ALN - HPC

In. Co.
Facultad de Ingeniería
Universidad de la República

Temario:

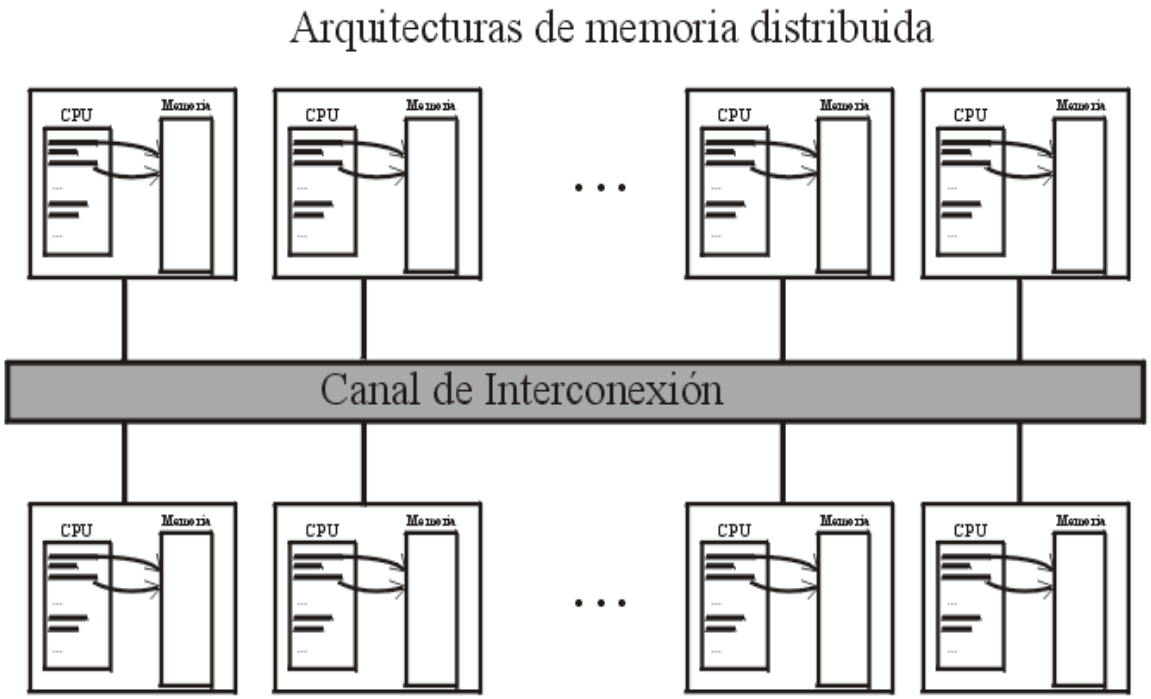
- **Arquitecturas paralelas**
- **Modelo de programación paralela**
- **Técnicas de programación paralela**
- **Medidas de performance**
- **Scheduling y balance de cargas**
- **Herramientas**

Arquitecturas paralelas

- Modelo de la computación estándar : Arquitectura de Von Neumann.
  - CPU única
    - Ejecuta un programa (único).
    - Accede a memoria.
  - Memoria única.
    - Operaciones read/write.
    - Dispositivos.
- Modelo robusto, independiza al programador de la arquitectura subyacente.
- Permitió el desarrollo de las Técnicas de Programación (estándar).

Arquitecturas paralelas

- Extendiendo el modelo a la computación paralela
  - Para lograr abstraer el hardware subyacente.
- Varias alternativas
  - Multicomputador
    - Varios nodos (CPUs de Von Neumann).
    - Un mecanismo de interconexión entre los nodos.



Arquitecturas paralelas

- Extendiendo el modelo a la computación paralela
- Otras alternativas
  - Computador masivamente paralelo.
    - Muchísimos nodos (sencillas CPUs estilo Von Neumann).
    - Topología específica para interconexión entre los nodos.
  - Multiprocesador de memoria compartida.
    - Nodos de Von Neumann.
    - Memoria única.
  - Cluster.
    - Multiprocesador que utiliza una red LAN como mecanismo de interconexión entre sus nodos.

Arquitecturas paralelas

CATEGORIZACION DE FLYNN

		instrucciones	
		SI	**MI**
datos	**SD**	SISD	(MISD)
	MD	SIMD	MIMD

S=single, M=multi, I=Instrucción, D=Datos

- SISD - Modelo convencional de Von Neumann.
- SIMD - Paralelismo de datos, Computación Vectorial.
- MISD - Arrays sistólicos.
- MIMD – Modelo general, varias implementaciones.

Arquitecturas paralelas

- SISD = Máquina de Von Neumann
  - Un procesador capaz de realizar acciones secuencialmente, controladas por un programa el cual se encuentra almacenado en una memoria conectada al procesador.
  - Este hardware esta diseñado para dar soporte al procesamiento secuencial clásico, basado en el intercambio de datos entre memoria y registros del procesador, y la realización de operaciones aritméticas en ellos.

Arquitecturas paralelas
¿ Por qué el modelo SISD no fue suficiente ?

- Los problemas crecieron, o surgió la necesidad de resolver nuevos problemas de grandes dimensiones (manejando enormes volúmenes de datos, mejorando precisión de las grillas, etc.).

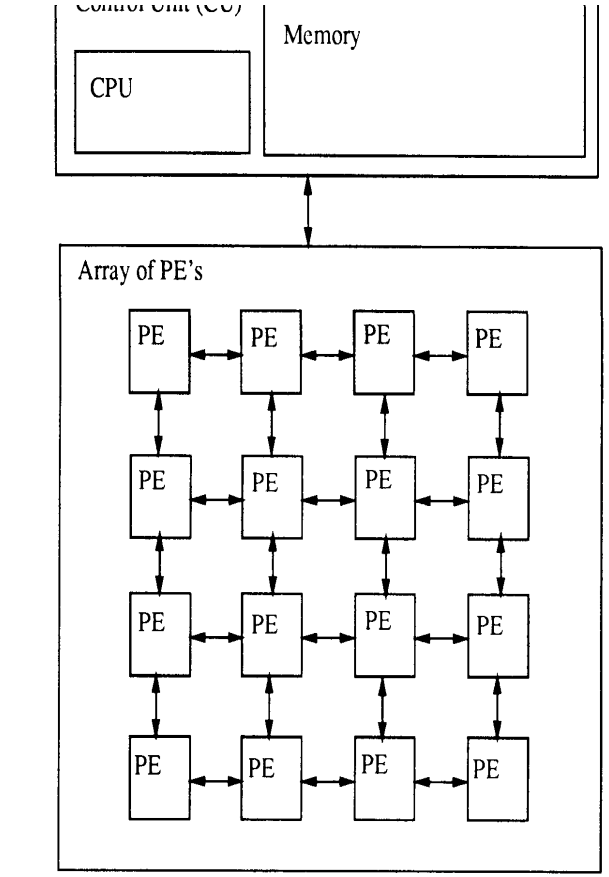
- Si bien las maquinas SISD mejoraron su performance
  - Compiladores optimizadores de código.
  - Procesadores acelerando ciclos de relojes, etc.

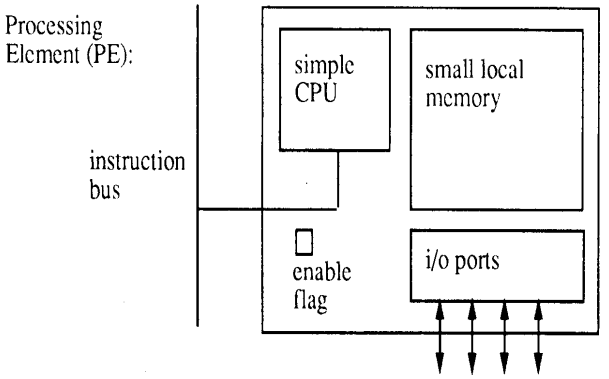
- Aún no fue suficiente, y se prevé que el ritmo de mejoramiento se desacelere (debido a limitaciones físicas)

En este contexto se desarrollaron las maquinas paralelas

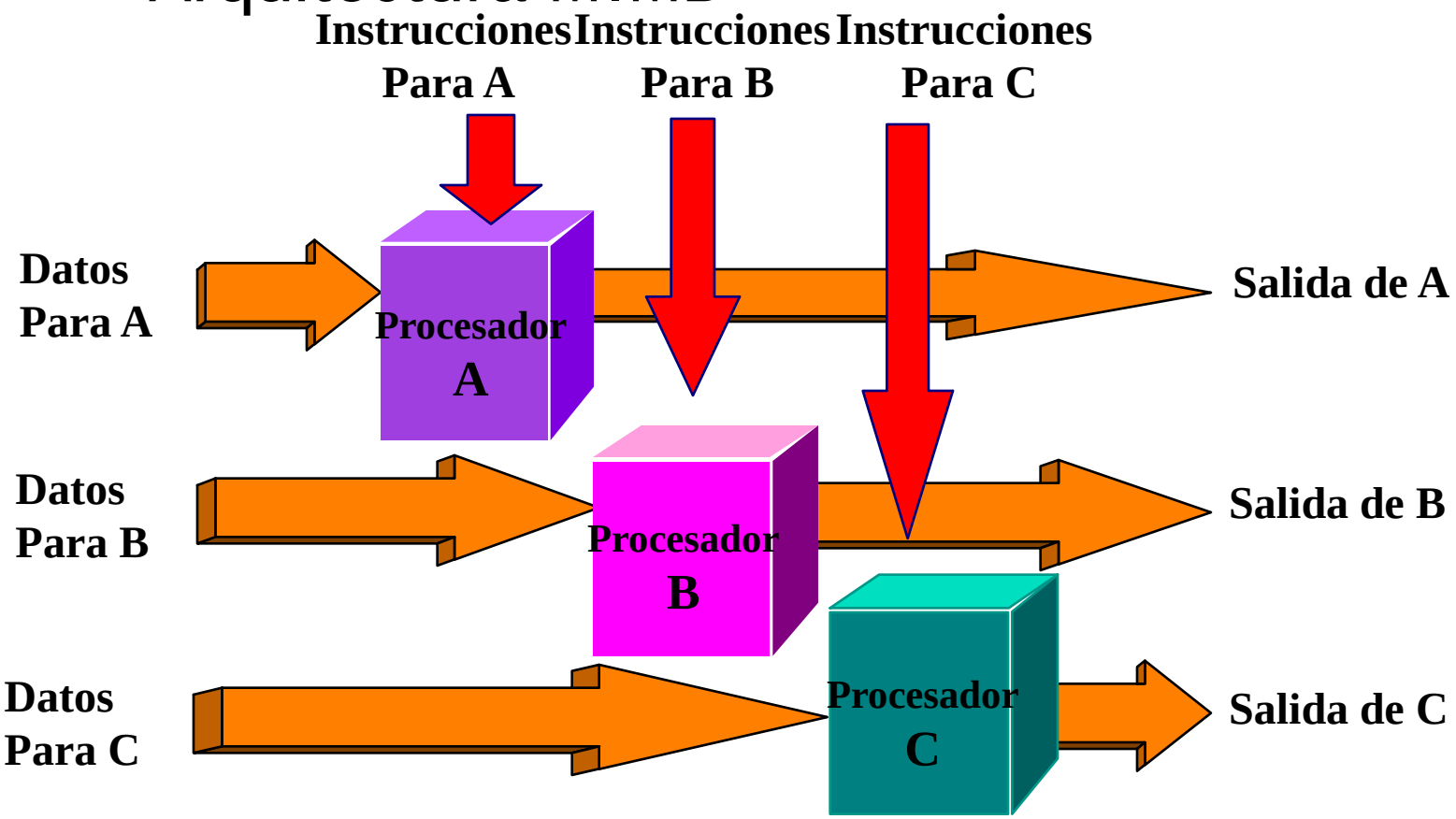
Arquitecturas paralelas

- SIMD
  - Único programa controla los procesadores.
  - Útil en aplicaciones uniformes.
  - Aplicabilidad limitada por las comunicaciones fijas entre procesadores : procesamiento de imágenes, diferencias finitas, etc.





Arquitectura MIMD



A diferencia de SISD y MISD las computadoras MIMD trabajan asincrónicamente.

- MIMD de memoria compartida (fuertemente acopladas)
- MIMD de memoria distribuída (poco acopladas)

Arquitectura MIMD con memoria compartida

COMO SE COMPARTEN LOS DATOS ?

- UMA = Uniform Memory Access
  - Acceso uniforme (todos los procesadores acceden a la memoria en el mismo tiempo).
  - Multiprocesadores simétricos (SMP).
  - Pocos procesadores (32, 64, 128, por problemas de ancho de banda del canal de acceso).
- NUMA = Non-Uniform Memory Access.
  - Colección de memorias separadas que forman un espacio de memoria direccionable.
  - Algunos accesos a memoria son más rápidos que otros, como consecuencia de la disposición física de las memorias (distribuidas físicamente).
  - Multiprocesadores masivamente paralelos (MPP).

Arquitectura MIMD con memoria distribuida

- No existe el concepto de memoria global.
- La comunicación y sincronización se realiza a través del pasaje de mensajes explícitos (mayor costo que en memoria compartida).
- Arquitectura escalable para aplicaciones apropiadas para esta topología (decenas de miles de procesadores).

Conectividad entre procesadores

- FACTORES QUE DETERMINAN LA EFICIENCIA
  - Ancho de banda
    - Número de bits capaces de transmitirse por unidad de tiempo.
  - Latencia de la red
    - Tiempo que toma a un mensaje transmitirse a través de la red.
  - Latencia de las comunicaciones
    - Incluye tiempos de trabajo del software y retardo de la interfaz.
  - Latencia del mensaje
    - Tiempo que toma enviar un mensaje de longitud cero.
  - Valencia de un nodo
    - Número de canales convergentes a un nodo

Conectividad entre procesadores

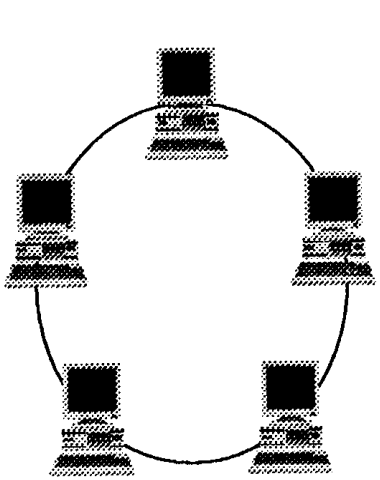
- FACTORES QUE DETERMINAN LA EFICIENCIA
  - Diámetro de la red
    - Número mínimo de saltos entre los nodos más alejados.
    - Permite estimar el peor caso de retardo de un mensaje.
  - Ancho de bisección
    - Número mínimo de enlaces que, en caso de no existir, separarían la red en dos componentes conexas.
  - Largo máximo de un tramo de comunicación.
  - Costo
    - Cantidad de enlaces de comunicación.

Conectividad entre procesadores

- FACTORES QUE DETERMINAN LA EFICIENCIA
  - CONFIGURACIÓN ÓPTIMA
    - Ancho de banda grande.
    - Latencias (de red, comunicación y mensaje) bajas.
    - Diámetro de la red reducido.
    - Ancho de bisección grande.
    - Valencia constante e independiente del tamaño de la red.
    - Largo máximo de tramo reducido, constante e independiente del tamaño de la red.
    - Costo (monetario) mínimo.

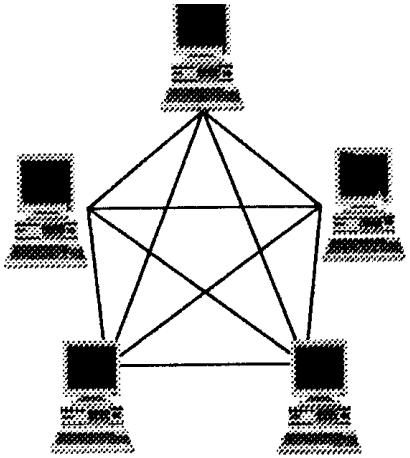
Conectividad entre procesadores

Modelos de conectividad entre procesadores



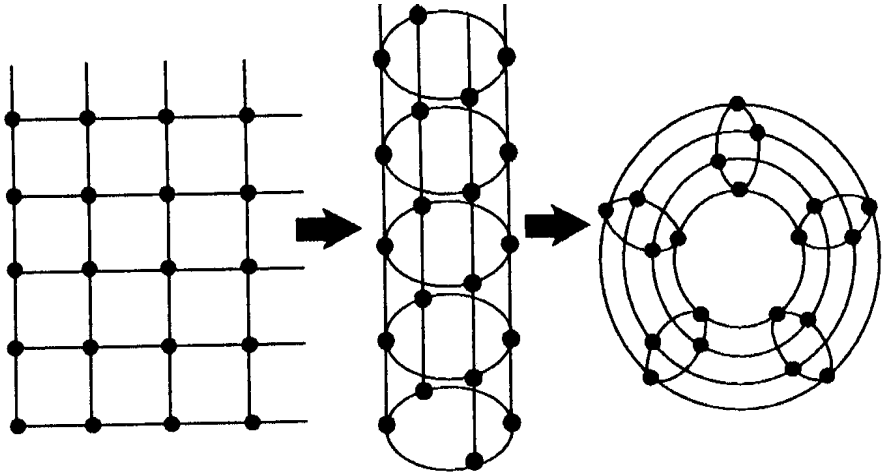
RING

Anillo



STAR

Estrella

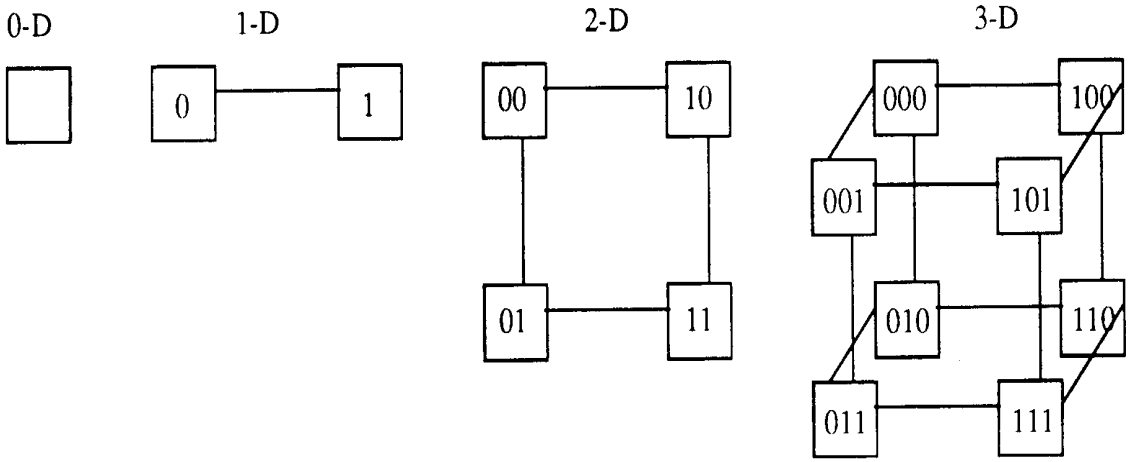


2D CYLINDER TORUS

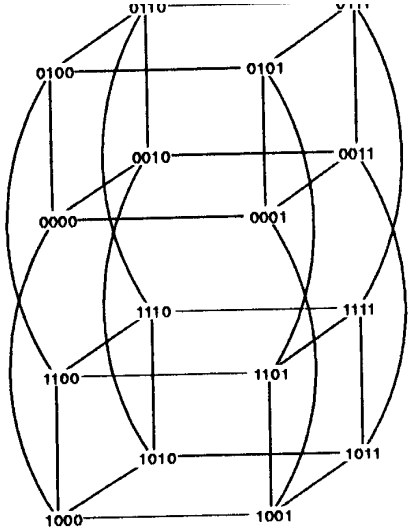
Topologías geométricas (Mallas)

Conectividad entre procesadores

Modelos de conectividad entre procesadores



Topologías “dimensionales” (distancia constante)



Hipercubo

MODELO DE PROGRAMACIÓN PARALELA

Modelo de Programación Paralela

- Programación en máquina de Von Neumann
  - Secuencia de operaciones (aritméticas, read/write de memoria, avance de PC).
  - Abstracciones de datos e instrucciones.
  - Técnicas de programación modular.

- Programación en máquina paralela
  - Incluye complicaciones adicionales.
    - Multiejecución simultánea.
    - Comunicaciones y sincronización.
  - La modularidad pasa a ser fundamental, para manejar la (potencialmente) enorme cantidad de procesos en ejecución simultánea.

Modelo Conceptual de Paralelismo

- El paradigma de diseño y programación difiere del utilizado para diseñar y programar aplicaciones secuenciales.
  - Una buena estrategia de división del problema puede determinar la eficiencia de un algoritmo paralelo.
  - Es importante considerar la disponibilidad de hardware.

- Respecto a los mecanismos de comunicación entre procesos, existen dos paradigmas de computacion paralela
  - MEMORIA COMPARTIDA
  - PASAJE DE MENSAJES

  (Existen también MODELOS HÍBRIDOS, que combinan ambas técnicas)

Niveles de Aplicación del Procesamiento Paralelo

- El paralelismo puede aplicarse :
  - A nivel intrainstrucción (hardware, pipelines).
  - A nivel interinstrucción (SO, optimización de código, compilador).
  - A nivel de procedimientos o tareas (algorítmico – tareas).
  - A nivel de programas o trabajos (algorítmico – funcional).

- Los niveles 3 y 4 de aplicación de las técnicas de paralelismo (enfocado al diseño y programación de algoritmos paralelos).

Modelo Conceptual de Paralelismo

- GRAFOS DIRIGIDOS ACICLICOS (ADGs)

  nodos = tareas o procesos (partes de código que ejecutan secuencialmente)

  aristas = dependencias (algunas tareas preceden a otras)

El problema a resolver se divide en tareas a ejecutar cooperativamente en múltiples procesadores.

Se debe :

Definir tareas que pueden ejecutar concurrentemente.

Lanzar la ejecución y detener la ejecución de tareas.

Implementar los mecanismos de comunicación y sincronización.

Modelo Conceptual de Paralelismo

- No siempre es una buena decisión partir de un algoritmo secuencial (“paralelizar” una aplicación). En ocasiones será necesario diseñar un nuevo algoritmo muy diferente.

- Resumiendo las etapas de diseño de aplicaciones paralelas :
  - Identificar el trabajo que puede hacerse en paralelo.
  - Partir el trabajo y los datos entre los procesadores.
  - Resolver los accesos a los datos, comunicación entre procesos y sincronizaciones.

- DESCOMPOSICIÓN – ASIGNACIÓN – ORQUESTACIÓN – MAPEO

Modelo Conceptual de Paralelismo

- DESCOMPOSICIÓN
  - Identifica concurrencia y decide el nivel y la forma en la cual se explotará.
    - Cantidad de tareas.
    - Tareas estáticas o dinámicas.
    - Criterios de división y utilización de recursos.

- ASIGNACIÓN
  - Asigna datos a procesos
  - Criterios de asignación
    - Estáticos o dinámicos.
    - Balance de cargas.
    - Reducción de costos de comunicación y sincronización.

Modelo Conceptual de Paralelismo

- ORQUESTACIÓN
  - Decisión sobre parámetros de arquitectura, modelo de programación, lenguaje o biblioteca a utilizar.
    - Estructuras de datos, localidad de referencias.
    - Optimizar comunicación y sincronización.

- MAPEO (SCHEDULING)
  - Asignación de procesos a procesadores.
  - Criterios de asignación
    - Performance.
    - Utilización.
    - Reducción de costos de comunicación y sincronización.

Modelo Conceptual de Paralelismo

- Los mecanismos para definir, controlar y sincronizar tareas deben formar parte del lenguaje a utilizar o ser intercalados por el compilador.
- Estos mecanismos deben permitir especificar
  - El control de flujo de ejecución de tareas.
  - El particionamiento de los datos.
- Algunos ejemplos
  - Definición de tareas, lanzado y detención : fork & join (C), parbegin-parend (Pascal concurrente), task (Ada), spawn (PVM).
  - Coordinación y sincronización : Semáforos, monitores, barreras (memoria compartida), mensajes asincrónicos (C, PVM, MPI) y mensajes sincrónicos (rendezvouz de Ada) (memoria distribuida).
  - El particionamiento de los datos es una tarea usualmente asignada al diseñador del algoritmo paralelo.
- El modelo se complica por características particulares de la computación paralela – distribuida.

Problemas con la computación paralela - distribuida

- CONFIABILIDAD
  - Varios componentes del sistema pueden fallar
    - nodos, interfases, tarjetas, caches, bridges, routers, repeaters, gateways, medios físicos.
  - Problemas de utilizar equipamiento no dedicado
    - problemas de uso y tráfico (propio y ajeno), etc.
    - no repetibilidad de condiciones de ejecución.

- NO DETERMINISMO EN LA EJECUCIÓN

  los programas tienen muchos estados posibles: adiós a "diagramas de flujo" para especificar secuencias de control

Problemas con la computación paralela - distribuida

- SEGURIDAD
  - Acceso a equipos remotos.
  - Datos distribuidos.
- DIFICULTAD DE ESTIMAR LA PERFORMANCE FINAL DE UNA APLICACIÓN
  - Como consecuencia del no determinismo en la ejecución.
  - Criterios estadísticos.
- DIFICULTAD DE TESTEAR / DEBUGGEAR PROGRAMAS
  - difícil reproducir escenarios (múltiples estados, asincronismo).
  - datos y procesos no centralizados (un debugger en cada nodo).

Problemas con la computación paralela - distribuida

- INCOMPATIBILIDADES ENTRE PRODUCTOS Y PLATAFORMAS (para sistemas heterogéneos)
  - Sistemas Operativos: AIX, Solaris, Linux, W2K, NT, OSF, VMS, etc.
  - Protocolos de comunicaciones: TCP/IP, DEC-NET, IPX, NetBEUI, etc....
  - Herramientas de software: PVM, MPI, C, HPF, etc....

TÉCNICAS DE PROGRAMACIÓN PARALELA

Técnicas de programación paralela

Introducción

Técnica de descomposición de dominio

Técnica de descomposición funcional

Técnicas Híbridas

Pipeline

Introducción

- Analizaremos las técnicas de DESCOMPOSICIÓN o PARTICIONAMIENTO, que permiten dividir un problema en subproblemas a resolver en paralelo.

- El objetivo primario de la descomposición será dividir en forma equitativa tanto los cálculos asociados con el problema como los datos sobre los cuales opera el algoritmo.

Introducción

- ¿ Cómo lograr el objetivo de la descomposición ?
  - Definir al menos un orden de magnitud más de tareas que de procesadores disponibles (utilización de recursos)
  - Evitar cálculos y almacenamientos redundantes.
  - Generar tareas de tamaño comparable.
  - Generar tareas escalables con el tamaño del problema.
  - Considerar varias alternativas de descomposición, en caso de ser posible.
- Según se enfoque principalmente en la descomposición de datos o de tareas, resultará una técnica diferente de programación paralela.

Descomposición de Dominio

- Se concentra en el particionamiento de los datos del problema (data parallel).
  - Se trata de dividir los datos en piezas pequeñas, de (aproximadamente) el mismo tamaño.
- Luego se dividen los cálculos a realizar
  - Asociando a cada operación con los datos sobre los cuales opera.
- Los datos a dividir pueden ser
  - La entrada del programa.
  - La salida calculada por el programa.
  - Datos intermedios calculados por el programa.

Descomposición de Dominio

- Si bien no existe una regla general para determinar como realizar la división de datos, existen algunas sugerencias obvias dadas por:
  - La estructura o “geometría” del problema.
  - La idea de concentrarse primero en las estructuras de datos más grandes o las accedidas con más frecuencia.

Descomposición de Dominio

- La técnica de descomposición de dominio se asocia comúnmente con la estrategia de “Divide & Conquer” y los modelos SIMD y SPMD de programas paralelos.

- Ejemplo 2 : Resolución de ecuaciones
  - Discretización de la solución.
  - División de dominios de cálculo.
  - Mismo programa en cada dominio.
  - Comunicación necesaria para cálculos en los bordes.

Descomposición de Dominio

Aplicable en modelos de programa SIMD y SPMD.

SIMD: Single Instruction Multi Data

SPMD: Single Program Multi Data

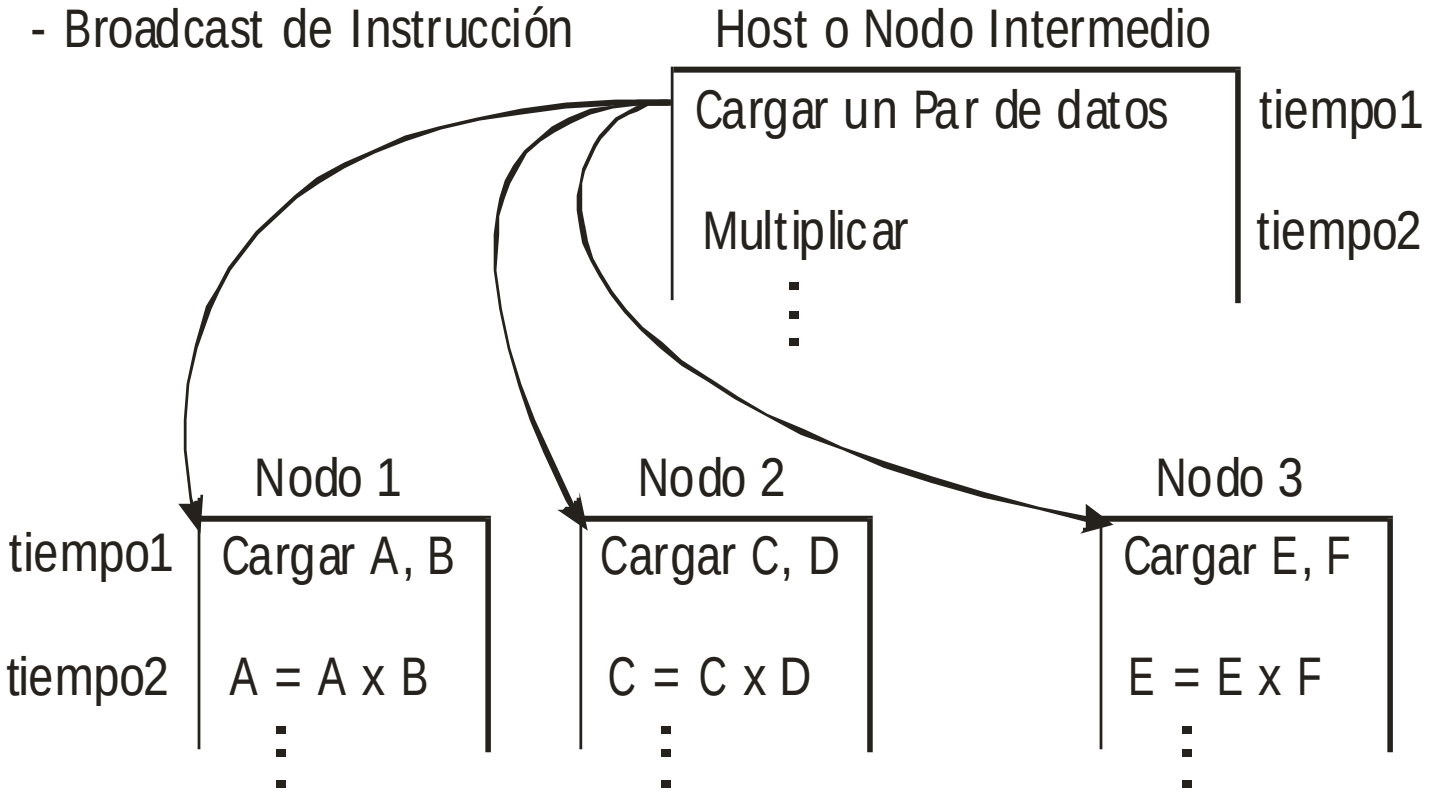
SIMD sobre arquitecturas de memoria compartida.

SPMD sobre arquitecturas de memoria distribuida.

Descomposición de Dominio

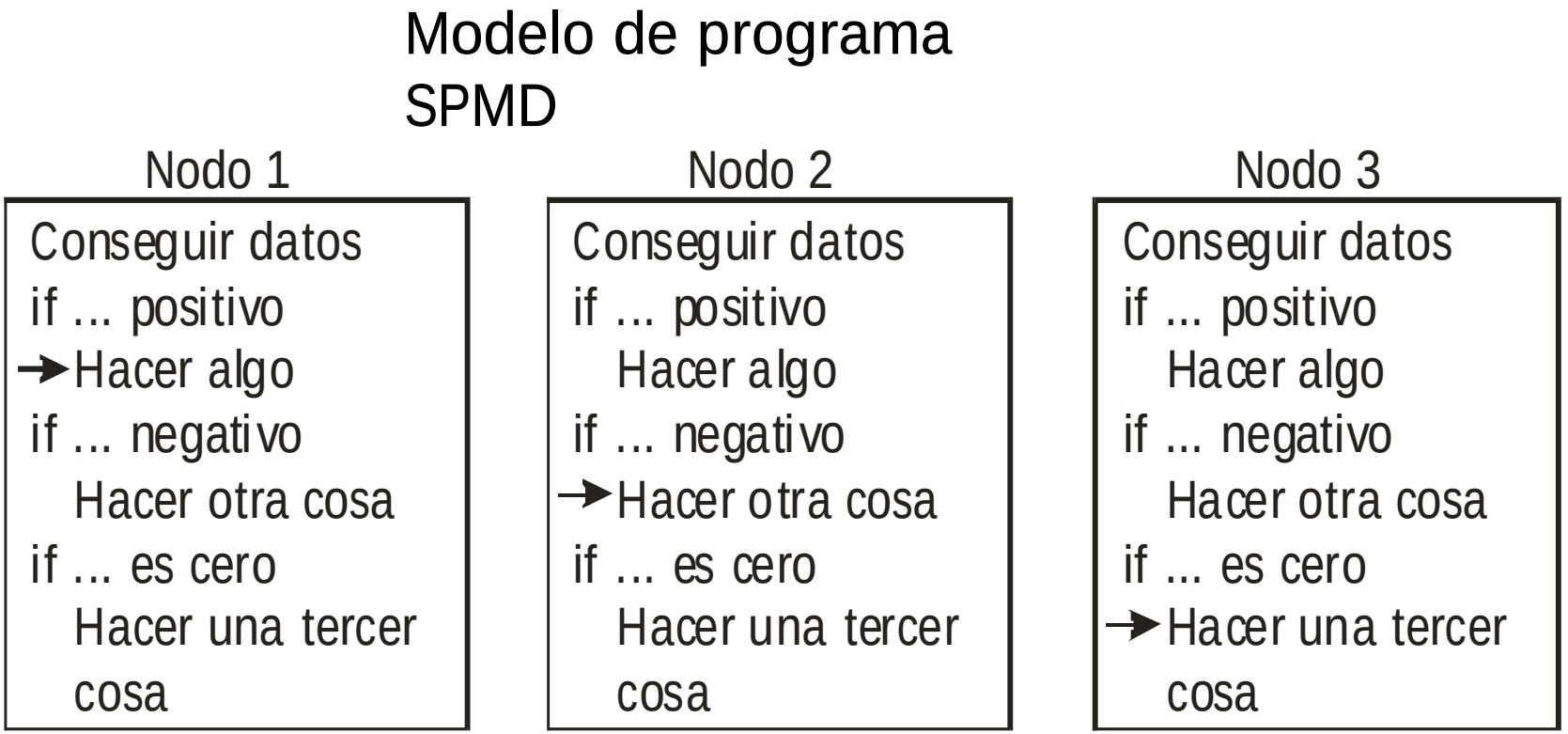
Modelo de programa SIMD

- Cargar Programa
- Broadcast de Instrucción



- Los Nodos reciben y ejecutan

Descomposición de Dominio

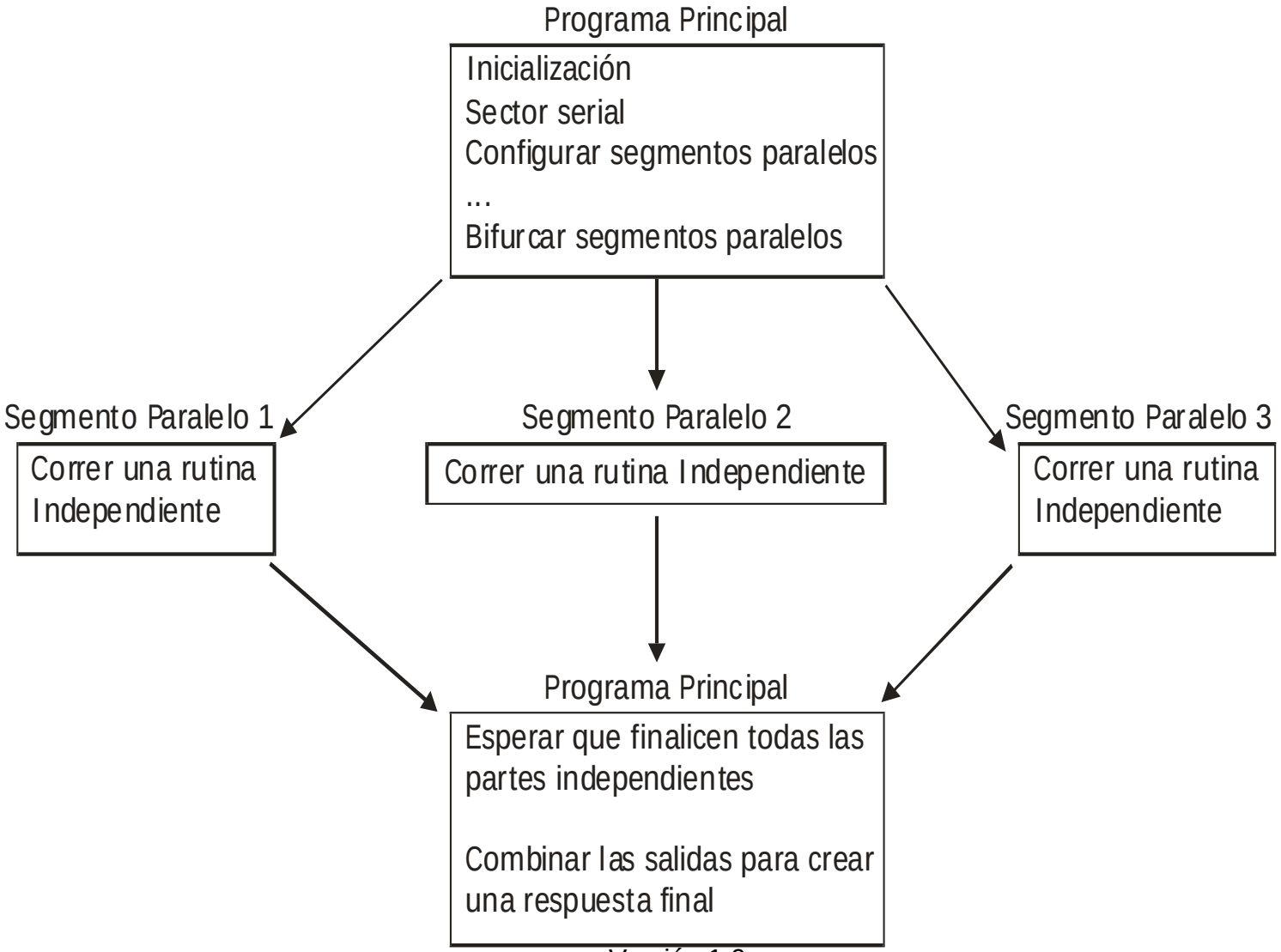


- Todos los nodos ejecutan el mismo programa, pero no las mismas instrucciones

Descomposición funcional

- Se concentra en el particionamiento de las operaciones del problema (control parallel).
  - Se trata de dividir el procesamiento en tareas disjuntas.
- Luego se examinan los datos que serán utilizados por las tareas definidas.
  - Si los datos son disjuntos, resulta un PARTICIONAMIENTO COMPLETO.
  - Si los datos NO son disjuntos, resulta un PARTICIONAMIENTO INCOMPLETO. Se requiere replicar los datos o comunicarlos entre los procesos asociados a las diferentes tareas.

Descomposición funcional



Modelos Híbridos

- Dos o más tipos de programación paralela dentro del mismo programa.
- Comúnmente utilizados en programas paralelos distribuidos en Internet (donde casi siempre existe la posibilidad de conseguir recursos *ociosos* adicionales).

Diseño y “paralelización”

- No siempre se dispone de recursos (en especial de tiempo) para diseñar una aplicación paralela – distribuida de acuerdo a los criterios y técnicas de programación especificadas.
- En múltiples ocasiones, se trata de obtener resultados de eficiencia computacional aceptable adaptando programas secuenciales a los modelos de programación paralela:

  “Paralelizar” una aplicación existente.
- Problemas

  Utilización de un código existente que no fue diseñado para ejecutar sobre múltiples recursos computacionales.

  Modelos enmarañados, “contaminación” del código heredado.

Paralelizando aplicaciones existentes

- Deben analizarse varios aspectos
  - ¿ Existe una partición funcional evidente ?
    - El código modular es el más fácil de paralelizar.
  - ¿ Existe forma de particionar los datos ?
    - Si existe, cual es la relación procesamiento/datos ?
  - ¿ Existen muchas variables globales ?
    - Cuidado !!! El manejo de recursos compartidos es un problema a resolver.
    - Considerar el uso de un servidor de variables globales

Modelos de comunicación entre procesos

- Cómo comunicar y/o sincronizar procesos paralelos.
- Modelos de comunicación:

  Modelo maestro-esclavo (master-slave).

  Modelo cliente-servidor.

  Modelo peer to peer.

Modelos de comunicación entre procesos

Master-slave

- Uno de los paradigmas de comunicación más sencillo
- Modelo en el cual se generan un conjunto de subproblemas y procesos que los resuelven
- Un proceso distinguido (*master*, maestro) y uno o varios procesos idénticos (*slaves*, esclavos).
- El proceso master controla a los procesos slave.
- Los procesos slave procesan.

Modelo master-slave

- El proceso master lanza a los esclavos y les envía datos.
- Luego de establecida la relación master/slave, la jerarquía impone la dirección del control del programa (del master sobre los slaves).
- La única comunicación desde los esclavos es para enviar los resultados de la tarea asignada.
- Habitualmente no hay dependencias fuertes entre las tareas realizadas por los esclavos (poca o nula comunicación entre esclavos).
- Modelos sincrónicos y asincrónicos.

Modelo master-slave

- Características:
  - El paradigma es sencillo, pero requiere programar el mecanismo de lanzado de tareas, la distribución de datos, el control del master sobre los slaves y la sincronización en caso de ser necesaria.
  - La selección de recursos es fundamental para la performance de las aplicaciones master/slave (balance de cargas).

Modelo cliente/servidor

- Modelo de comunicación que identifica dos clases de procesos. Procesos de una clase (los *clientes*) solicitan servicios a procesos de otra clase (los *servidores*), que atienden los pedidos.
- Puede ser utilizado para comunicar procesos que ejecutan en un único equipo, pero es una idea potencialmente más poderosa para comunicar procesos distribuidos en una red.
- El modelo C/S provee un mecanismo para comunicar aplicaciones distribuidas (que funcionen simultáneamente como clientes y servidores para diferentes servicios), convenientemente, de acuerdo a las características de la red.