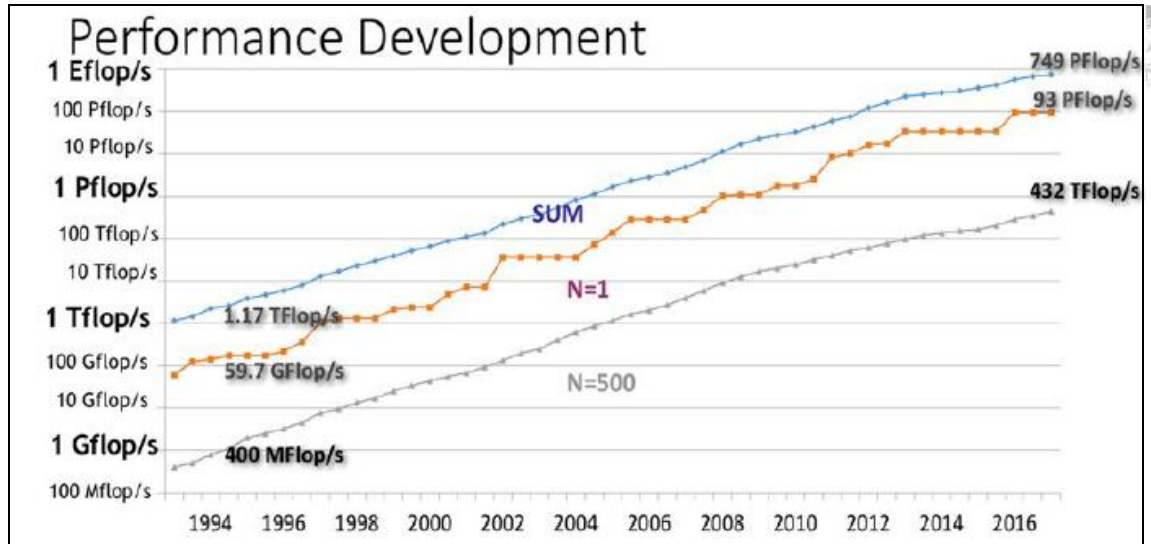


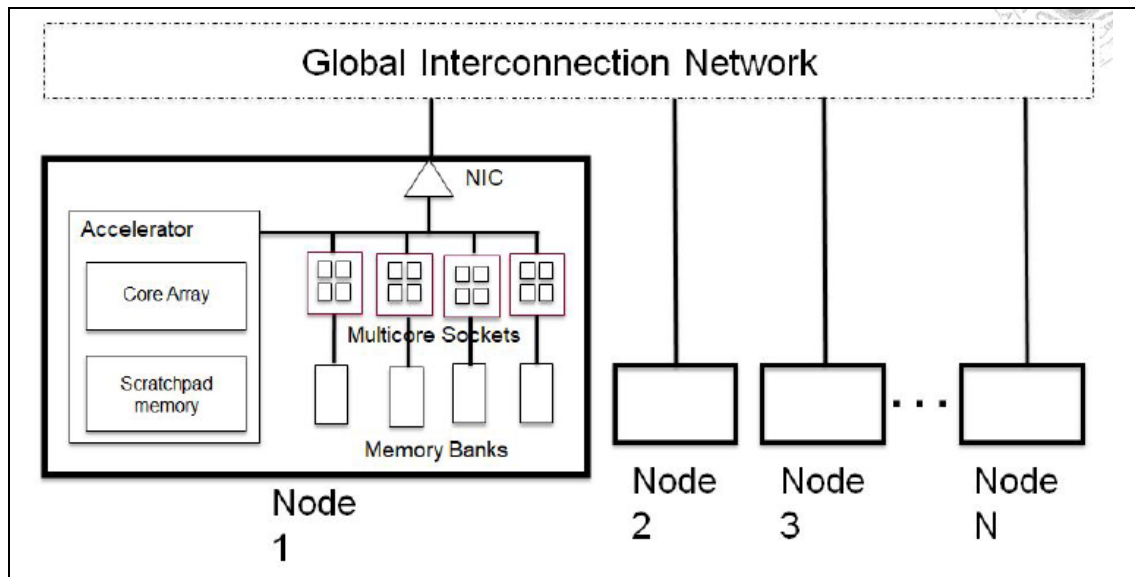
# 1 – INTRODUCCIÓN

## EVOLUCIÓN DEL RENDIMIENTO:



**Supercomputador:** sistema de computación de alto rendimiento con grandes recursos y capacidades en términos de tecnología, coste, energía y confiabilidad.

## ARQUITECTURA DE UN SISTEMA HETEROGÉNEO CONVENCIONAL:



## PARÁMETROS QUE AFECTAN AL RENDIMIENTO:

- Rendimiento pico.
- Capacidad RAM.
- Ancho de banda.
- Capacidad de almacenamiento secundario.

- Organización:
  - Clase de sistema (fabricante).
  - # nodos.
  - #procesadores por nodo.
  - Topología.

#### **ÉPOCA I - CÁLCULO MECÁNICO (1650-1950):**

- Tecnología: mecánica, tarjetas perforadas.
- Conceptos: secuenciación de instrucciones, almacenamiento de valores numéricos.
- Logros: ábaco ... ENIAC.

#### **ÉPOCA II – ARQUITECTURA VON NEUMANN (1950-1960):**

- Tecnología: válvulas de vacío, memoria de mercurio.
- Conceptos: modelo Von Neumann, computabilidad de Turing, lógica booleana.
- Logros: UNIVAC.

#### **ÉPOCA III – PARALELISMO / INSTRUCCIÓN (1960-1975):**

- Tecnología: transistor, semiconductores.
- Conceptos: paralelismo, mejorar la velocidad de reloj → tecnología.
- Logros: utilizar el transistor, CDC 6600, 1 MFLOP.

#### **ÉPOCA IV – PROCESAMIENTO VECTORIAL (1975-1980):**

- Tecnología: SSI (Small Scale Integration), MSI (Medium Scale Integration), RTL, TTL, etc.
- Conceptos: pipeline, modelos de ejecución vectorial, incrementar frecuencia de reloj.
- Logros: CRAY 1, 80MHz, 136MFLOPS.

#### **ÉPOCA V – SIMD-ARRAY / PVP (1980-1990):**

- Tecnología: LSI (Large Scale Integration).
- Conceptos: SIMD (Single Instruction Multiple Data), PVP (Parallel Vector Processing).
- Logros: GFLOPS, CRAY XMP, Maspar.

#### **ÉPOCA VI - CSP (1990-2005):**

- Tecnología: VLSI (Very Large Scale Integration), 32/64 bits microprocesador, SAN (System Area Network).
- Conceptos: CSP (Communicating Sequential Processing), memoria distribuida, paso de mensajes, BSP (Bulk Synchronous Parallel).
- Logros: commodity clusters, de 10s GFLOPS a PFLOPS.

#### **ÉPOCA VII – MULTICORE / HETEROGÉNEO (2005-2020):**

- Tecnología: multicore.
- Conceptos: ejecución híbrida.
- Logros: supercomputadores enormes, interfaces de programación híbrida.

#### **ÉPOCA VIII - EXASCALE (2020-2030):**

- Tecnología: apilamiento 3D, PIM (Processor-in-memory).
- Conceptos: nuevos modelos de programación, multi-threading.
- Logros: 3D-stack en EPFL.

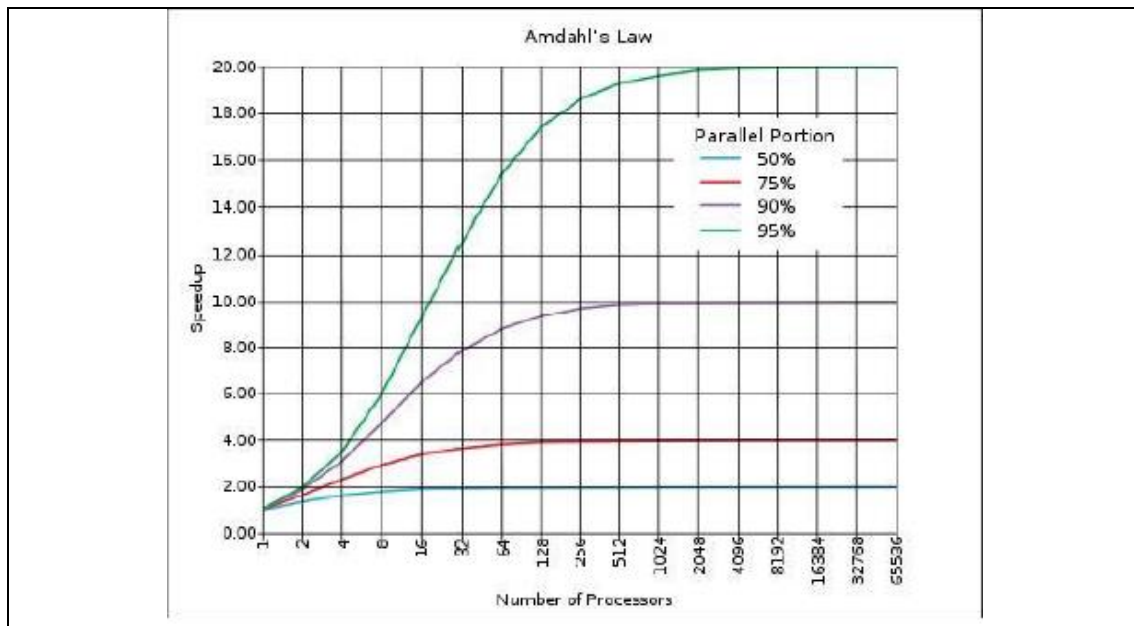
## ÉPOCA IX – COMPUTACIÓN CUÁNTICA (> 2030):

- Tecnología: criogénica, grafeno.
- Conceptos: mecánica cuántica.

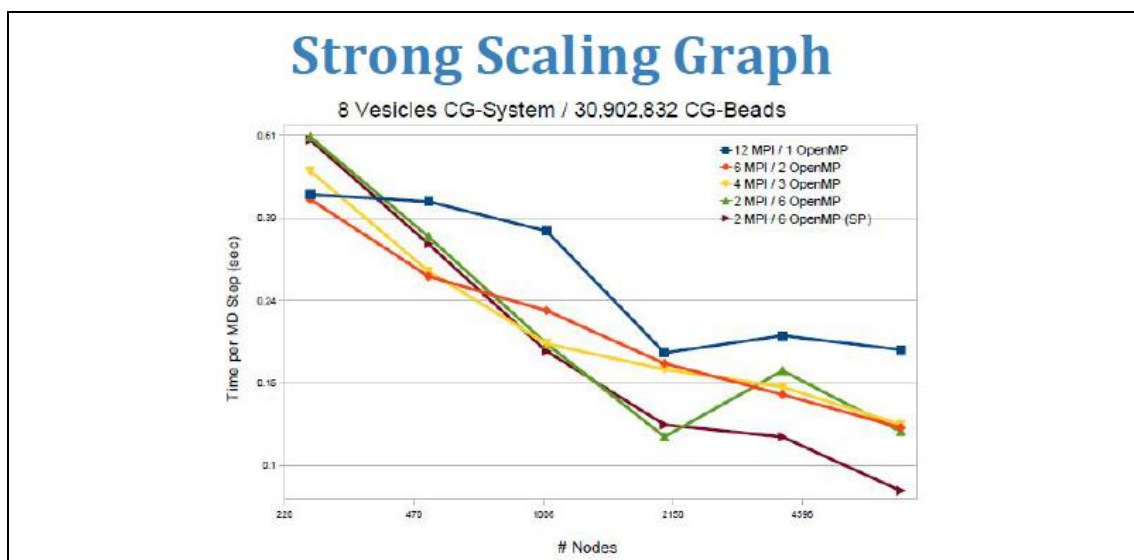
Logros: qubits.

### LEY DE AMDAHL:

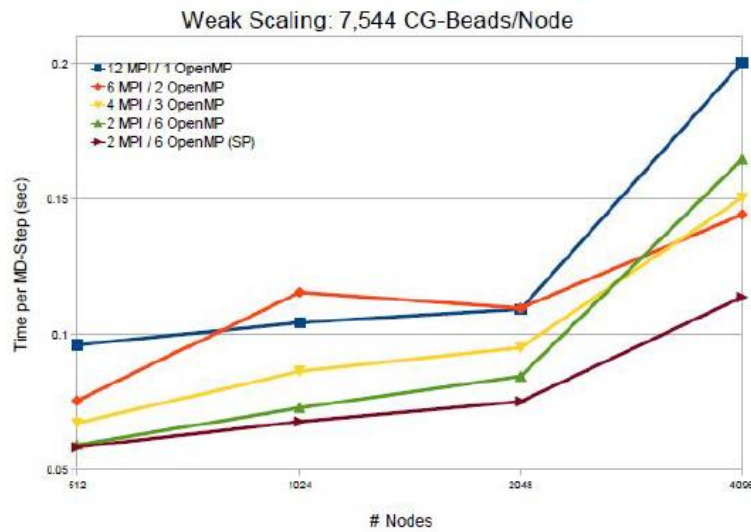
El *speedup* de un programa paralelo está limitado por la parte secuencial.



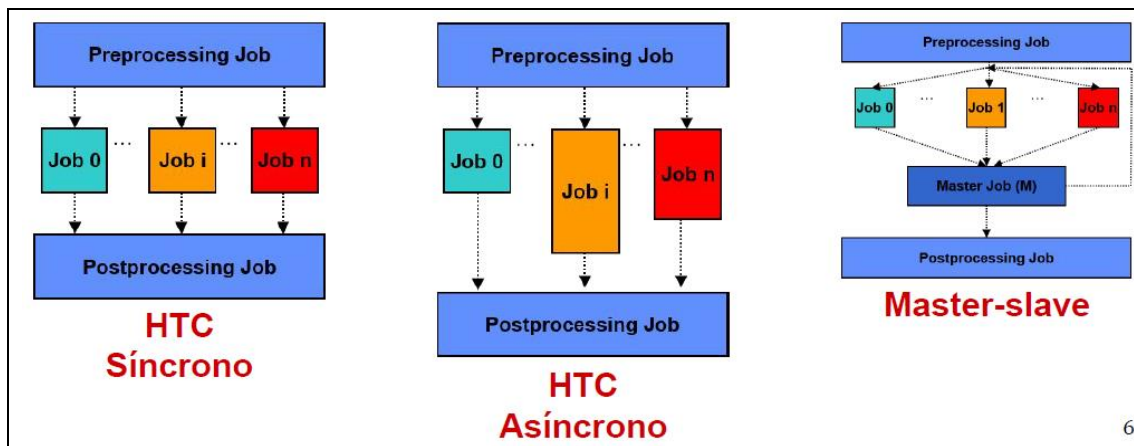
- Strong scaling: cómo varía el tiempo de ejecución en función del número de procesadores para un tamaño de problema total fijado. Mide el *speedup*.
- Weak scaling: cómo varía el tiempo de ejecución en función del número de procesadores para un tamaño de problema por procesador fijado. Mide si la velocidad es la misma.



## Weak Scaling Graph



### TIPOS DE APLICACIONES:



### SPEEDUP:

- Ley de Amdahl: la eficiencia obtenida viene limitada por la parte secuencial.

$$\beta = \frac{T_{\text{paralelo}}}{T_{\text{total}}} = 0.9667$$

$$\alpha = 1 - \beta = 0.0333$$

$$S_n = \frac{n}{1 + \alpha \cdot (n - 1)}$$

- Ley de Gustafson: el tamaño de problema crece con el número de procesadores.

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

### **ARQUITECTURAS MULTIPROCESADOR:**

- Memoria compartida
- Memoria distribuida
- Servidores HPC
  - Ventajas: Interconexión con ancho de banda alto y latencia baja  
Acceso uniforme al sistema gracias a una única copia del sistema operativo
  - Inconvenientes: Baja escalabilidad (para SMPs)  
Modelos complejos de programación (para MPPs)  
Precio alto

### **ARQUITECTURAS CLUSTER:**

- Clusters dedicados
  - Ventajas: Mejor relación coste/rendimiento para aplicaciones HTC  
Mayor escalabilidad
  - Inconvenientes: Requieren modelos de programación de memoria distribuida  
(librerías de paso de mensajes como MPI) para aplicaciones HPC
- Clusters no dedicados
  - Ventajas: Mínima relación coste/rendimiento para aplicaciones HTC  
Mayor escalabilidad
  - Inconvenientes: Interconexión con ancho de banda bajo y latencia alta  
Requiere capacidades de gestión adaptativa para usar los tiempos ociosos de los recursos dinámicos

### **ARQUITECTURAS GRID:**

- Complementaria a las anteriores.
- Interconecta recursos en diferentes dominios de administración respetando sus políticas y software de gestión.

### **MODELOS DE PROGRAMACIÓN:**

PARADIGMAS	MODELOS DE PROGRAMACIÓN	ARQUITECTURA
MC	<ul style="list-style-type: none"><li>- Paralelismo en control con directivas</li><li>- Memoria compartida estándar</li></ul>	UMA Y NUMA
MD	<ul style="list-style-type: none"><li>- Paralelismo en datos con directivas</li><li>- Paso de mensajes estándar</li></ul>	UMA, NUMA y MD

## 2 – NIVELES DE PARALELISMO

### NIVELES DE PARALELISMO:

- Paralelismo explícito
  - Paralelismo grueso:
    - Nivel de aplicación dentro de un computador: entre diferentes aplicaciones independientes
    - Nivel de programa o tarea dentro de una aplicación: entre tareas que cooperan
  - Paralelismo fino:
    - Nivel de procedimiento dentro de un programa: entre fragmentos de código
    - Nivel de bucle dentro de un procedimiento: entre iteraciones dentro de un mismo bucle
- Paralelismo interno
  - Nivel de instrucción: entre instrucciones o sentencias
  - Nivel de fases de una instrucción: entre las fases de ejecución de una instrucción
  - Nivel de bit: entre los bits de un operando

### GRANULARIDAD:

- Nivel de aplicación – grano muy grueso → < decenas de miles de instrucciones  
Explotado por el S.O. multiprogramado
- Nivel de programa – grano grueso → < miles de instrucciones  
Explotado por el programador
- Nivel de procedimiento – grano medio → < mil instrucciones  
Explotado por el programador con ayuda del preprocesador
- Nivel de bucle – grano fino → < 500 instrucciones  
Explotado por el compilador
- Nivel de instrucción – grano muy fino → < 20 instrucciones  
Explotado por el compilador y el microprocesador
- Paralelismo de grano fino:
  - No requiere mucho conocimiento del código
  - Paralelización automática
  - Se pueden obtener buenas eficiencias en poco tiempo
  - Ejemplo: descomposición de bucles
- Paralelismo de grano grueso:
  - Requiere conocimiento del código
  - Paralelización de alto nivel
  - Se paraleliza más código
  - Mejor rendimiento
  - Ejemplo: descomposición en dominios

# 3 – TIPOS DE APLICACIONES PARALELAS

## PARALELISMO A NIVEL DE TAREA:

- La aplicación consta de una serie de tareas que se pueden realizar simultáneamente y de manera independiente.
- Características:
  - Cada tarea es independiente y tiene su propio conjunto de datos de entrada
- Ejemplo: algoritmo que actúa sobre varios conjuntos de entrada.
- Ventajas:
  - Se obtienen buenos rendimientos (no hay parte secuencial, no hay sobrecarga de comunicaciones, no hay desigualdad de carga)
  - El tiempo invertido en la paralelización es mínimo (se pueden usar algoritmos secuenciales, no hace falta usar un lenguaje de programación paralelo)

## GRANJAS DE PROCESOS:

- Sistema formado por los siguientes tres tipos de procesos:
  - Raíz: encargado de dividir las tareas entre los trabajadores
  - Trabajador: recibe la tarea de la raíz, la procesa y envía los resultados al recopilador
  - Recopilador: recibe los resultados de los trabajadores e informa a la raíz de que un trabajador ha quedado disponible
- Se suele emplear cuando las tareas requieren diferente cantidad de computación → balanceo de carga dinámico.
- Limitaciones y mejoras:
  - Para evitar que los trabajadores queden esperando se coloca un buffer para aumentar la productividad. Un trabajador mantiene la tarea sobre la que trabaja y la siguiente
  - Reparto de las tareas para que los trabajadores terminen a la vez
  - Los trabajadores se comunican entre sí (gestión complicada)

## PARALELISMO FUNCIONAL:

### Segmentado

- La aplicación realiza una única tarea que se puede segmentar en etapas.
- El paralelismo se introduce solapando el procesamiento de varios conjuntos de entrada y los resultados parciales se transmiten entre las etapas del pipe.
- Característica:
  - La aplicación se debe poder dividir en etapas.
- Ejemplo: fases de lectura y escritura de ficheros.
- Ventajas:
  - Paralelización sencilla y natural
- Inconvenientes:
  - Las ineficiencias se deben al tiempo de inicialización del pipe y a la desigualdad de duración entre las etapas
  - El número de procesadores que se pueden utilizar viene limitado por el número de etapas de la aplicación.

### **Simultáneo**

- La aplicación realiza una única tarea que se compone de varias funciones o fragmentos de código que se pueden ejecutar en paralelo.
- Características:
  - La aplicación se debe poder dividir en funciones independientes
- Ejemplo: varias fases en el tratamiento de una imagen.

### **PARALELISMO A NIVEL DE DATOS:**

#### **Totalmente síncrono o balanceado**

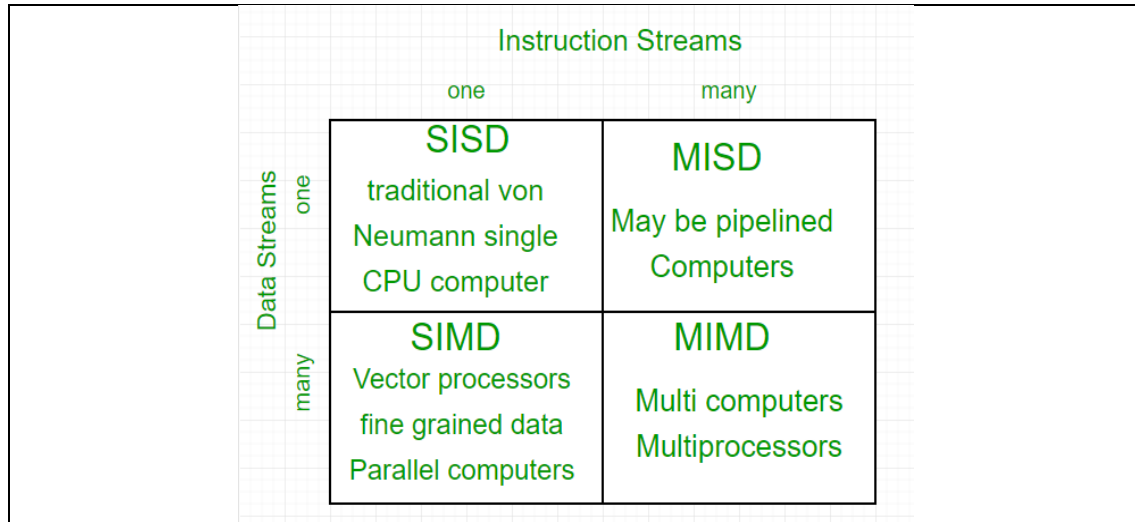
- Una operación se realiza de forma síncrona sobre todos los datos (los datos se reparten uniformemente entre los procesadores).
- Las ineficiencias son debidas a que el número de datos no es múltiplo del número de procesadores y al tiempo consumido en comunicación.
- Se requiere más trabajo de programación que en los casos anteriores.
- Ejemplo: cálculo de una integral.

#### **Débilmente síncrono o desbalanceado**

- Cada procesador realiza una parte de un problema heterogéneo, se produce una sincronización y se vuelve a tratar el problema.
- Las ineficiencias son debidas a la desigualdad dinámica del trabajo en cada procesador y al tiempo consumido en la interacción con los procesadores.
- Es más difícil de programar porque combina las dificultades de los paralelismos segmentado y totalmente síncrono



## 4 – ARQUITECTURAS PARALELAS



# 5 – OPTIMIZACIÓN EN MICROPROCESADORES

- Identificar las partes que consumen mayor cantidad de tiempo.
- Invertir el tiempo en las zonas computacionalmente más costosas (puntos calientes).
- Compilar el programa con la máxima optimización.
- Seleccionar el algoritmo óptimo.

## TÉCNICAS DIRECTAS:

- Reordenar las expresiones para minimizar el número de operaciones.
- Utilizar operaciones aritméticas menos costosas.
- Uso preferente de constantes frente a variables.
- Evitar exponenciación y conversiones de tipo.
- Eliminar los saltos y operaciones de entrada/salida dentro de los bucles.
- Minimizar el número de llamadas a subrutinas.
  - Problemas:
    - Producen fallos en la memoria cache de instrucciones
    - Alto overhead (salto y almacenamiento de contexto)
    - Disminución de la probabilidad de optimización en el compilador

## JERARQUÍA DE MEMORIA:

- El 90% del tiempo suele consumirse en el 10% del código.
- Localidad espacial (secuencial): acceso a instrucciones en programas y datos en arrays.
- Localidad temporal: bucles y subrutinas en programas y variables temporales.
- Según nos alejamos del procesador, la memoria es más lenta y de mayor capacidad.

## MEMORIA CACHE:

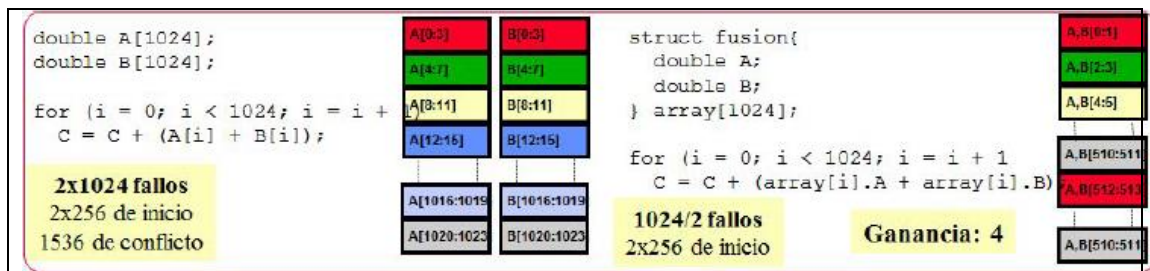
- Memoria pequeña y rápida situada entre el procesador y la memoria principal.
- Almacena la información de la memoria actualmente en uso.
- Disminuye el tiempo de acceso a memoria.
- Conceptos:
  - Bloque: unidad de transferencia
  - Marco de bloque: porción donde se ubican los bloques.
  - Acierto: el dato requerido se encuentra en memoria cache
  - Fallo: el dato requerido no se encuentra en memoria cache

## REGLAS FUNDAMENTALES:

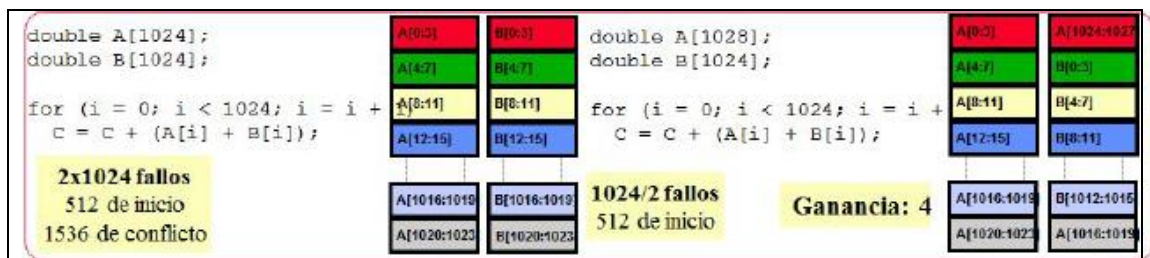
- Optimizar la localidad espacial: cuanto menor el stride de acceso a un array mejor.
- Optimizar la localidad temporal: utilizar los datos traídos lo máximo posible.

## OPTIMIZACIÓN DE LA MEMORIA CACHE:

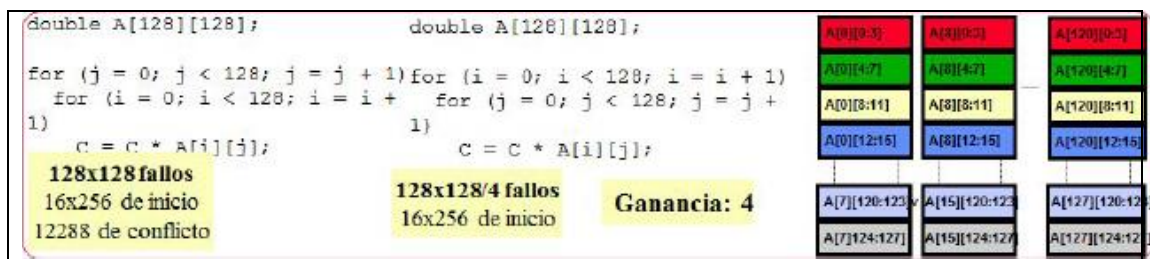
- Técnicas por parte del compilador o del programador.
- Fusión de arrays: mejora la localidad espacial para disminuir los fallos de conflicto.
  - Coloca las mismas posiciones de array en posiciones contiguas de memoria



- Alargamiento de array: mejora la localidad espacial y disminuye los fallos de conflicto.
  - Impide que en cada iteración se compita por el mismo marco de bloque



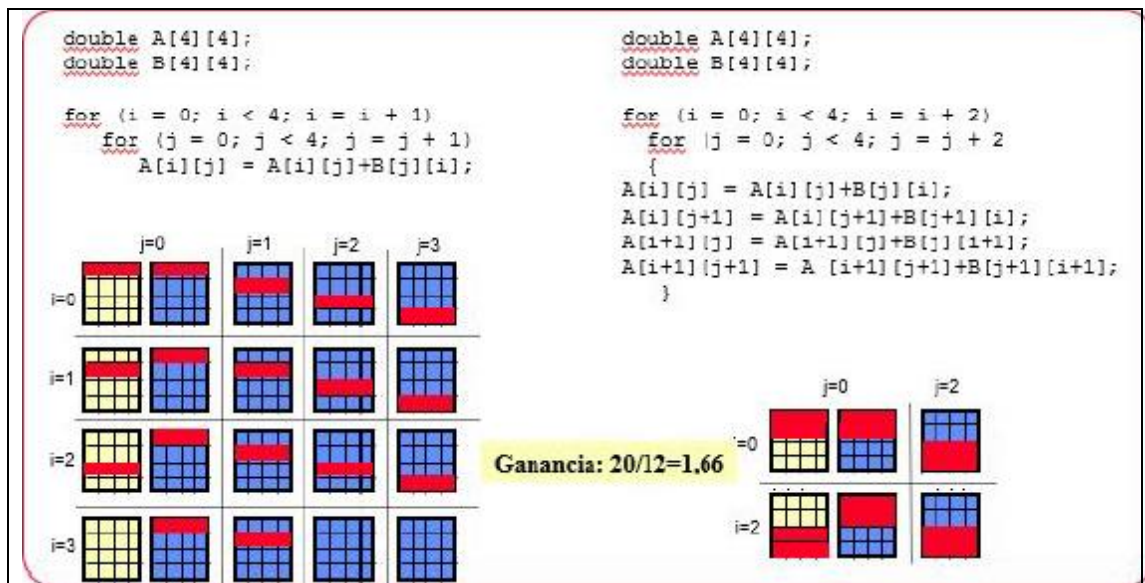
- Intercambios de bucles: mejora la localidad espacial y disminuye los fallos de conflicto.
  - Las matrices se almacenan por filas y por tanto es recomendable su recorrido en ese orden



- Fusión de bucles: mejora la localidad temporal para disminuir los fallos de conflicto.
  - Fusionar los bucles que usen los mismos arrays



- Bloqueo de matrices: mejora la localidad temporal y disminuye los fallos de conflicto.
  - Operar con submatrices



- Bit reversal: esta técnica puede usarse si el número de dimensiones de los array coincide con el número de bucles anidados. Consiste en trasponer la inicialización de los array.

### MEMORIA VIRTUAL:

- Facilita la compartición de la memoria por varios procesos.
- Funcionamiento:
  - Las direcciones virtuales generadas por el procesador deben traducirse a direcciones físicas.
  - La traducción se gestiona dividiendo el espacio en páginas.
- Las técnicas de optimización son las mismas que para cache.
- Bloqueo de matrices: se aplica cuando el número de bucles anidados es mayor que el número de dimensiones de las matrices.
- Bit reversal: se aplica cuando el número de bucles anidados es igual que el número de dimensiones de las matrices involucradas.

### PROCESADORES:

- Ejecución secuencial.
- Ejecución segmentada o escalar:
  - Ejecuta una instrucción por ciclo
  - Problemas con planificación estática: dependencias estructurales, de datos LDE y de control
  - Problemas con planificación dinámica: emisión en orden, pero lectura de operandos y ejecución en desorden. Dependencias de datos EDE y EDL y predicción de saltos
- Ejecución superescalar:
  - Ejecución de múltiples instrucciones por ciclo
  - Aumenta los problemas del caso segmentado
  - Superescalar: emisión de varias instrucciones por ciclo
  - Supersegmentado: pipes más profundos dividiendo una etapa en varias

- VLIW: una instrucción del procesador contiene varias que se pueden ejecutar simultáneamente

#### **OPTIMIZACIÓN DE PROCESADORES:**

- Ayuda a las unidades funcionales segmentadas.
- Disminuir el recargo de las instrucciones de control (minimizar saltos, incrementos de contadores y punteros...).
- Ayuda a los procesadores superescalares.
- Desenrollado de bucles.
- Fusión/Distribución de bucles.
- Software pipelining.

#### **OPTIMIZACIÓN DE ENTRADA/SALIDA:**

- Segmentar entre las diferentes fases de la entrada/salida.
- Paralelizar la propia entrada/salida por medio de threads.
- Utilizar discos y ficheros en memoria principal.
- Comprimir/escribir y leer/descomprimir.
- Utilizar formatos binarios en vez de ASCII.

## 6 – VISIÓN GLOBAL DE MODELOS DE PROGRAMACIÓN

- Memoria compartida (sin threads).
- Memoria compartida (con threads).
- Memoria distribuida / Paso de mensajes.
- Datos paralelos.
- Híbrido.
- SPMD.
- MPMD.
- Estos modelos no pertenecen a un tipo particular de máquina o arquitectura de memoria.

### **MEMORIA COMPARTIDA (SIN THREADS):**

- En este modelo de programación, los procesos/tareas comparten un espacio de direcciones común.
- Mecanismos como cerrojos o semáforos se usan para controlar el acceso a la memoria compartida, resolver condiciones de carrera e interbloqueos
- Ventaja: todos los procesos ven y tienen igual acceso a la memoria compartida.
- Desventaja: en términos de rendimiento, es difícil comprender y gestionar la localidad de los datos.

### **MEMORIA COMPARTIDA (CON THREADS):**

- Un proceso “pesado” puede tener múltiples vías “ligeras” de ejecución concurrentes.
- Cada hebra dispone de datos locales, pero también comparte los recursos. Cada hebra se beneficia de una visión global de la memoria porque comparte el espacio.
- Las hebras se comunican entre sí a través de la memoria global. Esto requiere de sincronización.
- Las hebras pueden implementarse como funciones de biblioteca invocadas desde un código fuente paralelo o como directivas de compilación.
- POSIX threads (basado en biblioteca) y OpenMP (basado en directivas de compilador).

### **MEMORIA DISTRIBUIDA / PASO DE MENSAJES:**

- Conjunto de tareas que usan su propia memoria local. Pueden residir en la misma máquina física o distribuidas entre varias máquinas.
- Las tareas intercambian datos enviando y recibiendo mensajes.
- Una operación de envío tiene que tener una operación de recepción.

### **DATOS PARALELOS:**

- También se conoce como PGAS (Partitioned Global Address Space).
- Las tareas realizan la misma operación sobre una porción del dataset global.
- En memoria compartida se accede al dataset a través de la memoria global.
- En memoria distribuida, cada tarea tiene una copia local de su porción del dataset.

### **HÍBRIDO:**

- El modelo híbrido combina más de uno de los modelos vistos previamente.
- Actualmente, uno de los modelos más usados combina MPI y OpenMP.
  - Las hebras OpenMP se encargan de la computación más intensiva
  - La sincronización entre procesos se resuelve con MPI
- Este modelo encaja muy bien en la naturaleza heterogénea actual de un clúster.
- Otros ejemplos: MPI con pthreads, MPI con aceleradores no-GPU.

### **SPMD:**

- Es un modelo de alto nivel que se puede construir con cualquier combinación de los modelos anteriores.
- Single Program: todas las tareas ejecutan su copia del mismo programa simultáneamente.
- Multiple Data: todas las tareas pueden usar diferentes datos.
- Los programas SPMD tienen lógica interna necesaria para seleccionar la porción de código con la que deben trabajar.

### **MPMD:**

- También es un modelo de alto nivel que se puede construir con cualquier combinación de los modelos anteriores.
- Multiple Program: las tareas pueden ejecutar programas diferentes de forma simultánea.
- Multiple Data: todas las tareas pueden usar diferentes datos.
- Los programas MPMD no son tan comunes como los SPMD. Se ajustan mejor a una partición funcional del programa paralelo.

# 7- MODELO DE PROGRAMACIÓN BASADO EN MEMORIA COMPARTIDA

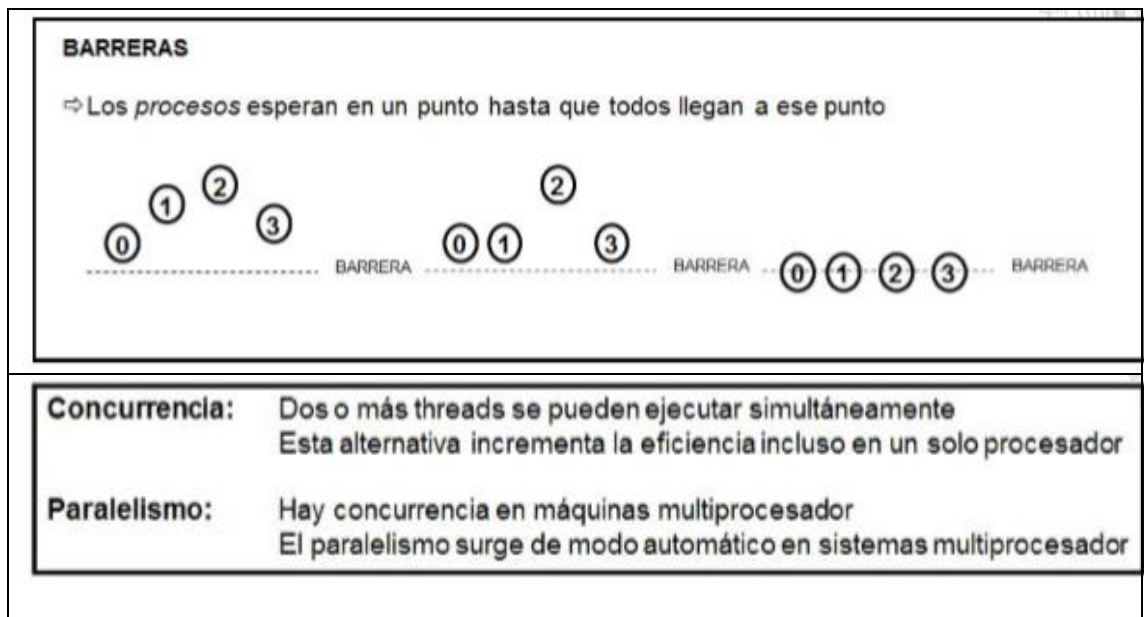
- Un sistema multiprocesador donde todos los procesos pueden ejecutarse con el mismo privilegio se denomina multiprocesador simétrico (SMP).
- Todos los recursos se comparten (memoria, E/S).
- Un sistema operativo soporta multiprocesamiento simétrico siempre que se puedan proteger regiones críticas.
- El planificador distribuye los procesadores disponibles entre los procesos (threads).
- El paralelismo aparece cuando un programa ha sido codificado (o compilado) de modo que fragmentos del mismo se ejecutan como procesos (threads) independientes.
- Computación basada en sincronización para garantizar la consistencia de los datos.
- La comunicación se realiza mediante compartición de los datos.

## FUNCIONES DE SINCRONIZACIÓN DE ALTO NIVEL:

- Garantizan el acceso secuencial a las secciones críticas.
- Ejemplos: cierres, semáforos y barreras.

<b>CIERRES</b>  <b>Variable cierre:</b> <i>flag</i> => ON(1) o OFF(0)  <b>Funciones:</b> <i>Lock(flag):</i> Si <i>flag</i> == ON (activado) entonces espera a que <i>flag</i> == OFF Si <i>flag</i> == OFF (desactivado) entonces <i>flag</i> = ON  <i>Unlock(flag):</i> <i>flag</i> = OFF	<div>P_padre: SUM = 0           for k =1 to N do               <b>Fork</b> P_hijo(k)           end for</div> <div>P_hijo: <b>Lock</b> (flag)           SUMA = SUMA + A(k)           <b>Unlock</b>(flag)           <b>Join</b></div>
<b>SEMÁFOROS</b>  <b>Variable semáforo:</b> S <i>Booleano:</i> 0 ó 1 (como los cierres) <i>General:</i> Entero >= 0  <b>Funciones:</b> <i>Wait(S) o P(S)</i> Si S == 0 entonces espera a que S > 0 Si S > 1 entonces S = S - 1  <i>Signal(S) o V(S):</i> S = S + 1  <b>Semáforos generales:</b> <ul style="list-style-type: none"><li>• Si se inicializa con M, puede haber M procesos en su sección crítica simultaneamente</li><li>• Problemas productores-consumidores o lectores-escritores</li></ul>	<div>P1: P(S)       Sección crítica de P1       v(S)</div> <div>P2: P(S)       Sección crítica de P2       v(S)</div>





### VENTAJAS E INCONVENIENTES:

- Mejora de la respuesta de las aplicaciones.
  - Permite la ejecución de funciones en threads independientes
- Uso de sistemas multiprocesador de modo más eficiente.
- Mejora la estructura de los códigos.
- Uso de menos recursos del sistema.
- Mejora del rendimiento.
  - En un multiprocesador los programas se ejecutan en menor tiempo de ejecución
  - En monoprocesador el tiempo de respuesta es menor, al poder solapar acciones (cómputo con E/S)

### PROCESOS VS THREADS:

#### Procesos:

- Espacio de direcciones con uno o más threads de control.
- Un contexto software y hardware.
- La sobrecarga por creación o carga de proceso es muy alta.
- Hay mecanismos artificiales para que varios procesos compartan alguna página de memoria.

#### Threads:

- Flujo de control dentro de un proceso.
- Los threads comparten un único espacio de direcciones.
- Se caracterizan por un contexto hardware.
- La sobrecarga por la creación o cambio de thread es muy pequeña.
- Los threads comparten memoria de modo natural.
- Un thread puede afectar a otro.
- Cada thread tiene su propia pila.
- Los threads se ejecutan de modo independiente.

# 8- MODELO DE PROGRAMACIÓN BASADO EN DIRECTIVAS CON PARALELISMO

- Paralelización explícita de un programa secuencial mediante la inclusión de directivas de compilación.
- Se trata de un paralelismo en control:
  - El trabajo se reparte entre los procesadores (iteraciones de bucle o subrutinas)
  - Los procesadores sincronizan periódicamente sus actividades

## TIPOS DE DIRECTIVAS:

- Directivas de control de repartición de subrutinas (grano medio y grueso):
  - Indican los fragmentos de código, funciones o subrutinas, que se deben ejecutar concurrentemente
- Directivas de control de repartición de iteraciones de un bucle (grano fino):
  - Indican cómo distribuir las iteraciones de un bucle entre los threads

## VENTAJAS:

- Es la manera más rápida de ejecutar en paralelo un programa secuencial.
- No se modifica el programa secuencial (portabilidad).
- Apropiado para todo tipo de grano.
- En una primera aproximación se puede realizar una paralelización automática.

## INCONVENIENTES:

- Se obtienen eficiencias pobres si la paralelización es a grano fino.
- Solo válido para sistemas con multiprocesamiento simétrico.
- En computadores ccNUMA no se tiene en cuenta la distribución de los datos.

## LOCAL Y COMPARTIDA:

- Una variable o array es local (privado) cuando es local cada una de las iteraciones de un bucle (thread).
  - El valor asignado en una iteración no se propaga al resto
  - Cada thread tiene una copia local no inicializada de las variables y arrays. Sólo existen durante la ejecución del bucle y el contador recorre todo el espacio de iteraciones asignado al thread
- Una variable o array compartida es una variable compartida por todas las iteraciones.
  - El valor asignado puede ser visto por otras iteraciones del bucle
  - Todos los threads comparten la única copia de las variables y arrays
  - No garantiza la exclusión mutua en el acceso a los datos compartidos

## ÚLTIMA LOCAL:

- Especifica variables o arrays que son locales y cuyo valor en la última iteración el bucle DO se usará tras la terminación del bucle.
- Puede producir resultados diferentes a la ejecución secuencial.

- El procesador que realiza la última iteración no tiene por qué coincidir con el procesador que contiene el último valor actualizado.

#### **REDUCCIÓN:**

- Una variable de reducción es aquella cuyo valor parcial se puede calcular localmente en cada thread, y cuyo valor final se obtiene operando con estos valores parciales.

#### **PRECISIÓN NUMÉRICA EN LAS OPERACIONES DE REDUCCIÓN:**

- Errores de desbordamiento: el rango de representación está acotado por el exponente.
- Errores de redondeo: los dígitos son finitos.
  - Eliminan la propiedad conmutativa de las operaciones: acumulación, anulación catastrófica

#### **SOLO LECTURA:**

- Especifica variables y arrays solo de lectura dentro del bucle (sólo compartidas).
- No se modifican durante la ejecución del bucle.
- El compilador replica el dato para cada thread y optimiza el acceso a memoria.

#### **TIPOS:**

- La estrategia de planificación especifica la distribución de las iteraciones de un bucle sobre los procesadores (threads).
- Muy importante para balancear la carga sobre los threads.
- Chunk: tamaño mínimo a repartir.
- Tipos:
  - Estáticas: reparto de iteraciones al comienzo del bucle y fijo durante la ejecución.
  - Dinámicas: reparto de iteraciones durante la ejecución del bucle.

#### **PLANIFICACIÓN ESTÁTICA:**

- Bloques: distribución uniforme de las iteraciones sobre los threads.
- Cíclica estática: distribuye un subconjunto de iteraciones en cada thread de modo circular. El tamaño del subconjunto es un valor por defecto.

#### **PLANIFICACIÓN DINÁMICA:**

- Cíclica dinámica: distribuye dinámicamente un subconjunto de iteraciones en cada thread de modo circular. Mejor balanceo cuando la carga es diferente en cada iteración.
- Factorizada: con  $n$  iteraciones y  $k$  threads distribuye dinámicamente  $n/2k$  iteraciones en cada thread. Existe un tamaño mínimo por defecto para los subconjuntos.
- Auto-planificación guiada: al primer thread le distribuye  $n/k$  iteraciones. El resto/ $k$  al siguiente y así sucesivamente. Buen balanceo y reduce el número de accesos a sección crítica.

#### **PILA DEL PROGRAMA:**

- El programa mantiene una pila para el programa principal y una por cada thread.
- Pueden ser estáticas o dinámicas.

- Cuando se realizan múltiples llamadas a la misma subrutina desde diferentes threads se crean interferencias al acceder todos a las mismas variables locales estáticas.
  - Es necesario hacer todas las variables automáticas.
  - Cada thread usará una copia local almacenada en su pila.

### **BUCLES PARALELIZABLES:**

#### **Requisitos**

- Es un bucle DO.
- Los valores de las variables array en cada iteración no dependen de los valores variables array en cualquier otra iteración.
- Si el bucle modifica un escalar, éste no debe ser utilizado al final del bucle ya que no se garantiza su valor final.
- Para cada iteración, cualquier subprograma invocado dentro del bucle no referencia o modifica valores de variables array para otra iteración.
- El índice contador del bucle deber ser un entero.
  
- Bucles con poca computación (pocas iteraciones): sobrecarga en la iniciación y sincronización de los threads.
- Bucles con dependencias: la ejecución simultánea de sus iteraciones produce resultados diferentes a la ejecución secuencial.
- En un anidamiento de bucles solo se paraleliza un nivel (de exterior a interior).

### **DEPENDENCIA DE DATOS:**

Un bucle no tiene dependencia de datos si:


1. Ninguna iteración escribe en una posición que es leída o escrita por otra iteración.
2. Las iteraciones pueden leer en una misma posición siempre que ninguna escriba sobre la posición.
3. Una misma iteración puede leer y escribir en una posición repetidamente.

#### **Tipos de bucles:**

- Imposibles de paralelizar (secuenciales).
- Paralelizables (sin dependencias).
- Paralelizables en función de una variable (indexaciones indirectas).
- Paralelizables si reestructuramos el código.
- Si una dependencia no se puede eliminar reestructurando debemos:
  - Sacar del bucle: fisión de bucles.
  - Si hay anidamiento intentar paralelizar el otro bucle.

### **DEPENDENCIA SERIE:**


- Una variable escalar se acumula de iteración en iteración.
- Se puede calcular el escalar en cada iteración sin referenciar valores anteriores.



**Negrita = local**

<p><b>SIN DEPENDENCIA</b></p> <div style="border: 1px solid black; padding: 5px; margin: 5px;"> <pre>DO 10 I = 1, N   10  A(I) = X +     B(I)*C(I)</pre> </div> <p style="text-align: right;"><b>paralelo</b></p>	<p><b>DEPENDENCIA DE DATOS</b></p> <div style="border: 1px solid black; padding: 5px; margin: 5px;"> <pre>DO 20 I = 2, N   20  A(I) = B(I) -     A(I-1)</pre> </div> <p style="text-align: right;"><b>secuencial</b></p>
<p><b>STRIDE MAYOR QUE UNO</b></p> <div style="border: 1px solid black; padding: 5px; margin: 5px;"> <pre>DO 20 I = 2, N, 2   20  A(I) = B(I) -     A(I-1)</pre> </div> <p style="text-align: right;"><b>paralelo</b></p>	<p><b>VARIABLE LOCAL</b></p> <div style="border: 1px solid black; padding: 5px; margin: 5px;"> <pre>DO I = 1, N   X = A(I)*A(I) +   B(I)   B(I) = X + B(I)*X END DO</pre> </div> <p style="text-align: right;"><b>paralelo</b></p>
<p><b>LLAMADA A FUNCIÓN</b></p> <div style="border: 1px solid black; padding: 5px; margin: 5px;"> <pre>DO 10 I = 1, N   X = SQRT(A(I))   B(I) = X*C(I) +   X*D(I) 10 CONTINUE</pre> </div> <p style="text-align: right;"><b>depende de la función</b></p>	<p><b>DEPENDENCIA SERIE</b></p> <div style="border: 1px solid black; padding: 5px; margin: 5px;"> <pre>INDX = 0 DO I = 1, N   INDX = INDX + I   A(I) = B(I) +   C(INDX) END DO</pre> </div> <p style="text-align: right;"><b>reestructurar</b></p>

37



**Negrita = local**

<p><b>SALTO INCONDICIONAL</b></p> <div style="border: 1px solid black; padding: 5px; margin: 5px;"> <pre>DO I = 1, N   IF (A(I) .LT. EPSILON) GOTO 320   A(I) = A(I) * B(I) END DO 320 CONTINUE</pre> </div> <p style="text-align: right;"><b>secuencial</b></p>	<p><b>DIRECCIONAMIENTO INDIRECTO</b></p> <div style="border: 1px solid black; padding: 5px; margin: 5px;"> <pre>INDEX = SELECT(N) DO I = 1, N   A(I) = A(INDEX) END DO</pre> </div> <p style="text-align: right;"><b>depende de INDEX</b></p>
<p><b>ANÁLISIS COMPLICADO</b></p> <div style="border: 1px solid black; padding: 5px; margin: 5px;"> <pre>DO I = K+1, 2*K   W(I) = W(I) + B(I,K) * W(I-K) END DO</pre> </div> <p style="text-align: right;"><b>paralelo</b></p>	
<p><b>ARRAY LOCAL</b></p> <div style="border: 1px solid black; padding: 5px; margin: 5px;"> <pre>DO I = 1, N   D(1) = A(I,1) - A(J,1)   D(2) = A(I,2) - A(J,2)   D(3) = A(I,3) - A(J,3)   TOTAL_DISTANCE(I,J) = SQRT(D(1)**2 + D(2)**2 +   D(3)**2) END DO</pre> </div> <p style="text-align: right;"><b>paralelo</b></p>	

38

### RECURRENCIA:

- Actualización en una iteración depende valores calculados en otras iteraciones.
- Recurrencia hacia adelante:  $X(I) = X(I-1)*B(I)+A(I) \rightarrow$  reestructurar.
- Recurrencia hacia atrás:  $X(I) = X(I+1)*B(I)+A(I) \rightarrow$  crear nuevo array.

### REDUCCIÓN:

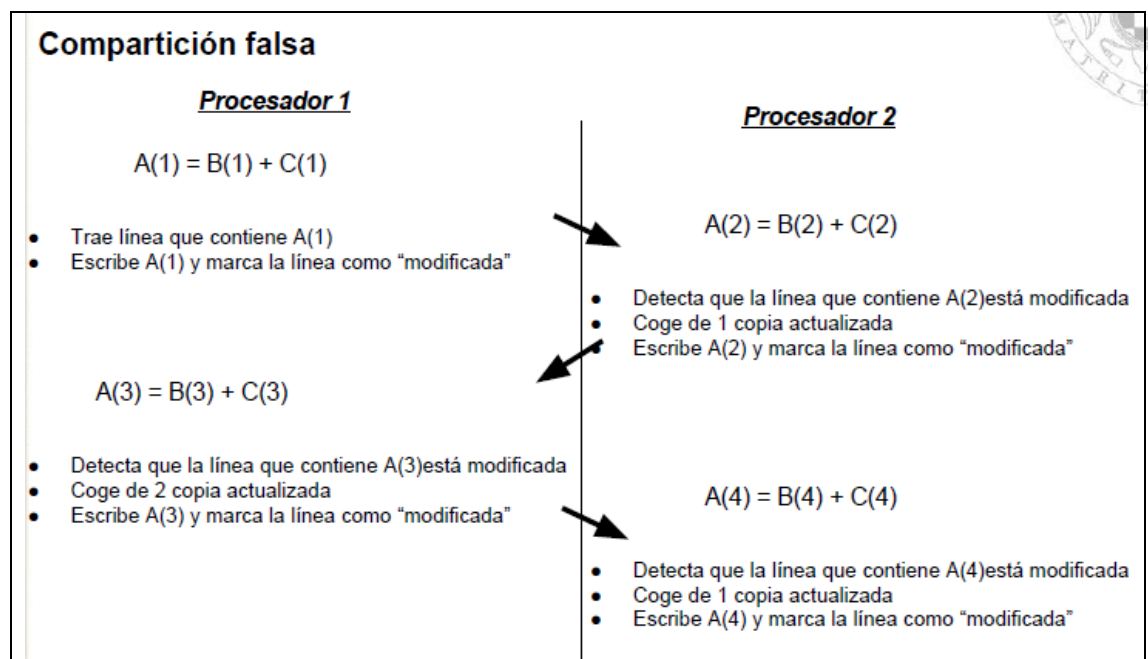
- Reducen arrays de dimensión  $d$  a  $d-1$  (exige una ejecución en orden de las iteraciones).
- Problemas:
  - Grado de paralelismo
  - Actualizaciones de variables compartidas

### DIRECCIONAMIENTO INDIRECTO:

- El uso de vectores de índices puede producir dependencias en tiempo de ejecución.
- Puede exigir una ejecución en orden de las iteraciones.

### FALSA COMPARTICIÓN:

- El problema no reduce el tiempo de ejecución con el número de procesadores.
- Uso ineficiente de la memoria cache. Los procesadores leen la misma línea de cache.
- Puede ocurrir con matrices compartidas que se modifican en un bucle.



# 9- MODELO DE PROGRAMACIÓN BASADO EN EL PASO DE MENSAJES

- Consiste en la replicación del paradigma de programación secuencial.
- Se divide la aplicación en varios procesos que se ejecutan en diferentes procesadores sin compartir memoria y comunicándose por medio de mensajes.

## VENTAJAS:

- Portable de modo eficiente a cualquier tipo de arquitectura: computador paralelo, red de estaciones y a una única estación de trabajo.

## INCONVENIENTES:

- Bastante más complicado de programar y depurar que memoria compartida.

## SPMD vs MPMD:

- Muchos computadores actuales solo soportan SPMD, el mismo ejecutable en cada procesador.
- Muchos sistemas actuales no soportan tiempo compartido, los procesadores están siendo empleados por un único usuario (cola de trabajos en cada partición).

## CARACTERÍSTICAS DE LOS MENSAJES:

- Un mensaje es una transferencia de datos de un proceso a otro.
- Información que caracteriza un mensaje:
  - Send
    - En qué variable están los datos que se envían
    - Cuántos datos se envían
    - Qué proceso recibe el mensaje
    - Cuál es el tipo de mensaje que se envía
  - Receive
    - Cuál es el tipo de mensaje que se envía
    - Qué proceso envía el mensaje
    - Dónde almacenar los datos que se reciben
    - Cuántos datos espera recibir el proceso receptor
- Partes de un proceso que se comunica por medio de mensajes:
  1. Acceso: el proceso debe unirse al sistema de paso de mensaje, obtiene un número y el número de procesos en el grupo.
  2. Ejecución: el proceso realiza operaciones interactuando con el resto por medio de las rutinas de paso de mensajes.
  3. Finalización: antes de terminar, el proceso debe abandonar el sistema de paso de mensajes.

### **COMUNICACIÓN PUNTO A PUNTO:**

Envío síncrono y asíncrono (condición de finalización):

- Síncrona: el proceso que realiza el envío recibe información sobre la recepción del mensaje. La comunicación se completa cuando el mensaje ha sido recibido.
- Asíncrona: el proceso únicamente conoce cuando se envía el mensaje. La comunicación se completa tan pronto como el mensaje ha sido enviado.

Operaciones bloqueantes y no bloqueantes (espera o no la condición de finalización):

- Operación no bloqueante: se inicia la operación y se vuelve al programa, por medio de otras funciones se puede comprobar la finalización de la operación.
- Operación bloqueante: solo se vuelve al programa cuando la operación ha finalizado.

### **COMUNICACIONES COLECTIVAS:**

- Barrera
- Broadcast
- Reducción



# 10- MODELO DE PROGRAMACIÓN CON DIRECTIVAS DE DISTRIBUCIÓN DE DATOS

## PARALELISMO EN DATOS SOBRE LOS CLÁSICOS VECTORIALES Y SIMD:

- Misma operación sobre un conjunto de datos.
- Flujo secuencial.
- Opciones:
  - Aumentar el lenguaje con sentencias que realizan operaciones sobre conjuntos
  - Instrucciones tipo MATLAB
  - Lenguaje secuencial con directivas

## PARALELISMO EN DATOS CON DIRECTIVAS DE DISTRIBUCIÓN:

- En los computadores tipo NUMA la distribución de los datos es fundamental para obtener buenos rendimientos.
- En los computadores de memoria distribuida es el único modo de programar distribuido por medio de directivas.
- El programador especifica:
  - La distribución de los datos compartidos
  - Qué variables son privadas
  - Fortran-Plus, CRAFT, Viena Fortran → El estándar HPF

# **11- GESTIÓN DE RECURSOS**

## **TIPOS DE RECURSOS GESTIONADOS:**

- Nodos de computación.
- Núcleos de procesado.
- Interconexiones.
- Almacenamiento permanente y E/S.
- Aceleradores.

## **NODOS DE COMPUTACIÓN:**

- Cada uno aporta una computación y almacenamiento.
- Incrementar el número permite el escalado de la aplicación.
- Los recursos varían en función del nodo:
  - Tipo de CPU y frecuencia de reloj
  - Capacidad de memoria principal
  - Ancho de banda de interconexión y latencia

## **NÚCLEOS DE PROCESADO:**

- Proporcionan paralelismo local:
  - Múltiples núcleos por procesador
  - Múltiples procesadores por nodo
- Compartición de elementos de procesado entre diferentes aplicaciones:
  - Permite un mejor aprovechamiento de los recursos
  - El modo exclusivo destina el nodo completo a una única aplicación

## **INTERCONEXIÓN:**

- Normalmente un tipo por sistema:
  - 10G Ethernet
  - InfiniBand
  - 1G Ethernet (GigE)
- Son posibles instalaciones con múltiples redes.

## **ALMACENAMIENTO PERMANENTE:**

- Normalmente implementado como un sistema de ficheros compartido.
- Pueden proporcionarse múltiples sistemas para distintos escenarios de uso.
- Almacenamiento localizado puede proporcionar mejor ancho de banda para datos agregados.

## **ACELERADORES:**

- Procesadores de uso especial:
  - Heterogéneos
  - Gran número de elementos de procesado de grano fino.
  - Consumen menos energía y tiempo para determinadas tareas.
- Variables comunes:
  - GPGPU (General Purpose Graphics Processing Unit)

- Intel Xeon Phi
- FPGA (Field Programmable Gate Array)
- La distribución de aceleradores en el sistema hardware puede no ser uniforme.

#### **TAREA DE CÓMPUTO:**

- Unidad de trabajo que puede necesitar unos datos de entrada y produce un resultado durante su ejecución.

#### **COLA DE TAREAS:**

- Define el orden en el que la máquina ejecutará las tareas:
  - Normalmente FIFO (“first-in,first-out”).
  - Los planificadores pueden alterar el orden preestablecido para incrementar la productividad o por la sobreescritura de operadores.
- Agrupa tareas que utilizan los mismos recursos

#### **PLANIFICADOR DE TAREAS:**

- Parámetros que afectan a la planificación:
  - Disponibilidad de los recursos
  - Prioridad de la tarea
  - Recursos destinados
  - Número máximo de tareas
  - Tiempo de ejecución solicitado
  - Tiempo de ejecución transcurrido
  - Dependencias y prerrequisitos de la tarea
  - Eventos específicos
  - Disponibilidad del operador
  - Disponibilidad de la licencia software

#### **GESTORES DE RECURSOS COMUNES:**

- SLURM (Simple Linux Utility for Resource Management).
- PBS (Portable Batch System).
- LFS (OpenLava bases on the Load Facility Sharing).
- Moab Cluster Suite.
- Tivoli Workload Scheduler LoadLeveler.
- Univa Grid Engine.
- HTCondor.
- OAR,
- YARN (Hadoop Yet Another Resource Navigator).

#### **SLURM:**

- Utilizado en el 60% de las máquinas del TOP500.
- Complejas opciones de configuración:
  - Diferentes tipos de interconexión
  - Algoritmos de planificación
  - Implementaciones MPI
- Escalable hasta 10 millones de núcleos.
- Gestión de hasta 1000 tareas y 500 ejecuciones de tareas por segundo.

- Múltiples estrategias para optimización energética.
- Decisiones dependientes de la arquitectura del nodo (NUMA, distribución de núcleos).
- Gestión dinámica de recursos para las tareas.
- Algoritmos de planificación.

#### **ORGANIZACIÓN DE LA CARGA DE TRABAJO:**

- Particiones.
- Puede utilizarse la totalidad o una fracción de los nodos asignados.
- Los arrays de tareas sirven para gestionar gran cantidad de tareas similares.

#### **PLANIFICACIÓN:**

- Basada en eventos por defecto.
- Planificación de ajustes:
  - Inicia tareas de baja prioridad si no afecta en el inicio de las de mayor prioridad
  - Incrementa la utilización
- Planificación en grupo:
  - Para tareas similares fija los mismos recursos.
- Prevención:
  - Detiene tareas de baja prioridad para iniciar las de mayor prioridad
- Planificación dirigida por recursos:
  - Basada en la disponibilidad de los dispositivos hardware
- Computación elástica:
  - Permite la gestión dinámica de los recursos consumidos,
- Computación de alta productividad.

#### **PBS:**

- Capaz de ejecutar en grid usando Globus toolkit.
- Adición/eliminación dinámica de nodos de ejecución.
- Compatible con TORQUE (Terascale Open-source Resource and QUEUE manager).

# 12- SERVICIOS CLOUD

## TIPOS DE SERVICIO CLOUD:

- Software como servicio → acceso bajo demanda a una aplicación → usuario final.
- Plataforma como servicio → plataforma para desarrollo y ejecución → desarrollador.
- Infraestructura como servicio → recursos IT → administrador de sistemas.