado.

d) A nivel de tarea: consiste en asignar distintas tareas a cada uno de los procesadores de un sistema de cómputo. En consecuencia, cada procesador efectuará su propia secuencia de operaciones.

El paralelismo se da cuando dos o más procesos o hilos se ejecutan al mismo tiempo, con el paso del tiempo y el avance tecnológico que se tiene, podemos encontrar dos tipos de paralelismo para arquitecturas de microprocesadores con varios núcleos:

a) Múltiples procesos. Comunicación a través de IPCs (*Inter-Process Communication*).

b) Un solo proceso múltiples hilos. Comunicación a través de memoria compartida.

Es muy importante saber que en el paralelismo existen limitantes y una de sus principales limitantes es que los procesos tienen una parte, la cual no es posible paralelizar, además, hay que tomar en cuenta que para paralelizar un proceso se requiere de dos factores que son el software adicional y la comunicación entre procesadores, por tal razón, resulta imprescindible saber que estos factores suponen pérdidas de tiempo añadidas.

## 3.2 Rendimiento de las computadoras en paralelo

La velocidad con que operan las computadoras es medida por el número de operaciones básicas que pueden realizar por unidad de tiempo. Pero, dado que en ocasiones existen diferencias entre las computadoras, resulta imposible contar con una sola unidad para medir el rendimiento de las mismas; por tal motivo se cuenta con varias unidades para poder realizar este tipo de mediciones. Una unidad para medir esta velocidad es el MIPS (1 millón de instrucciones/segundo), sin embargo, este tipo de unidad tiene el

inconveniente de no ser homogénea, puesto que las instrucciones de un procesador pueden ser mucho más potentes que las de otro.

Existe otra unidad, la cual pude resultar ser más fiable, el MFLOPS (1 millón de instrucciones de punto flotante/segundo) que, aunque en menor medida, también padece del mismo defecto que la unidad anterior. Otra unidad alternativa que también es utilizada es el VUP (VAX Unit Performance), y debe su nombre a la computadora VAX-11/780, en la que se encuentra basada, y toma como unidad la velocidad de dicha computadora. En la actualidad, para conocer el rendimiento de este tipo de computadoras se utilizan unos programas de prueba específicos denominados *benchmarks*; en donde el tiempo de ejecución de estos programas nos dará una medida del rendimiento de un sistema. Para medida del CPU, el que se utiliza actualmente es el SPEC CPU2006 (por sus siglas, Standard Performance Evaluation Corporation).

Una medida bastante utilizada para medir la eficiencia de un sistema y, que sólo afecta a parámetros arquitectónicos del mismo sin entrar en la calidad de la tecnología empleada, es el CPI (ciclos por instrucción), el cual es el número medio de ciclos de reloj necesario para ejecutar una instrucción. En otras palabras, se describe de la siguiente manera:

$$\mathrm{CPI} = \frac{\text{Número de ciclos de reloj consumidos}}{\text{Número de instrucciones ejecutadas}}$$

Entonces, si un programa tiene *n* instrucciones, su tiempo de ejecución vendrá dado por:

$$\mathrm{t} = \mathrm{CPI} \cdot \mathrm{n} \cdot \frac{1}{\mathrm{f}}$$

Donde:

*f* representa la frecuencia del reloj, por lo tanto, su inverso es el tiempo de ciclo de reloj.

Una forma de medir la calidad de un sistema paralelizado consiste en comparar la velocidad conseguida con el sistema paralelo (con N procesadores), con la velocidad conseguida de un solo procesador. Para ello, se definirá la ganancia de velocidad de un sistema de N procesadores de la siguiente forma:

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

Donde:

*t(1)* es el tiempo empleado para ejecutar el proceso si se ejecuta en un solo procesador.
*t(N)* es el tiempo empleado para ejecutarlo en el sistema paralelo con N procesadores.

En condiciones ideales,

$$t(N) = \frac{1}{N},$$

con lo que, en esas mismas condiciones, la ganancia de velocidad dada por la ecuación anterior será:

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

Una forma de medir el rendimiento del sistema será, comparar la ganancia de velocidad del sistema con la ganancia de velocidad ideal, a esta medida se la denomina eficiencia y vendrá dada por:

$$E(N) = \frac{S(N)}{S(N)ideal} = \frac{S(N)}{N} = \frac{t(1)/t(N)}{N} = \frac{t(1)}{Nt(N)};$$

Esta fórmula indica la medida en que se aprovechan los recursos.

## 3.3 Ley de Amdahl

El aumento de rendimiento que puede ser obtenido al mejorar alguna parte de la computadora y, puede ser calculado mediante el uso de la Ley de Amdahl (Amdahl, 1967), la cual establece que, la mejora obtenida en el rendimiento de un sistema debido a la alteración de uno de sus componentes o al utilizar un modo de ejecución más rápido, se encuentra limitada por la fracción de tiempo que utiliza dicho componente, o que pueda utilizar ese modo de ejecución más rápido.

La fórmula original de la ley de Amdahl es la siguiente:

$$F = Fa \cdot \left( (1 - Fm) + \frac{Fm}{Am} \right)$$

Donde:

*F* es el tiempo de ejecución mejorado.
*Fa* es el tiempo de ejecución antiguo.

Esta fórmula puede ser reescrita utilizando la definición del incremento de la velocidad que viene dado por:

$$A = \frac{Fa}{F}$$

Donde:
*A* es la aceleración (*speedup*).
*Fa* es el tiempo de ejecución antiguo.
*F* es el tiempo de ejecución mejorado.

La aceleración indica la rapidez con que se realizará una tarea utilizando una computadora con la mejora respecto a la máquina original. La ley de Amdahl da una forma rápida de calcular la aceleración, la cual depende de dos factores:

- La fracción del tiempo que pueda utilizarse para aprovechar la mejora. Este valor, que se llama fracción mejorada, es siempre menor o igual que 1.
- La optimización lograda por el modo de ejecución mejorado, es decir, cuán más rápido se ejecutará la tarea si solamente se utiliza el modo mejorado. Este valor es el tiempo del modo original con respecto al tiempo del modo mejorado, el valor es siempre mayor que 1 y se llama aceleración mejorada.

Por lo que la formula anterior se puede reescribir de la siguiente forma:

$$A = \frac{1}{(1 - Fm) + \frac{Fm}{Am}}$$

Donde:

*A* es la aceleración o ganancia en velocidad conseguida en el sistema completo debido a la mejora de uno de sus subsistemas.
*Am* es el factor de mejora que se ha introducido en el subsistema mejorado.
*Fm* es la fracción de tiempo que el sistema utiliza del subsistema mejorado.

La ley de Amdahl describe el límite máximo de aceleración (*speedup*) con ejecución paralela, siendo así, un modelo general del desempeño de un procesador en paralelo. Es importante saber que no todos los programas pueden ser ejecutados en paralelo.

## 3.4 Ley de Gustafson-Barsis

La ley de Gustafson-Barsis (Gustafson, 1988) establece que un problema grande, con un conjunto de datos repetitivos puede ser eficientemente paralelizado. Esta ley se encuentra ligada a la Ley de Amdahl, la cual pone límite a la mejora del *Speedup* (aceleración), que puede tener debido a la paralelización. La Ley de Gustafson-Barsis ofrece una visión positiva de las ventajas acerca del procesamiento paralelo y es descrita de la siguiente forma:

$$S(P) = P - \propto \cdot (P - 1)$$

Donde:

*P* es el número de procesadores.
*S* es el speedup.

$\propto$ es la parte no paralelizable del proceso.

La ley de Gustafson-Barsis (Gustafson, 1988) propone que los programadores fijen el tamaño de los problemas para utilizar el equipo disponible al solucionar problemas dentro de un tiempo fijo práctico. Por lo tanto, si un equipo más rápido está disponible, problemas más grandes se pueden solucionar en el mismo tiempo.

*Implementación de la Ley de Gustafson-Barsis*

Siendo *n* la medida del tamaño del problema. La ejecución del programa en una computadora en paralelo se descompone en:

$$a(n) + b(n) = 1$$

Donde:

*a* es la fracción secuencial.
*b* es la fracción paralelo.
(Ignorando la sobrecarga por ahora.)

En una computadora secuencial el tiempo relativo sería:

$$a(n) + \ p \cdot b(n)$$

Donde:

*p* es el número de procesadores en el caso paralelo.

El *speedup* entonces es:

$$\big(a(n) + p \cdot b(n)\big)\ (\text{paralelo, en comparación al secuencial } a(n) + b(n) = 1)$$

Por lo tanto:

$$S = a(n) + p \cdot (1 - a(n))$$

Donde:

*a(n)* es la función serial.

Al asumir que la función serial *a(n)* disminuye con el tamaño n del problema, entonces el *speedup* se aproxima a *p* como *n* se aproxima a infinito, según lo deseado.

La ley de Gustafson-Barsis sostiene que incluso usando sistemas masivos de computadoras en paralelo no influye en la parte serial y se ve esta parte como constante, en comparación con lo que la hipótesis de la Ley de Amdahl resulta de la idea de que, la influencia de la parte serial, incrementa con el número de procesadores.

## 3.5 Dependencia entre procesos

Para que varios procesos (refiriéndose como proceso a cualquier fragmento de un programa en ejecución y de cualquier nivel descrito previamente en el punto 3.2) puedan ejecutarse de forma paralela, se requiere que los procesos sean independientes entre sí. Sin embargo, no siempre será posible que este tipo de ejecuciones se

realicen de forma independiente, dado que existen dependencias entre los procesos.

Estas dependencias pueden tener diferentes formas y son descritas a continuación:

a) ***Dependencia de datos:*** este tipo de dependencia hace referencia a situaciones en el que, de acuerdo al orden en que los datos son obtenidos, marca el orden en que se ejecutarán los procesos. Existes varios tipos de dependencia de datos y son los descritos a continuación:

- *Dependencia de flujo:* esta dependencia se da cuando un proceso $P_2$ es dependiente por flujo de otro proceso $P_1$, si $P_2$ sigue en el orden de programa a $P_1$ y una salida del proceso $P_1$ se utiliza como entrada en el proceso $P_2$.
- *Anti-dependencia:* esta dependencia se da cuando un proceso $P_2$ es anti-dependiente de otro proceso $P_1$, si $P_2$ sigue en el programa a $P_1$ y una salida de $P_2$ es utilizada como entrada en $P_1$.
- *Dependencia de salida:* esta dependencia se da cuando dos procesos son dependientes por la salida si escriben en la misma variable.
- *Dependencia de entrada/salida:* este tipo de dependencia se origina cuando dos procesos acceden a un mismo fichero o dispositivo para leer o escribir datos.
- *Dependencias desconocidas:* este tipo de referencia se da cuando existen situaciones en que hay peligro de dependencias de datos, pero no son posibles conocerlas antes de la ejecución. Un ejemplo de este tipo de dependencia se presenta en las instrucciones que afectan a

variables cuyos subíndices también se encuentran sub-indexados.

b) ***Dependencia de control:*** este tipo de dependencia se origina cuando no es posible conocer el orden de ejecución de los procesos, sino hasta el momento en el que se realiza la ejecución, lo cual genera una afectación a la ejecución de instrucciones o procedimientos afectados por instrucciones condicionales.

c) ***Dependencia por recursos:*** este tipo de dependencia es producida debido a conflictos en el uso de recursos compartidos, tales recursos pueden ser los de almacenamiento; como las áreas de memoria compartida, recursos de ejecución, unidades aritméticas, etc.

### 3.5.1 Condiciones de Bernstein

A. J. Bernstein definió unas condiciones (Bernstein, 1966) con las cuales es posible determinar si dos conjuntos de instrucciones son posibles de ejecutar en forma concurrente. Como se ha descrito con anterioridad, resulta imposible que todas las partes de un programa sean ejecutadas de forma concurrente.

Antes de abordar las condiciones que Bernstein, es indispensable complementar este punto mediante un ejemplo previo, compuesto por dos fragmentos de programa.

- Si se considera el fragmento de programa siguiente:

```
x:= x+1;
y:= x+2;
```

Se puede observar que la primera sentencia se debe ejecutar antes que la segunda.

- Ahora, considerando el siguiente fragmento de programa:

```
x:= 1;
y:= 2;
z:= 3;
```

Al observar este segundo fragmento de programa, podemos decir que el orden en el que se ejecuten cada una de las sentencias, no afectará el resultado final.

Ahora, en el supuesto caso de que tuviésemos tres procesadores, sería posible ejecutar cada una de las sentencias en cada uno de los procesadores, de tal forma que, la velocidad del sistema se incrementaría. Una vez que se han explicado los dos casos anteriores y se ha analizado la diferencia que existe entre cada uno de los fragmentos de programa, uno puede distinguir en dónde es posible ejecutar de forma concurrente y en dónde resulta imposible ejecutar de forma concurrente. Por tal razón, el ejemplo previo nos resulta de gran importancia para comprender las condiciones de Berstein, las cuales son útiles para determinar si dos conjuntos de procesos $P_i$ *y* $P_j$ son posibles de ejecutar de manera paralela.

Para que sea posible determinar si dos conjuntos de procesos pueden ser ejecutados de forma concurrente es indispensable definir previamente los siguientes conjuntos:

- $R(P_k) = \{a_1,a_2,\ldots,a_n\}$, definido como el conjunto de lectura o salida del conjunto de procesos $P_k$, el cual se encuentra formado por todas las variables, cuyos valores son referenciados durante la ejecución de los procesos en $P_k$.

- $W(P_k) = \{b_1, b_2, \ldots, b_n\}$, definido como el conjunto de escritura o salida del conjunto de procesos $P_k$, que se encuentra formado por las variables cuyos valores son actualizados durante la ejecución de los procesos en $P_k$.

Tanto el conjunto de lectura como el conjunto de escritura se encuentran formados por operandos que pueden encontrarse en los registros del procesador, en la memoria e, inclusive, en los ficheros.

Para que dos conjuntos de procesos $P_i$ *y* $P_j$ puedan ser ejecutados de forma concurrente, es necesario que se cumplan las siguientes ecuaciones:

a) $R(Pi) \cap W(Pj) = \emptyset$

b) $W(Pj) \cap R(Pi) = \emptyset$

c) $W(Pi) \cap W(Pj) = \emptyset$

Las condiciones de Bernstein son similares a las ecuaciones descritas anteriormente, sólo que, se encuentran expresadas de forma distinta; éstas expresan los diferentes tipos de dependencias de datos en los accesos a memoria en la misma ubicación, son las siguientes:

$$O_1 \cap I_2 = \emptyset$$
$$I_1 \cap O_2 = \emptyset$$
$$O_1 \cap O_2 = \emptyset$$

Existe una dependencia de datos cuando dos accesos a la memoria pueden hacer referencia a la misma posición de memoria y uno de los accesos, es una escritura.

La dependencia de datos se clasifica en de la siguiente forma:

- Dependencia verdadera (leer después de escribir): el primer acceso almacena en una ubicación que será leída después por el segundo acceso. X=…; …=X;
- Anti-dependencia (escribir después de leer): el primer acceso lee una ubicación en la cual el segundo acceso almacenará después. …=X; X=…;
- Dependencia de salida (escribir después de escribir): ambos accesos escriben en una ubicación y el orden de escritura debe ser preservado de modo tal que, en cualquier acceso posterior se lea el valor correcto. X=…; X=…;

A través del siguiente ejemplo se explicará cómo es que son aplicadas las condiciones de Bernstein, en el cual tenemos los siguiente procesos:

$$P1 \rightarrow a := x + y;$$
$$P2 \rightarrow b := z - 1;$$
$$P3 \rightarrow c := a - b;$$
$$P4 \rightarrow w := c + 1;$$

Primero se calculan los conjuntos de lectura y escritura, que son los siguientes:

$$R(P_1) = \{x, y\}$$
$$R(P_2) = \{z\}$$
$$R(P_3) = \{a, b\}$$
$$R(P_4) = \{c\}$$
$$W(P_1) = \{a\}$$
$$W(P_2) = \{b\}$$

$W(P_3) = \{c\}$

$W(P_4) = \{w\}$

Al aplicar las condiciones de Bernstein a cada par de sentencias, obtenemos lo siguiente:

Entre $P_1$ y $P_2$:

$R(P_1) \cap W(P_2) = \emptyset$

$W(P_1) \cap R(P_2) = \emptyset$

$W(P_1) \cap W(P_2) = \emptyset$

Entre $P_1$ y $P_3$:

$R(P_1) \cap W(P_3) = \emptyset$

$W(P_1) \cap R(P_3) = a \neq \emptyset$

$W(P_1) \cap W(P_3) = \emptyset$

Entre $P_1$ y $P_4$:

$R(P_1) \cap W(P_4) = \emptyset$

$W(P_1) \cap R(P_4) = \emptyset$

$W(P_1) \cap W(P_4) = \emptyset$

Entre $P_2$ y $P_4$:

$R(P_2) \cap W(P_4) = \emptyset$

$W(P_2) \cap R(P_4) = \emptyset$

$W(P_2) \cap W(P_4) = \emptyset$

Entre $P_2$ y $P_3$:

$R(P_2) \cap W(P_3) = \emptyset$

$W(P_2) \cap R(P_3) = b \neq \emptyset$

$W(P_2) \cap W(P_3) = \emptyset$

Entre $P_3$ y $P_4$:

$R(P_3) \cap W(P_4) = \emptyset$

$W(P_3) \cap R(P_4) = c \neq \emptyset$

$W(P_3) \cap W(P_4) = \emptyset$

| | **$P_1$** | **$P_2$** | **$P_3$** | **$P_4$** |
|---|---|---|---|---|
| **$P_1$** | - | Sí | No | Sí |
| **$P_2$** | - | - | No | Sí |
| **$P_3$** | - | - | - | No |
| **$P_4$** | - | - | - | - |

**Tabla 3.1** Resultados de los pares de procesos que es posible ejecutar de forma paralela

### 3.5.2 Grafos de Dependencia de Datos

Un grafo de dependencia de datos es un una notación gráfica representada mediante grafos dirigidos (nodos y arcos), en donde los nodos representan los procesos y, los arcos describen las relaciones de dependencia de datos que existen entre ellos.

Mediante el uso de una flecha dirigida de un nodo A hasta un nodo B, se interpreta que el nodo B sólo se podrá ejecutar cuando el nodo A haya finalizado; en cambio, si aparecen dos nodos en paralelo, esto significa que pueden ejecutarse ambos nodos de forma paralela.

Para entender el concepto de grafos de dependencia de datos se realizará el siguiente ejemplo:

Partiendo de que debemos evaluar la siguiente expresión:

$$z = \frac{v * w - x}{v * w + y}$$

Para que un programa pueda resolver esta expresión debería realizarlo mediante la siguiente secuencia de procesos:

$P_1 a = v * w$

$P_2 b = a - x$

$P_3 c = a + y$

$P_4 z = \frac{b}{c}$

Cuando estos procesos son ejecutados en forma secuencial, se ejecutan uno por uno, pero es posible que no sean ejecutados en orden (primero el proceso $P_1$, luego el proceso $P_2$, seguido del proceso $P_3$ y, finalmente, el proceso $P_4$) pues los procesos $P_2$ y $P_3$ son posibles de intercambiar, esto quiere decir, ejecutar primero el proceso $P_3$ y después el proceso $P_2$; pues el orden en el que estos dos procesos se ejecuten no altera el resultado.

En cambio, si estos cuatro procesos son ejecutados de forma paralela, los procesos $P_2$ y $P_3$ pueden ser ejecutados concurrentemente, siempre y cuando, el proceso $P_1$ haya finalizado su ejecución. Esto es debido a la dependencia de datos que existe entre los procesos $P_1$ y, los procesos $P_2$ y $P_3$. En cambio, el proceso $P_4$ no podrá ejecutarse hasta que los procesos $P_2$ y $P_3$ hayan concluido la ejecución de su proceso. En la figura 3.1 se muestra el grafo de dependencia de datos de la expresión anterior, en el cual se puede ver de forma clara la dependencia de datos que existe entre las operaciones.

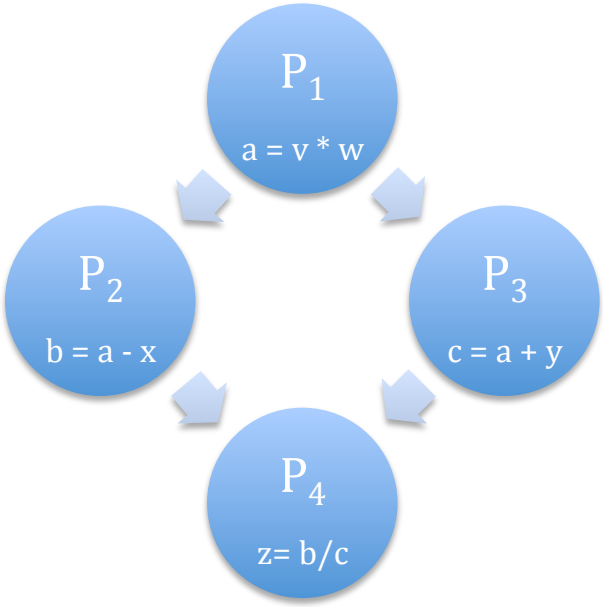



**Figura 3.1** Grafo de dependencia de datos para la evaluación de la expresión $\mathbf{Z} = \frac{\mathbf{v*w-x}}{\mathbf{v*w+y}}$

## 3.6 Procesamiento en Paralelo

El procesamiento en paralelo implica tomar una tarea grande, dividirla en tareas pequeñas y, después, trabajar con cada una de esas tareas pequeñas de forma simultánea. El objetivo del procesamiento en paralelo es dividir y completar una gran tarea en menos tiempo de lo que nos costaría hacerlo de la forma habitual. El procesamiento en paralelo no sólo incrementa el poder de procesamiento, también ofrece otras ventajas cuando es implementado adecuadamente, tales como (Rajasekaran & Reif, 2008):

- Un mayor rendimiento.
- Una mayor tolerancia a los fallos que puedan ocurrir en el sistema.
- Un mejor costo/rendimiento.

Para poder llevar a cado el cómputo en paralelo es de suma importancia cumplir con los requerimientos básicos.

a) Hardware diseñado para trabajar con múltiples procesadores y que provea comunicación entre esos procesadores.
b) Un sistema operativo capaz de administrar múltiples procesos.
c) Un software que sea capaz de partir largas tareas en múltiples tareas pequeñas y, que éstas a su vez, puedan funcionar en paralelo.

Resulta de suma importancia saber por qué es importante el procesamiento en paralelo, saber cómo es que surge y las necesidades que puede llegar a satisfacer. La idea del procesamiento en paralelo surge a partir de las limitantes físicas que se tienen, en este caso, resulta ser que la velocidad de procesamiento en los procesadores depende de la transmisión de la velocidad de la información en los componentes electrónicos con el procesador. La velocidad, a su vez, es limitada por la velocidad de la luz, que es 299.792.458 metros por segundo. Otro factor limitante, es la densidad de los transistores dentro del procesador, ya que pueden ser sometidos hasta cierto límite, más allá del límite, los transistores crean interferencia electromagnética entre ellos.

Las mejorías en el ciclo de reloj y el diseño de circuitos alcanza un nivel óptimo, los diseñadores de hardware buscan otras alternativas para incrementar el rendimiento. Por tal razón, el paralelismo es el resultado a esos esfuerzos, ya que, habilita múltiples procesadores trabajando simultáneamente en varias partes de la tarea, en orden para completar más rápido de lo que podrían realizarse aparte de esa tarea.

### 3.6.1 Tipos de paralelismo

El paralelismo puede ser clasificado en dos tipos: paralelismo explícito y paralelismo implícito. La diferencia entre estos dos tipos de paralelismo radica en que, el paralelismo implícito involucra única y exclusivamente al procesador, lo cual quiere decir que el procesador es quien decide cómo paralelizar las instrucciones; en cambio, en el paralelismo explícito, el procesador ya no decide cómo paralelizar las instrucciones, en este caso es el compilador quien decide.

El paralelismo implícito puede llevarse a cabo mediante la segmentación y la división funcional. En la segmentación, se divide la función a realizar en una serie de sub-funciones que se pueden ejecutar de manera independiente. Este tipo de paralelismo se realiza en arquitecturas segmentadas, conocidas también, como pipelines o tuberías, ya que benefician al encadenamiento del proceso de ejecución de las instrucciones. La división funcional consiste en usar múltiples unidades funcionales, casi siempre de tipo aritmético-lógico, siendo posible ejecutar distintas instrucciones al mismo tiempo.

En el paralelismo explícito, los sistemas disponen de múltiples procesadores, así, el software del sistema es quien decide qué instrucción se va a ejecutar en cada procesador. Al dar soporte a la multiplicidad de sucesos simultáneos que se presentan en los sistemas de computadoras, es posible tener diversos esquemas de arquitecturas, según la clasificación de Michael Flynn, la cual se explicó a detalle en el capítulo 2.

### 3.6.2 Características de los sistemas en paralelo

Todo sistema paralelo cumple con ciertas características, de las cuales se abordarán las siguientes (Dye, 1999):

- Escalabilidad
- Eficiencia
- Grado de paralelismo
- Granularidad
- Fiabilidad
- Disponibilidad
- Tasa de utilización del sistema

La *escalabilidad* se define como la capacidad que tiene un sistema en paralelo de ampliar o mejorar su rendimiento, sin que sea necesario devaluar sus prestaciones al aumentar el tamaño del problema y el número de procesadores. Dicha escalabilidad se puede evaluar mediante los siguientes factores:

- Facilidad de ampliación del sistema respecto a los elementos de proceso que puedan serle añadidos.
- Linealidad que existe entre la relación de la ganancia de velocidad y el número de elementos de proceso.

La *eficiencia* es definida como la medida de porcentaje de tiempo que es empleado por cada máquina para resolver un procesamiento en paralelo.

La *fiabilidad* de un sistema es definida como la probabilidad que existe para que, en un instante (t), durante cierto tiempo, dicho sistema no presente ningún problema durante un intervalo de tiempo.

La *disponibilidad* se define como la probabilidad de que el sistema presente un funcionamiento adecuado durante un instante (*t*).

El *grado de paralelismo* es definido como el número de elementos de un proceso, que son necesarios para llevar a cabo el paralelismo de un programa en su máxima expresión en un instante *(t).*

La *tasa de utilización del sistema* es la relación existente entre el grado de paralelismo medio  y el grado máximo de paralelismo.

El *grano de paralelismo* es definido como el número de elementos de un proceso que componen un sistema paralelo.

La *granularidad* de los esquemas internos es el grado de detalle de éstos, en función del esquema lógico. Cuanto más fina es la granularidad (a nivel de campo de registro es más fina, que a nivel de registro completo) se consigue mayor grado de independencia de los datos.

## 3.7 Ley de Moore

En el año de 1965 el Sr. Gordon Moore elaboró un artículo (Moore, 1965), en el cual anticipaba que la complejidad de los circuitos integrados se duplicaría cada año con una reducción de costo conmensurable. Conocida como la *Ley de Moore*, su predicción ha hecho posible la proliferación de la tecnología en todo el mundo y ,hoy se ha convertido en el motor del rápido cambio tecnológico. El artículo que Moore escribió se titula: "Meter más componentes en los circuitos integrados" y, su predicción es la siguiente: "La complejidad de los componentes se ha multiplicado aproximadamente por 2, cada

años. A corto plazo, se puede esperar que esta tasa se mantenga, o incluso que aumente. A largo plazo, la tasa de aumento es un poco más incierta, aunque no hay razón para creer que no permanecerá constante, por lo menos durante 10 años. Esto significa que para 1975, el número de componentes en cada circuito integrado, de mínimo coste, será de 65000. Creo que un circuito tan grande puede construirse en una sola oblea." (aquí te hace falta una nota al pie, para saber de dónde lo sacaste)

Moore actualizó su predicción en 1975 para señalar que el número de transistores en un chip se duplica cada dos años y, esto se sigue cumpliendo hoy. Además de proyectar cómo aumenta la complejidad de los chips (medida por transistores contenidos en un chip de computador), la Ley de Moore sugiere también una disminución de los costos. A medida que los componentes y los ingredientes de las plataformas con base de silicio crecen en desempeño, se vuelven, exponencialmente, más económicos de producir y, por lo tanto, más abundantes, poderosos y transparentemente integrados en nuestras vidas diarias.

La Ley de Moore es sinónimo de más rendimiento, la capacidad de proceso, medida en millones de instrucciones por segundo (MIPS), ha subido debido a cómputos de transistores en aumento. Pero la Ley de Moore también significa costes en descenso, a medida que los ingredientes de las plataformas y los componentes basados en el silicio obtienen más rendimiento, resulta exponencialmente más barato fabricarlos y, por consiguiente, son más abundantes, más potentes y están más integrados en nuestra vida cotidiana. Los microprocesadores de hoy día equipan todo tipo de artículos, desde juguetes hasta semáforos.

La Ley de Moore ha tenido sus variaciones, pero su equilibrio promedio ha servido incluso como referencia para otras leyes similares aplicadas a otros componentes (como la Ley de Kryder para los discos duros). Es importante saber que esta ley no será eterna y, que llegará el momento en que no puedan seguir incrementando el número de transistores en los chips; una nueva tecnología vendrá a suplir a la actual, ya que con las actuales tecnologías seria difícil reducir los chips por debajo de los 12 nanómetros de superficie.

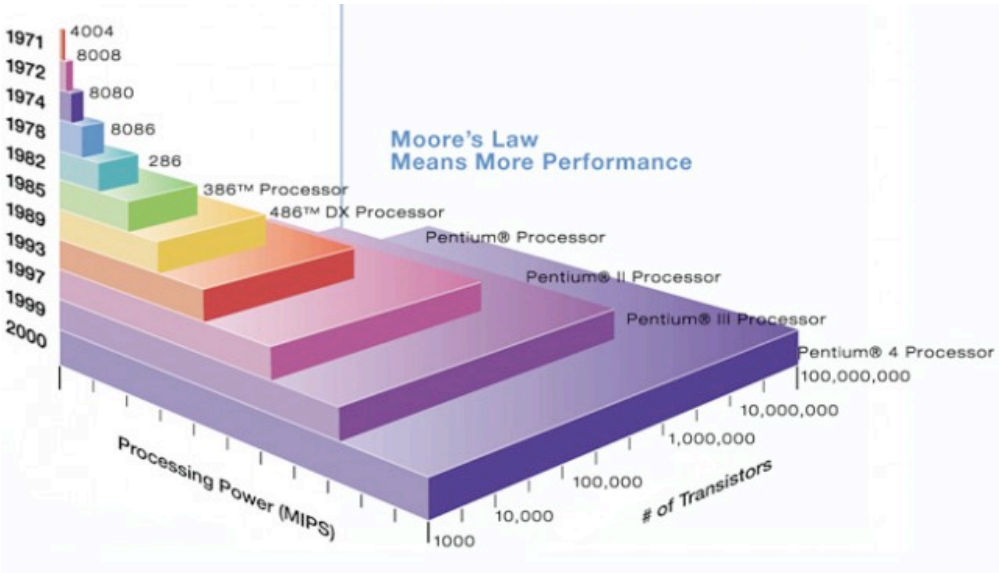



**Figura 3.2** Ley de Moore en procesadores Intel

## 3.8 Metodología de Diseño de Foster.

La metodología para diseñar y realizar programas en paralelo consta de 4 pasos, descrito por Ian Foster, los cuales son los siguientes:

a) Particionar. Consiste en dividir el problema que se tiene, en el particionado se realizará la división de tareas y datos en muchas partes pequeñas.

b) Comunicación. Consiste en el intercambio de datos entre cómputos, lo cual implica determinar la cantidad y el patrón de comunicación. El objetivo es minimizar el coste de las comunicaciones.

c) Aglomeración. Consiste en agrupar tareas para poder mejorar el rendimiento, lo que implica combinar tareas; el objetivo de la aglomeración es mantener la escalabilidad sin modificar el diseño, reducir los costes de ingeniería de software y minimizar las comunicaciones al combinar grupos de tareas, que consistirán en menor cantidad de mensajes, pero con mayor capacidad.

d) Mapeo. Consiste en asignar tareas a los procesadores, lo que significa asignar tareas aglomeradas a los hilos generados. El objetivo del mapeo es maximizar el uso de los procesadores para poder equilibrar la carga y minimizar las comunicaciones, al tener tareas conectadas a un mismo procesador.

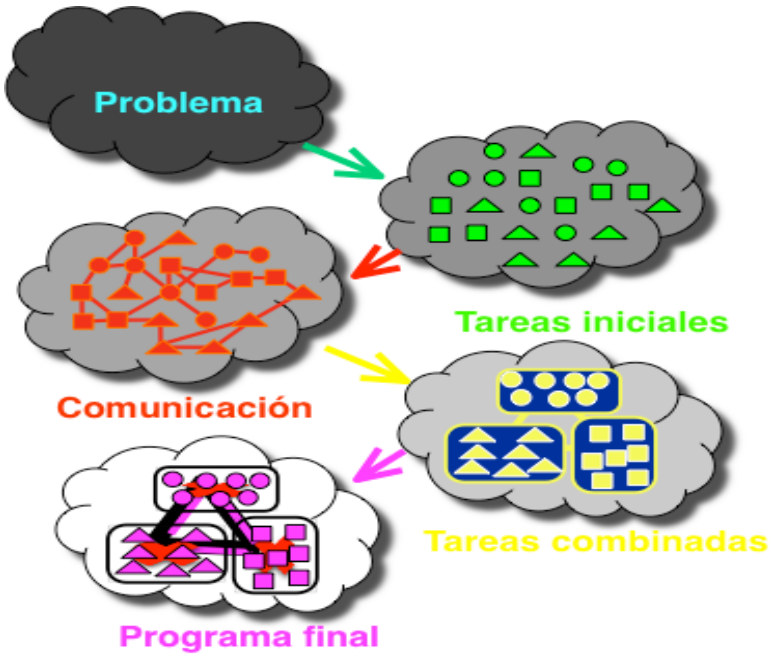



**Figura 3.3** Metodología de diseño de Foster

## 3.9 Módulos de programación paralelos.

En la realización de programación en paralelo existen algunos tipos de módulos, que son los siguientes:

a) Descomposición funcional
   - Paralelismo de tareas.
   - Dividir el cómputo y asociarle datos.
   - Tareas independientes del mismo problema.

b) Descomposición de datos.
   - La misma operación, pero ejecutando diferentes datos.
   - Dividir datos en piezas y asociarles cómputo.

## 3.10 Métodos de descomposición

En la realización de programación en paralelo existen diversos modelos de descomposición, que son útiles al pensar acerca de cuándo romper el cómputo y empezar el trabajo independiente. Algunas veces es claro saber cual es el modelo que se debe emplear, pero algunas otras, resulta complicado saber qué modelo usar; todo depende de la naturaleza del problema.

a) Descomposición de Tarea
- Este tipo de descomposición, también llamada descomposición funcional, divide el problema por el tipo de tarea a realizar y, luego, asigna una tarea particular a cada trabajador en paralelo.
- Enfocarse al cómputo, puede revelar la estructura en un problema.

b) Descomposición por Dominio

- Enfocarse en la estructura de datos, más grande o más frecuentemente accedida.
- Paralelismo en los datos (la misma operación aplicada a todos los datos).

c) Descomposición por Pipeline

- La computación es realizada en etapas independientes.
- Descomposición funcional:
    - Los hilos se asignan a una etapa a computar.
    - Línea de ensamble de automóviles.

- Descomposición de datos:
    - Los hilos procesan todas las etapas de una sola instancia.
    - Un trabajador construye un auto completo.

## 3.11 Arquitecturas de Hardware en Paralelo

Como ya se ha descrito con anterioridad, el procesamiento en paralelo se refiere al uso de múltiples procesadores, con el fin de reducir el tiempo que requiere completar una tarea requerida. En vez de que un solo procesador ejecute toda la tarea, varios procesadores trabajan en conjunto en una parte de la tarea. Para poder realizar el procesamiento en paralelo se requiere que el hardware de la computadora pueda soportar más de un solo procesador. A lo largo de los años se han desarrollado diversas arquitecturas para sistemas con múltiples procesadores. A continuación, se describen cuatro arquitecturas que se encuentran funcionando en la actualidad (Mahapatra & Mishra, 2000):

- **Sistemas de multiprocesamiento Simétrico** (*Symmetric Multiprocessing Systems*/ SMP).

Este tipo de arquitectura se caracteriza por tener múltiples CPU's, los cuales comparten la misma memoria, el bus y, la entrada y salida del sistema. El intercambio se lleva a cabo mediante el uso de un bus de sistema de alta velocidad. Una copia del sistema operativo se ejecuta en el ordenador y controla todos los procesadores. Este sistema operativo debe estar diseñado para soportar múltiples procesadores, y debe tener un algoritmo de programación que utilice todos los procesadores en el sistema de manera uniforme. La arquitectura de SMP también se conoce como una arquitectura de memoria compartida o bien acoplados. (Mahapatra & Mishra, 2000)

Cada procesador, en un sistema SMP, puede ejecutar programas de forma independiente, con cada procesador accediendo a un área separada de la memoria. Un sistema de un solo procesador que funciona de un modo de tiempo compartido, puede trabajar muy bien en una sola tarea a la vez. Esto, logra la ilusión de hacer varias cosas a la vez debido al rápido ciclo a través de todos los diferentes trabajos que se están ejecutando. Un sistema de SMP, en cambio, puede trabajar muy bien en varios trabajos simultáneamente, dado que procesadores separados están dedicados a cada trabajo (esta forma de trabajo es equivalente al paralelismo *inter-query*, el cual se abordará más adelante), porque la asignación de trabajo del procesador es totalmente transparente a las aplicaciones. Las aplicaciones de las bases de datos pueden tener los beneficios de un mayor rendimiento en este tipo de sistemas (SMP); sin necesidad de que los cambios se realicen para el software RDBMS. Además, varios procesadores en un sistema SMP pueden ser organizados para trabajar cooperativamente en un solo programa grande, con el fin de completar más rápidamente lo que un único procesador podría hacer

trabajando de forma individual (lo cuál es equivalente al paralelismo *intra-query*, el cual, también se abordará más adelante). Sin embargo, el paralelismo *intra-query* requiere de un apoyo específico del software RDBMS y de las aplicaciones de la base de datos.

Ventajas

- SMP proporciona una manera tranquila para aumentar el rendimiento de un sistema, ya que sólo se tienen que añadir más procesadores. También es posible agregar más memoria y capacidad de disco, si es necesario.
- El software diseñado para sistemas monoprocesadores puede trabajar en un sistema SMP sin ninguna modificación.
- Los sistemas SMP representan una manera de costo efectivo para lograr la escalabilidad en caso de que un sistema monoprocesador no pueden cumplir con los requisitos de rendimiento.
- La arquitectura SMP es madura y ampliamente utilizada.
- Debido a que sólo una copia de los sistemas operativos se ejecuta en un sistema SMP, la sobrecarga administrativa de la gestión de un sistema SMP es similar a la de la gestión de un sistema de un procesador.

Desventajas

- La arquitectura SMP no escala bien a un gran número de procesadores.
- La arquitectura SMP no proporciona alta disponibilidad.

El número de procesadores que se pueden agregar a un sistema SMP es limitado, ya que todos los procesadores comparten la misma memoria. El añadir procesadores de más, provoca cuellos de botella de memoria y el bus del sistema. Además, cabe resaltar que el número máximo de procesadores que se pueden configurar en un

sistema SMP varía de un sistema operativo y de un proveedor, a otro. Dado que, sólo una copia del sistema operativo se ejecuta en un sistema SMP, el sistema operativo representa un único punto de fallo. Además, ciertos fallos de hardware también pueden hacer fallar al sistema. En consecuencia, los sistemas SMP por sí mismos no representan una buena opción cuando la prioridad es tener una alta disponibilidad.

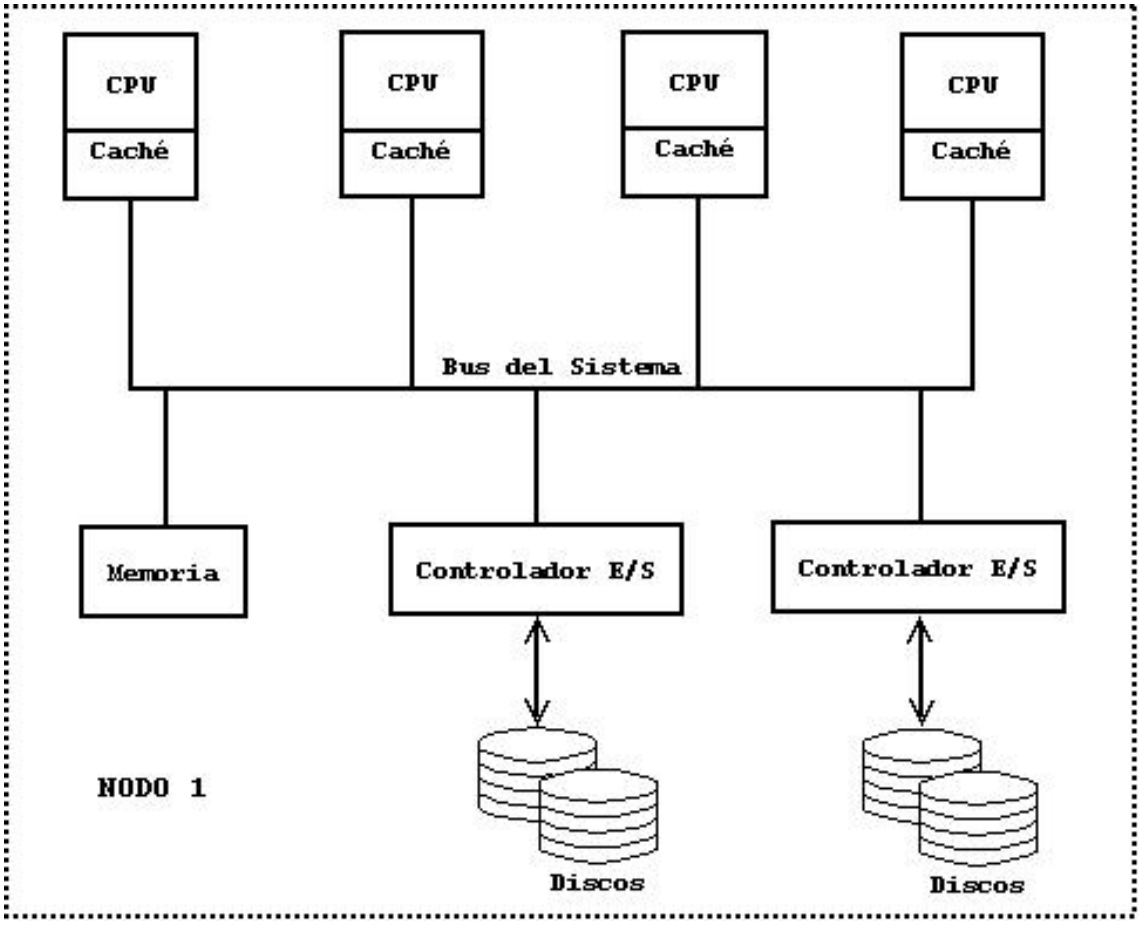



**Figura 3.4** Arquitectura SMP **(Oracle, 2008)**

- **Sistemas de Procesamiento Masivamente Paralelo** (*Massively Parallel Processing Sistems*/ MPP).

Este tipo de arquitectura se caracteriza por tener varios nodos conectados a través de su propia red de alta velocidad. Cada nodo tiene su propio CPU y su propia memoria, bus, discos, entrada y

salida del sistema. Cada nodo corre con su propia copia del sistema operativo y, el número de nodos, varia para cada sistema.

Debido a que cada nodo tiene su propia memoria, este tipo de sistemas MPP se conocen como sistemas de memoria distribuida. Cada nodo ejecuta su propia copia del sistema operativo y, la comunicación entre nodos, se realiza a través del paso de mensaje.

Los sistemas MPP fueron desarrollados originalmente para el cálculo científico, altamente paralelizable; pero, cada vez se utilizan más, para aplicaciones de almacenamiento de datos. (Bauer, 1999)

Los sistemas MPP y clúster tienen muchas similitudes desde el punto de vista arquitectónico, ya que, ambos utilizan una arquitectura de memoria distribuida y, en ambos casos, cada nodo ejecuta su propia copia del sistema operativo. En ambas arquitecturas se utiliza el paso de mensajes para la comunicación entre nodos y requieren un esfuerzo adicional de programación de aplicaciones para explotar el paralelismo. La principal diferencia entre la arquitectura MPP y las arquitecturas de clúster es que el número de nodos que se puede configurar en la arquitectura MPP es mayor al número de nodos que se puede configurar en una arquitectura clúster. Además de que las características de interconexión son diferentes entre ambas arquitecturas.

Ventajas

- La ventaja principal de este tipo de sistemas es que no experimentan cuellos de botella de memoria, que es lo que limita la escalabilidad en los sistemas SMP, ya que cada nodo en un sistema MPP tiene su propia memoria.
- Los sistemas MPP son adecuados para grandes aplicaciones de almacenamiento de datos, la configuración de un sistema MPP

grande puede soportar fácilmente miles de usuarios y grandes transacciones por minuto.

Desventajas

- Debido a que la memoria de cada nodo, en un sistema MPP, es independiente de los otros, hay características especiales que se deben incorporar en el sistema operativo para permitir la comunicación entre los nodos.
- Los sistemas MPP son más difíciles de manejar que los sistemas SMP, ya que cada nodo en un sistema MPP ejecuta su propia copia del sistema operativo y su propia copia del software de la base de datos. Por lo tanto, la arquitectura MPP resulta ser una mejor opción al alcanzarse los límites y la escalabilidad de un sistema SMP o clúster.

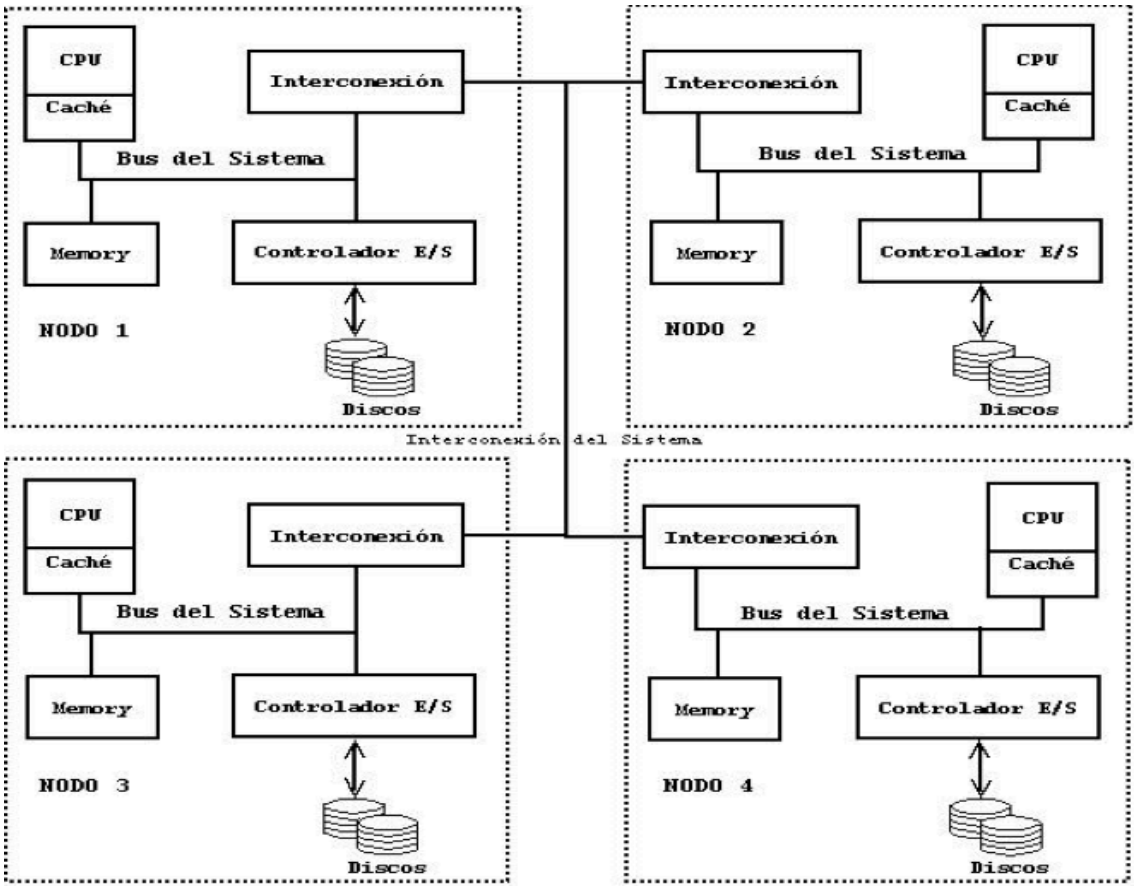



**Figura 3.5** Arquitectura MMP **(Oracle, 2008)**

- **Sistemas Clúster (*Clustered Systems*)**

Consiste en un sistema clúster con varios nodos, débilmente acoplados, usando una conexión de área local (LAN), tecnológicamente interconectado. Cada uno de los nodos puede ser una máquina única de procesador o un SMP. En un clúster el sistema de software balancea la carga de trabajo entre los nodos y provee gran disponibilidad. (Bauer, 1999)

En las arquitecturas en clúster, un conjunto de nodos está interconectado y comparte dispositivos de almacenamiento. Ya que cada nodo es independiente del sistema, tiene su propio procesador y memoria independiente, además, ejecuta su propia copia del sistema operativo. Cada nodo en este tipo de arquitectura puede ser un sistema con un único procesador o un sistema SMP.

Ventajas

- Proporciona un mayor rendimiento y escalabilidad que puede ser posible con cualquier sistema SMP. La escalabilidad de un sistema clúster se puede aumentar de dos maneras: aumentando el número de procesadores dentro de cada nodo o aumentando el número de nodos en el clúster.
- Los sistemas clúster pueden proporcionar un alta disponibilidad, ya que la red entre los nodos se realiza con conexión dual para proporcionar redundancia. Si un nodo del clúster falla, los otros nodos en el clúster continuarán operando y, los usuarios conectados al nodo que falló, pueden ser redistribuidos a los nodos que se encuentran funcionando.

Desventajas

- Se requiere software de sistema adicional para gestionar y monitorizar los sistemas en clúster.

- La gestión de un sistema clúster resulta ser más compleja que la gestión de un sistema SMP. La complejidad aumenta a medida de que el número de nodos incrementa en la arquitectura.
- En comparación con un sistema SMP, un sistema clúster requiere más esfuerzo de programación para coordinar el procesamiento en los diferentes nodos; así como para equilibrar la carga de trabajo de procesamiento a través de los nodos.

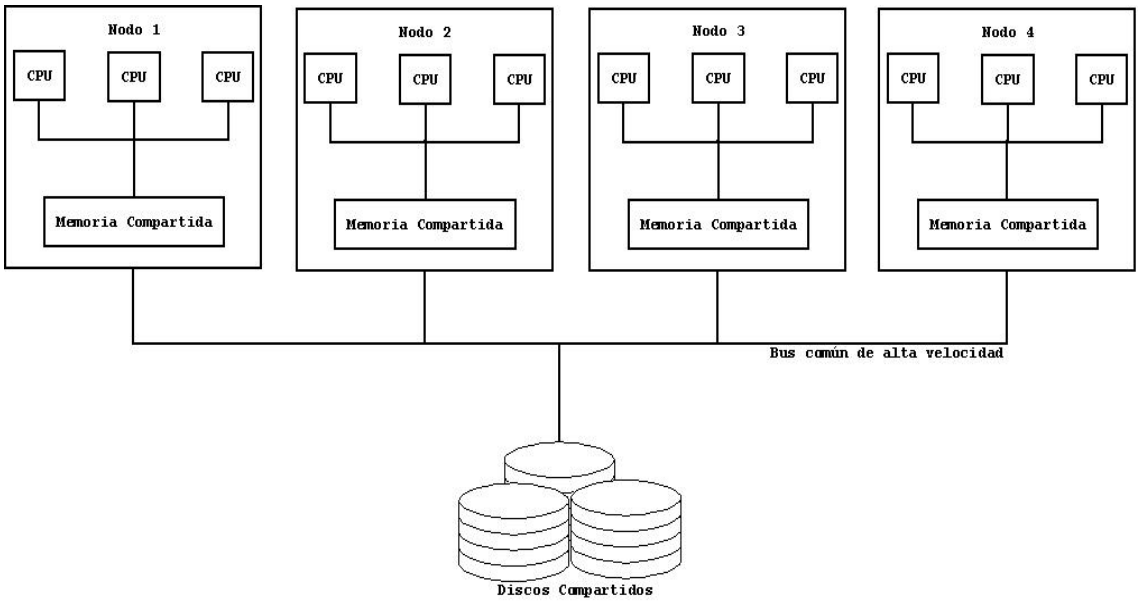



**Figura 3.6** Arquitectura de un Sistema Clúster **(Oracle, 2008)**

- **Sistemas de Acceso a Memoria no Uniforme** (*Non Uniform Memory Acces Systems*/ NUMA).

Estos sistemas corren una copia del sistema operativo que cruza todos los nodos. (Rajasekaran & Reif, 2008)

En una arquitectura NUMA, múltiples sistemas SMP están interconectados de manera ordenada para formar un sistema grande. En este tipo de sistemas, toda la memoria de los grupos SMP está

conectada en conjunto, para formar un espacio grande de memoria (memoria contigua).

Para los procesadores, toda la memoria parece la misma, la única diferencia está en el tiempo de acceso. Ya que, cuando un procesador accede a la memoria local (memoria en su propio grupo SMP), el acceso es rápido; en cambio, cuando un proceso accede a memoria no local (memoria de otro grupo SMP), el acceso tiene que ir a través de la interconexión y, como resultado, es mucho más lenta. Aquí es donde la arquitectura NUMA recibe su nombre, ya que el tiempo de acceso a la memoria no es uniforme.

Es importante recordar que los sistemas SMP se basan en una arquitectura de memoria compartida; los sistemas MPP y los sistemas clúster representan arquitecturas de memoria distribuida, debido a que la memoria se distribuye entre los nodos. Los sistemas NUMA representan una arquitectura híbrida, debido a que se refiere a una arquitectura de memoria compartida distribuida. La arquitectura NUMA intenta combinar lo mejor de las arquitecturas distribuidas y compartidas. Los sistemas NUMA tienen la simplicidad de los sistemas SMP, ya que funcionan como un solo sistema grande. A medida que aumentan las necesidades de su procesamiento es posible hacer crecer un sistema NUMA añadiendo más grupos SMP.

Ventajas

- Proporciona una escalabilidad de acoplamiento flexible como el de una arquitectura MPP.
- Las aplicaciones de software no tienen que cambiar para adaptarse a la arquitectura.

Desventajas

- Mientras que, el acceso memoria local es rápido, el acceso a memoria no local, no lo es. Un alto porcentaje de accesos a memoria no local puede ralentizar el sistema.
- El rendimiento no sólo depende de la interconexión de alta velocidad, sino también del ajuste adecuado del sistema operativo.

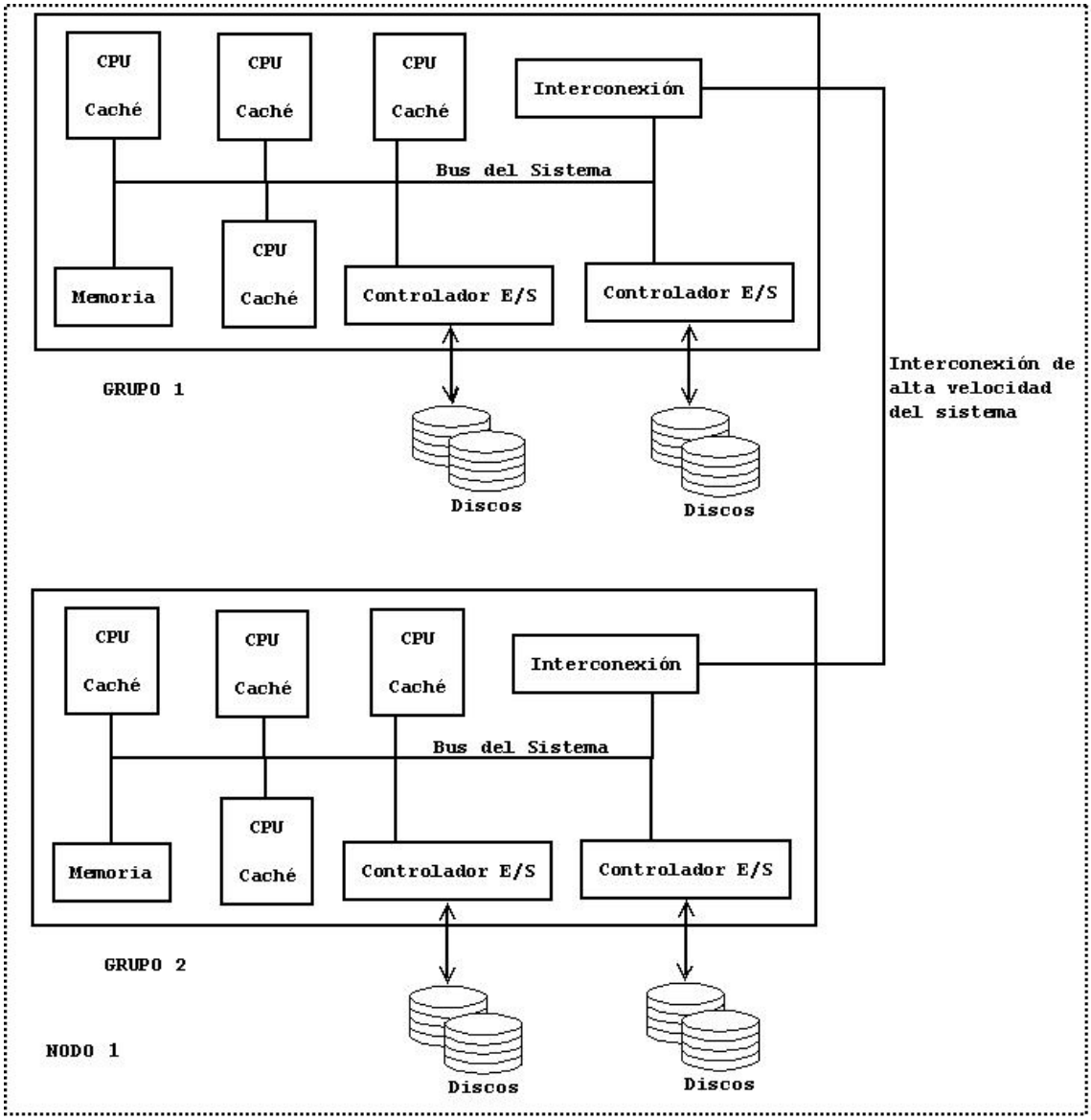



**Figura 3.7** Arquitectura de un Sistema NUMA **(Oracle, 2008)**

## 3.12 Paralelismo en las bases de datos

El paralelismo en las bases de datos puede ser definido como la partición de la base de datos, teniendo como único fin, el de procesar una sola operación en forma paralela para mejorar la velocidad en la ejecución de las consultas. Esto puede realizarse mediante el uso de varios procesadores, extrayendo información de uno o más discos, a manera de reducir el tiempo de respuesta de la consulta solicitada.

La implementación de la base de datos en paralelo ha ido incrementando con el paso de los años, al día de hoy, diversas marcas de bases de datos ofrecen este producto. Esto es, debido al aumento en las transacciones que son realizadas diariamente por las organizaciones y la manipulación de grandes volúmenes de datos; deseando obtener una respuesta en un menor tiempo, aprovechando así, el avance tecnológico que se ha dado en la implementación de múltiples procesadores, provocando que éstos ejecuten una misma consulta.

En este tipo de bases de datos es importante verificar y asegurar que los procesadores no actualicen de forma simultánea los mismos datos, de lo contrario, podría repercutir gravemente en la base de datos, por tal motivo, es importante que el SGBD garantice que tiene la última versión de los datos en la memoria intermedia.

# Capítulo 4. Procesamiento en paralelo en las bases de datos.


## EXTRACTO

En este cuarto se tratarán temas tales como el procesamiento de consultas, las principales partes que forman el procesamiento en paralelo, la importancia del procesamiento en paralelo en las bases de datos, los obstáculos del paralelismo, diversos tipos de paralelismo que existen en las bases de datos, algunos modelos analíticos y, finalmente, se describirán dos manejadores de bases de datos que soportan este tipo de procesamiento.

# 4 Introducción

Las bases de datos de hoy día han incrementado su tamaño de forma exponencial respecto a las bases de datos del pasado, dicho crecimiento se irá incrementando en el futuro; impactando así, la caída constante del costo de almacenamiento acompañado de un rápido incremento en la capacidad de almacenamiento. Años atrás, una base de datos de un terabyte era considerada como una base de datos "muy grande", cuestión que en nuestros días podría considerarse pequeña; dado que los volúmenes diarios que se manejan en algunas bases de datos son medidas en terabytes, por tal motivo, en un futuro, manipular bases de datos que procesen información en medidas como el petabyte o exabyte será común.

Con volúmenes de datos de tal magnitud resulta evidente que el procesamiento secuencial es insuficiente para procesar tal magnitud de información; por consiguiente, para gestionar eficazmente tales volúmenes de datos, es necesario asignar recursos múltiples para su procesamiento de forma masiva. Por lo tanto, el alto rendimiento de procesamiento de la consulta se centra en el procesamiento de la consulta, incluyendo las consultas en la base de datos y las transacciones de la misma.

En estos tiempos resulta común ver cómo las bases de datos crecen de forma exponencial y son utilizadas por una gran número de usuarios. Cuando consideramos una base de datos de 2 terabytes de tamaño, utilizando un único procesador con la capacidad de procesamiento a una velocidad de 1 megabyte/segundo, esto representaría 120 días y noches de tiempo de procesamiento. Si este procesamiento necesita ser reducido en tiempo, entonces el procesamiento paralelo es una alternativa.

## 4.1 Procesamiento de consultas

El objetivo principal del procesamiento de consultas es el de transformar una consulta escrita en un lenguaje de alto nivel, normalmente SQL, en una estrategia de ejecución correcta y eficiente expresada en un lenguaje de bajo nivel (implementación de álgebra relacional) y, ejecutar dicha estrategia para extraer los datos requeridos.

Un aspecto de gran importancia en el procesamiento de consultas es la optimización de la consulta, la cual consiste en elegir la transformación que minimice el uso de recursos; como podría ser el reducir el tiempo de ejecución de la consulta, que es la suma de los tiempos de ejecución de todas las operaciones individuales que componen la consulta. El uso de recursos también podría medirse según el tiempo de respuesta de la consulta.

El procesamiento de consultas puede ser dividido en cuatro partes principales:

- Descomposición (compuesta de análisis sintáctico y validación)
- Optimización
- Generación de código
- Ejecución

### *Descomposición de Consultas*

Esta es la primera fase del procesamiento de consultas, el objetivo de la descomposición es el de transformar una consulta de alto nivel en una consulta de álgebra relacional y, comprobar que, dicha consulta sea sintáctica y semánticamente correcta. La descomposición de consultas se divide en cinco etapas que son los siguientes:

- Análisis
- Normalización
- Análisis semántico
- Simplificación
- Reestructuración

*Análisis*. En esta etapa se analiza la consulta léxica y sintácticamente, utilizando las técnicas de los compiladores de los lenguajes de programación; se verifica que las relaciones y atributos especificados en la consulta se encuentran definidos en el catálogo del sistema, además de verificar que todas las operaciones aplicadas a los objetos de la base de datos sean apropiadas para el tipo de objeto.

*Normalización*. Esta etapa convierte la consulta en una forma normalizada para pueda manipularse más fácilmente.

*Análisis Semántico.* El objetivo del análisis semántico es rechazar las consultas normalizadas que estén incorrectamente formuladas o que sean contradictorias. Una consulta será incorrectamente formulada si sus componentes no contribuyen a la generación del resultado, lo que puede suceder si faltan algunas especificaciones de combinación y, una consulta será contradictoria si su predicado no puede ser satisfecho por ninguna tupla (conjunto de valores relacionados, uno por cada atributo).

*Simplificación*. En esta etapa se detectan las cualificaciones redundantes, se eliminan las sub-expresiones comunes y se transforma la consulta en otra consulta que sea semánticamente equivalente pero que se pueda calcular de forma más fácil y eficiente. Normalmente, las restricciones de acceso, las definiciones de vista y las restricciones de integridad se toman en consideración en esta

etapa, pudiendo algunas de esas restricciones y definiciones introducir redundancia.

*Reestructuración*. Esta es la etapa final de la descomposición de una consulta, la consulta se reestructura para obtener una implementación más eficiente.

## 4.2 Procesamiento Paralelo en las Bases de Datos

Para abordar este tema es importante conocer tres necesidades que son de gran importancia en el ambiente del procesamiento en paralelo en las bases de datos.

- Necesidad de una mayor velocidad. El tamaño de las bases de datos siempre se encuentra en constante crecimiento, las consultas (queries o query en singular) se vuelven mas complejos y el software utilizado en la base de datos debería poder cumplir con la demanda a este incremento, dando un resultado satisfactorio a este tipo de complejidades.
- Necesidad de la escalabilidad. Este tipo de requerimiento va de la mano con el desempeño, dado que, las bases de datos crecen de forma rápida y, por tal motivo, los usuarios necesitan de un camino fácil y a un costo razonable para poder escalar sus sistemas cuando éstos se encuentran en crecimiento.

- Necesidad de una alta disponibilidad. El alta disponibilidad hace referencia a la necesidad de conservar un buen funcionamiento en la base de datos cuando ésta comienza a funcionar y se encuentra en funcionamiento con el mínimo

tiempo o, mientras se encuentre en un periodo de inactividad. Con el incremento en el uso de Internet, los usuarios requieren de una base de datos que se encuentre disponible en todo momento, ya sea de noche o de día.

## 4.3 Aceleración o incremento de velocidad (Speedup)

Como se ha mencionado con anterioridad, el tamaño de las bases de datos tiene un crecimiento constante; es muy común encontrar cientos de gigabytes de información en tales bases de datos, algunas de ellas son denominadas como Bases de datos de gran escala (*Very Large Data Bases*- VLDB's), en las cuales, *queries* complejos son ejecutados. Estos *queries* requieren demasiado tiempo de procesamiento para ser ejecutados, si se ejecutaran en paralelo, el tiempo transcurrido podría ser reducido mientras lo requiera el tiempo de procesador. (Bauer, 1999)

La aceleración es definida como la relación que existe entre el tiempo de ejecución (*run-time*) con un procesador y el tiempo de ejecución, usando múltiples procesadores. La medida en el aumento del rendimiento ganado al usar múltiples procesadores en vez de un procesador. En otras palabras, se refiere a la ejecución de una tarea dada en menos tiempo, aumentando el grado de paralelismo. La aceleración es una medida utilizada para medir el rendimiento de consultas de lectura solamente (recuperación de datos) y, se calcula mediante la siguiente fórmula:

$$\text{Speed up} = \frac{\text{Tiempo 1}}{\text{Tiempo m}},$$

Donde:

$Tiempo_1$ es el tiempo que toma ejecutar una tarea usando únicamente un procesador.
$Tiempo_m$ es el tiempo que toma ejecutar la misma tarea usando m procesadores.

Por ejemplo si una consulta requiere 6 minutos para completar su tarea usando un solo procesador, usando 6 procesadores solamente le tomará 1 minuto.
*Speed Up* = 6 / 1 = 6.

Es importante saber que este tipo de resultados se dan solamente en casos ideales, casi nunca se da en la vida real.

La eficiencia es igual al resultado obtenido de la aceleración dividida entre el número de procesadores que son utilizados; por ejemplo, en el caso anterior, se obtuvo una aceleración (*Speed up*) igual a 6, lo cual quiere decir que la eficiencia sería del 100%, esto representa un caso ideal.

La aceleración lineal (*Linear-Speed up*) se refiere a la mejora en el rendimiento creciente, linealmente con más recursos; es decir, un incremento de velocidad de N, cuando un sistema grande tiene N veces los recursos del sistema más pequeño. La aceleración sub-lineal (*Sublinear Speed up*) se refiere a la aceleración, cuando es menor que N. Y, la aceleración súper-lineal (*Superlinear Speed up*) se refiere a la aceleración, cuando es mayor que N, pero, este caso es muy raro encontrarlo, ocasionalmente puede ser visto debido a alguna característica única de la arquitectura, la cual favorece la formación paralela, como memoria adicional en el sistema multiprocesador.

## 4.4 Escalabilidad

La escalabilidad es la capacidad de mantener los niveles de rendimiento conforme la carga de trabajo incrementa, aumentando más capacidad al sistema (como sería el incremento de discos o de procesadores). En los sistemas que tienen un único procesador, resulta más complicado lograr la escalabilidad, mientras que, en los sistemas paralelos resulta mejor escalar el sistema.

En varias aplicaciones, el número de usuarios de la base de datos y el volumen de transacciones incrementan con el tiempo. La demanda para incrementar el poder de procesamiento al manipular el incremento de la carga, sin perdida en el tiempo de respuesta, es posible hallarla mediante el uso de sistemas paralelos.

*Aumento Escalar (Scale up)*

El aumento escalar hace referencia al manejo de grandes tareas, aumentando así, el grado de paralelismo. Es la capacidad de procesar grandes tareas en la misma cantidad de tiempo, proporcionando más recursos o, aumentando el grado de paralelismo. En otras palabras, es la capacidad de una aplicación de mantener el tiempo de respuesta cuando el tamaño de trabajo o el volumen de transacción aumentan, al incrementar procesadores o discos al sistema. (Dye, 1999)

La diferencia que existe entre la aceleración (*speed up*) y el aumento escalar (*scale up*) es que, mientras se calcula la aceleración, el tamaño del problema se conserva fijo; mientras que, el aumento escalar es calculado al incrementase el tamaño del problema o el volumen de la transacción. El aumento escalar es medido en términos de cuánto volumen de transacción puede ser incrementado

por la adición de mas procesadores mientras se mantiene contante el tiempo de respuesta. Este tipo de medida usualmente es utilizada en los sistemas de procesamiento de transacciones (manipulación de datos) y, es calculado mediante el uso de la siguiente formula:

$$\text{Scale up} = \frac{\text{Tp}}{\text{Tmp}} ,$$

Donde:

$T_p$ representa el tiempo transcurrido en un monoprocesador en un sistema pequeño.
$T_{mp}$ representa el tiempo transcurrido en un multiprocesador en un sistema grande.

El aumento escalar lineal hace referencia a la capacidad de mantener el mismo nivel de rendimiento cuando la carga de trabajo (*workload*) y los recursos son proporcionalmente incrementados. Utilizando la fórmula del aumento escalar (*Scale up*), cuando el aumento escalar es igual a 1 se dice que dicho aumento escalar es lineal; cuando el aumento escalar es menor a 1 entonces se dice que se tiene un sub-aumento escalar lineal (*Sub-linear scale up*).
Existen dos tipos de aumento escalar, el aumento escalar de transacción (*transaction scale up*) y el aumento escalar de dato (*data scale up*), ambos son relevantes en las bases de datos en paralelo, dependiendo de cómo se mide el tamaño de la tarea.

*Aumento escalar de transacción (transaction scale up)*

Este tipo de aumento escalar se refiere a incrementar la velocidad en donde las operaciones son procesadas, el tamaño de la base de datos puede incrementar proporcionalmente a medida que se van recibiendo transacciones. Así como N-veces tantos usuarios están

presentando N-veces tantas solicitudes, o transacciones, contra una base de datos de N-veces más grande. Este tipo de escalamiento es relevante en los sistemas de procesamiento de transacciones en donde las transacciones son pequeñas actualizaciones. El procesamiento de transacción es especialmente bien adaptado al procesamiento en paralelo; desde que distintas transacciones pueden ejecutarse concurrentemente e independientemente en procesadores separados y, cada transacción toma una pequeña parte de tiempo, incluso si la base de datos crece. (Taniar, H.C Leung, Rahayu, & Goel, 2008)

*Aumento escalar de dato (data scale up)*

Este tipo de aumento escalar se refiere al incremento en el tamaño de la base de datos y, la tarea es un trabajo grande, cuyo tiempo de ejecución depende de qué tan grande sea la base de datos. Por ejemplo, al ordenar una tabla cuyo tamaño es proporcional al tamaño de la base de datos, el tamaño de la base de datos es la medida de la magnitud. Es común encontrar este tipo de aumento escalar en procesamiento analítico en línea (OLAP- *online analytical processing*) en el almacenamiento de datos, en donde la tabla de hechos es normalmente muy grande en comparación con todas las tablas de dimensiones combinadas. (Taniar, H.C Leung, Rahayu, & Goel, 2008)

## 4.5 Alto Rendimiento

Las bases de datos son requeridas en una gran cantidad de áreas debido a su gran utilidad, las cuales ofrecen aplicaciones en beneficio de los usuarios, como suelen ser: bancos, líneas aéreas, universidades, etc. Algunas aplicaciones de las bases de datos requieren tener una disponibilidad de 24 horas al día, los 7 días de la semana, todo el año; por estas razones es fundamental contar con

bases de datos que tengan una alta disponibilidad, lo cual resulta crucial en el éxito cierto tipo de empresas.

Trabajar con bases de datos en paralelo en un sistema paralelo multi-nodo es una opción que provee una gran disponibilidad. Con una base de datos en paralelo, cuando un nodo se cae, afecta únicamente al subconjunto de usuarios conectados al nodo que falló y, los usuarios de este nodo pueden tener acceso a la base de datos después de reactivar alguno de los nodos que siguen funcionando.

## 4.6 Precio / Desempeño

La parte económica es un factor que juega un papel importante respecto al cómputo en paralelo, ya que, el usuario, al adquirir un sistema, busca obtener el mejor desempeño por el precio que ofrece.

El hacer procesadores más rápidos significa realizar una inversión monetaria importante, después de un cierto límite, incrementar el poder del procesamiento en un sistema CPU con un único procesador se vuelve técnicamente muy difícil de realizar.

## 4.7 Obstáculos del paralelismo

El decidir realizar operaciones en paralelo en las bases de datos, conlleva tomar en cuenta ciertos factores que representan un riesgo, pues, tanto la velocidad como el aumento escalar pueden disminuir, tales riesgos pueden presentarse en cualquiera de los siguientes tres escenarios (Mahapatra & Mishra, 2000) (Taniar, H.C Leung, Rahayu, & Goel, 2008):

a) Costos del arranque y consolidación
b) La interferencia y comunicación
c) Sesgo

*Costos de arranque y consolidación*

El costo del arranque se encuentra asociado con la iniciación de múltiples procesos y una operación en paralelo consiste en múltiples procesos; entonces, el tiempo de arranque puede eclipsar el tiempo de procesamiento real, afectando así, la velocidad, especialmente si miles de procesos deben de iniciar; inclusive si un número reducido de procesos en paralelo han iniciado. Si el tiempo de procesamiento actual es muy corto, el costo de arranque puede dominar el tiempo de procesamiento global.

El costo de consolidación se refiere a aquellos costos asociados con la recolección de los resultados obtenidos por cada procesador, en un procesador central.

El procesamiento en paralelo normalmente comienza con la ruptura de una tarea principal en múltiples sub-tareas, en las que, cada sub-tarea se lleva a cabo por un elemento de procesamiento diferente. Después de que las sub-tareas se han completado, es necesario consolidar los resultados producidos por cada sub-tarea que es presentada al usuario. Dado que, el proceso de consolidación se lleva a cabo por un único elemento de procesamiento, normalmente el procesador central, esto se realiza sin aplicar el paralelismo, por consiguiente se ve afectada la aceleración global del proceso.

Tanto el arranque como la consolidación de costos hacen referencia a partes secuenciales de procesos y no pueden ser

paralelizados. Esto es una manifestación de la ley de Amdahl, en qué estados, el tiempo de cómputo puede ser dividido en una parte paralela y cuál en una parte serial. Sin importar cuán alto sea el grado de paralelismo en el primero, la aceleración puede ser limitada asintóticamente por éste último, el cual deberá ejecutarse en un elemento único de procesamiento.

*Interferencia y Comunicación*

Dado que los procesos que son ejecutados en un sistema en paralelo usualmente tienen acceso a recursos compartidos, puede haber una desaceleración debido a la interferencia de cada proceso nuevo, ya que, cada proceso nuevo compite con los procesos existentes por tener los recursos, así, se ven afectados por dicho fenómeno, tanto la aceleración como el aumento escalar. Es muy común que un proceso deba comunicarse con otros procesos y en un ambiente sincronizado. El proceso en espera que desea comunicarse con otros procesos puede ser obligado a esperar a que otros procesos se encuentren listos para comunicarse. Este tiempo de espera puede afectar al proceso global, ya que algunas tareas se encuentran inactivas por esperar a otras tareas.

*Sesgo*

El sesgo en el procesamiento de bases de datos en paralelo se refiere a la irregularidad que existe en la partición en la carga de trabajo. En los sistemas en paralelo, la carga de trabajo es igual entre todos los elementos de procesamiento y es uno de los factores críticos para conseguir una aceleración lineal. Cuando la carga de un elemento de procesamiento es más pesada que las otras, el tiempo total transcurrido para una tarea en particular pude ser determinada por este elemento de procesamiento y, la finalización del resto de

elementos de procesamiento tendrían que esperar, lo cual hace que sea una situación indeseable. El sesgo en estos sistemas de procesamiento en paralelo surge comúnmente por una distribución irregular de los datos. (Taniar, H.C Leung, Rahayu, & Goel, 2008)

## 4.8 Tipos de paralelismo en las bases de datos

Existen diferentes formas de realizar el procesamiento en paralelo en las bases de datos. Dependiendo de la tarea que se requiera realizar, será el tipo de paralelismo que se opte por usar. Dentro de las formas que serán abordadas en este documento están las siguientes (Taniar, H.C Leung, Rahayu, & Goel, 2008):

a) Paralelismo Inter-Query
b) Paralelismo Intra-Query
c) Paralelismo de Inter-Operación
d) Paralelismo de Intra-Operación

a) Paralelismo Inter-Query

Este tipo de paralelismo es definido como la capacidad de usar múltiples procesadores para ejecutar simultáneamente e independientemente varios *queries*. Es importante saber que en este tipo de paralelismo cada *query* es ejecutado por un único procesador.

El siguiente esquema muestra cómo es que funciona el paralelismo inter-query, ejemplifica cómo tres *queries* independientes pueden ser ejecutados de forma simultánea por tres diferentes procesadores.

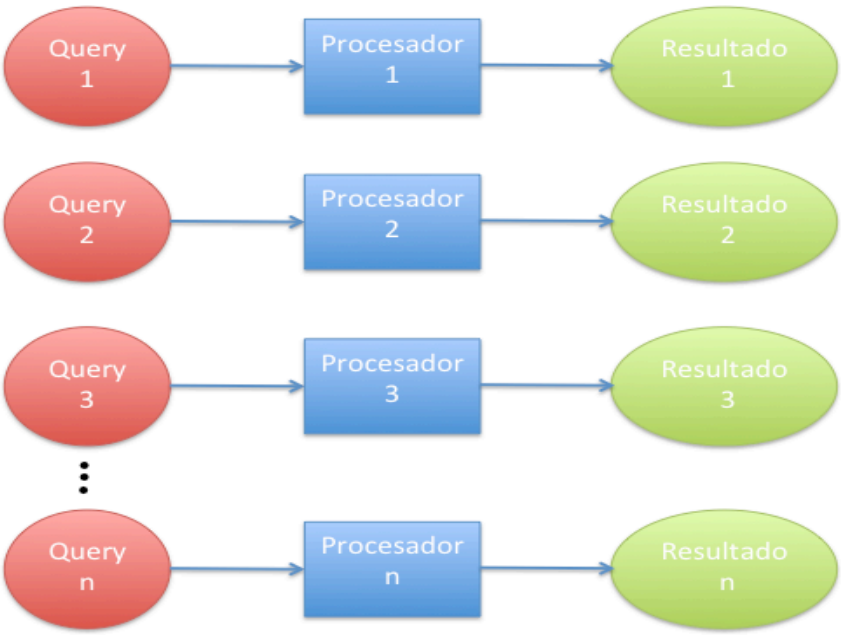



**Figura 4.4** Paralelismo Inter-query **(Mahapatra & Mishra, 2000)**

En las aplicaciones de procesamiento de transacciones en línea (OLTP), cada *query* es independiente y toma poco tiempo ejecutarlo; al incrementar el número de usuarios OLTP, más *queries* son generados, los cuales puedes ser distribuidos sobre todos los procesadores, utilizando el paralelismo inter-query. En cambio, sin el paralelismo inter-query tendríamos que ejecutar todos los *queries* en un único procesador de modo tiempo-compartido; lo cual generaría un tiempo de respuesta menor. Por tal razón, desde que los *queries* pueden ser ejecutados simultáneamente por múltiples procesadores, el tiempo de respuesta es satisfactorio.

b) Paralelismo Intra-Query

Este tipo de paralelismo se define como la capacidad de dividir un único *query* en sub-tareas y ejecutar, dichas sub-tareas en forma paralela, utilizando un procesador diferente para cada sub-tarea,

obteniendo como resultado el reducir el tiempo global transcurrido ocupado al ejecutar un único query.

El siguiente esquema muestra cómo un *query* grande puede ser descompuesto en dos *sub-queries*, los cuales son ejecutados de forma simultánea, utilizando "n" procesadores. Los resultados de estas dos sub-tareas es que se vuelven a unir para generar el resultado original del *query.*

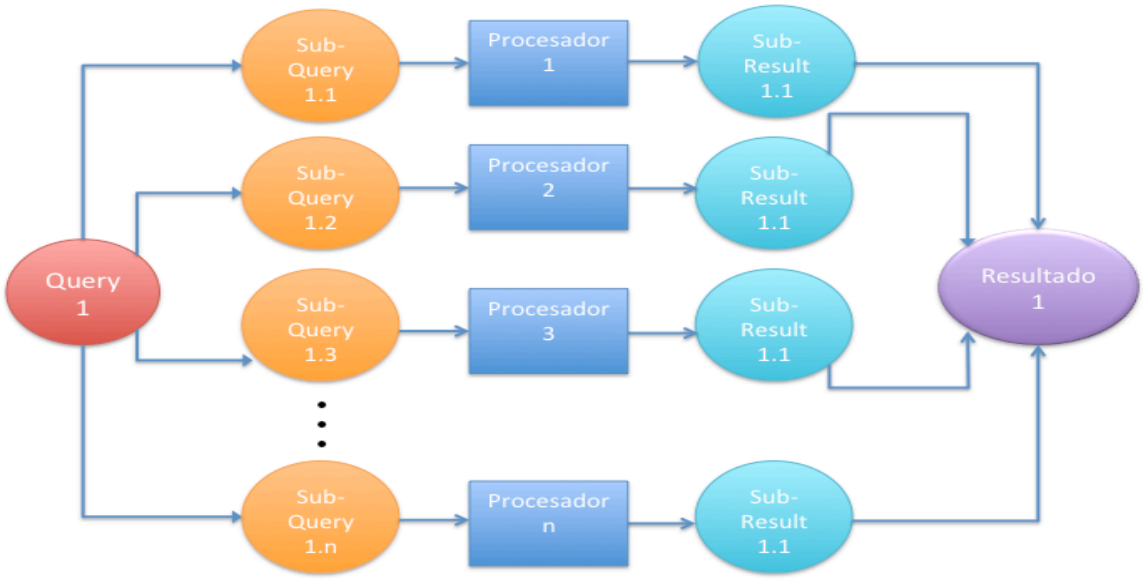



**Figura 4.5** Paralelismo Intra-query **(Mahapatra & Mishra, 2000)**

c) Paralelismo Inter-Operación

Esta forma de paralelismo se origina cuando el paralelismo es creado mediante la ejecución concurrente de diversas operaciones dentro de la misma consulta u operación. Es posible acelerar el procesamiento de la consulta mediante la ejecución en paralelo de las diferentes operaciones en una expresión de consulta, por ejemplo, realizar una consulta de clasificación (*sort*) y búsqueda (*search*) simultáneamente. Existen dos formas de paralelismo de interoperabilidad: el paralelismo pipeline y el paralelismo independiente.

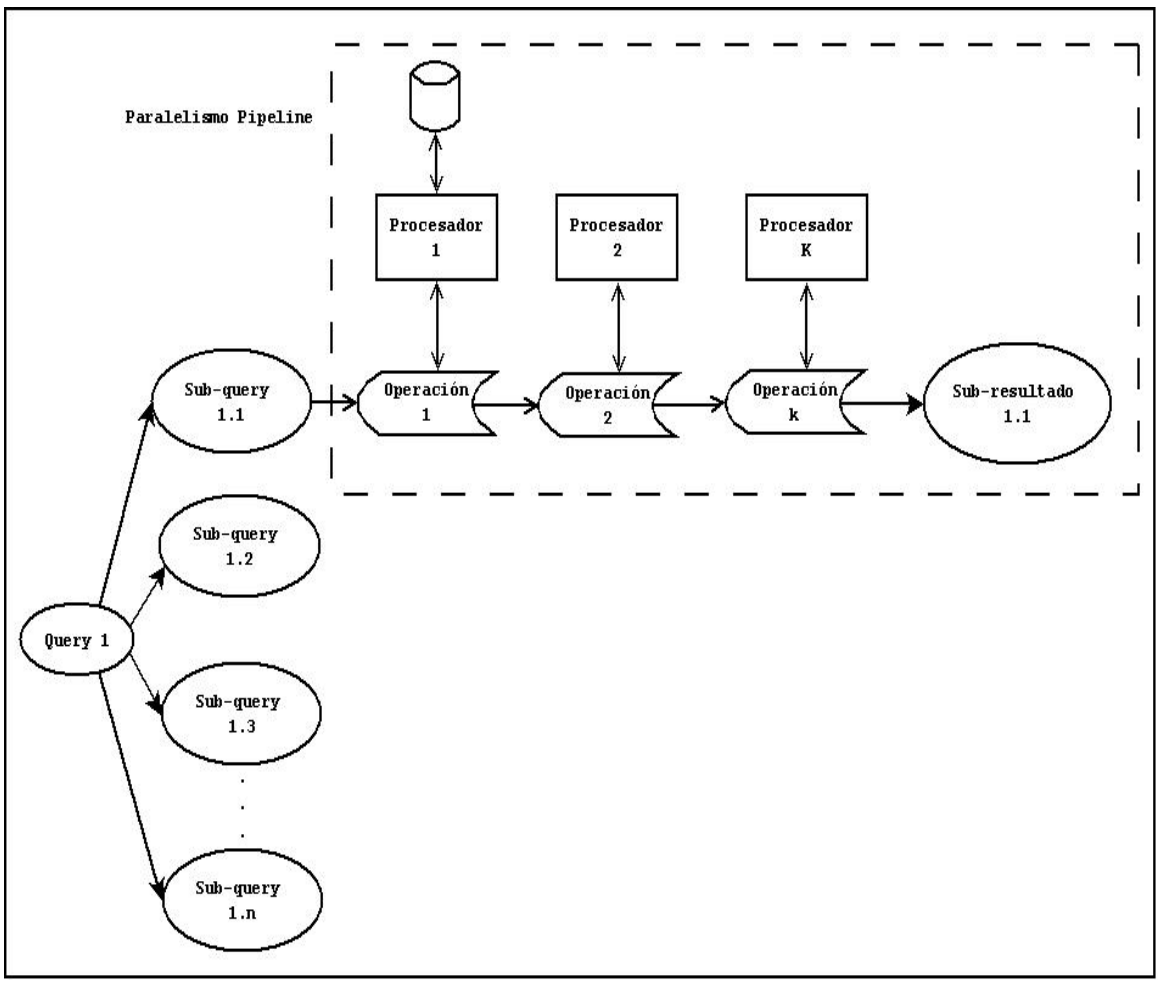



**Figura 4.6** Paralelismo pipeline **(Taniar, H.C Leung, Rahayu, & Goel, 2008)**

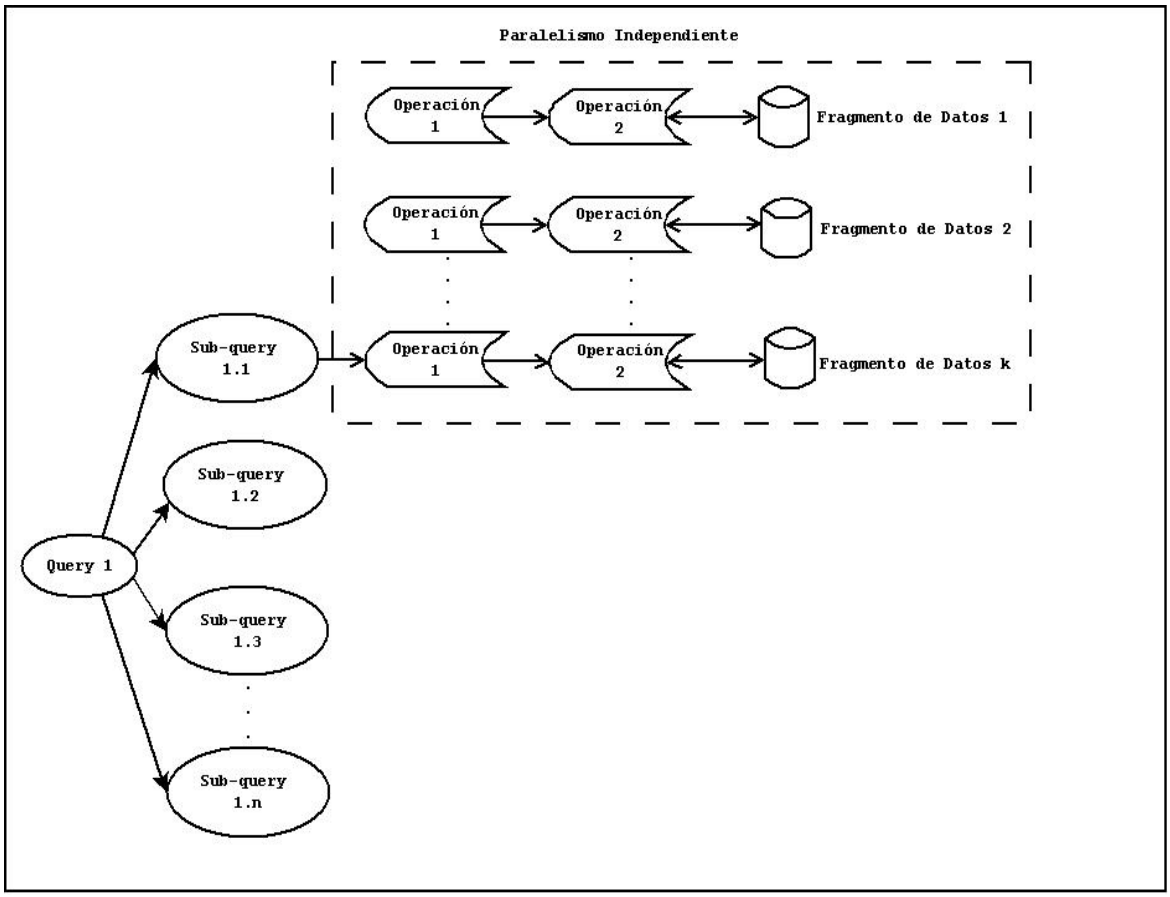



**Figura 4.7** Paralelismo independiente **(Taniar, H.C Leung, Rahayu, & Goel, 2008)**

d) Paralelismo Intra-Operación

Este tipo de paralelismo se origina debido a que las operaciones de trabajo en la base de datos en las tablas, contienen grandes conjuntos de datos de los registros; es posible paralelizar las operaciones mediante la ejecución en paralelo de éstas, en diferentes subconjuntos de la tabla. Por lo tanto, el paralelismo Intra-operación es comúnmente llamado paralelismo particionado, debido a que los datos son particionados.

Es posible acelerar el procesamiento de una consulta mediante la ejecución paralelizada de cada operación individual, por ejemplo, una ordenación en paralelo (*parallel sort*), una búsqueda en paralelo (*parallel search*), etc.

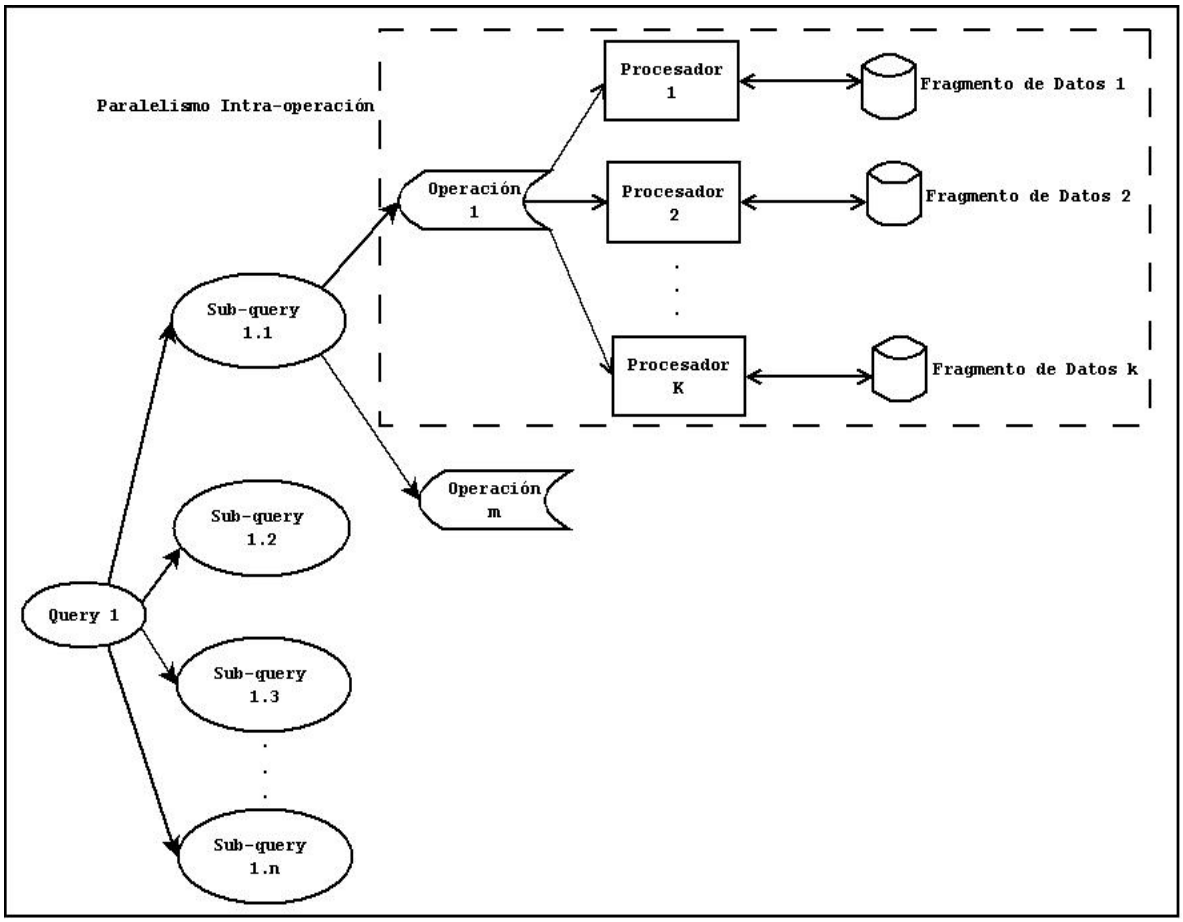



**Figura 4.8** Paralelismo Intra-operación **(Taniar, H.C Leung, Rahayu, & Goel, 2008)**

## 4.9 Modelos Analíticos

Los modelos analíticos son ecuaciones o fórmulas de costo que se utilizan para calcular el tiempo transcurrido de una consulta utilizando un algoritmo especial para el procesamiento en paralelo. Una ecuación de costo está compuesta de variables, que son sustituidas por valores específicos en el tiempo de ejecución de la consulta. Estas variables indican los costos de los componentes del procesamiento de consultas en paralelo. (Taniar, H.C Leung, Rahayu, & Goel, 2008)

*Modelos de Costo*

Para medir la efectividad de paralelismo del procesamiento de consulta de la base de datos es necesario proporcionar modelos de costos, los cuales pueden describir el comportamiento de cada algoritmo de consultas en paralelo. Aunque los modelos de costos pueden ser usados para calcular el rendimiento de una consulta, es primordial utilizarlos para describir el proceso que se encuentra implicado y realizar una comparación. Los modelos de costos también sirven como herramientas para examinar todos los factores de costo a mayor detalle. El costo se encuentra expresado, principalmente, en términos del tiempo transcurrido que le toma responder a una consulta. (Taniar, H.C Leung, Rahayu, & Goel, 2008)

*Notaciones de Costo*

Las ecuaciones de costo constan de un número de componentes, en particular los siguientes:

- Parámetros de datos
- Parámetros de sistemas
- Parámetros de consultas
- Costo de unidad de tiempo
- Costo de comunicación

Cada uno de estos componentes se encuentra representado por una variable, a la cual se asigna un valor en el tiempo de ejecución. Las notaciones utilizadas se muestran en la tabla siguiente:

| **Símbolo** | **Descripción** |
|---|---|
| *Parámetros de Datos* | |
| R | Tamaño de la tabla en bytes |
| $R_i$ | Tamaño del fragmento de la tabla en bytes en el procesador i |
| \|R\| | Número de registros en la tabla R |
| $\|R_i\|$ | Número de registros en la tabla R en el procesador i |
| *Parámetros de Sistemas* | |
| N | Número de procesadores |
| P | Tamaño de página |
| H | Tamaño de la tabla Hash |
| *Parámetros de Consultas* | |
| **π** | Proyectividad |
| σ | Selectividad |
| *Coste de Unidad de Tiempo* | |
| IO | Tiempo efectivo para leer una página desde el disco |
| $t_r$ | Tiempo para leer un registro de la memoria principal |
| $t_w$ | Tiempo para escribir un registro en la memoria principal |
| $t_d$ | Tiempo para calcular el destino |
| Costo de Comunicación | |
| $m_p$ | Costo del protocolo de mensaje por página |
| $m_l$ | Costo latente por una página |

**Tabla 4.1** Notaciones de Costos **(Taniar, H.C Leung, Rahayu, & Goel, 2008)**

*Parámetros de Datos*

Los parámetros de datos mas importantes son los siguientes:

- Número de registros en una tabla (|R|) y
- Tamaño actual (en bytes) de la tabla (R).

El procesamiento de datos en cada procesador se basa en el número de registros. Por ejemplo, la evaluación de un atributo se realiza a nivel de registro. En cambio, en los sistemas de procesamiento, tales como I / O (de lectura / escritura de datos, desde / hacia el disco) y la distribución de datos en una red interconectada, se realiza a nivel de página, donde, una página se compone normalmente de varios registros.

En términos de sus notaciones, para conocer el tamaño real de una tabla, ésta debe ser identificada por una letra mayúscula, tal como puede ser R, S, etc. Por ejemplo, si dos tablas están involucradas en una consulta, éstas podrían identificarse por las letras R y S, las cuáles nos indicaran que se está haciendo referencia a las tablas 1 y 2, respectivamente. Como ya se mencionó, el tamaño de la tabla se mide en bytes y, se sabe que 1 gigabyte = 1024 bytes. Por lo tanto, si el tamaño de la tabla R es 4 gigabytes, al calcular una ecuación de coste variable R serán sustituidos por 4 X 1024 X 1024 X 1024.

Para el número de registros se utiliza la notación de valor absoluto. Por ejemplo, el número de registros de la tabla R se indica mediante la expresión |R|. Si la tabla S se utiliza en la consulta, entonces |S| indica el número de registros de la tabla S. Al calcular el coste de una ecuación, si hay 1 millón de registros en la tabla R, la variable |R| tendrá un valor de 1,000,000.

En un entorno de multiprocesador, la tabla es fragmentada en múltiples procesadores, por lo tanto, el número de registros y el tamaño real de la tabla para cada una, son divididos entre los procesadores que haya en el sistema. Para indicar el tamaño del fragmento de tabla en un procesador en particular, se utiliza un subíndice. Por ejemplo, $R_i$ indica el tamaño del fragmento de la tabla en el procesador *i*. Subsecuentemente, el número de registros en la tabla R en el procesador *i* es indicado por $|R_i|$.

Como el subíndice *i* indica el número de procesador, entonces $R_1$ representa el tamaño del fragmento de la tabla y $|R_1|$ representa el número de registros de la tabla R en el procesador 1. Sin embargo, en el procesamiento de la consulta con una base de datos en paralelo, el tiempo transcurrido del procesamiento de la consulta es determinado por el mayor tiempo empleado en un procesador. El cálculo del tiempo transcurrido, se refiere sólo a los procesadores que tienen el mayor número de registros a procesar. Por lo tanto, para *i* = *1…n*, elegimos el mayor $R_i$ y $|R_i|$ para representar el tiempo transcurrido más largo en el procesador que tiene la carga más pesada. Si la tabla R ya se encuentra dividida en partes iguales para todos los procesadores, entonces calcular $R_i$ y $|R_i|$ resulta sencillo, solamente se debe dividir R y |R| entre el número de procesadores.

*Parámetros de Sistema*

Como se puede observar en la figura x, existen tres parámetros de sistema, los cuáles son denominados número de procesadores (N), tamaño de página (P) y tamaño de la tabla hash (H). El número de procesadores es el más importante de los parámetros de sistema, ya que nos indica el número de procesadores que son utilizados en el procesamiento de una consulta. Si se quiere calcular $R_i$ y $|R_i|$, asumiendo que los datos se encuentran uniformemente distribuidos,

tanto R como |R| se deben dividir entre N para poder obtener $R_i$ y $|R_i|$. Por ejemplo, hay un millón de registros y se están utilizando 10 procesadores, entonces se desea saber el número de registros por cada procesador:

$$|R| = 1{,}000{,}000$$

$$N = 10$$

$$|R_i| = \frac{|R|}{N} = \frac{1{,}000{,}000}{10} = 100{,}000 \text{ registros}$$

El tamaño de la página es representado por el símbolo P y representa el tamaño de datos en bytes que tiene una página; el contenido de dicha página es un lote de registros. Cuando los registros son cargados del disco a la memoria principal, éstos no se cargan registro por registro, sino página por página.

Para calcular el número de páginas de una tabla dada, se divide el tamaño de la tabla entre el tamaño de página. Por ejemplo: $1024^2$

$$R=4 \text{ gigabytes}= 4 * 1024^3 \text{ bytes}$$

$$P = 4 \text{ kilobytes} = 4 * 1024 \text{ bytes}$$

$$\frac{R}{P} = 1024^2 \text{ número de páginas}$$

El tamaño de la tabla hash es representado por el símbolo H y, es el tamaño máximo de la tabla hash que puede caber en la memoria principal. Esto se mide normalmente por el número máximo de registros. El tamaño de la tabla hash es un parámetro importante para el procesamiento en paralelo de consultas de bases de datos grandes. Si la base de datos es grande, es probable que todos los datos no quepan en la memoria principal, ya que normalmente el

tamaño de la memoria principal es mucho más pequeño que el tamaño de una base de datos. Por lo tanto, en el modelo de costo es importante conocer la capacidad máxima de la memoria principal; de manera que, se pueda calcular con precisión cuántas veces un lote de registros necesita ser intercambiado de la memoria principal al disco. Entre más grande sea la tabla hash, resultará menos probable que el intercambio de registro sea necesario, lo que mejora el rendimiento general.

*Parámetros de Consulta*

Hay dos parámetros de consulta que son de gran importancia, denominados proyectividad ($\pi$) y selectividad ($\sigma$). (Taniar, H.C Leung, Rahayu, & Goel, 2008)

La proyectividad ($\pi$) es la razón entre el tamaño de atributo proyectado y la longitud original del registro. El rango de valor de la proyectividad varía de 0 a 1. Por ejemplo, suponiendo que el tamaño del registro de la tabla R es 100 bytes y el tamaño del registro de salida es 45 bytes; la proyectividad ($\pi$) es de 0.45.

La selectividad ($\sigma$) es la razón entre los registros totales de salida, que son determinados por el número de registros en el resultado de la consulta y el número total de registros originales. Al igual que la proyectividad, la selectividad varía en rangos de 0 a 1. Por ejemplo, suponiendo que inicialmente hay 1000 registros ($|R_i|$ = 1000 registros), y la consulta produce 4 registros, entonces la selectividad ($\sigma$) es de 4/1000 = 1/250 = 0.004.

La importancia de la proyectividad y la selectividad en el procesamiento de la consulta se debe a que están asociados con el número de registros, antes y después del procesamiento.

Adicionalmente, el número de registros es un parámetro de coste importante, que determina el tiempo de procesamiento en la memoria principal.

*Costos de Unidad de Tiempo*

Los costos por unidad de tiempo son el tiempo que se requiere para procesar una unidad de datos, son los siguientes:

- Tiempo de lectura o escritura de una página desde el disco (IO).
- Tiempo para leer un registro de la memoria principal (tr).
- Tiempo para escribir un registro en la memoria principal (tw).
- Tiempo para calcular el cómputo en la memoria principal.
- Tiempo para hallar el destino de un registro (td).

El tiempo de lectura/escritura de una página de/a disco es básicamente el tiempo asociado con la entrada/salida del proceso. La variable que se usa para identificar este tipo de coste es IO, es importante tomar en cuenta que, IO trabaja a nivel de página. Por ejemplo, para leer una tabla entera desde el disco a la memoria principal, se divide el tamaño de la tabla y el tamaño de la página, posteriormente se multiplica por el IO. Esto es (R / P) * IO.

El tiempo para leer/escribir un registro de/a la memoria principal son indicados por $t_r$ y $t_w$, respectivamente. Estos dos tipos de costes se encuentran relacionados con los registros de lectura, los cuales están ya en la memoria principal; también son utilizados en la obtención de los registros de la página de datos. Es importante tomar en cuenta que estos dos tipos de costes unitarios trabajan a nivel de registro y no a nivel de página.

El tiempo que se requiere para calcular el cómputo en la memoria principal varía de un tipo de cómputo a otro; se denota mediante una *t* seguida de un subíndice que indique el tipo de cómputo a realizarse. El tiempo de cómputo, en este caso, es el tiempo que se requiere para computar un único proceso en el CPU.

Por último, el tiempo necesario para computar el destino de un registro se denota por $t_d$, este tipo de coste unitario es utilizado cuando un registro debe ser distribuido o transferido de un procesador a otro.

*Costos de Comunicación*

Los costos de comunicación pueden ser clasificados mediante los siguientes elementos:

- Costo del protocolo de mensaje por página ($m_p$).
- Costo de latencia por una página ($m_l$).

Ambos elementos trabajan a nivel de página, como con el disco. El primero de ellos (costo del protocolo de mensaje) es el costo asociado a la iniciación de transferencia de mensaje, mientras que el segundo (costo de latencia), se encuentra asociado con el tiempo de transferencia del mensaje actual.

Los gastos de comunicación se dividen en dos componentes principales, uno para el emisor y el otro para el receptor. El costo emisor se refiere al costo total de envío de registros de páginas, el cual es calculado mediante la multiplicación del número de páginas que se envían y los costos unitarios de comunicación, mencionados anteriormente. Por ejemplo, para enviar toda la tabla R, el coste sería R / P * ($m_p$ + $m_l$). Es importante tomar en cuenta que el tamaño de

la tabla debe ser dividido entre el tamaño de la página con el fin de calcular el número de páginas que se envían. El costo unitario para el envío es la suma de los dos componentes de costes de comunicación.

El coste del receptor es el costo total de recibir los registros de páginas, el cual se calcula mediante la multiplicación del número de páginas recibidas y el coste del protocolo de mensaje por página. Es importante tomar en cuenta que en el coste del receptor, la latencia de mensajes no está incluida, por lo tanto, el coste de recepción sería $R / P * m_p$.

En un entorno multiprocesador, el coste de envío es el costo de enviar datos de un procesador a otro, y el costo de envío vendrá de la carga del procesador más pesado, el cual envía el mayor volumen de datos.

*Operaciones básicas en las bases de datos en paralelo*

Las operaciones en sistemas de bases de datos en paralelo, normalmente, siguen los siguientes pasos:

- Carga de datos (escaneo) desde el disco.
- Obtención de registros de la página de datos en la memoria principal.
- Cómputo de datos y distribución de datos.
- Escritura de registros (resultados de la consulta) de la memoria principal a la página de datos.
- Escritura de datos en el disco.

Operaciones en disco. Este primer paso corresponde a la lectura y escritura de datos en el disco.

## 4.10 Servidor paralelo de Oracle (*Oracle Parallel Server*)

Oracle es un sistema de gestión de base de datos objeto-relacional, se considera a Oracle como uno de los sistemas de bases de datos más completos, destacando el soporte de transacciones, estabilidad, escalabilidad y soporte multiplataforma. La mayoría de los sistemas de gestión de base de datos más comerciales han implementado características de paralelismo, por lo que, Oracle implementó un nuevo sistema llamado *Oracle Parallel Server* (OPS), el cual soporta el procesamiento en paralelo, y puede ser dividido en *Parallel execution* (Ejecución en Paralelo), que se refiere al paralelismo intra-query, y en *Parallel server* (servidor en paralelo), que se refiere al uso de múltiples instancias al abrir una única base de datos (Mahapatra & Mishra, 2000).

Ejecución en paralelo (*Parallel Execution*)

La ejecución en paralelo de Oracle permite dividir una tarea entre múltiples procesos para poder completar la tarea de forma más rápida, lo cual permite tomar ventaja en el uso de múltiples procesadores en una máquina. Una forma de ejemplificar el modo en que trabaja la ejecución en paralelo de Oracle, es el siguiente:

```
SQL> SELECT COUNT (*) FROM orders;
```

Si uno ejecuta la declaración anterior de forma serial, un único proceso buscará el orden de la tabla y contará el número de renglones, sin embargo, si uno cuenta con una máquina con cuatro procesadores y utiliza la ejecución en paralelo de Oracle, el orden de la tabla podría ser separado en cuatro partes; lo que significa que el proceso comenzaría en cada CPU y las cuatro partes de la tabla

serían escaneadas de forma simultánea, obteniendo un resultado por cada procesador, que al final se unirían para regresar la cuenta total.

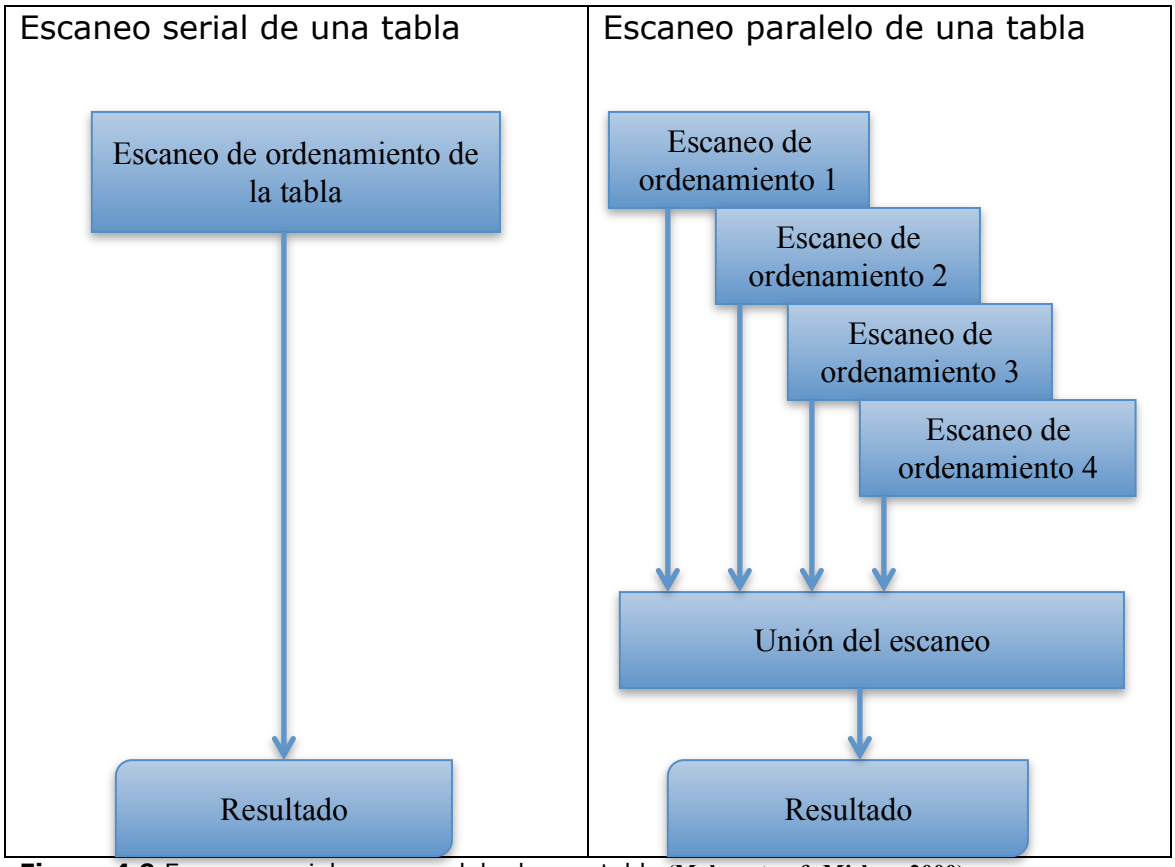



**Figura 4.9** Escaneo serial y en paralelo de una tabla **(Mahapatra & Mishra, 2000)**

Esta herramienta de Oracle soporta una gran variedad de declaraciones que pueden ser procesadas en paralelo, dentro de las cuales se encuentran las siguientes:

Query en paralelo (*Parallel Query*)

Los *queries* grandes pueden ser divididos en pequeñas tareas y ser ejecutados en paralelo, mediante múltiples procesos esclavo en orden; para reducir el tiempo transcurrido global, tal y como fue explicado en el ejemplo anterior. (Sonawalla, 1999)

DML en paralelo (*Parallel* DML).

Además de las declaraciones SELECT, esta herramienta pude ejecutar operaciones DML como INSERT, UPDATE and DELETE en paralelo. Sin embargo, existen algunas restricciones en este tipo de operaciones, por ejemplo, en las operaciones UPDATE y DELETE pueden ser paralelizadas únicamente en tablas que se encuentran particionadas; las declaraciones, INSERT INTO…SELECT…FROM pueden ser paralelizadas tanto en tablas no particionadas como en tablas particionadas. Es importante mencionar que el DML en paralelo resulta de gran ayuda en el desarrollo de *data warehouse* ya que guarda el resumen histórico de las tablas, además de que el tiempo necesario para reconstruir o mantener esas tablas es reducida, ya que el trabajo puede ser realizado en paralelo. Para habilitar el uso de declaraciones DML en paralelo es necesario utilizar el siguiente comando ALTER SESSION como se muestra a continuación (Sonawalla, 1999):

```
SQL> ALTER SESSION ENABLE PARALLEL DML;
```

DLL en paralelo (*Parallel* DLL)

Con la implementación del *Oracle Parallel Server* es posible paralelizar tablas y crear índices en paralelo, por ejemplo, ahora se pueden paralelizar sentencias como la siguiente:

```
CREATE TABLE… AS SELECT… FROM
CREATE INDEX
ALTER INDEX REBUILD
```

Cargador de datos en paralelo (*Parallel data loading*).

Es posible cargar una gran cantidad de datos de forma paralela mediante la separación del dato de entrada en múltiples archivos y activar múltiples sesiones de SQL*Loader para poder cargar los datos en una tabla; esta característica resulta de gran ayuda, ya que reduce el tiempo para cargar ese dato (Sonawalla, 1999). Un ejemplo de cómo se realiza la carga en paralelo es el siguiente:

```
SQLLOAD scott/tiger CONTROL=parte1.ctl DIRECT=TRUE
PARALLEL=TRUE
SQLLOAD scott/tiger CONTROL=parte2.ctl DIRECT=TRUE
PARALLEL=TRUE
SQLLOAD scott/tiger CONTROL=parte3.ctl DIRECT=TRUE
PARALLEL=TRUE
```

Recuperación en paralelo (*Parallel recovery*).

Esta característica puede reducir el tiempo necesario por instancia y el medio de recuperación, con la recuperación en paralelo; múltiples procesos esclavo en paralelo pueden ser utilizados para trabajar en operaciones de recuperación. El recuperar una base de datos grande toma demasiado tiempo y, con el uso de esta característica, uno puede reducir el tiempo. La cláusula PARALLEL puede ser usada con el comando RECOVER para paralelizar el medio de recuperación. Dicha cláusula se usa para especificar el grado o el número de procesos esclavos paralelos que se utilizaran. Es posible utilizar la cláusula PARALLEL con los comandos RECOVER DATABASE, RECOVER TABLESPACE y RECOVER DATAFILE, como se muestra a continuación (Sonawalla, 1999):

```
RECOVER DATABASE PARALLEL (DEGREE d INSTANCES DEFAULT);
RECOVER TABLESPACE tablespace_name PARALLEL (DEGREE d
INSTANCES i);
RECOVER DATAFILE 'datafile_name' PARALLEL (DEGREE d);
RECOVER DATABASE PARALLEL (DEGREE DEFAULT);
```

Propagación de replicación en paralelo (*Parallel replication propagation*).

Esta característica puede ser utilizada por administradores que suelen usar la replicación para mantener copias de los objetos de la base de datos, en múltiples bases de datos. Es posible usar la propagación en paralelo para actualizar esas copias eficientemente; hacer cambios en una base de datos puede propagarse a otra base de datos, usando múltiples procesos esclavo para acelerar la propagación (Mahapatra & Mishra, 2000).

Servidor en paralelo (*Parallel server, OPS*)

El servidor en paralelo de Oracle permite a una base de datos ser montada y abierta al mismo tiempo por varias instancias. Cada instancia OPS es como cualquier otra instancia de Oracle independiente y se ejecuta en un nodo separado que tiene su propio CPU y memoria. La base de datos reside en un subsistema de disco compartido por todos los nodos, OPS toma el paralelismo a un plano superior, ya que permite difundir el trabajo no sólo sobre múltiples CPU's, sino también, a través de múltiples nodos.

En la figura siguiente se ilustra una base de datos en un servidor en paralelo, compuesta de dos nodos. Cada nodo ejecuta una instancia de Oracle, en la que cada instancia tiene su propio conjunto de procesos de fondo y su propia Área Global del Sistema.

Ambas instancias montan y abren una base de datos que reside en un subsistema de disco compartido.

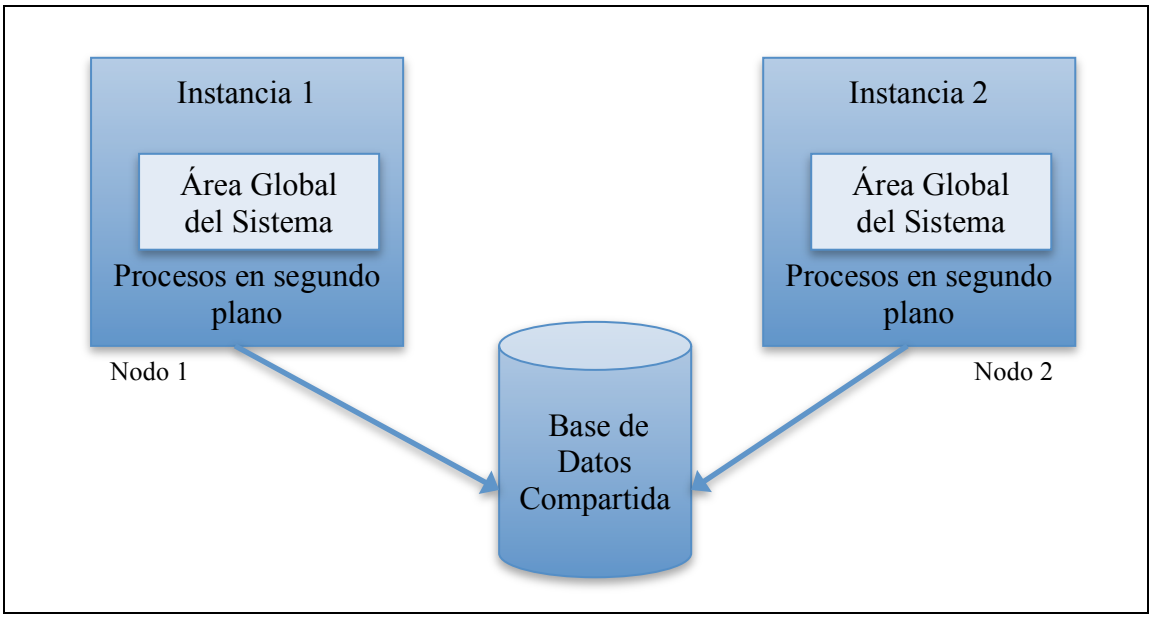



**Figura 4.10** El servidor en paralelo de Oracle permite operar varias instancias en una sola base de datos **(Mahapatra & Mishra, 2000)**

## 4.11 SQL Server (*Parallel Query Processing*/ Procesamiento de consulta en paralelo)

Microsoft SQL Server es un sistema para la gestión de bases de datos producido por Microsoft, está basado en el modelo relacional. SQL Server al igual que otros SGBD con el fin de aprovechar al máximo el avance de tecnología en cuanto a hardware, proporciona a los usuarios la opción de poder gestionar consultas en paralelo, con el fin de optimizar la ejecución de consultas y operaciones de índice en los equipos que cuentan con más de un microprocesador (CPU). Dado que, SQL Server puede realizar una operación de consulta o índice en paralelo mediante varios subprocesos del sistema operativo, la operación se puede realizar de forma rápida y eficiente.

Durante la optimización de consultas, SQL Server busca consultas u operaciones de índice que podrían beneficiar a la ejecución en paralelo. Para estas consultas, SQL Server inserta operadores de cambio en el plan de ejecución de consultas para preparar la consulta para la ejecución en paralelo. Un operador de cambio es un operador en el plan de ejecución de consultas que ofrece la administración de procesos, la redistribución de datos y control de flujo. El operador de cambio incluye tres diferentes tipos de operadores lógicos que son: el flujo de distribución (*distribute streams*), el flujo de repartición (*repartition streams*) y flujo de agrupación (*gather streams*). (Microsoft, 2012)

Un operador lógico describe el álgebra relacional utilizada para procesar una instrucción o sentencia. En otras palabras, los operadores lógicos describen conceptualmente qué operación se debe realizar.

El operador de flujo de distribución es un operador lógico que sólo se utiliza en los planes de consulta en paralelo. Este tipo operador lleva un único flujo de entrada de registros y produce múltiples flujos de salida. El contenido y el formato de registro no se modifican. Cada registro del flujo de entrada aparece en uno de los flujos de salida. Este operador conserva automáticamente el orden relativo de los registros de entrada en los flujos de salida.

Después de que los operadores de cambio se insertan, el resultado es un plan de ejecución de consultas en paralelo, el cual puede utilizar más de un hilo. Un plan de ejecución serial, utilizado por una consulta no paralela, requiere un solo hilo para su ejecución. El número real de hilos utilizados en una consulta en paralelo se encuentra determinado en la inicialización del plan de ejecución de

consulta y está determinado por la complejidad del plan y el grado de paralelismo. El grado de paralelismo determina el número máximo de CPU's que se están utilizando, sin embargo, esto no implica el número de hilos que se están utilizando. El valor del grado de paralelismo se fija en el nivel de servidor y puede ser modificado mediante el procedimiento almacenado del sistema "*sp_configure*", que tiene la siguiente sintáxis: (Microsoft, 2012)

```
sp_configure [ [ @configname = ] 'option_name'      [ , [
@configvalue = ] 'value' ] ]
```

[ **@configname=** ] **'**option_name**'**
Es el nombre de una opción de configuración. option_name es de tipo varchar(35) y su valor predeterminado es NULL. SQL Server Database Engine (Motor de base de datos de SQL Server) reconoce cualquier cadena única que forme parte del nombre de configuración. Si no se especifica, se devuelve la lista completa de opciones.

[ **@configvalue=** ] **'**value**'**
Es la nueva configuración. value es de tipo int y su valor predeterminado es NULL. El valor máximo depende de la opción individual.

SQL *Server* cuenta con un optimizador de consultas llamado SQL *Server Query Optimizer*, el cual es un optimizador basado en costos. Este optimizador analiza una serie de planes de ejecución candidatos para una determinada consulta, estima que el costo de cada uno de estos planes y, selecciona el plan con el menor costo de las opciones consideradas. (Microsoft, 2012)

El optimizador de consultas de SQL *Server* no utiliza un plan de ejecución en paralelo para una consulta, si, alguna de las siguientes condiciones es verdadera:

- El costo de ejecución de serie de la consulta no es lo suficientemente alta como para considerar una alternativa, el plan de ejecución en paralelo.
- Un plan de ejecución serial se considera más rápido que cualquier otro plan de ejecución en paralelo para una consulta particular. (Microsoft, 2012)

# Capítulo 5. Laboratorio de pruebas.

## EXTRACTO

En este quinto y último capítulo se mostrarán diversas pruebas que se realizaron en un Sistema Gestor de Base de Datos con procesamiento secuencial y procesamiento en paralelo, con el fin de sustentar lo que se explicó de forma teórica.

# 5 Introducción

En este capítulo se describirá de forma detallada el equipo de hardware y software que se empleo durante la implantación del laboratorio de pruebas, el cuál que abarca desde la instalación de cada una de las herramientas de software hasta la ejecución de cada una de las pruebas, guiando al usuario paso a paso en las diversas instalaciones efectuadas. También se explicará en que consistió cada una de las pruebas (o eventos), cómo se evaluaron, los resultados obtenidos y la interpretación de los resultados de cada uno de los eventos.

El laboratorio de pruebas tiene como principal objetivo comparar el tiempo de ejecución transcurrido durante una consulta a la base de datos tanto con procesamiento secuencial como con procesamiento en paralelo, a fin de conocer las ventajas y desventajas que presenta cada uno de los dos tipos de procesamiento, y poder conocer cuando es conveniente utilizar uno y cuando el otro.

## 5.1 Características del equipo de cómputo

En la implantación del laboratorio de pruebas se utilizaron diversas herramientas de hardware y software para poder llevar a cabo cada la parte experimental, a continuación se describen las herramientas utilizadas, las cuáles se dividen en las herramientas de hardware y software.

Herramientas de hardware

El laboratorio de pruebas fue montado en una computadora MacBook Pro, la cuál tiene las siguientes características:

- Nombre del procesador: Intel Core 2 Duo
- Velocidad del procesador: 2.26 GHz
- Número de procesadores: 1
- Número total de núcleos: 2
- Caché de nivel 2: 3 MB
- Memoria RAM: 5 GB (DDR3, 1066 MHz)
- Velocidad del bus: 1.07 GHz

Herramientas de software

El software utilizado en este laboratorio se describe a continuación:

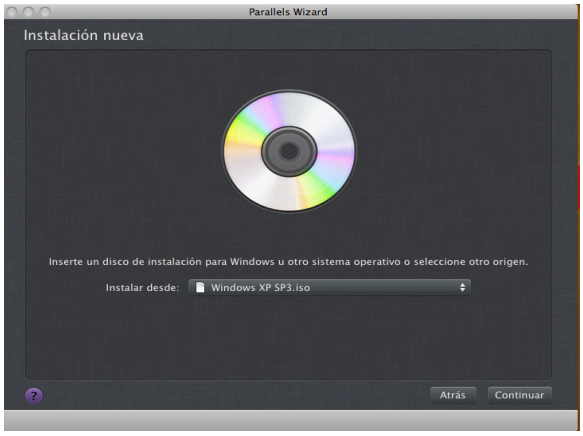
- Sistema operativo de la computadora: Mac OS X Snow Leopard versión 10.6.8
- Máquina Virtual: Parallels Desktop versión 8.0
- Sistema operativo de la máquina virtual: Windows XP SP3 (32 bits)
- Base de datos: Oracle 11g R2 Enterprise Edition para Windows de 32 bits

## 5.2 Instalación de la Máquina Virtual

La primera fase para la construcción del laboratorio de pruebas es instalar la máquina virtual en la cuál se va a realizar la parte experimental. Los pasos a seguir son los siguientes:

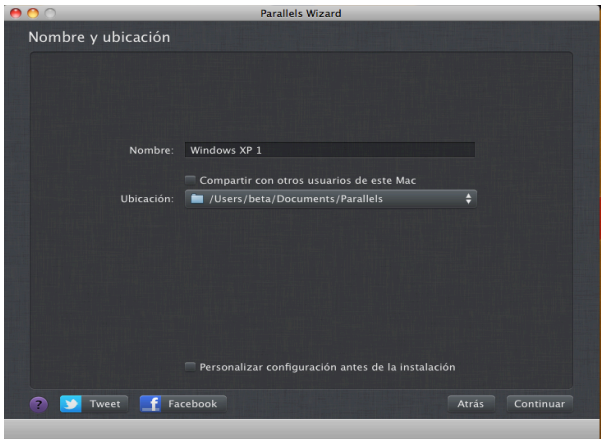
1. El primer paso consiste en iniciar la aplicación denominada "Parallels Desktop" y crear una nueva máquina virtual


2. Seleccionar la ubicación desde dónde se va a instalar la máquina virtual

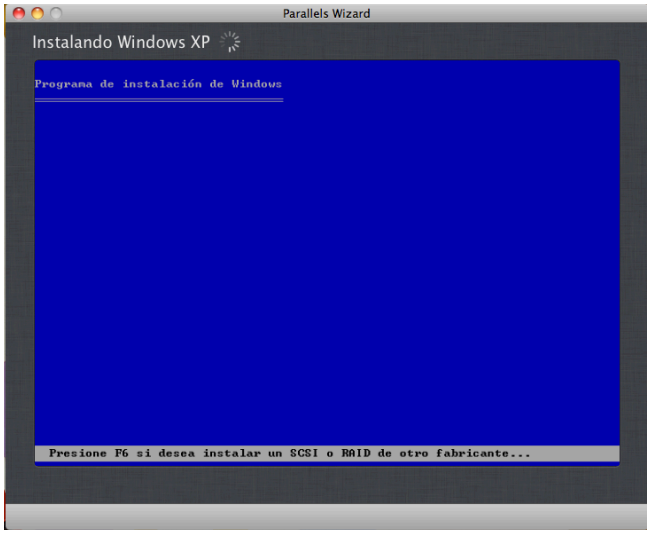



3. Seleccionar el tipo de integración de la máquina virtual con nuestro sistema operativo (en nuestro caso Mac OS X)

4. Elegir un nombre y una ubicación para esta nueva máquina virtual.




5. Esperar a que comience a instalarse la máquina virtual con el sistema operativo deseado (en nuestro caso se utilizó Windows XP SP3)

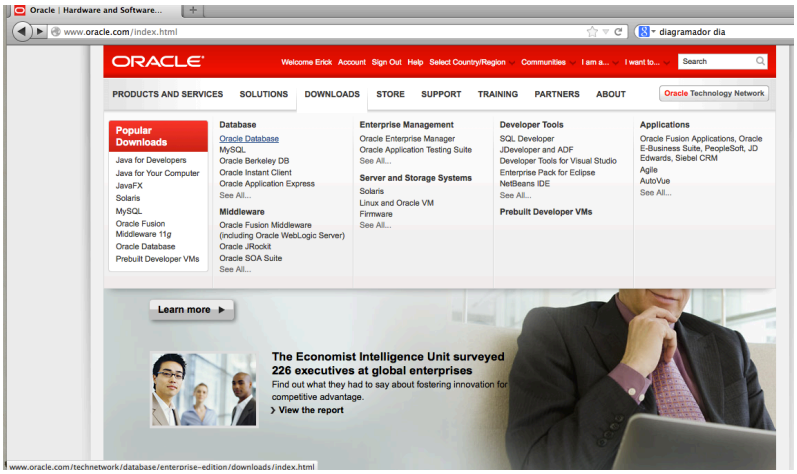


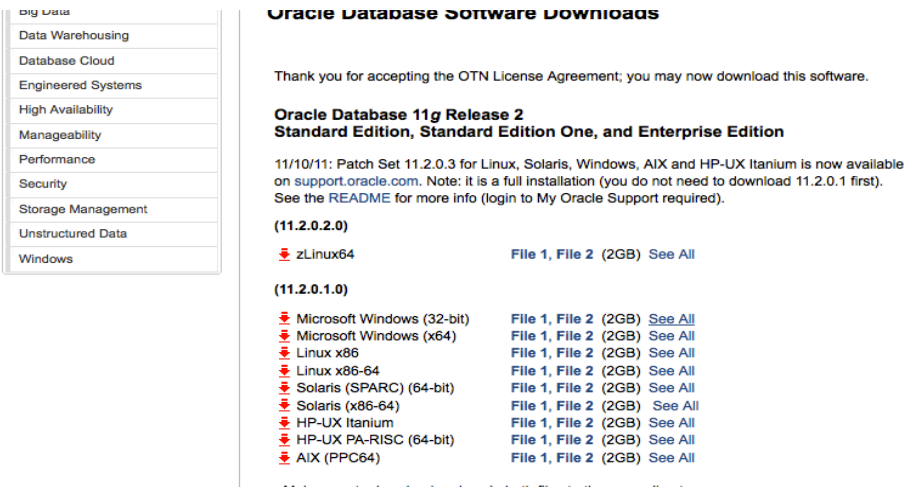

6. Finalmente tenemos instalado el sistema operativo en la máquina virtual, y se encuentra listo para utilizarse


## 5.3 Instalación de Oracle 11gR2 para Windows

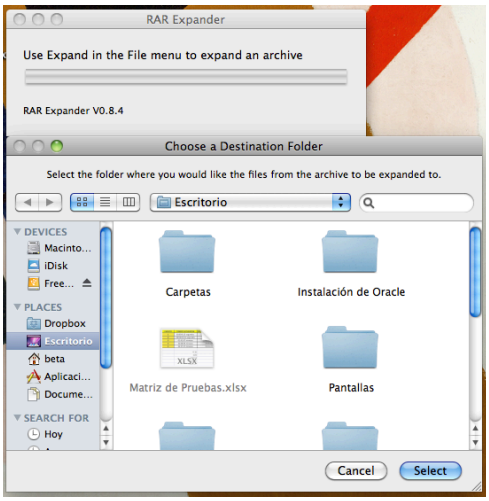
El siguiente paso en la creación de nuestro laboratorio de pruebas es la instalación de Oracle 11gR2 en la máquina virtual que se instaló con anterioridad. Los pasos a seguir son los siguientes:

1. El primer paso consiste en descargar el software de la página oficial de Oracle (www.oracle.com), y descargar la versión que se vaya a utilizar (en nuestro caso elegiremos Microsoft Windows de 32 bits)

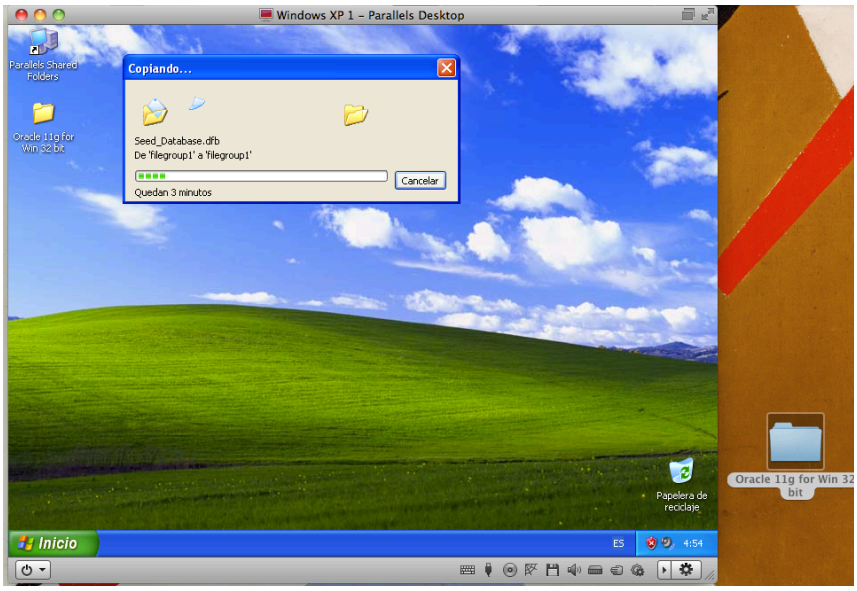





2. Una vez que finaliza la descarga del archivo, el siguiente paso es descomprimir el archivo con alguna herramienta para la descompresión de archivos (en nuestro caso utilizaremos la herramienta RAR Expander), después se selecciona el lugar en dónde se va a crear la carpeta Oracle 11g (nosotros seleccionaremos el Escritorio)

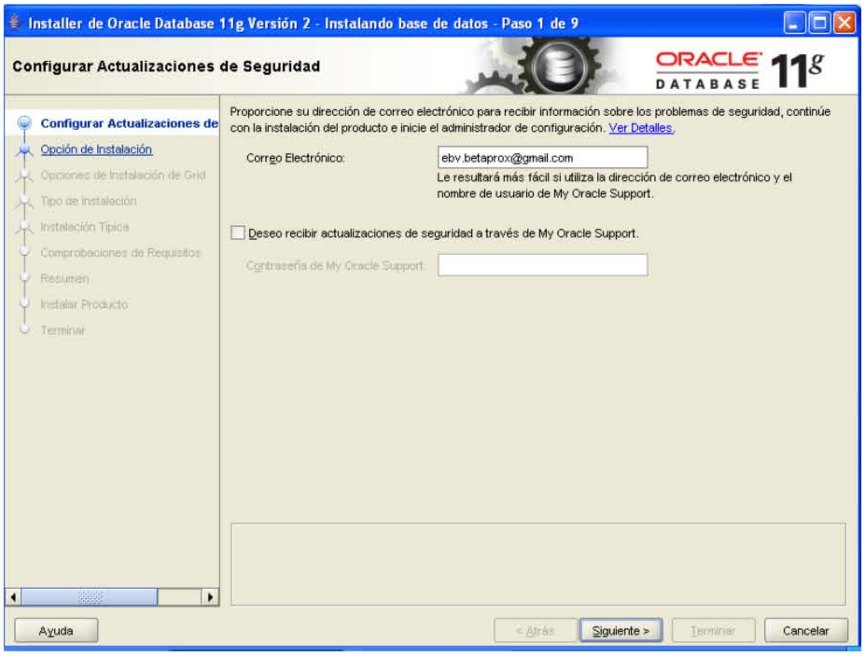


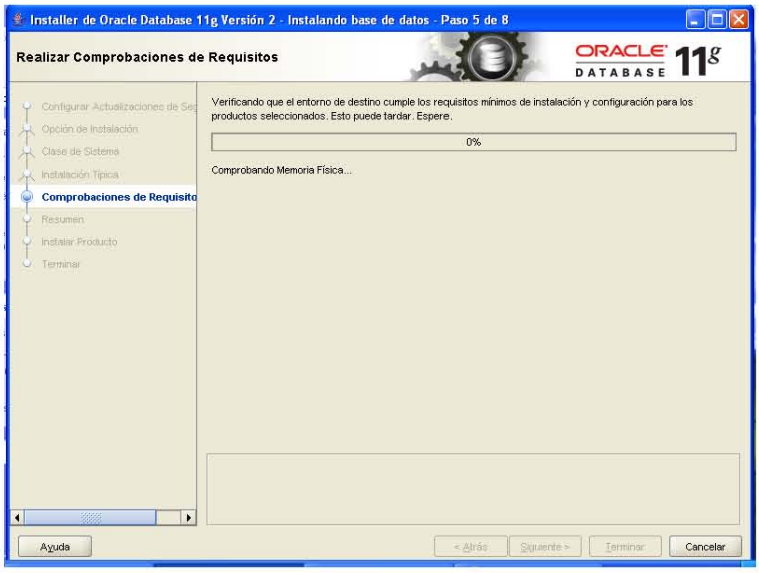

3. Una vez que se descomprime el archivo y se crea la carpeta, seleccionamos la carpeta y la arrastramos hasta nuestra máquina virtual

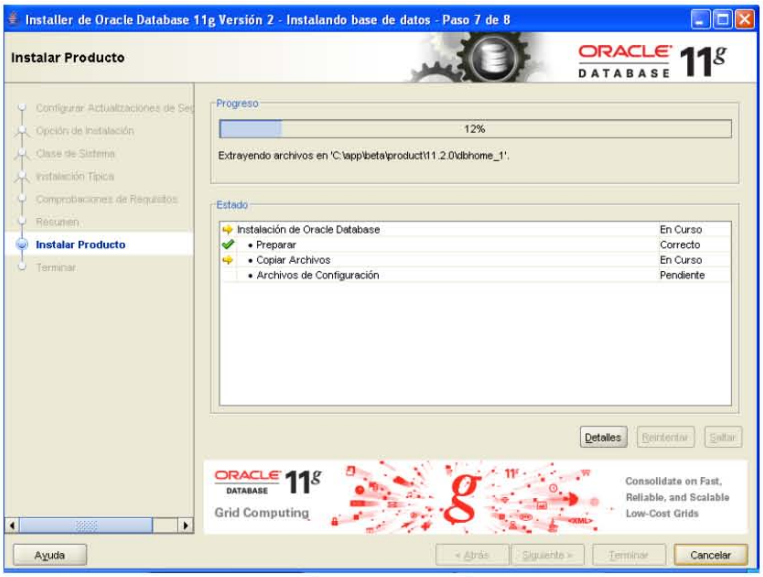


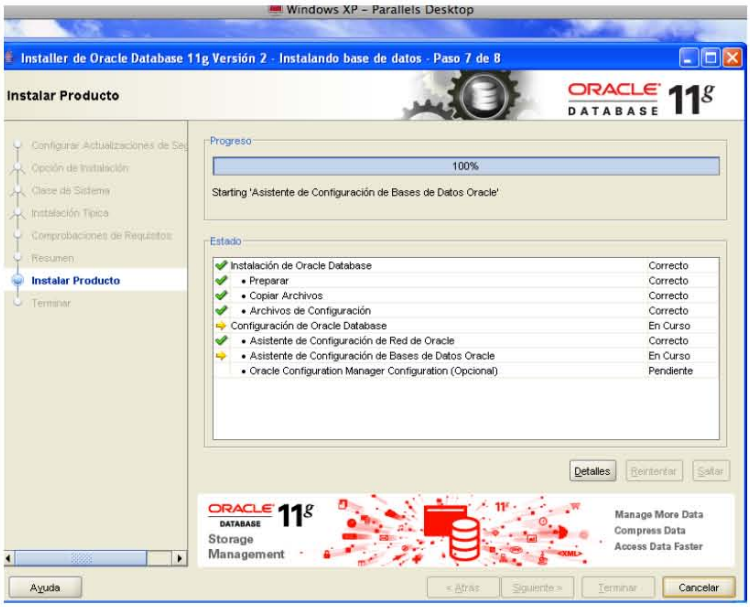

4. El siguiente paso consiste en abrir la carpeta que se copio en la máquina virtual y buscar el OUI (Oracle Universal Installer – Instalador Universal de Oracle), ejecutarlo y comenzar con la instalación de Oracle 11g en la máquina virtual

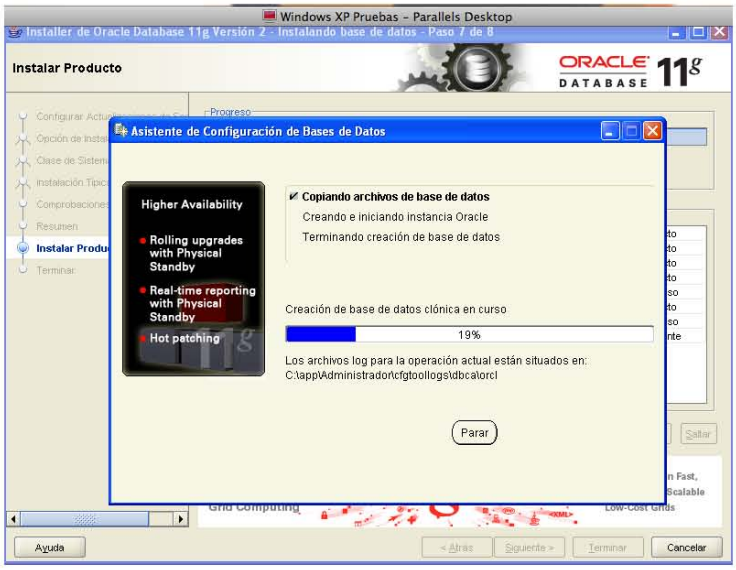

5. El siguiente paso es completar los campos que requiere el instalador y seguir los pasos para instalar la base de datos de Oracle, los cuáles nos va proporcionando el mismo instalador y esperar a que se termine la instalación


Installer de Oracle Database 11g Versión 2 - Instalando base de datos - Paso 1 de 9
Configurar Actualizaciones de Seguridad
ORACLE DATABASE 11g
Configurar Actualizaciones de
Opción de Instalación
Opciones de Instalación de Grid
Tipo de Instalación
Instalación Típica
Comprobaciones de Requisitos
Resumen
Instalar Producto
Terminar
Proporcione su dirección de correo electrónico para recibir información sobre los problemas de seguridad, continúe con la instalación del producto e inicie el administrador de configuración. Ver Detalles.
Correo Electrónico:
ebv.betaprox@gmail.com
Le resultará más fácil si utiliza la dirección de correo electrónico y el nombre de usuario de My Oracle Support.
Deseo recibir actualizaciones de seguridad a través de My Oracle Support.
Contraseña de My Oracle Support:
Ayuda
< Atrás
Siguiente >
Terminar
Cancelar

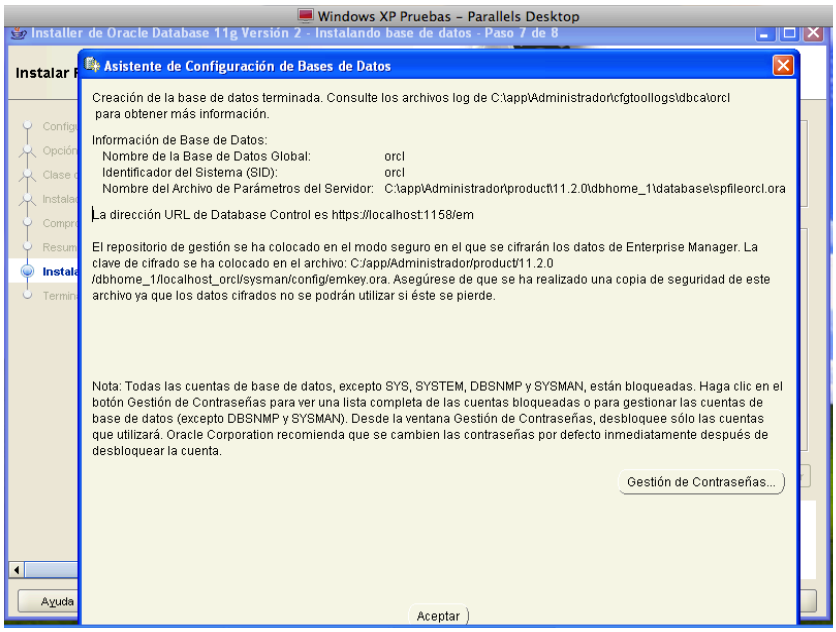

Installer de Oracle Database 11g Versión 2 - Instalando base de datos - Paso 5 de 8
Realizar Comprobaciones de Requisitos
ORACLE DATABASE 11g
Configurar Actualizaciones de Seg
Opción de Instalación
Clase de Sistema
Instalación Típica
Comprobaciones de Requisito
Resumen
Instalar Producto
Terminar
Verificando que el entorno de destino cumple los requisitos mínimos de instalación y configuración para los productos seleccionados. Esto puede tardar. Espere.
0%
Comprobando Memoria Física...
Ayuda
< Atrás
Siguiente >
Terminar
Cancelar


Installer de Oracle Database 11g Versión 2 - Instalando base de datos - Paso 7 de 8
Instalar Producto
ORACLE 11g DATABASE
Progreso
12%
Extrayendo archivos en 'C:\app\beta\product\11.2.0\dbhome_1'.
Estado
Instalación de Oracle Database En Curso
• Preparar Correcto
• Copiar Archivos En Curso
• Archivos de Configuración Pendiente
Detalles
Reintentar
Saltar
ORACLE 11g DATABASE
Grid Computing
Consolidate on Fast, Reliable, and Scalable Low-Cost Grids
Ayuda
« Atrás
Siguiente »
Terminar
Cancelar


Windows XP - Parallels Desktop
Installer de Oracle Database 11g Versión 2 - Instalando base de datos - Paso 7 de 8
Instalar Producto
ORACLE 11g DATABASE
Progreso
100%
Starting 'Asistente de Configuración de Bases de Datos Oracle'
Estado
Instalación de Oracle Database Correcto
• Preparar Correcto
• Copiar Archivos Correcto
• Archivos de Configuración Correcto
Configuración de Oracle Database En Curso
• Asistente de Configuración de Red de Oracle Correcto
• Asistente de Configuración de Bases de Datos Oracle En Curso
• Oracle Configuration Manager Configuration (Opcional) Pendiente
Detalles
Reintentar
Saltar
ORACLE 11g DATABASE
Storage Management
Manage More Data
Compress Data
Access Data Faster
Ayuda
« Atrás
Siguiente »
Terminar
Cancelar


Windows XP Pruebas - Parallels Desktop
Instalar Producto
ORACLE 11g DATABASE
Asistente de Configuración de Bases de Datos
Higher Availability
Rolling upgrades with Physical Standby
Real-time reporting with Physical Standby
Hot patching
Copiando archivos de base de datos
Creando e iniciando instancia Oracle
Terminando creación de base de datos
Creación de base de datos clónica en curso
19%
Los archivos log para la operación actual están situados en:
C:\app\Administrador\cfgtoollogs\dbca\orcl
Parar
Ayuda
« Atrás
Siguiente »
Terminar
Cancelar

6. En esta ventana se nos muestra el Nombre Global de la Base de Datos y el Identificador del Sistema (SID), que es "orcl" para ambos en nuestro caso




7. Finalmente tenemos instalado Oracle 11gR2 en la máquina virtual, y se encuentra listo para poder utilizarse, como parte final de la instalación abrimos la herramienta de desarrollo de Oracle, denominada Oracle SQL Developer, para verificar que la herramienta se encuentra funcionando correctamente

## 5.4 Creación de tablas e inserción de datos

Para efectos de las pruebas se crearan tres tablas en el esquema (schema) "Scott", el cuál es un esquema que se encuentra definido en la base de datos desde que se creó, es importante mencionar que existen otros esquemas que al igual que el esquema Scott ya vienen predefinidos en la instalación de la base de datos, tales como el esquema HR,BI, etc. De igual forma el usuario que se utilizará en las pruebas será el usuario denominado "Scott".
Las tablas que se crearán en el esquema Scott son la tabla "personal", "tipo_pagos" y "entidad federativa". las dos primeras tablas se encuentran compuestas por diez columnas cada una y la tercera por dos columnas. A continuación se describen las columnas que lleva cada una de las tablas y si :

Tabla "PERSONAL"

- Empleado, Anio, Mes_Anio, Dia, Sexo, Id_Nacimiento, Id_Trabajo, Apellido_Paterno, Apellido_Materno, Nombre

Tabla "TIPO_PAGOS"

- Pago, Empleado, Interna, Parte, Agno, Mes, Quincena, Percepcion, Deducción, Líquido

Tabla "ENTIDAD_FEDERATIVA"

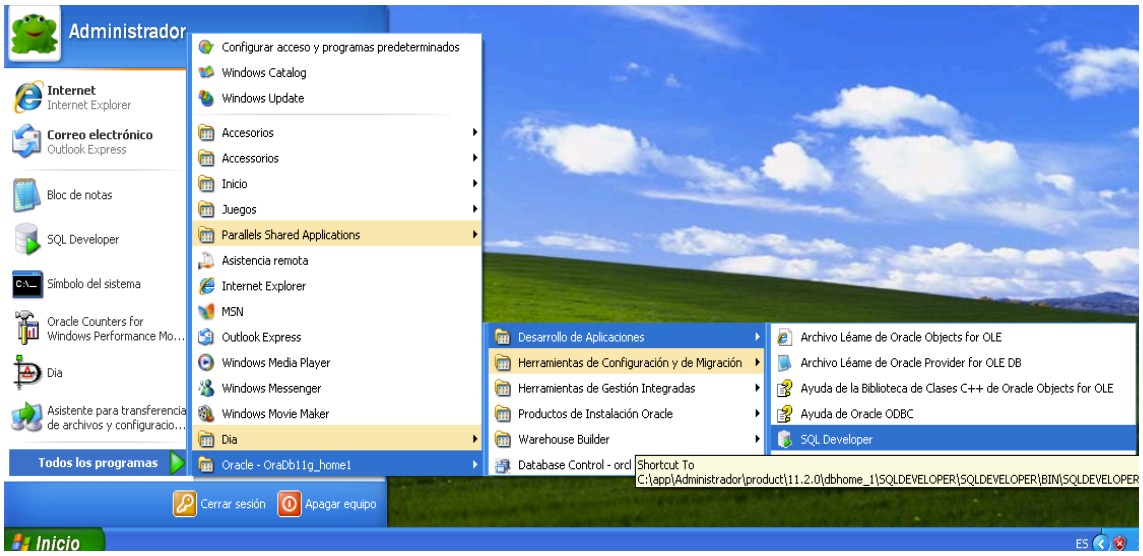
- Entidad, Estado

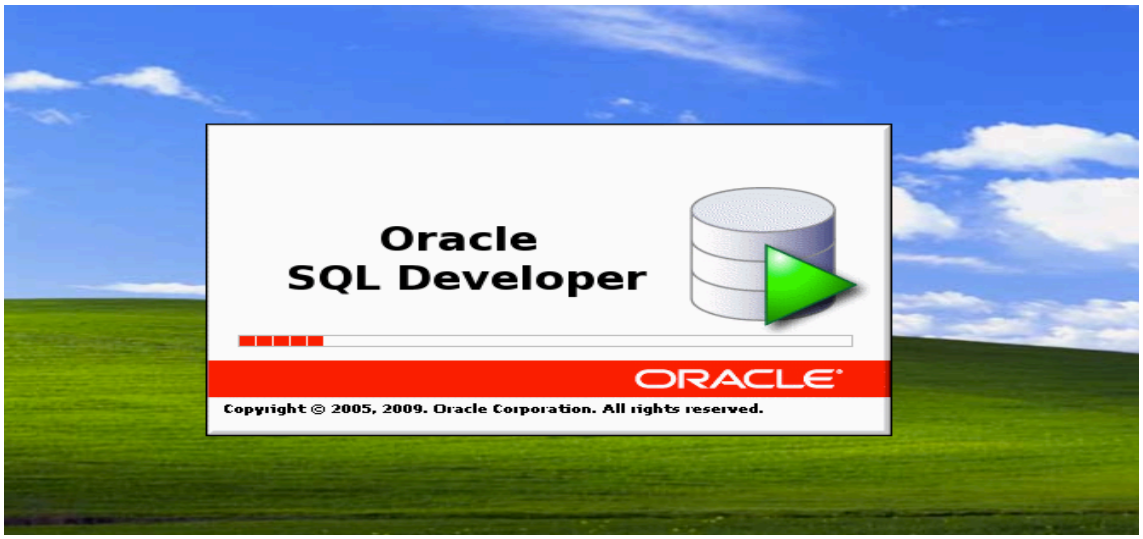
Oracle nos proporciona dos formas de poder interactuar con la base de datos, la primera es mediante la herramienta de desarrollo Oracle SQL DEVELOPER y la segunda es la herramienta SQL*PLUS. Ambas herramientas nos sirven para el desarrollo en bases de datos Oracle, la diferencia que existe entre ellas es que Oracle SQL Developer nos presenta una interfaz gráfica y SQL*PLUS es una herramienta utilizada mediante línea de comandos.

En nuestra caso se hizo uso de ambas herramientas para la ejecución de las pruebas. Las tablas se van a crear mediante la herramienta Oracle SQL Developer. Posteriormente la carga de datos, la alteración en la ejecución de las consultas y el tiempo de ejecución se hará a través de la herramienta SQL*PLUS.

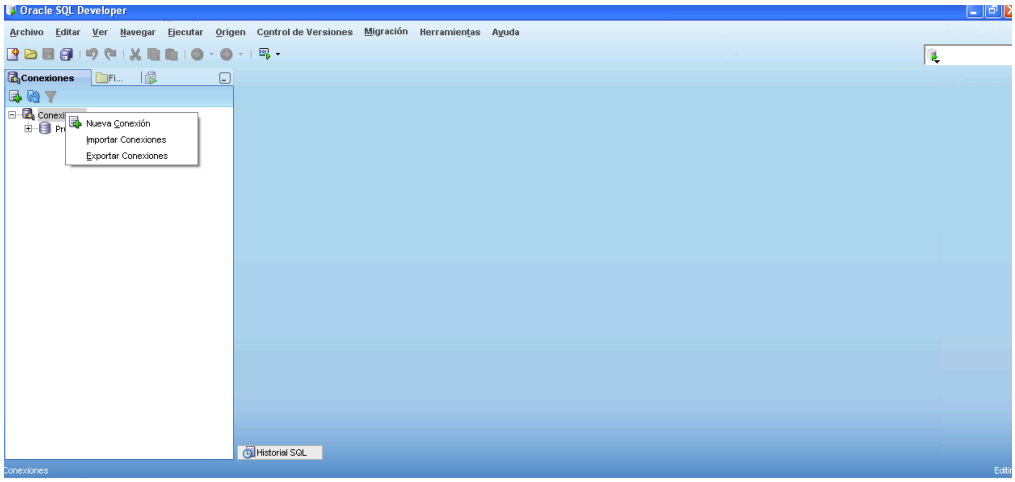
Para la creación de las las tablas se deben seguir los siguientes pasos:

1. Abrir la interfaz Oracle SQL Developer, como se muestra a continuación

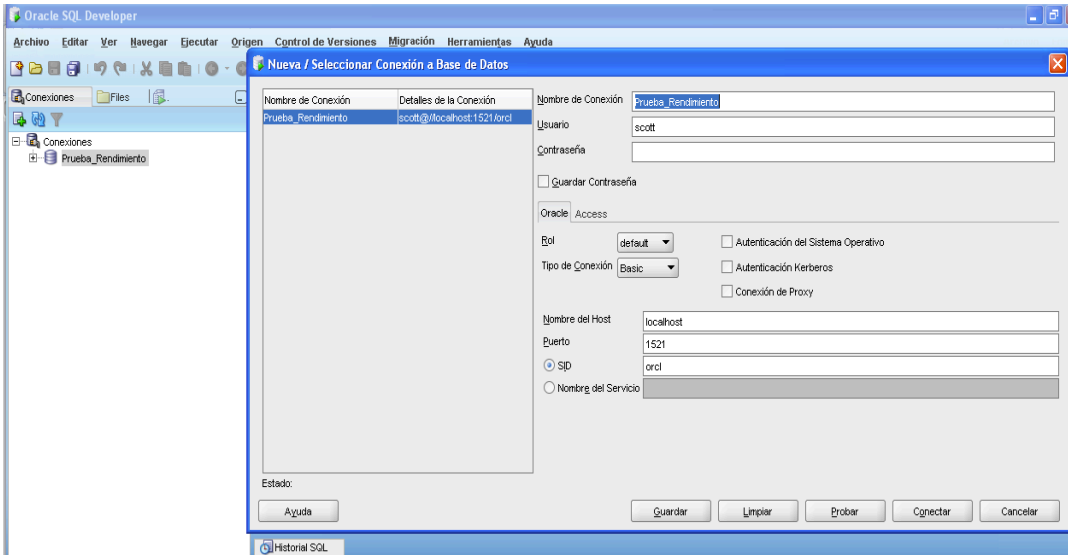


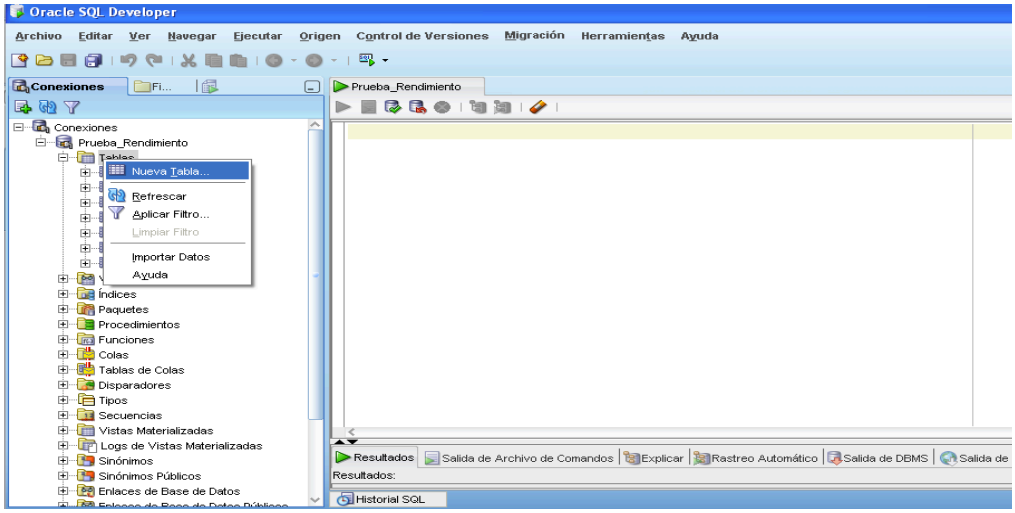
2. Una vez abierta la herramienta Oracle SQL Developer, se ubica el área de conexiones, y en dónde dice "Conexiones" se da clic segundario y se elige la opción *Nueva Conexión*

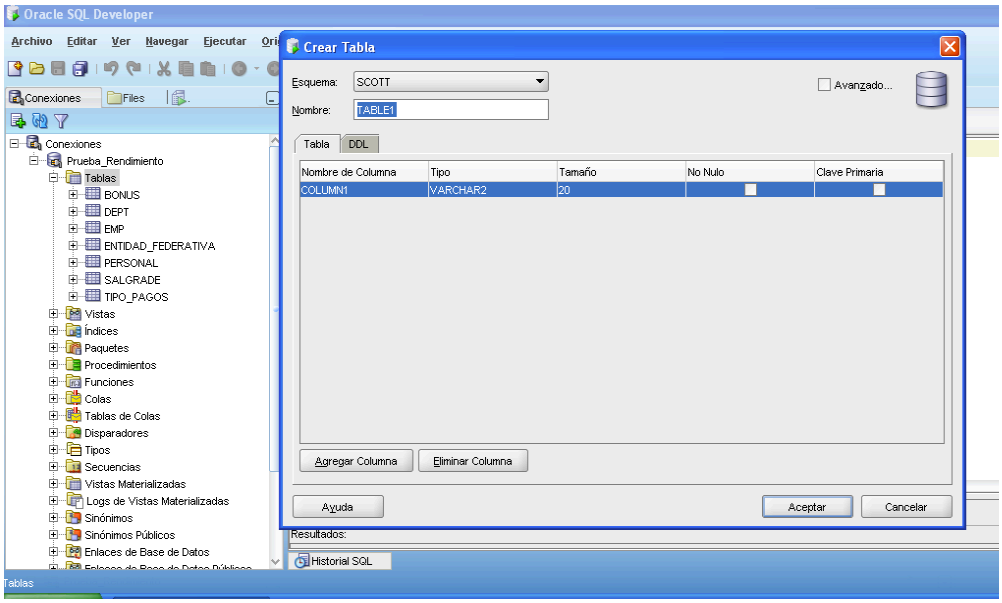



3. La herramienta requiere que proporcionemos cierta información como el nombre de conexión, usuario, contraseña y el SID, una vez proporcionada la información solicitada se da clic sobre el botón *Probar*, para verificar si es correcta la conexión, posteriormente damos clic sobre el botón *Guardar* y finalmente damos clic sobre el botón *Conectar*, para poder conectarnos a la base de datos

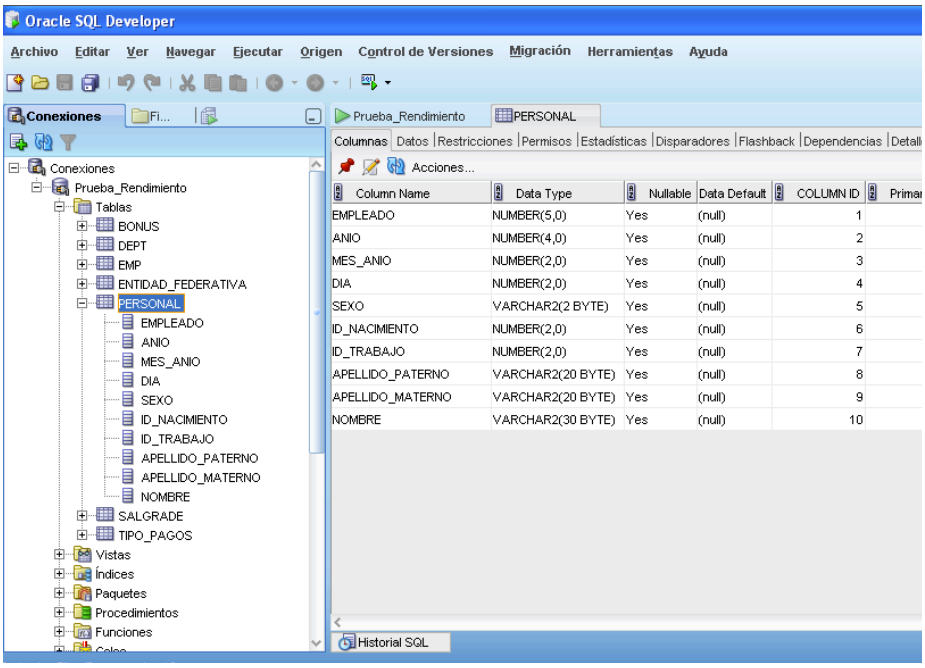


4. Habiendo realizado la conexión, nos situamos en el botón de *Tablas* y efectuamos un clic secundario y elegimos la opción *Nueva Tabla*, posteriormente nos va a desplegar una ventana que nos solicitará la información que deseamos incluir en la tabla a crear, tal como el nombre de la tablas, el nombre de la columna, el tipo de dato, el tamaño, etc., en la parte inferior de la ventana se puede agregar o eliminar columnas, una vez que especificamos la información que va a tener la tabla damos clic en el botón *Aceptar* y nuestra tabla habrá sido creada finalmente, a continuación se muestran las impresiones de pantalla de lo que se ha descrito:



Al crearse la tabla, está se verá de la siguiente forma:

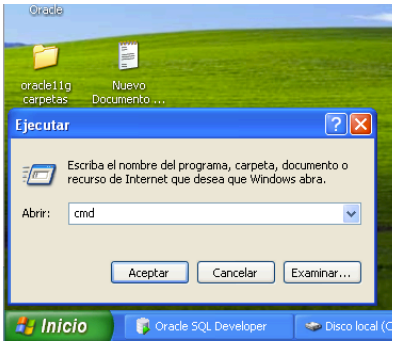



Después de crear las tres tablas que se usarán en las pruebas a realizar, lo que prosigue es cargar los datos en cada una de las tablas. Como se mencionó en la parte introductoria del capítulo, se medirán los tiempos de ejecución de algunas consultas realizadas a la base de datos, para poder efectuar las pruebas se utilizaran dos archivos que contienen una cantidad considerable de información, sobre todo en las tablas de "TIPO_PAGOS" la cuál tiene 399999 filas y "PERSONAL" que tiene 21501 filas. Esta información será cargada a las tablas a partir de dos archivos en formato *.csv* (formato de Windows en el cuál los valores se encuentran separados por comas), mediante la herramienta SQL LOADER, la cuál es proporcionada por Oracle para la carga de datos.

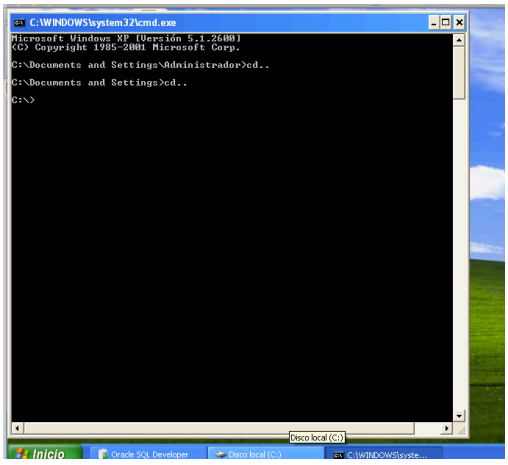
5. Este paso consiste en realizar la carga de datos a través de la herramienta SQL LOADER. Para poder cargar los datos en cada una de las tablas es necesario generar un archivo de control. Esto se realiza de la siguiente manera:

- Efectuamos un clic sobre el botón de *Inicio* y se mostrara una ventana con el menú del botón de inicio, localizamos el apartado en dónde dice ejecutar y damos clic


- Se mostrará una ventana, en dicha ventana escribimos *cmd*

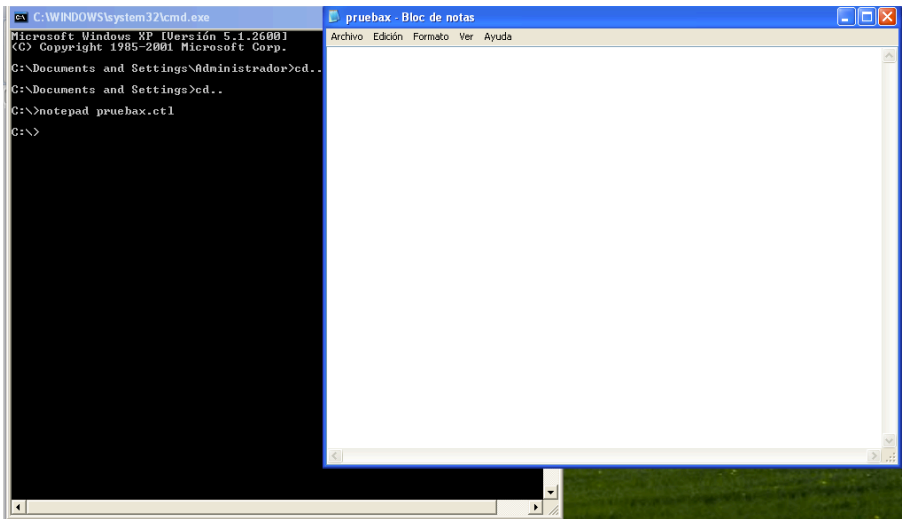



- Y nos abrirá una ventana de línea de comandos (también conocida como consola MS-DOS)



- Una vez abierta la línea de comandos tecleamos lo siguiente: *C:\> notepad pag.ctl*

EL efecto que tiene lo escrito en la línea de comandos es abrirnos un programa llamado *Bloc de Notas* el cuál fue invocado mediante la palabra *notepad*, la palabra *prueba* hace referencia al nombre con el cuál vamos a llamar a nuestro archivo de control, el sistema identifica que es un archivo de control dado que determinamos su extensión mediante los que no encuentra un archivo con el nombre de *pag.ctl* (esto es debido a que no existe, pues apenas se va a crear), y nos pregunta si deseamos crear un archivo nuevo, para responder a la pregunta damos clic sobre el botón *Sí.*



- El archivo de control nombrado "pag.ctl", debe tener la siguiente estructura:

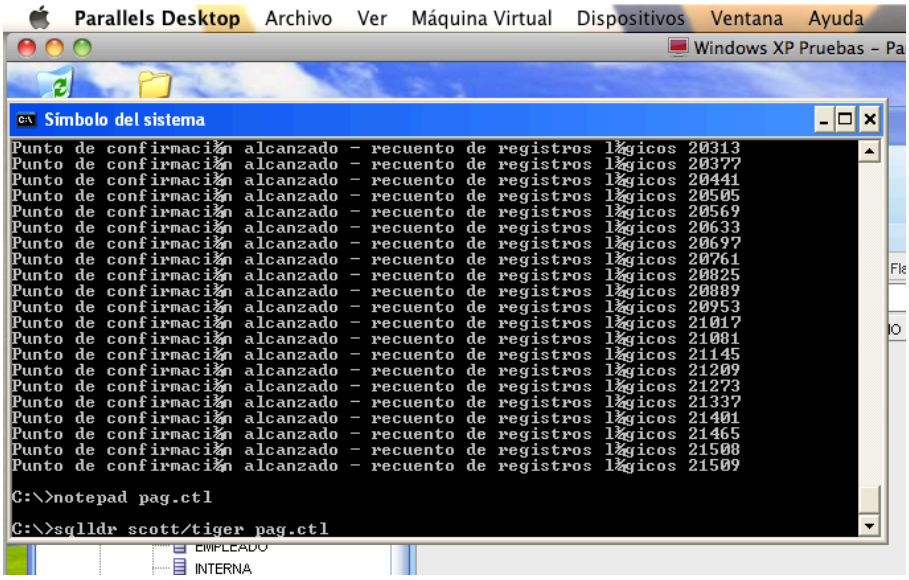
```
load data
infile 'c:\PAGOS.csv'
append
into table TIPO_PAGOS
fields terminated by ","
(
PAGO,
EMPLEADO,
INTERNA,
PARTE,
AGNO,
MES,
QUINCENA,
PERCEPCION,
DEDUCCION,
LIQUIDO
)
```

Es importante explicar que es lo que se realiza en cada una de las líneas del archivo de control. En la primera línea aparece la palabra *load data*, ésta le dice a SQL Loader que se van a insertar datos a la base, en la segunda línea aparece *infile 'c:\PAGOS.csv'*, esto es la ubicación (que es c:\) del archivo llamado *PAGOS.csv* el cuál vamos a cargar en la base de datos. En la tercera línea se encuentra la palabra *append*, esto significa que la inserción de datos debe realizarse sin borrar datos preexistentes. En la cuarta línea aparece *into table TIPO_PAGOS*, esto es en nombre de la tabla en la cuál se van a cargar los datos. En la quinta línea aparece *fields terminated by ","* esto significa que los campos se encuentran separados por una coma, y en el resto de las líneas se aprecian el nombre de las columnas que conforman la tabla en la que se van a cargar los datos. Los mismos pasos se deben de seguir para generar el archivo de control de la tabla "PERSONAL".

6. Una vez que se ha generado el archivo de control, lo siguiente es realizar la carga, la cuál se efectúa escribiendo desde la línea de comandos lo siguiente:

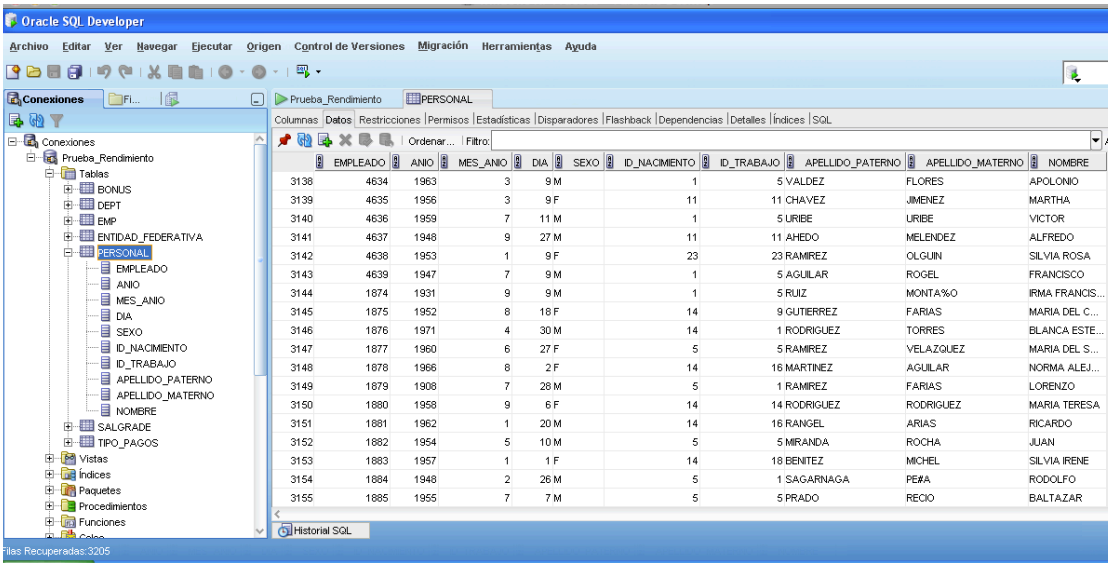
*C:\sqlldr scott/tiger control = pag.ctl*

Primero se invoca a la herramienta SQL LOADER (*sqllldr)*, después se escribe el nombre del usuario y su contraseña (*scott/tiger*) y finalmente se especifíca el nombre del archivo de control (*control = pag.ctl*). Y obtenemos un resultado como el que se muestra a continuación:




El mismo procedimiento debe realizarse para la carga de datos de la tabla “PERSONAL”.

7. Una forma gráfica de corroborar si los datos cargados se encuentran en la tabla es visualizando la misma mediante la herramienta Oracle SQL Developer, esto es eligiendo la tabla a visualizar y en el apartado de datos se da un clic y podemos ver desplegados en pantalla todos los datos que se encuentran almacenados en la tabla, como se muestra a continuación:




| EMPLEADO | ANIO | MES_ANIO | DIA | SEXO | ID_NACIMIENTO | ID_TRABAJO | APELLIDO_PATERNO | APELLIDO_MATERNO | NOMBRE |
|---|---|---|---|---|---|---|---|---|---|
| 3138 | 4634 | 1963 | 3 | 9 M | 1 | 5 | VALDEZ | FLORES | APOLONIO |
| 3139 | 4635 | 1956 | 3 | 9 F | 11 | 11 | CHAVEZ | JIMENEZ | MARTHA |
| 3140 | 4636 | 1959 | 7 | 11 M | 1 | 5 | URIBE | URIBE | VICTOR |
| 3141 | 4637 | 1948 | 9 | 27 M | 11 | 11 | AHEDO | MELENDEZ | ALFREDO |
| 3142 | 4638 | 1953 | 1 | 9 F | 23 | 23 | RAMIREZ | OLGUIN | SILVIA ROSA |
| 3143 | 4639 | 1947 | 7 | 9 M | 1 | 5 | AGUILAR | ROGEL | FRANCISCO |
| 3144 | 1874 | 1931 | 9 | 9 M | 1 | 5 | RUIZ | MONTA%O | IRMA FRANCIS... |
| 3145 | 1875 | 1952 | 8 | 18 F | 14 | 9 | GUTIERREZ | FARIAS | MARIA DEL C... |
| 3146 | 1876 | 1971 | 4 | 30 M | 14 | 1 | RODRIGUEZ | TORRES | BLANCA ESTE... |
| 3147 | 1877 | 1960 | 6 | 27 F | 5 | 5 | RAMIREZ | VELAZQUEZ | MARIA DEL S... |
| 3148 | 1878 | 1966 | 8 | 2 F | 14 | 16 | MARTINEZ | AGUILAR | NORMA ALEJ... |
| 3149 | 1879 | 1908 | 7 | 28 M | 5 | 1 | RAMIREZ | FARIAS | LORENZO |
| 3150 | 1880 | 1958 | 9 | 6 F | 14 | 14 | RODRIGUEZ | RODRIGUEZ | MARIA TERESA |
| 3151 | 1881 | 1962 | 1 | 20 M | 14 | 16 | RANGEL | ARIAS | RICARDO |
| 3152 | 1882 | 1954 | 5 | 10 M | 5 | 5 | MIRANDA | ROCHA | JUAN |
| 3153 | 1883 | 1957 | 1 | 1 F | 14 | 18 | BENITEZ | MICHEL | SILVIA IRENE |
| 3154 | 1884 | 1948 | 2 | 26 M | 5 | 1 | SAGARNAGA | PE#A | RODOLFO |
| 3155 | 1885 | 1955 | 7 | 7 M | 5 | 5 | PRADO | RECIO | BALTAZAR |

La misma acción debe realizarse para poder visualizar cualquier tabla que se desee. Sólo para como parte de confirmación a continuación se muestra una impresión de pantalla, en la cuál haciendo uso de la herramienta de SQL*PLUS se muestra que las tablas han sido creadas de las tablas, así como las columnas que contiene cada una de ellas.

Lo que se hace es lo siguiente, desde la línea de comandos de Windows abrimos la herramienta SQL*PLUS tecleando:

*C:\sqlplus*

Se abre la herramienta y nos solicita un nombre de usuario y una contraseña. Una vez conectados a la base de datos de Oracle con el usuario correspondiente, le indicamos a ésta que nos muestre las tablas que se encuentran almacenadas en la base de datos correspondientes al esquema con el cuál estamos trabajando. Esto se hace mediante el siguiente comando forma:

```
SQL> SELECT table_name FROM user_tables;
```

La respuesta que obtenemos por parte de la base de datos es la siguiente:

```
TABLE_NAME
------------------------------
DEPT
EMP
BONUS
SALGRADE
PERSONAL
TIPO_PAGOS
ENTIDAD_FEDERATIVA

7 filas seleccionadas.
```

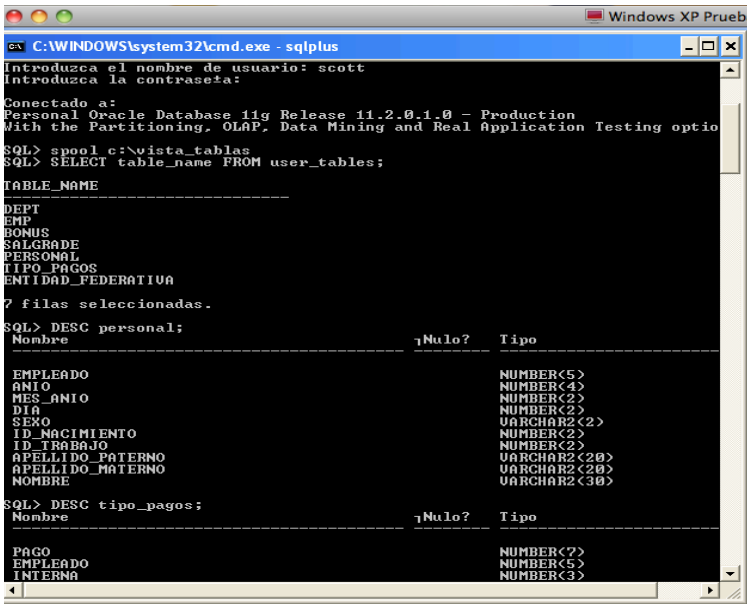
Como se puede observar hay siete tablas diferentes en este esquema, incluyendo las tres tablas que creamos con anterioridad, las cuáles serán las únicas que se utilizaran en la ejecución de las pruebas. Posteriormente le indicamos a la base de datos que deseamos visualizar la descripción de la tabla “PERSONAL”.

```
SQL> DESC personal;
```

El resultado que se arroja la base de datos es el siguiente:

```
Nombre                                    ¿Nulo?   Tipo
----------------------------------------- -------- ------------------
 EMPLEADO                                          NUMBER(5)
 ANIO                                              NUMBER(4)
 MES_ANIO                                          NUMBER(2)
 DIA                                               NUMBER(2)
 SEXO                                              VARCHAR2(2)
 ID_NACIMIENTO                                     NUMBER(2)
 ID_TRABAJO                                        NUMBER(2)
 APELLIDO_PATERNO                                  VARCHAR2(20)
 APELLIDO_MATERNO                                  VARCHAR2(20)
 NOMBRE                                            VARCHAR2(30)
```

Esto lo vemos ahora en una impresión de pantalla:




## 5.5 Ejecución de pruebas

La fase de ejecución de pruebas consiste en realizar diversos eventos en los cuáles pueda compararse el procesamiento secuencial y el procesamiento en paralelo, obteniendo así, una salida que permita analizar cada uno de los dos procesamientos.

La métrica principal a evaluar en esta fase, es el tiempo de respuesta que presenta cada uno de los procesamientos ante una consulta

realizada a la base de datos, para su estudio y análisis se ha determinado realizar 5 pruebas diferentes.

Las pruebas a realizarse son las siguientes:

A. La primera consulta que se debe realizar a la base de datos es contar el número de registros que tiene la tabla "PERSONAL. La consulta en lenguaje SQL es la siguiente:

```
SQL> SELECT COUNT(*) FROM personal;
```

B. La segunda consulta que se debe realizar a la base de datos es contar el número de registros que tiene la tabla "TIPO_PAGOS". La consulta en lenguaje SQL es la siguiente:

```
SQL> SELECT COUNT(*) FROM tipo_pagos;
```

C. La tercera consulta que se debe efectuar a la base de datos consiste en seleccionar todos los datos almacenados en la tabla "PERSONAL". La consulta en lenguaje SQL es la siguiente:

```
SQL> SELECT * FROM personal;
```

D. La cuarta consulta que se debe efectuar a la base de datos consiste en seleccionar todos los datos almacenados en la tabla "TIPO_PAGOS". La consulta en lenguaje SQL es la siguiente:

```
SQL> SELECT * FROM tipo_pagos;
```

E. La quinta y última consulta que se debe realizar a la base de datos consiste en obtener todas las filas de ocho columnas tanto de la tabla "PERSONAL" como de la tabla "TIPO_PAGOS"

sin importar si tienen o no correspondencia alguna. La consulta en lenguaje SQL es la siguiente:

```
SQL> SELECT tipo_pagos.pago, tipo_pagos.parte,
     tipo_pagos.interna, tipo_pagos.interna,
     tipo_pagos.agno, tipo_pagos.mes,
     tipo_pagos.quincena, tipo_pagos.liquido
     FROM tipo_pagos
     FULL JOIN personal
     ON tipo_pagos.empleado = personal.empleado
     ORDER BY tipo_pagos.pago
```

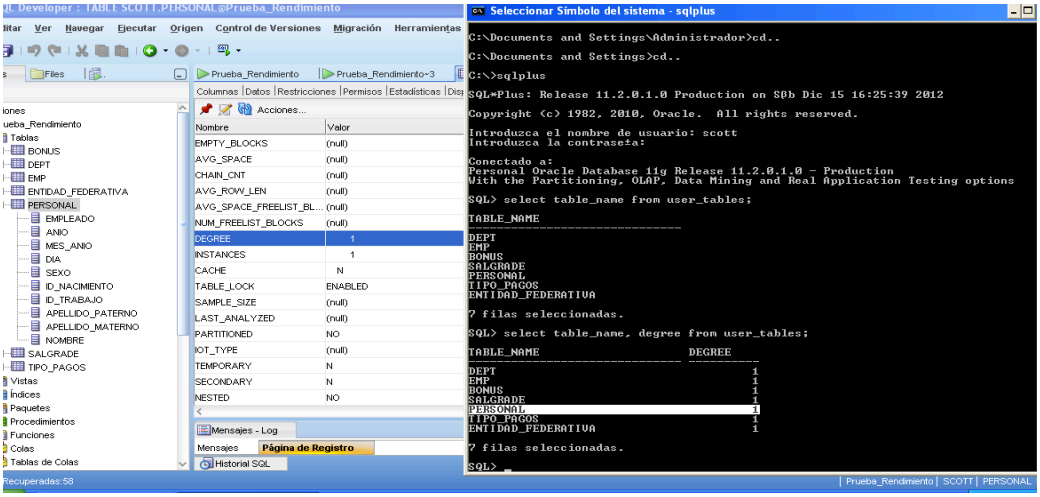
Los resultados obtenidos en las pruebas se muestran a continuación mediante el apoyo de impresiones de pantalla capturadas en el transcurso de cada una de las ejecuciones. La primera sección de pruebas corresponde a las que fueron efectuadas con procesamiento secuencial, posteriormente se presentaran las pruebas realizadas con procesamiento en paralelo. Para cada una de las pruebas se mostrará una pantalla en donde se muestra el grado con el que fueron efectuadas y el tiempo de transcurrido al ejecutar cada una de las consultas. En el caso especial de la prueba D, al ser la tabla que contiene la mayor cantidad de datos se presentará el plan de ejecución que arrojó la base de datos.

Pruebas con procesamiento Secuencial

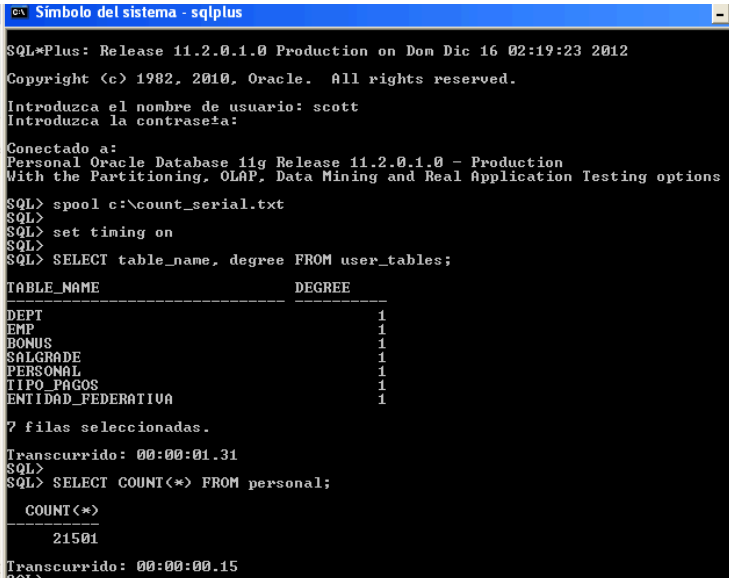
Las pruebas ejecutadas en este tipo de procesamiento son de grado uno, lo que significa que únicamente hubo un proceso esclavo durante la ejecución de cada una de las pruebas. Los resultados fueron los siguientes:

- Prueba A

En esta primera impresión de pantalla se muestra el grado de paralelismo con que antes de efectuar la consulta, el cuál puede observarse que es el mismo tanto en la herramienta Oracle SQL Developer como en la herramienta SQL*PLUS.

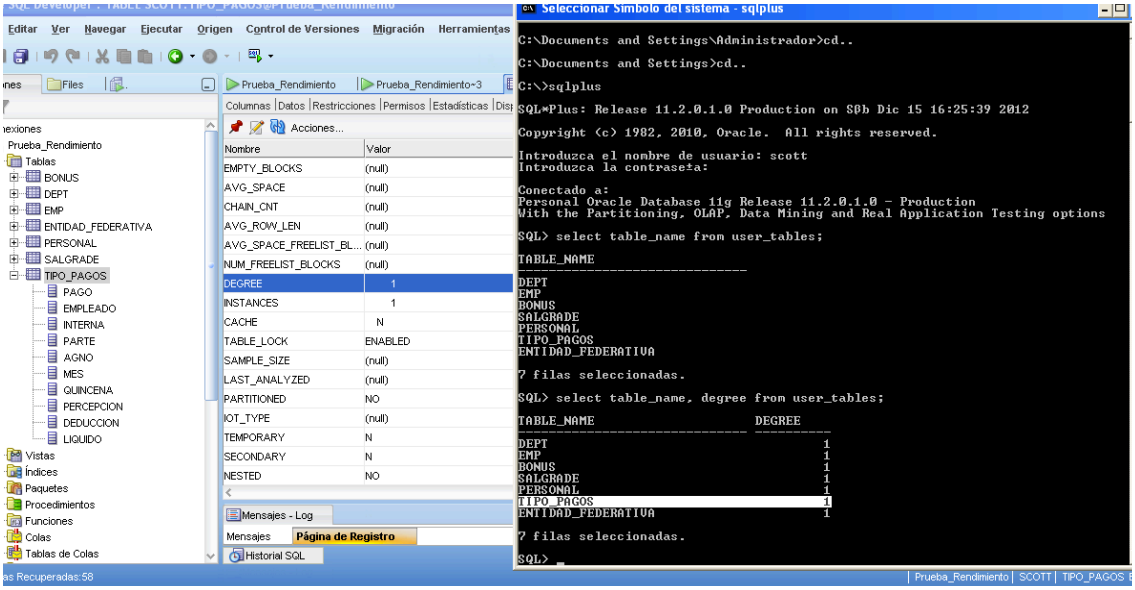



En esta segunda pantalla se muestra el tiempo transcurrido después de haber efectuado la consulta. El tiempo obtenido fue de 00:00:00.15 segundos.

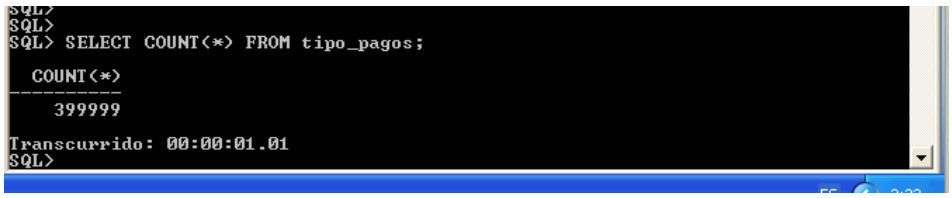


- Prueba B

En la primera impresión de pantalla se muestra el grado de paralelismo antes de efectuar la consulta.

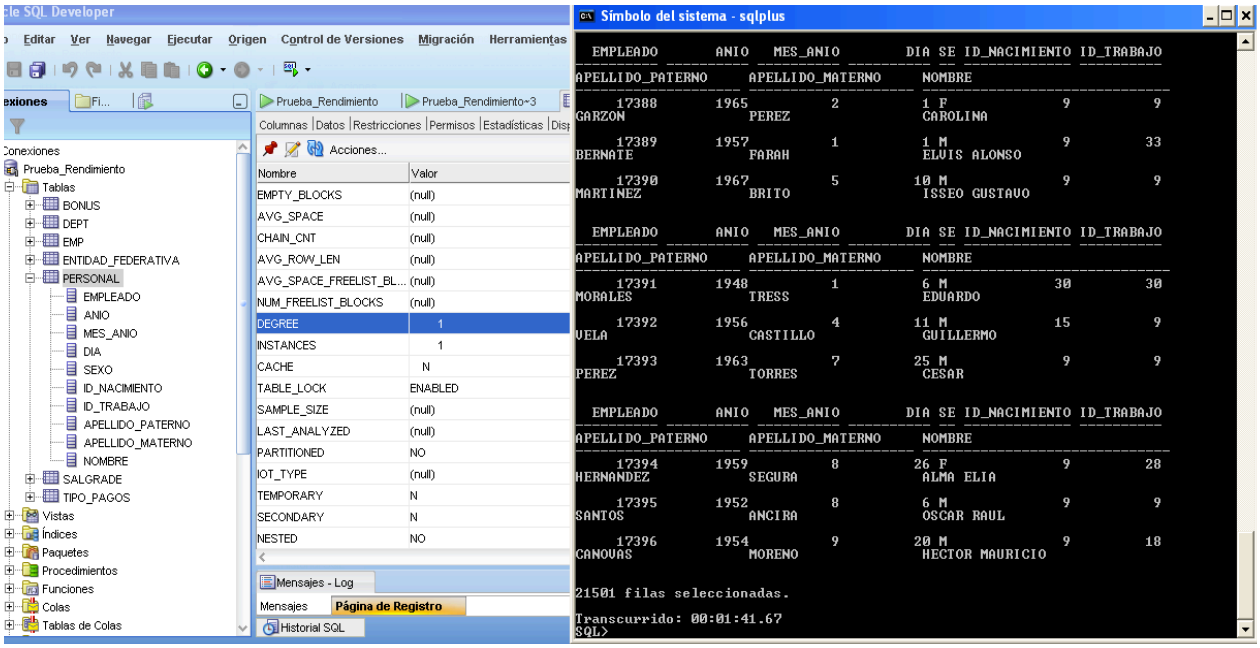



En la segunda pantalla se muestra el tiempo transcurrido después de haber efectuado la consulta. El tiempo obtenido fue de 00:00:01.01 segundos.



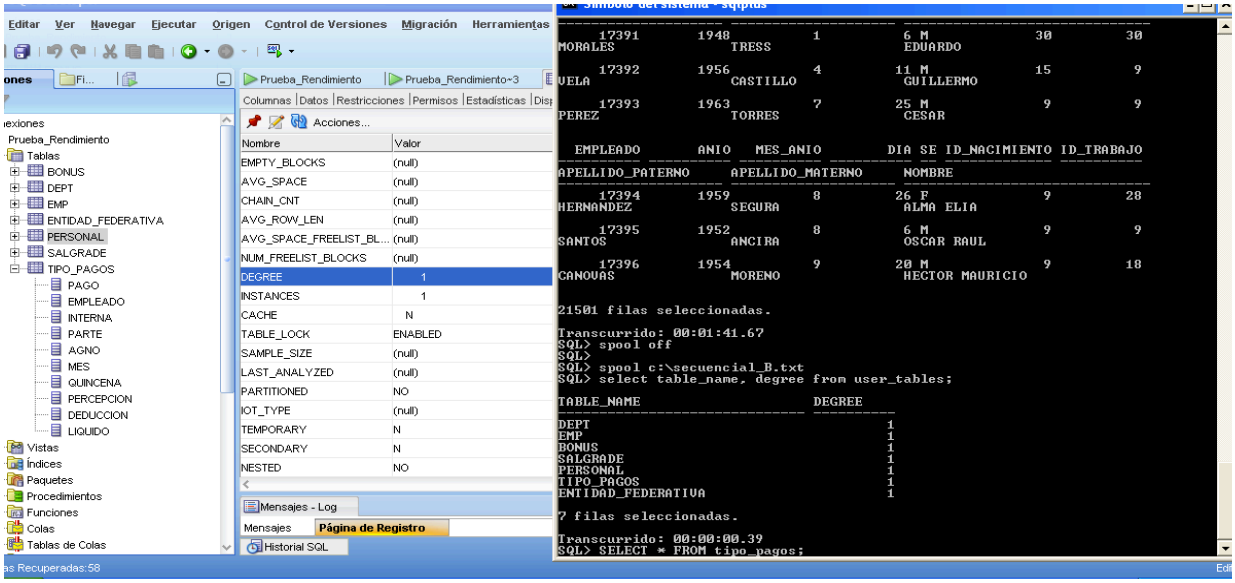

- Prueba C

En esta impresión de pantalla se muestra el grado al momento de efectuar la consulta y el tiempo transcurrido en la ejecución de la misma, el cual fue de 00:01:41.67 minutos.

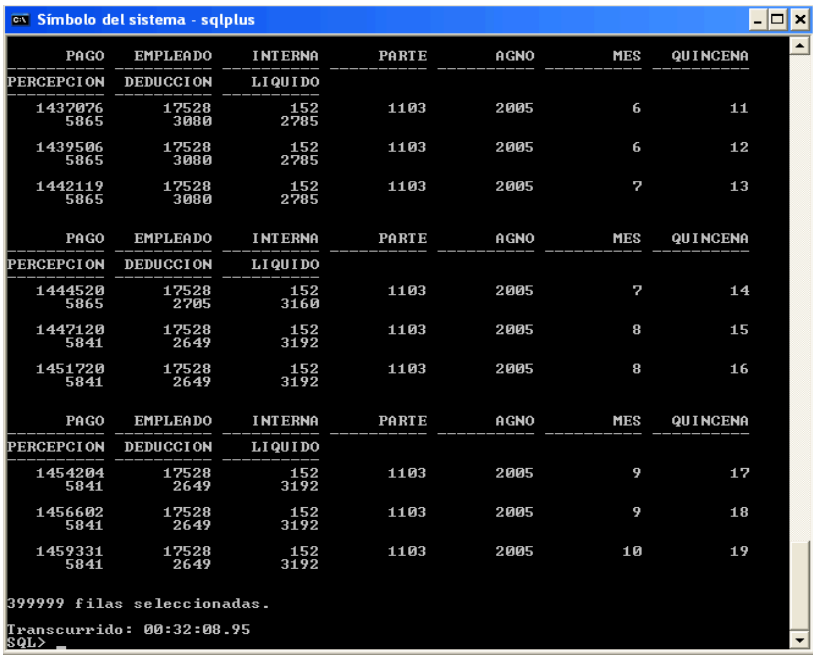



- Prueba D

En la primera impresión de pantalla se muestra el grado antes de efectuar la consulta.




En esta segunda impresión de pantalla se muestra el tiempo transcurrido de la consulta. El tiempo obtenido fue de 00:32:08.95 minutos.

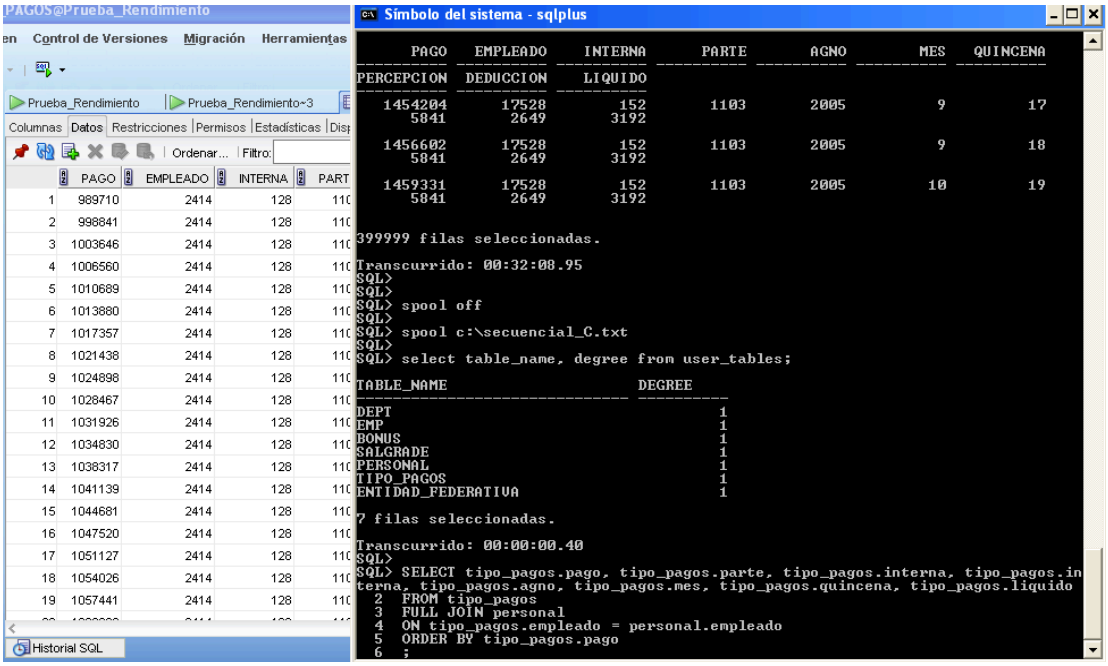



El plan de ejecución obtenido en la consulta fue el siguiente:

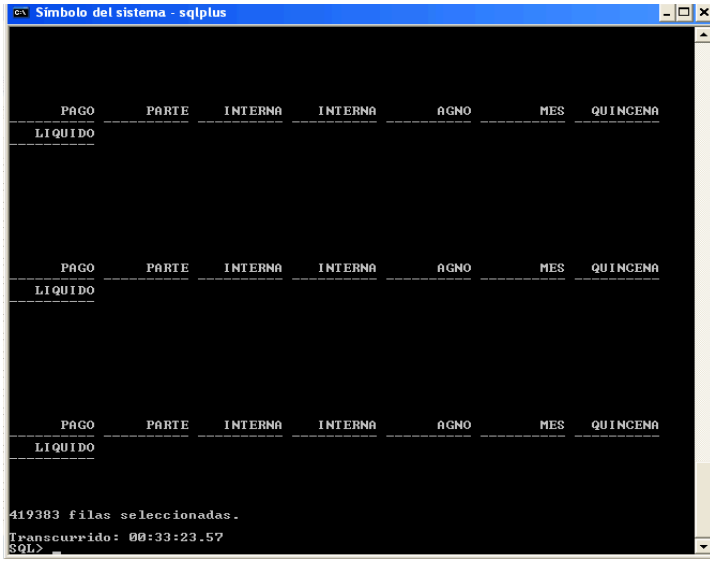
| Id | Operation | Name | Rows | Bytes | Cost (%CPU) | Time |
|---|---|---|---|---|---|---|
| 0 | Select Statement | | 365 K | 45 M | 689 (2) | 00:00:09 |
| 1 | Table Access Full | Tipo_pagos | 365 K | 45 M | 689 (2) | 00:00:09 |

- Prueba E

En la primera impresión de pantalla se muestra que ambas tablas se encuentran con el mismo grado antes de efectuar la consulta.



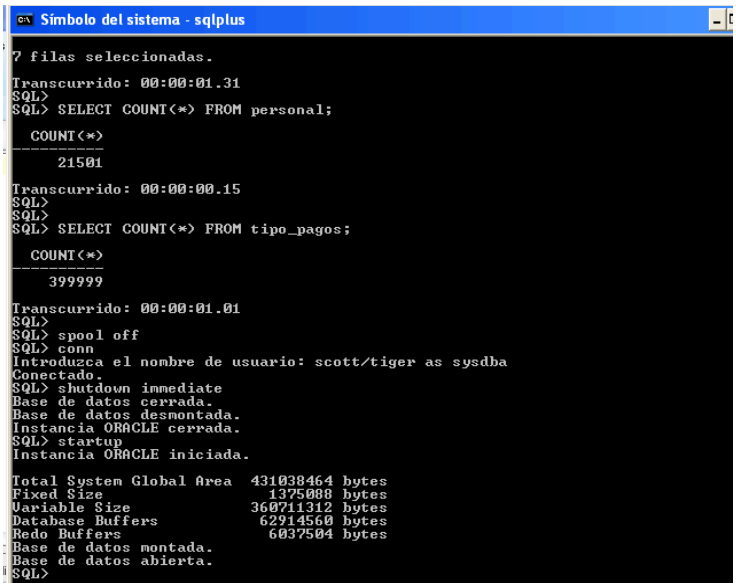

En esta segunda impresión de pantalla se muestra el tiempo transcurrido de la consulta. El tiempo obtenido fue de 00:33:23.57 minutos.




Pruebas con procesamiento en paralelo

Antes de realizar la prueba con procesamiento en paralelo se dió de baja la base de datos, y posteriormente se volvió a iniciar, con el

propósito de no interferir en los resultados, pues se efectuaron las mismas consultas en las mismas tablas, tal cuál se muestra a continuación:

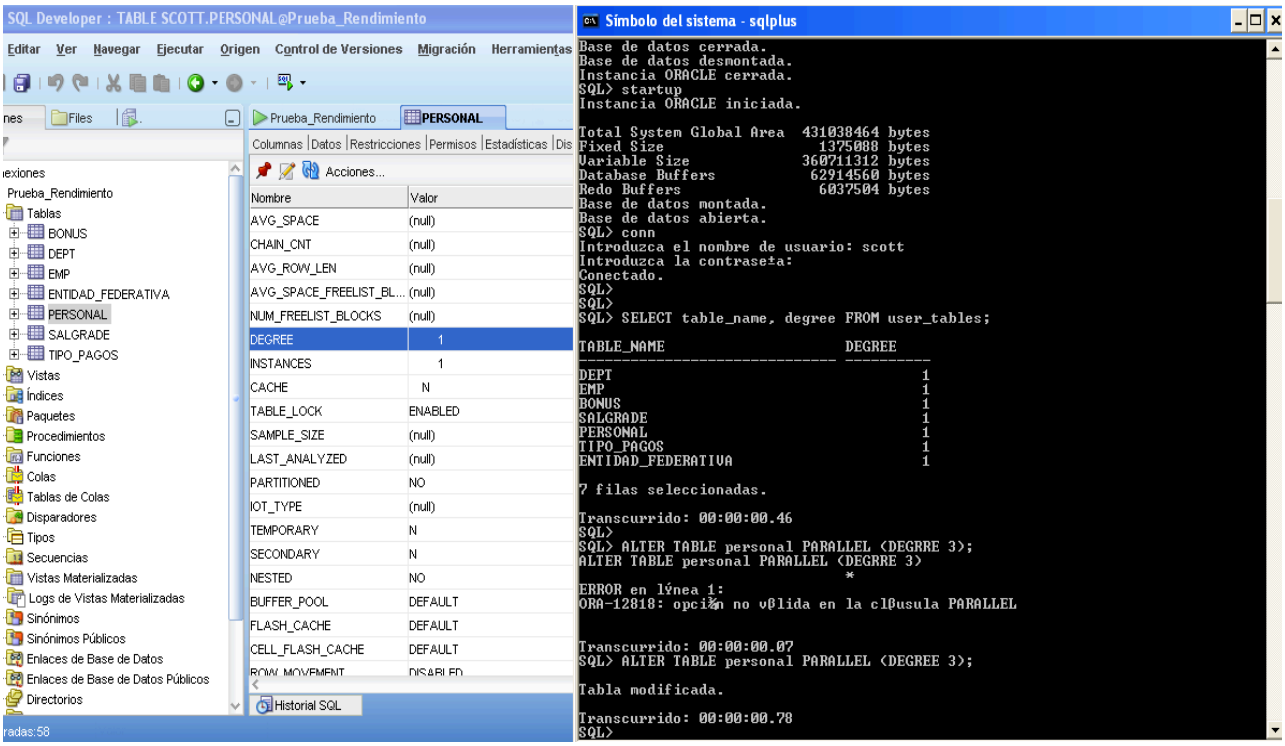



Después de haber levantado la base de datos y conectarnos a la misma, se debe alterar el grado de paralelismo de cada una de las tablas a utilizar, mediante las siguientes sentencias SQL:

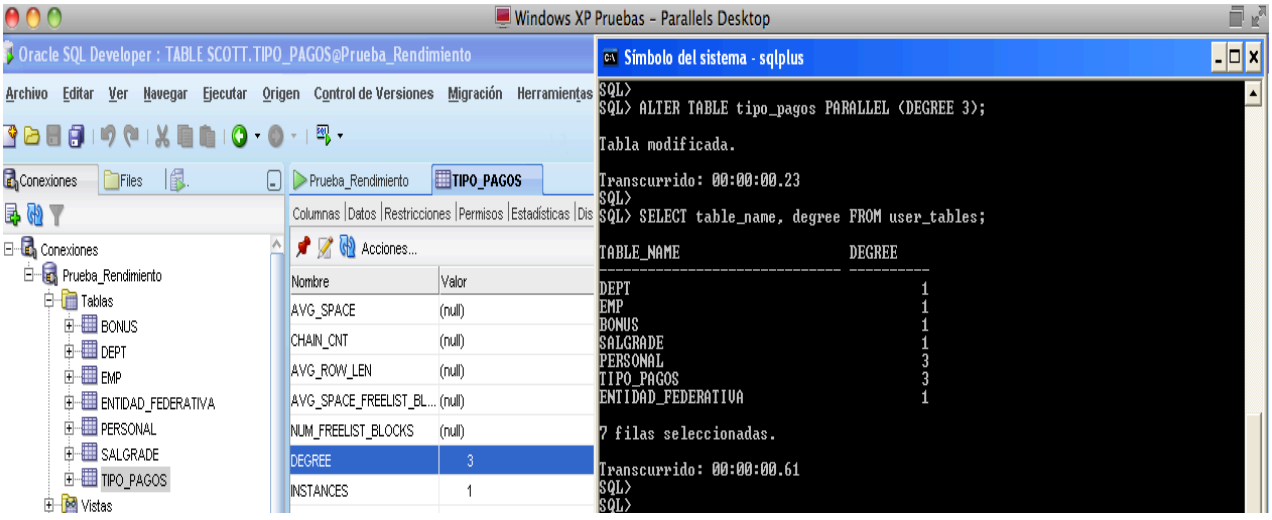
```
SQL> ALTER TABLE personal PARALLEL (DEGREE 3);
```

```
SQL> ALTER TABLE tipo_pagos PARALLEL (DEGREE 3);
```

En la siguiente impresión de pantalla se muestra el grado de cada una de las tablas antes de indicarle a la base de datos que se va a cambiar el grado de paralelismo de las tablas.




En la siguiente impresión de pantalla se muestran las tablas a las cuáles se les cambio el grado.

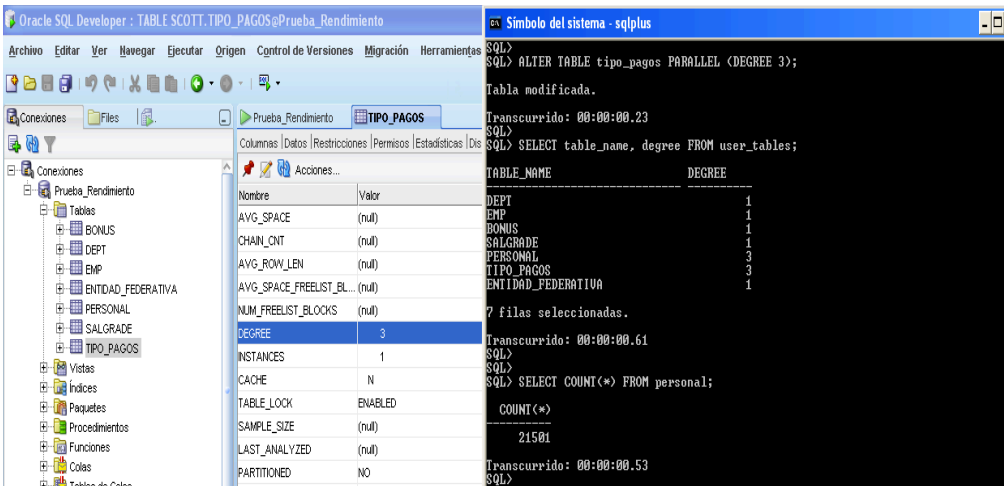



Una vez efectuados los cambios en las tablas "PERSONAL" y "TIPO_PAGOS", se procede a realizar las pruebas en este tipo de procesamiento con un grado de paralelismo de tres, esto significa que tres procesos esclavos estuvieron procesando de forma paralela

durante la ejecución de cada una de las pruebas. Los resultados fueron los siguientes:

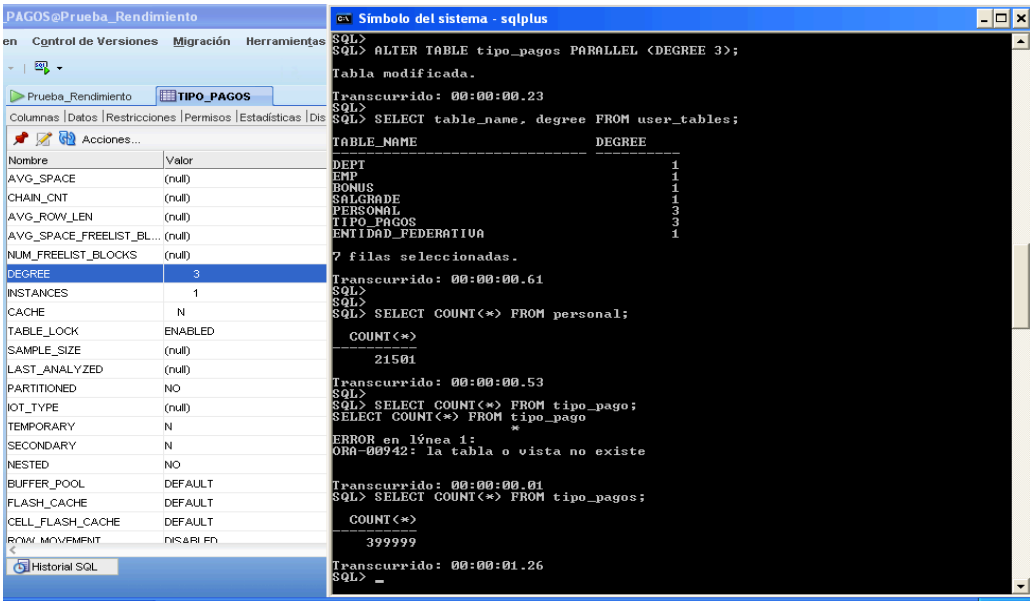
- Prueba A

En esta impresión de pantalla se muestra el grado con el que se efectuó la ejecución de la consulta y el tiempo transcurrido. El tiempo obtenido fue de 00:00:00.53 segundos.



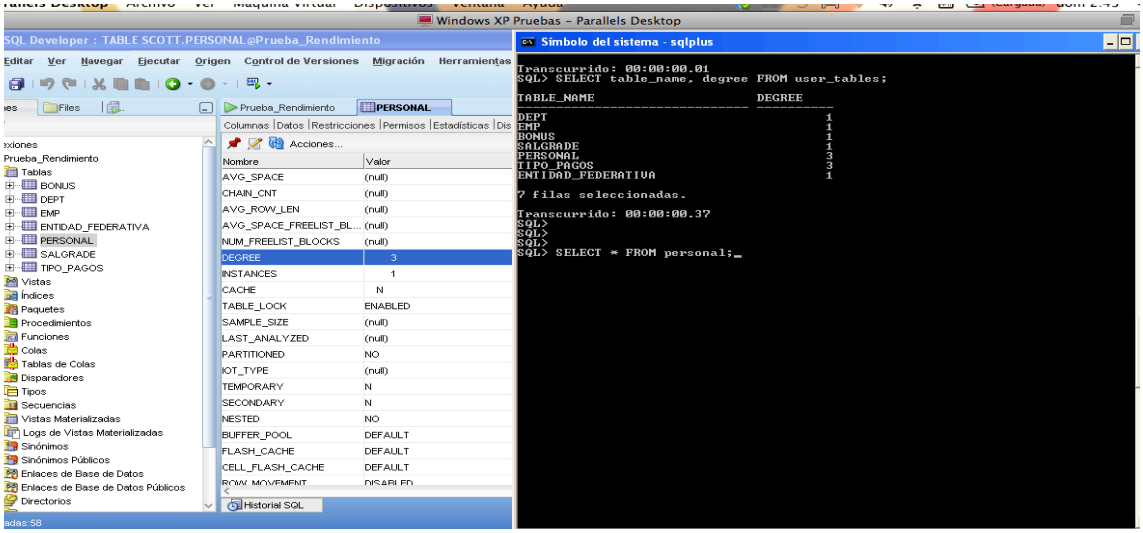

- Prueba B

En esta impresión de pantalla se muestra el grado con el que se efectuó la ejecución de la consulta y el tiempo transcurrido. El tiempo obtenido fue de 00:00:01.26 segundos.

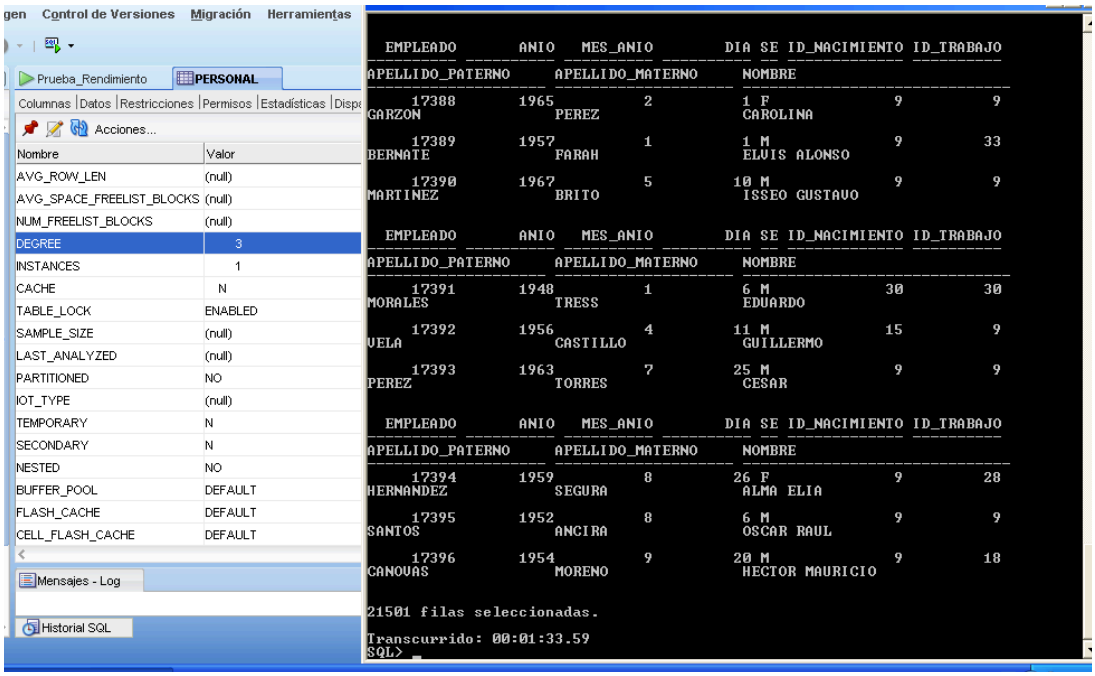



- Prueba C

En la primera impresión de pantalla se muestra el grado de la tablas antes de la ejecución.

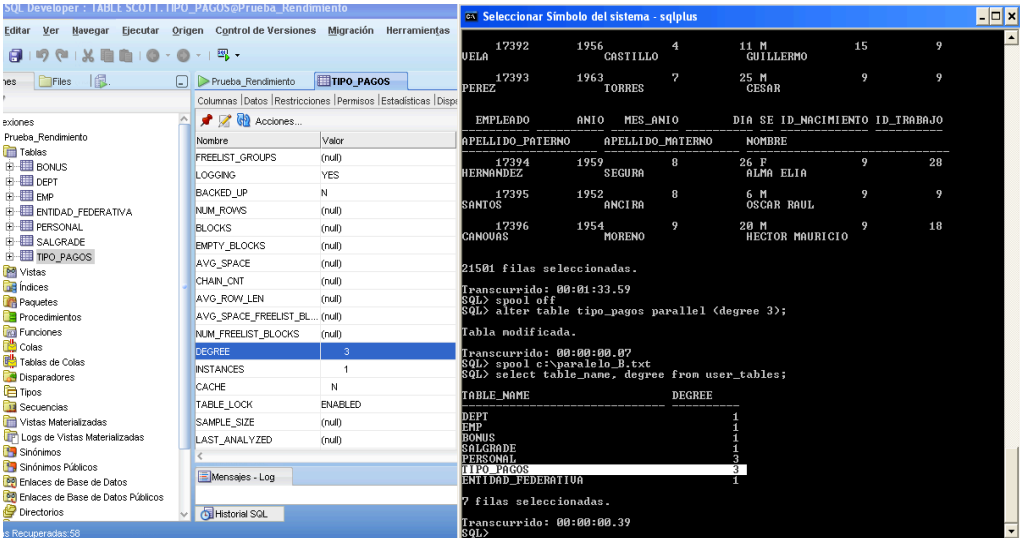



En esta segunda impresión de pantalla se muestra el tiempo transcurrido de la consulta. El tiempo obtenido fue de 00:01:33.59 minutos.

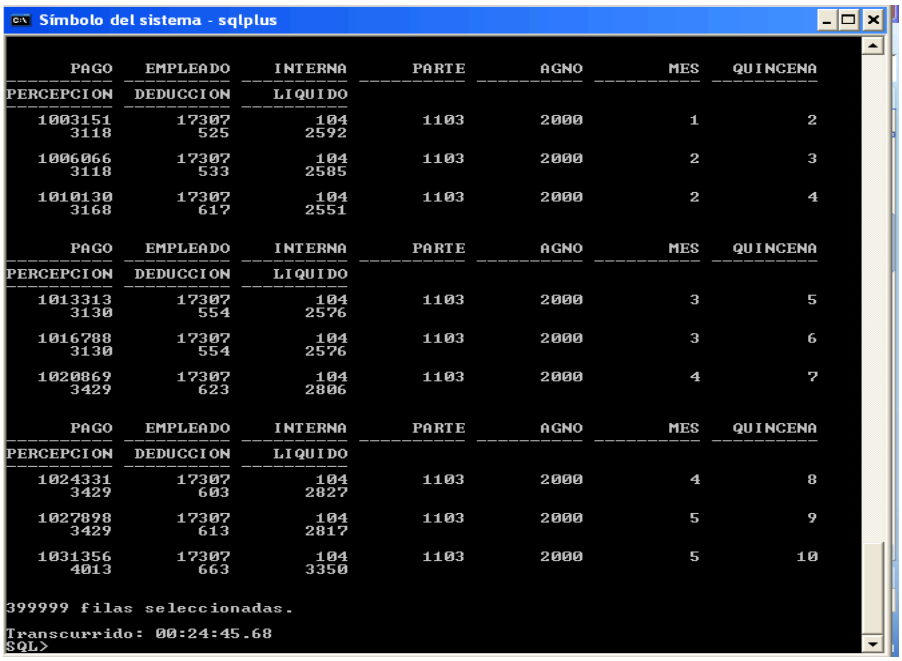



- Prueba D

En la siguiente impresión de pantalla se muestra el grado de la tabla antes de realizar la ejecución.




En esta impresión de pantalla se muestra el tiempo transcurrido de la consulta. El tiempo obtenido fue de 24:45.68 minutos

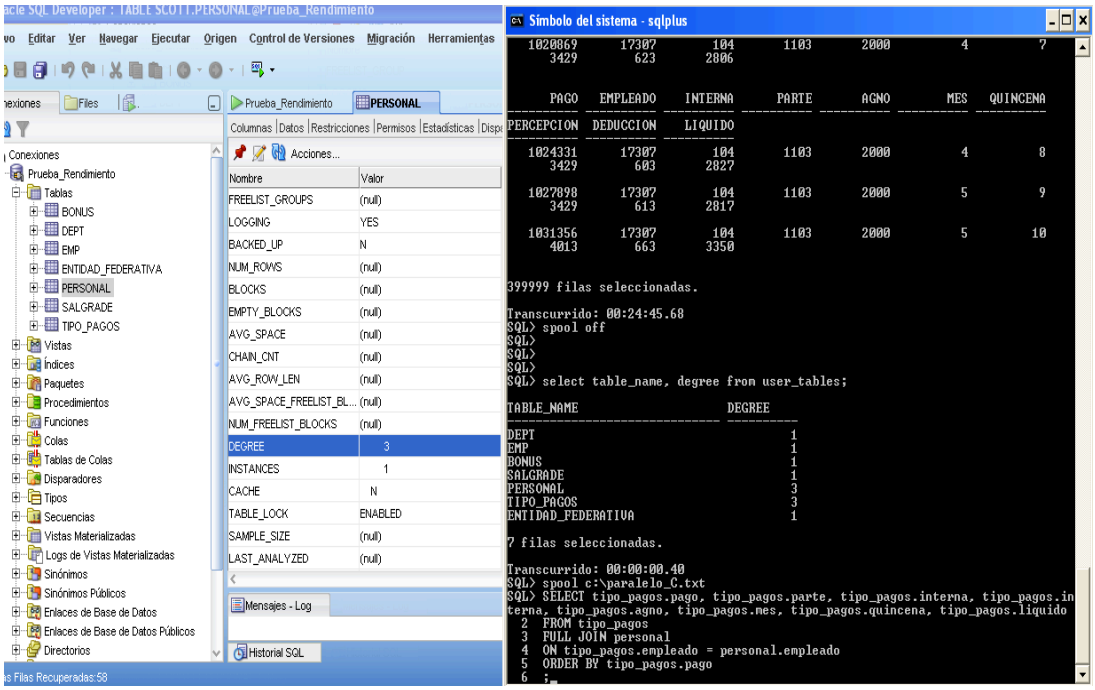



El plan de ejecución de la consulta obtenido fue el siguiente:

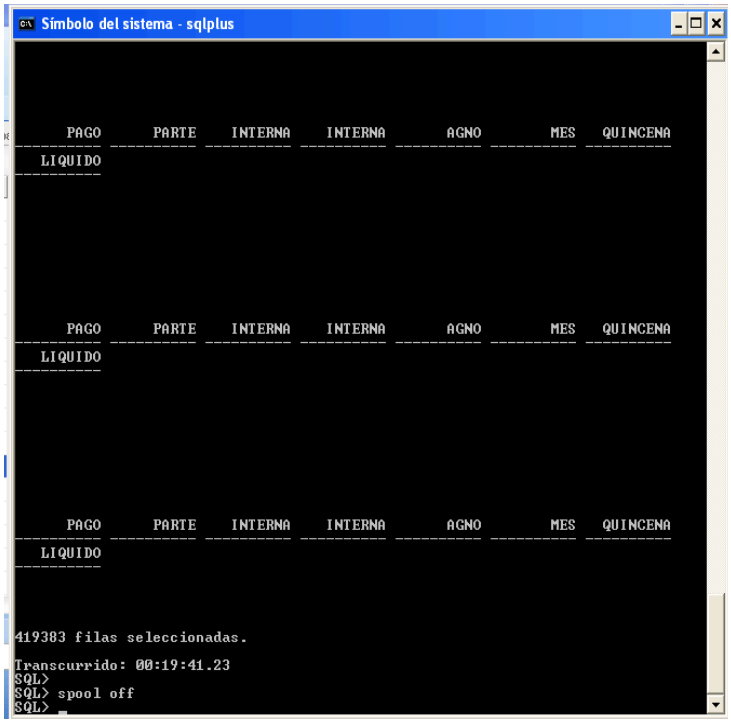
| Id | Operation | Name | Rows | Bytes | Cost (%CPU) | Time | TQ | IN-OUT | PQ Distrib |
|---|---|---|---|---|---|---|---|---|---|
| 0 | Select Statement | | 365K | 45 M | 255 (1) | 00:00:04 | | | |
| 1 | PX Coordinator | | | | | | | | |
| 2 | PX Send QC (Random) | :TQ10000 | 365K | 45 M | 255 (1) | 00:00:04 | Q1,00 | P->S | QC (RAND) |
| 3 | PX Block Iterator | | 365K | 45 M | 255 (1) | 00:00:04 | Q1,00 | PCWC | |
| 4 | Table Acces Full | Tipo_pagos | 365K | 45 M | 255 (1) | 00:00:04 | Q1,00 | PCWP | |

- Prueba E

En la siguiente impresión de pantalla se muestra el grado de las tablas antes de realizar la ejecución, así como la consulta a realizar.




En esta impresión de pantalla se muestra el tiempo transcurrido de la consulta. El tiempo obtenido fue de 19:41.23 minutos



## 5.6 Matriz de pruebas y análisis de resultados

En este punto se explican a detalle los resultados obtenidos en cada una de las pruebas realizadas. El parámetro a analizar es el tiempo de ejecución transcurrido en las pruebas efectuadas. Para poder realizar este análisis se utilizaron tablas y gráficas, las cuáles nos ayudan a interpretar de forma mas sencilla los resultados obtenidos de las pruebas.

En la tabla 5.1 se observa de manera ordenada los tiempos de ejecución que arrojó cada una de las consultas realizadas a la base de datos.

| Prueba | Tiempo Secuencial | Tiempo en paralelo | Mejor tiempo de procesamiento |
|---|---|---|---|
| **A** | 00:00:0.15 segundos | 00:00:0.53 segundos | Secuencial |
| **B** | 00:00:01.01 segundos | 00:00:01.26 segundos | Secuencial |
| **C** | 00:01:41.67 minutos | 00:01:33.59 minutos | Paralelo |
| **D** | 00:32:08.95 minutos | 00:24:45.68 minutos | Paralelo |
| **E** | 00:33:23.57 minutos | 00:19:41.23 minutos | Paralelo |

**Tabla 5.1** Resultados de los tiempos obtenidos en cada una de las pruebas

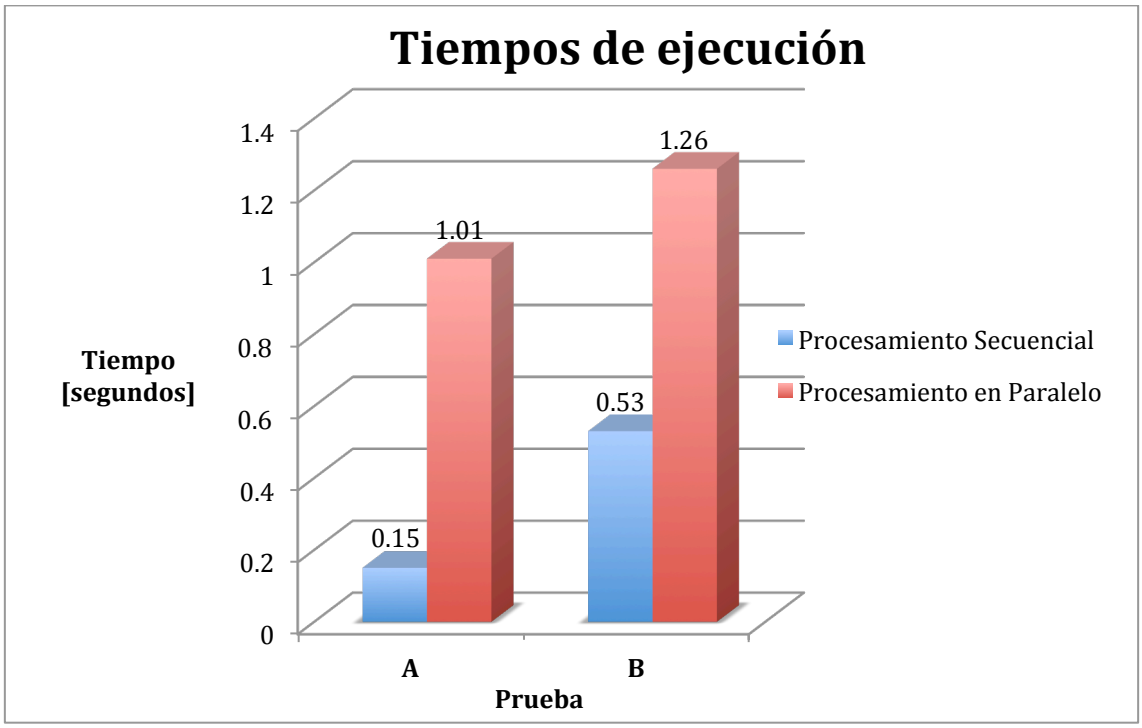
Como se puede observar la tabla consta de cuatro columnas y seis filas, en la primera columna se especifica el título de la prueba realizada, en la segunda columna se encuentra el tiempo obtenido al efectuar cada una de las pruebas con procesamiento secuencial (efectuado con un solo proceso esclavo), en la tercera columna se describen los tiempos obtenidos para cada una de las cinco pruebas efectuadas con procesamiento en paralelo (con tres procesos esclavos), en la cuarta y última columna se describe que tipo de procesamiento fue el que realizó la consulta en menor tiempo, siendo denominado como el mejor tiempo de procesamiento.

En la tabla 5.2 se muestran nuevamente los tiempos obtenidos para cada uno de los procesamientos y para cada prueba, sin embargo esta vez se presentan los tiempos en unidades de segundos, esto es debido a la importancia de tener un solo sistema de unidades.

| Prueba | Tiempo Secuencial [segundos] | Tiempo en paralelo [segundos ] | Diferencia de tiempo [segundos] |
|---|---|---|---|
| **A** | 0.15 | 0.53 | 0.38 |
| **B** | 1.01 | 1.26 | 0.25 |
| **C** | 101.67 | 93.59 | 8.08 |
| **D** | 1928.95 | 1485.68 | 443.27 |
| **E** | 2003.57 | 1181.23 | 822.34 |

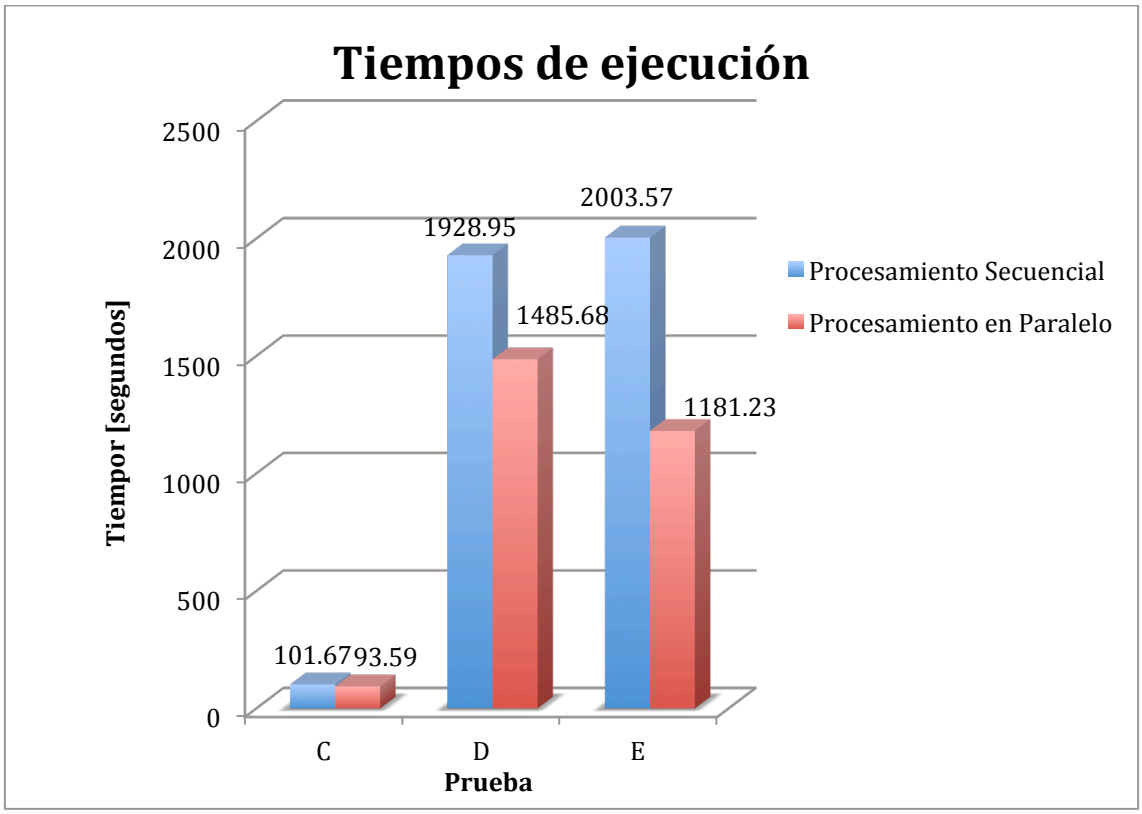
**Tabla 5.2** Tiempos de las pruebas obtenidos en unidades de segundos

En la gráfica siguiente se muestran los tiempos de ejecución de las pruebas A y B, en dónde es posible apreciar que los tiempo de ejecución con procesamiento secuencial son menores.




**Gráfica 5.1** Tiempos de ejecución de as pruebas A y B

Sin embargo, en la gráfica 5.2 es posible apreciar que los tiempos de ejecución de las pruebas C, D y E, el tiempo de ejecución con procesamiento en paralelo es el que presenta los menores tiempos.



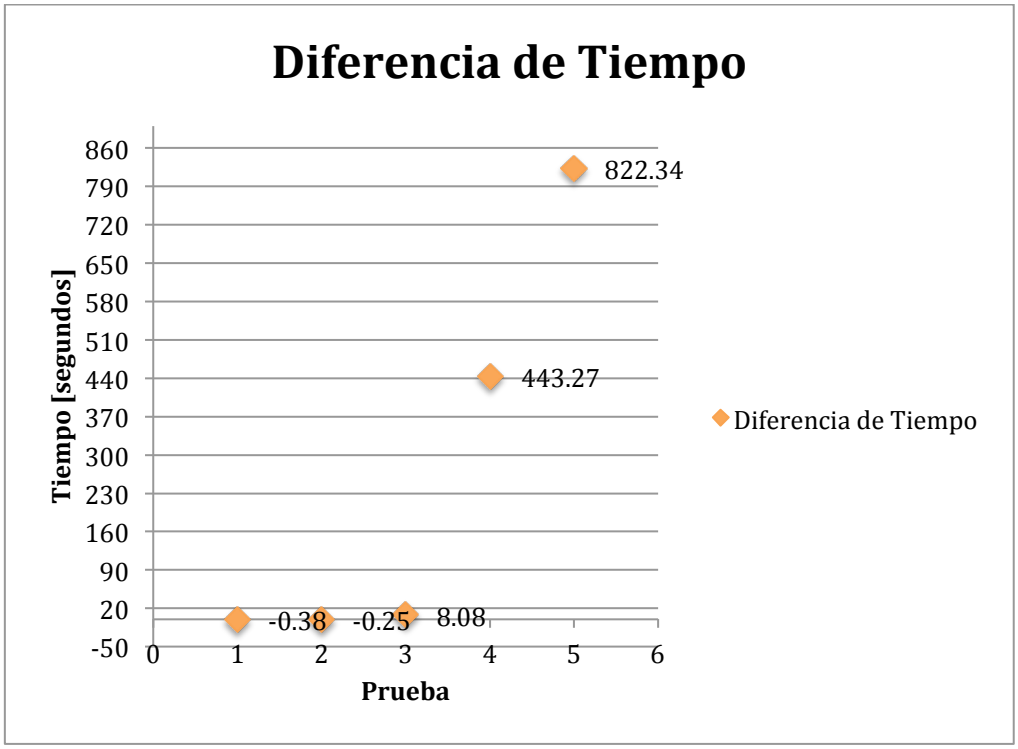

**Gráfica 5.2** Tiempos de ejecución de las pruebas C, D y E

Además de haber presentado de forma unificada las unidades de tiempo esta tabla incluye una nueva columna denominada diferencia de tiempo, en la cuál se describe la diferencia de tiempo obtenida al efectuar la siguiente operación entre las columnas dos y tres:

$$Diferencia\ de\ tiempo[segundos] = |Tiempo\ secuancial - Tiempo\ en\ paralelo|$$

En la gráfica 5.3 podemos apreciar la diferencia que existe entre los tiempos transcurridos de cada una de las pruebas realizadas. Los tiempos que aparecen en negativo significa que en esas pruebas (A y B), el tiempo de ejecución en secuencial fue menor que el tiempo de

ejecución en paralelo. Cabe resaltar que conforme la consulta efectuada involucraba menor cantidad de datos, como en las pruebas A y B, los resultados favorecieron al procesamiento en secuencial, sin embargo en las pruebas restantes (C, D y E) el panorama fue distinto, dado que los tiempos menores fueron obtenidos por el procesamiento en paralelo.




**Gráfica 5.3** Diferencia de tiempo en las consultas realizadas

La tabla 5.3 nos muestra dos columnas, la columna con el nombre de la prueba efectuada y la columna con el porcentaje de cambio que existe entre el tiempo obtenido con la ejecución del procesamiento en paralelo respecto al tiempo de ejecución con procesamiento secuencial. Estos valores se obtuvieron al realizar la siguiente operación entre los valores de las columnas dos y tres de la tabla anterior:

$$Porcentaje\ de\ Cambio = \frac{Tiempo\ secuencial - Tiempo\ en\ paralelo}{Tiempo\ secuencial} \times 100$$

| Prueba | Porcentaje de Cambio (respecto al tiempo secuencial) |
|---|---|
| **A** | -253% |
| **B** | -25% |
| **C** | 8% |
| **D** | 23% |
| **E** | 41% |

**Tabla 5.3** Porcentaje de cambio

Para la prueba A se obtuvo un porcentaje de cambio de -253%, esto significa que el tiempo transcurrido de la consulta con procesamiento en paralelo es 253% mayor que el tiempo obtenido con el procesamiento secuencial.

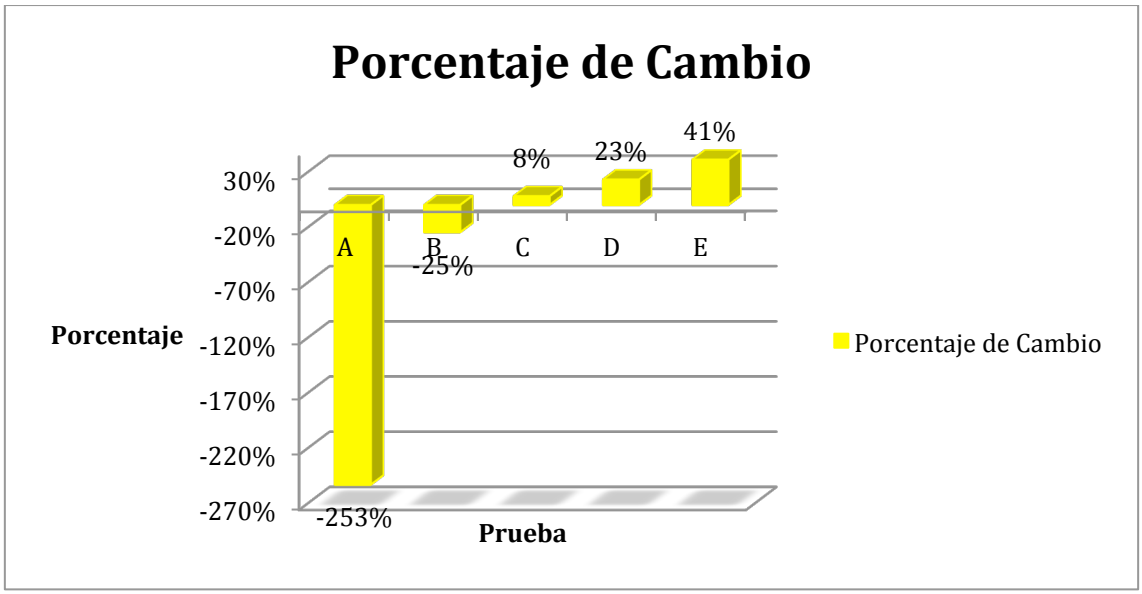
En la prueba B se observa un porcentaje de cambio de -25%, esto representa que el tiempo obtenido en la ejecución con procesamiento paralelo es 25% mayor respecto al tiempo con procesamiento secuencial.

En la prueba C se aprecia un porcentaje de cambio del 8%, lo que significa que el tiempo obtenido con el procesamiento en paralelo fue 8% menor con respecto al tiempo obtenido con el procesamiento secuencial.

En la prueba D el porcentaje de cambio fue del 23%, significando que el que procesamiento en paralelo obtuvo un porcentaje menor en la ejecución de las pruebas con respecto al tiempo obtenido con el procesamiento secuencial.

Y finalmente en la prueba E se obtuvo un porcentaje de cambio del 41%, lo que significa que el tiempo obtenido durante la ejecución en paralelo fue menor respecto al tiempo obtenido con la ejecución secuencial.

A continuación se muestra una gráfica en la que se puede apreciar el porcentaje de cambio obtenido en cada una de las pruebas.




**Gráfica 5.4** Porcentaje de cambio

Después de analizar las gráficas y tablas generadas con los resultados obtenidos en cada una de las pruebas, se puede puntualizar que cuando se maneja una menor cantidad de datos como en las pruebas A y B, la ejecución secuencial será la óptima a implementar. Por otra parte, cuando mayor cantidad de datos se encuentren involucrados la ejecución en paralelo será la que nos entregué un mayor rendimiento, pues como se demostró en las pruebas D, E y F con la implementación de la ejecución en paralelo el tiempo de respuesta obtenido fue menor con respecto a la ejecución secuencial.

# Conclusión

En el desarrollo de esta tesis se ha explicado, desde su inicio, la importancia de implementar bases de datos con procesamiento en paralelo. Se han descrito de forma concisa diversos conceptos que ayudan al entendimiento de este tipo de procesamiento en las bases de datos.

Si bien, el avance tecnológico respecto al hardware impacta de gran forma en el desempeño actual de las bases de datos, resulta fundamental concluir que no siempre se puede depender del desarrollo en cuanto a hardware.

A través de las pruebas realizadas en la parte práctica de esta tesis se pudo comprobar el efecto que causa la ejecución en paralelo en el rendimiento. Deduciendo así, que hay veces en que la ejecución en paralelo resulta conveniente cuando las cantidades de datos involucradas son mayores, sin embargo cuando pequeñas cantidades de datos están involucradas resulta ser una mejor opción la ejecución secuencial, tal y como se planteo en el aspecto teórico. Las pruebas nos corroboran lo descrito en la parte teórica en cuanto al rendimiento del procesamiento secuencial y el procesamiento en paralelo.

La implementación de bases de datos con procesamiento en paralelo resulta una gran alternativa a varios problemas, como lo es el procesamiento de información en un menor tiempo.

Las computadoras han sido creadas para poder resolver problemas de una forma más rápida de la que el ser humano podría hacerlo. Tomando en cuenta lo anterior, el poder procesar

información de manera más rápida y obtener un resultado en un menor tiempo es lo que se busca en la actualidad; en este punto, resulta de gran importancia la ejecución en paralelo en las bases de datos, pero, también es conveniente resaltar que la implementación de este tipo de ejecuciones no garantiza la mejora en el procesamiento de la información que se esté manejando, pues depende de la cantidad de datos que se manipulen; no es lo mismo implementarla en un comercio pequeño, en donde la información que se manipula es poca y en donde, realmente, no se verá reflejada la mejora al implementar con procesamiento en paralelo. En cambio, un banco que trabaja con millones de datos todos los días, podrá ver la mejora en las respuestas de la información que se solicite.

Como se mencionó en el capítulo tercero, el procesamiento en paralelo no incrementa únicamente el poder de procesamiento; de implementarlo de forma correcta, ofrece ventajas tales como un mayor rendimiento, una mayor tolerancia a los fallos que puedan suscitarle al sistema, etc.

Es importante que conocer los riesgos que implica la implantación de cualquier tipo de base de datos, así como sus ventajas y desventajas, y así, poder decidir qué tipo de base de datos resulta ser la mejor opción para nosotros o la que más conviene, según las necesidades que se tengan.

# Bibliografía

Collazo, J. (2001). *Diccionario Collazo de informática, computación y otras materias.* Ciudad de México: McGraw-Hill.

Connolly, T. M., & Begg, C. E. (2005). *Sistemas de bases de datos* (4ª edición ed.). Madrid: Pearson Addison Wesley.

Sonawalla, N. (1999). The Art of Tuning and Managing the Parallel Query Option. *Oracle Open World.*

Tanenbaum, A. S. (1988). *Sistemas Operativos: Diseño e implementación.* Ciudad de México.

Taniar, D., H.C Leung, C., Rahayu, W., & Goel, S. (2008). Hihg Performance Parallel Database Processing and Grid Databases. Canadá: Wiley.

Amdahl, G. (1967). Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities. *AFIPS Conference.*

Bauer, M. (1999). Oracle 8i Parallel Server Concepts and Administration.

Bernstein, A. J. (1966). Analysis of Programs for Parallel Processing. *IEEE Transactions on Computers*, *30.*

Dye, C. (1999). *Oracle Distributed Systems.* California: O'Reilly.

Española, R. A. (2012). *RAE*. From RAE: www.rae.es

Flynn, M. J. (1966). Very High-Speed Computer Systems. *Proceedings of the IEEE*, *54.*

Farina, M. V. (1971). *Diagramas de Flujo.* México: Prentice-Hall.

Gustafson, J. L. (1988). Reevaluating Amdahl's Law. *Communications of the ACM*, *31.*

Korth, H. F., & Silberschatz, A. (1993). *Fundamentos de bases de datos* (2ª ed.). Madrid: McGraw-Hill.

Mahapatra, T., & Mishra, S. (2000). *Oracle Parallel Processing.* California: O'Reilly.

Moore, G. E. (1965). Cramming more components onto integrated circuits.

Pratt, T., & Zelkowitz, M. (1998). *Lenguajes de Programación* (3ª ed.). Ciudad de México: Prentice Hall.

Oracle. (2008). *Oracle*.

Oracle. (2012). *Oracle*. From Oracle Web Site: www.oracle.com

Rajasekaran, S., & Reif, J. (2008). *Handbook of Parallel Computing.* New York: Chapman & Hall/CRC.