Universidad Francisco de Vitoria
Primer cuatrimestre del 2022

Profesor: Diego Gabriel Gachet
Correo electrónico: diegogabriel.gachet@ufv.es

Computación de Alto Rendimiento

4º Ingeniería Informática

—

Información de la asignatura

Conocimientos Previos

'Sistemas Operativos' y 'Arquitectura y Organización de Computadores'

Organización

En bloques según la siguiente tabla:

Bloque 1	Bloque 2	Bloque 3	Bloque 4	Bloque 5
Introducción	Concurrencia y Sincronismo	Computación Paralela	Sistemas de Tiempo Real	Tópicos Avanzados

Metodología

3 prácticas (código + documentación) y una extra en función del tiempo que conformarán el proyecto final de la asignatura. Estas prácticas tratarán sobre:

Práctica 1	Práctica 2 (main)	Práctica 3	Práctica Extra
POSIX sobre Linux	CUDA (Nvidia)/Numba	Free-RTOS ó MBED	MPI u OPENMP

Evaluación

Por porcentajes según la tabla, a la que se le añadirá un 5% de participación en clase:

Tipo de Convocatoria	40% (mínimo 5)	25% (mínimo 5)	30% (mínimo 5)
Evaluación Continua	2 Exámenes (media con 4)	Proyecto Final	Presentación del Proyecto Final
Convocatoria Ordinaria y Extraordinaria	Exámen Final	Proyecto Final	Presentación del Proyecto Final

—

Teoría

Introducción a la Computación de Alto Rendimiento

Tema introductorio que sirve a la vez como repaso de las asignaturas cursadas previamente y como introducción a los conceptos más básicos de la High Performance Computing.

Introducción al Paralelismo

El **paralelismo** consiste en combinar la potencia de cálculo de varios procesadores para aumentar el rendimiento. Se basa en:

- Distintas formas de interconexión.
- Información compartida.
- Datos compartidos.
- Paralelización del código.
- Reparto de la carga de trabajo.

Sirve para:

- Solución a problemas de optimización. Simulación (Ej: Programación Genética).
- Solución a problemas científicos con alta carga computacional (Ej: Proyecto Genoma Humano).

- Sistemas distribuidos en Internet altamente escalables (Ej: Aplicaciones comerciales o superordenadores, como el Mare Nostrum de Barcelona).

Tipos de Paralelismo

El objetivo del paralelismo es mejorar el **Speed-Up**:

$$S = \frac{T_s}{T_P}$$

y así poder abordar problemas con alta demanda de memoria.

Paralelismo Implícito

El **paralelismo implícito** es transparente al software, permanece dentro del procesador y no afecta al programa.

Dentro del paralelismo implícito encontramos las **arquitecturas segmentadas** (o pipelines) que se basan en el encadenamiento del proceso de ejecución de instrucciones.

Este tipo de paralelismo, además, se basa en la **división funcional**; es decir, la presencia de múltiples unidades funcionales aritmético-lógicas.

Paralelismo Explícito

En el **paralelismo explícito** se dispone de varios procesadores y el software define en cual se ejecuta la instrucción.

En este caso, **según la multiplicidad de procesos simultáneos** (es decir, según la taxonomía de Flynn), tenemos:

- **Single Instruction, Single Data (SISD)** → Monoprocesadores con arquitectura Von Neumann).
- **Single Instruction, Multiple Data (SIMD)** → Varios procesadores para la misma instrucción.
- **Multiple Instructions, Single Data (MISD)** → Procesadores independientes ejecutan instrucciones independientes.
- **Multiple Instructions, Multiple Data (MIMD)** → Múltiples procesadores ejecutan simultáneamente múltiples instrucciones.

Elementos del Ordenador Paralelo

Pueden estar basados en hardware o en software:

- Para Hardware:
    - Múltiples Procesadores
    - Múltiples Memorias
    - Redes de Interconexión (Datapaths)
- Para Software:
    - Sistemas Operativos Paralelos
    - Programas orientados a la concurrencia

Organización Lógica

La **Organización Lógica** es la visión que tiene el programador acerca de la máquina.

Modelos de Comunicación

Dentro de la organización lógica, contamos con 2 modelos de comunicación:

- Mediante **paso de mensajes**. Puede ser:
    - El procesador se comunica con un espacio de memoria propio.
    - Comunicación por mensajes entre Emisor y Receptor.
- Mediante **espacio de memoria compartida**. Puede ser:
    - Uniform Memory Access (UMA).
    - Non Uniform Memory Access (NUMA).

Organización Física

La organización física se basa en **Redes de Interconexión**; es decir, conexión entre los diferentes procesadores y la memoria del sistema.

Tipos de Interconexión

- **Estática** → Enlaces punto a punto (conectar nodos).
- **Dinámicas** → Elementos de conmutación (conectar nodos y/o bancos de memoria).

Métricas de Evaluación

- **Diámetro** → Distancia máxima entre cualquier nodo (cuanto menor, mejor).
- **Conectividad** → Mínimo número de arcos a eliminar para dividir la red en 2 (cuanto mayor, mejor).

- **Ancho de Bisección** → Mínimo de arcos a eliminar para dividir la red en 2 partes iguales (cuanto mayor, mejor).
- **Ancho de Banda de Bisección** → Mínimo volumen de comunicación entre 2 mitades cualesquiera (cuanto mayor, mejor).
- **Coste** → Número de enlaces de la red (cuanto menor, mejor).

Motivaciones del Paralelismo

- Ley de Moore → El número de transistores se dobla cada 2 años.
- Escala de Dennards → A lo largo de los transistores se hacen más pequeños, su potencia se mantiene.
- Inicio de la era Multicore (2005).

Pasado y Presente del Paralelismo

- Vector Massively Parallel Processing Transition
- Transición Terascale-Petascale
- Transición Petascale-Exascale → Supercomputadores Petascale-Exascale

Futuro del Paralelismo

- Lógica Reconfigurable
- Procesamiento centrado en Recursividad
- Computación Neuromórfica
- Computación Cuántica
- Computación Analógica
- Fotónica de Silicona

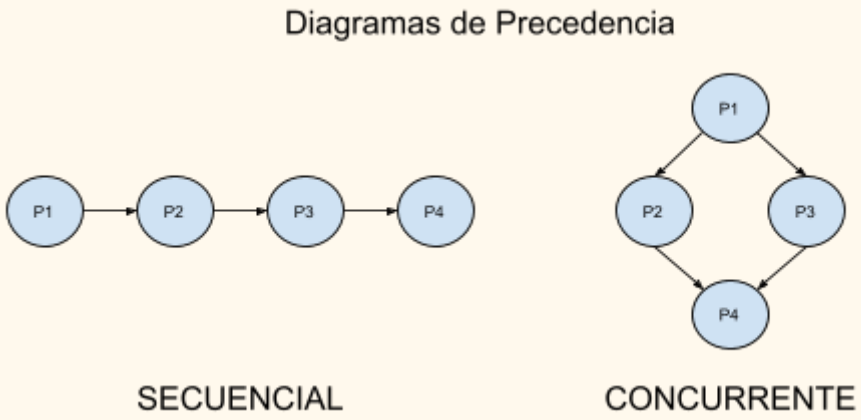
Conceptos de la Computación Concurrente

Tema introductorio que sirve a la vez como repaso de las asignaturas cursadas previamente y como introducción a los conceptos más básicos de la Computación Concurrente.

La computación concurrente se basa en procesos. Donde un proceso es la instancia de un programa en ejecución; la unidad de trabajo más pequeña individualmente planificable por un Sistema Operativo.

Las ventajas de las **operaciones multitarea** son:

- Ganancia de Velocidad
- Uso de dispositivos I/O (Entrada/Salida)
- Conveniencia para el usuario
- Multiprocesamiento
- Computación Distribuida



Sección Crítica → Una sección crítica es una parte del programa que accede a un dato o recurso compartido (un recurso crítico). En cualquier momento solo 1 proceso puede ejecutarse la Sección Crítica. Se necesita un permiso para que un proceso pueda entrar en la Sección Crítica. La Sección Crítica tiene: sección de entrada y sección de salida.

Exclusión Mútua → Actividad que realiza el sistema operativo para evitar que 2 o más procesos ingresen al mismo tiempo a un área de datos compartidos al mismo recurso. Si ningún proceso está en la sección crítica, puede entrar sin pedir permiso.

—

Ejercicios

- Visita la página del Top 500 Supercomputers (www.top500.org) y analiza los 5 primeros supercomputadores en relación al número de FLOPS.
- Listar los principales problemas que requieren el uso de computación de alto rendimiento en diferentes campos de la ciencia

Está en las transparencias

- Dar un modelo eficiente para la ejecución considerando el siguiente segmento de código para sumar 2 vectores:

```
1  for (i = 0; i < 1000; i++)
2          c[i] = a[i] + b[i];
```

En este ejemplo, varias iteraciones del bucle son independientes entre sí; es decir:

- c[0] = a[0]+ b[0]
- c[1] = a[1] + b[1]
- …
- c[999] = a[999] + b[999]

Por lo que pueden ejecutarse independientemente unos de otros. En consecuencia, si existe un mecanismo para ejecutar la misma instrucción, en este caso “sumar”, en todos los procesadores con los datos apropiados, podríamos ejecutar este bucle mucho más rápido.

- ¿Cómo modelar un sistema de memoria compartida en un sistema de paso de mensajes?

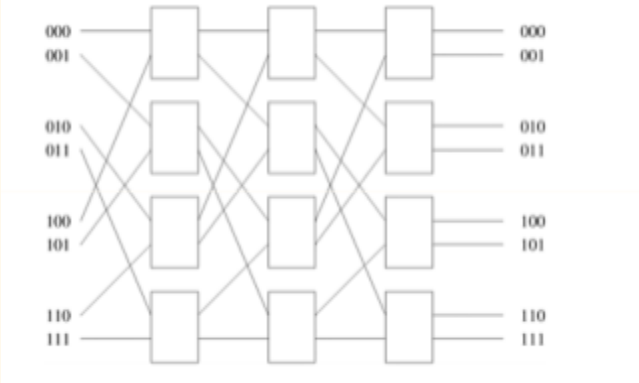
Es fácil emular una arquitectura de paso de mensajes que contiene nodos ***p*** en un equipo de espacio de direcciones compartido con un número idéntico de nodos.

Asumiendo nodos de un solo procesador, esto se puede hacer particionando el espacio de direcciones compartidas en ***p*** partes disjuntas y asignando una partición de este tipo exclusivamente a cada procesador. Un procesador puede entonces "enviar" o "recibir" mensajes

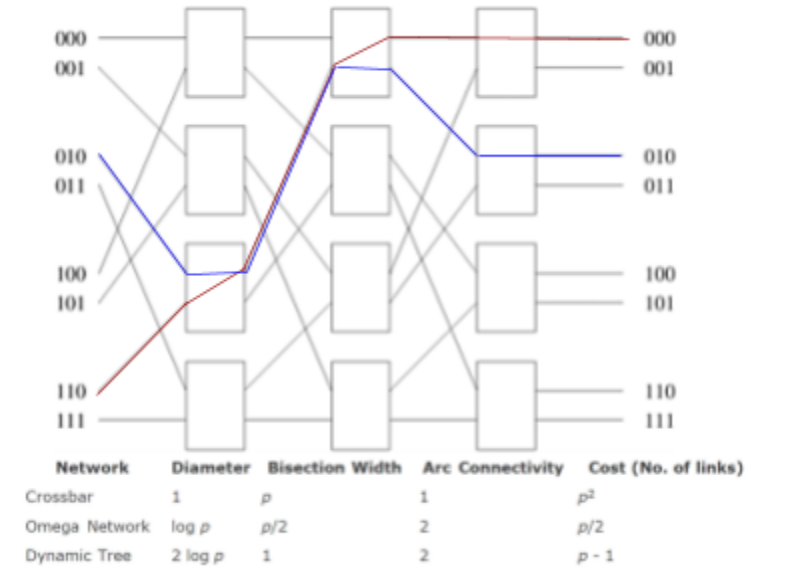
escribiendo o leyendo desde la partición de otro procesador mientras utiliza primitivas de sincronización apropiadas para informar a otros nodos de comunicación cuando ha terminado de leer o escribir los datos.

Sin embargo, emular una arquitectura de espacio de direcciones compartido en un equipo que pasa mensajes es costoso, ya que acceder a la memoria de otro nodo requiere enviar y recibir mensajes. En otras palabras habría que implementar toda una arquitectura de mensajes para acceder a las zonas de memoria de cada nodo, no es fácil.

- Construir una red OMEGA con 8 procesadores conectados a 8 bancos de memoria. Una vez construida la red queremos enrutar datos desde 010 a 010 y de la 110 a 000. ¿Existe la posibilidad de bloqueo?
  Calcular ancho de bisección, diámetro y coste de la topología.



De 010 a 010 y de 110 a 000 hay bloqueo:



- Sobre una topología completamente conectada con p = 8:
    a. Calcular el costes de la red
    b. Calcular el ancho de bisección

Network	Diameter	Bisection Width	Arc Connectivity	Cost (No. of links)
Completely-connected	1	$p^2/4$	p - 1	$p(p$ - 1)/2

- Considerar una red 2D con $\boldsymbol{p}$ procesadores, donde el patrón de comunicación que sigue el algoritmo implementado en cada nodo se comunica con sus nodos más cercanos (asumir link bidireccionales):
    a. Calcular el tiempo de comunicación para esta red asumiendo que cada mensaje contiene m palabras.
    b. Calcular el tiempo de comunicación de este sistema ahora considerando otro escenario distinto donde cada nodo se comunica con otro elegido de forma aleatoria.

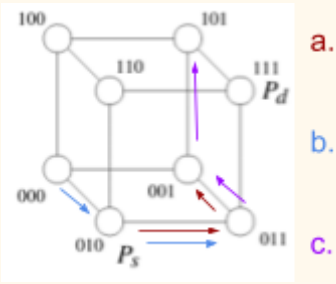
La red malla tendrá dimensiones $\sqrt{p} \times \sqrt{p}$ en la que cada nodo se comunica con su vecino, como los enlaces se utilizan sólo para una comunicación, el tiempo invertido en ello es ts + twm siendo m el número de palabras.

En el segundo caso la aleatoriedad implica que hay $\frac{p}{2}$ comunicaciones (o $\frac{p}{4}$ comunicaciones bidireccionales) que se producen en cualquier partición equivalente de la máquina (ya que el nodo con el que se comunica otro podría estar en cualquier mitad con la misma probabilidad). Sabemos que una malla 2-D tiene un ancho de bisección $\sqrt{p}$. De estas dos consideraciones podemos inferir que algunos enlaces tendrán que transportar $\frac{p/4}{\sqrt{p}} - \sqrt{\frac{p}{4}}$ mensajes asumiendo canales de comunicación bidireccional. Los mensajes, evidentemente van uno luego de otro, es decir se serializan sobre el enlace, si cada mensaje es de tamaño m el tiempo para esta operación será de al menos $t_s + t_s \times \sqrt{\frac{p}{4}}$. Este tiempo evidentemente no es el mismo que el modelo simplificado. Mirar en las transparencias el modelo de comunicación para redes congestionadas.

$$t_{com} = t_s + t_w \times m \times (\frac{número\ de\ mensajes}{ancho\ de\ bisección})$$

- Sobre una estructura de hipercubo de dimensión 3, muestra la ruta que seguirán los siguientes mensajes:
  a. 010 – 001
  b. 000 – 111
  c. 011 – 101

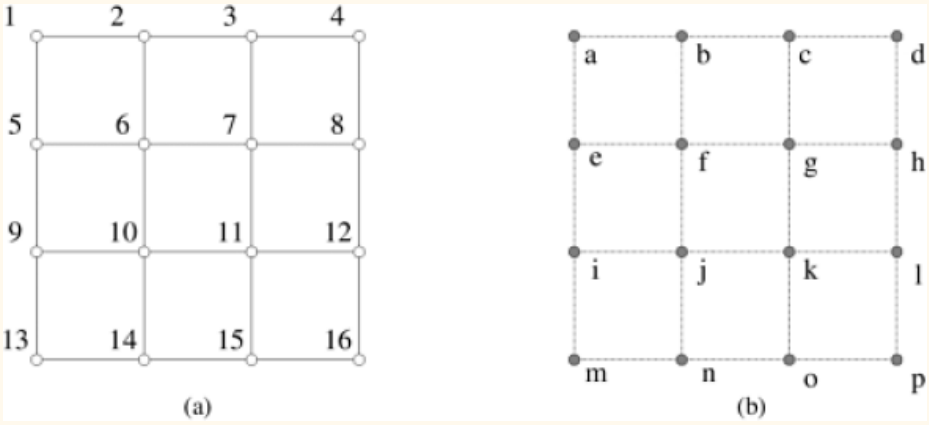
Aplicar el algoritmo E cube:

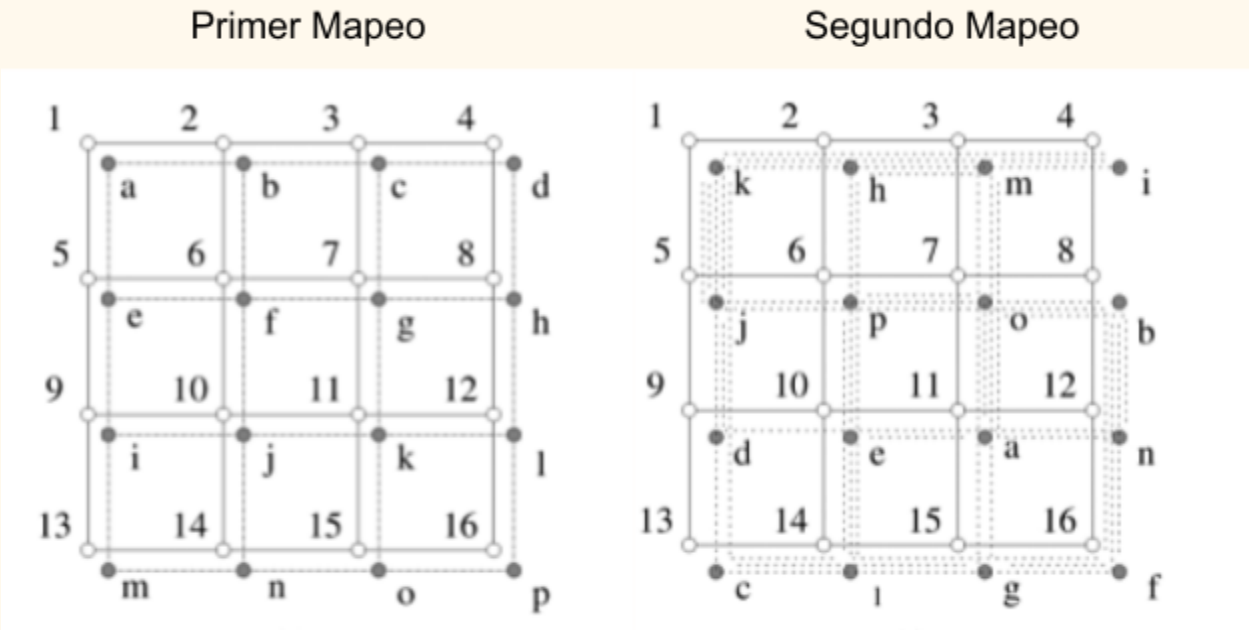


- Cuál es la diferencia entre un algoritmo de enrutado determinista y otro que no lo es (adaptativo).

Está en las transparencias

- Considera el escenario mostrado en la figura. Sobre una red de 16 nodos de procesamiento se quiere mapear un algoritmo que utiliza 16 procesos (a-p):
  a. Mapea los procesos sobre la red evitando congestión.
  b. Si el mapeo se hace de forma aleatoria, calcula la congestión máxima que podría darse sobre la red.





Es fácil ver que en este escenario un enlace podría transportar hasta 6 canales de comunicación por lo que esa sería la máxima congestión.

- Embeber una malla 4x4 en un hipercubo.
    a. Congestión
    b. Dilatación

Consider a $p \times q \times r$ mesh being embedded into a 2^d processor hypercube. Assume that $p = 2^x$, $q = 2^y$, and $r = 2^z$. Furthermore, since $p \times q \times r = 2^d$, $x + y + z = d$.
The embedding of the mesh can be performed as follows: Map processor (i, j, k) in the mesh to processor

Está en las transparencias

- Embeber una malla 2x4 en un hipercubo.
    a. Congestión
    b. Dilatación

Está en las transparencias

- Supongamos que se dispone de un número m de funciones polinomiales de grado n, de tal forma que la i-ésima función polinomial se puede representar como:

$$f'(x).\ i=0,1\ .....\ m\text{-}1$$

Se desea evaluar todas las funciones sobre un valor real $x = b$ y obtener el valor mínimo, es decir, obtener un valor y tal que:

$$V= mM\ \{f(x)\}\ para\ i(0,\ m\text{-}1\}$$

Para ello, los coeficientes de cada una de las funciones polinomiales se guardan en una matriz A de tamaño $m\ x\ (n+1)$ con la forma:

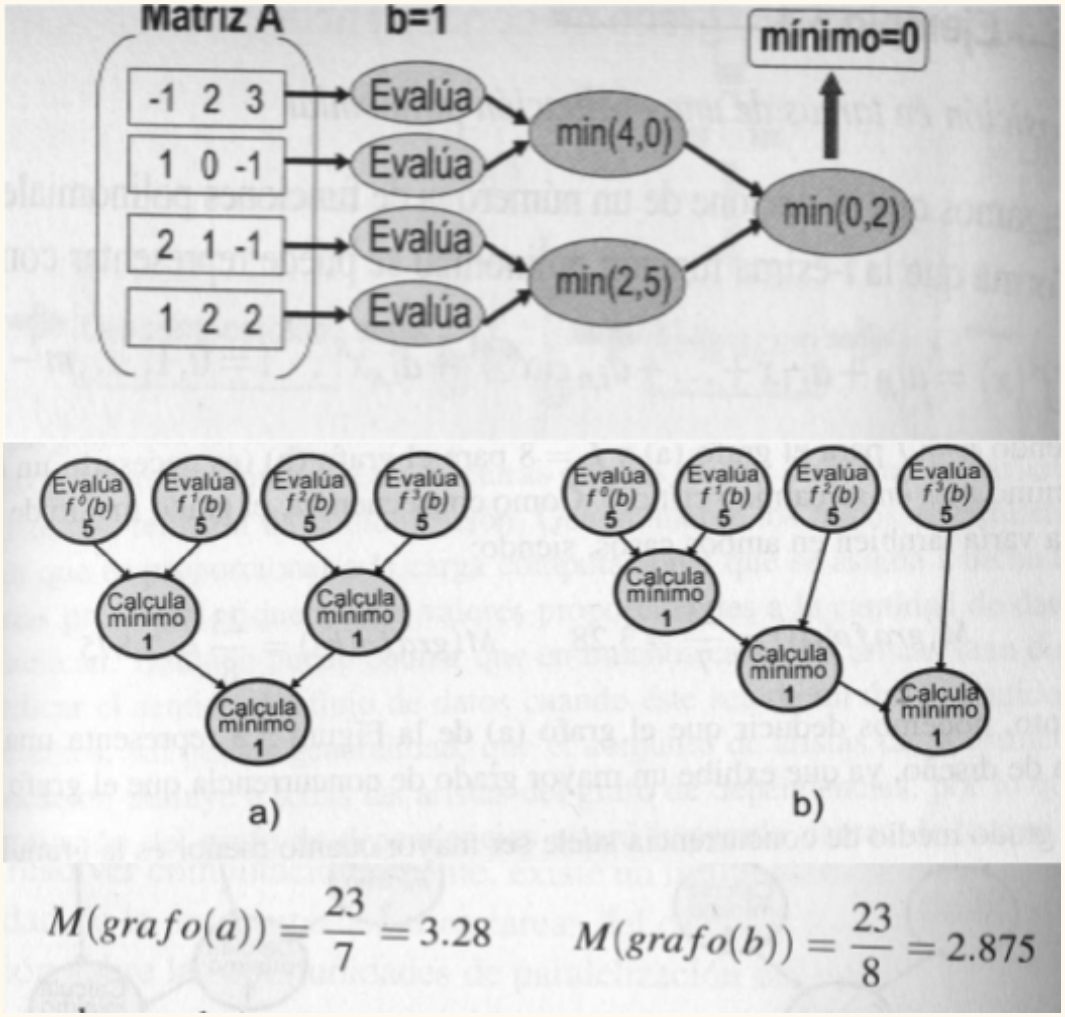
$$A = (a_{0,0}\ a_{0,1}\ ...\ a_{0,n};\ ...\ ;\ a_{m-1,0}\ a_{m-1,1}\ ...\ a_{m-1,n}\ )$$

a. Plantear la descomposición de tareas para el problema de la evaluación de polinomios.

b. Desarrollar grafo de dependencias para 4 polinomios de grado 2:

i. Máximo grado de concurrencia

ii. Grado medio de concurrencia



- Realiza la descomposición centrada en los datos de entrada para la realización del producto escalar de vectores:

$$X = (x_0\ x_1\ \ldots\ x_{n-1})\ y\ \ (y_0\ y_1\ \ldots\ y_{n-1})$$
$$x \cdot y = (x_0y_0 + x_1y_1 + \ldots + x_{n-1}y_{n-1})$$

Hay que centrarse necesariamente en los datos de entrada, ya que los datos de salida están formados por un único escalar, sería razonable asignar el mismo número de elementos de x y de y a cada tarea, podríamos pensar en p tareas, cada una gestionando un bloque de los vectores y realizando la operación:

$$d_i = \sum_{j=i\frac{n}{p}}^{(i+1)\frac{n}{p}-1} x_j y_j$$

Donde $\frac{n}{p}$ es el número de elementos que gestiona cada tarea, una vez obtenidas estas sumas parciales, las podemos consolidar con un enfoque recursivo igual que en el problema anterior.

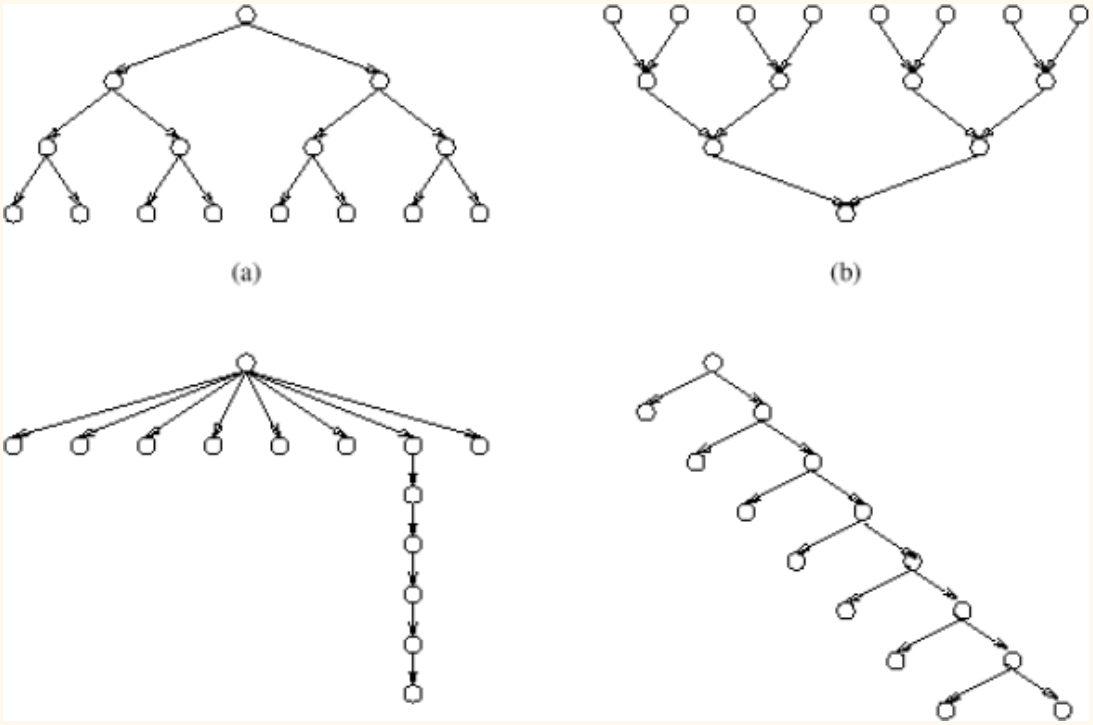
- Dado un arreglo A de longitud N, encontrar una descomposición recursiva de tareas que encuentre el mínimo elemento del arreglo. Comentar brevemente cómo funcionaría el algoritmo paralelo.

```
1.    procedure SERIAL_MIN (A, n)
2.    begin
3.        min = A[0];
4.    for i := 1 to n - 1 do
5.        if (A[i] < min) min := A[i];
6.    End for;
7.    return min;
8.    end SERIAL_MIN
```

Un programa en serie para encontrar el mínimo en una matriz de números A de longitud n.

```
1.   procedure RECURSIVE_MIN (A, n)
2.   begin
3.   if (n = 1) then
4.       min := A[0];
5.   else
6.       lmin := RECURSIVE_MIN (A, n/2);
7.       rmin := RECURSIVE_MIN (&(A[n/2]), n - n/2);
8.       if (lmin < rmin) then
9.           min := lmin;
10.      else
11.          min := rmin;
12.      endelse;
13.  endelse;
14.  return min;
15.  end RECURSIVE_MIN
```

- Para los grafos de dependencia de tareas de la Figura siguiente, determinar:
    a. Máximo grado de concurrencia.
    b. Longitud del camino crítico.
    c. Máximo Speed-Up alcanzable sobre un sistema monoprocesador si un número arbitrariamente grande de procesos están disponibles.
    d. Mínimo número de procesos para obtener el máximo speed-up.



Asumimos un peso de 1 en cada nodo:

1.- (a) 8, (b) 8, (c) 8, (d) 8

2.- (a) 4, (b) 4, (c) 7, (d) 8

3.- (a) 15/4, (b) 15/4, (c) 2, (d) 15/8

4.- (a) 15/4, (b) 15/4, (c) 8, (d) 15/8

- Supongamos que debemos evaluar la expresión:

$$x = \frac{a * b + c}{a * b - d}$$

  Genera un programa capaz de ejecutarse en un computador paralelo mostrando el grafo de dependencia de datos para la expresión anterior.

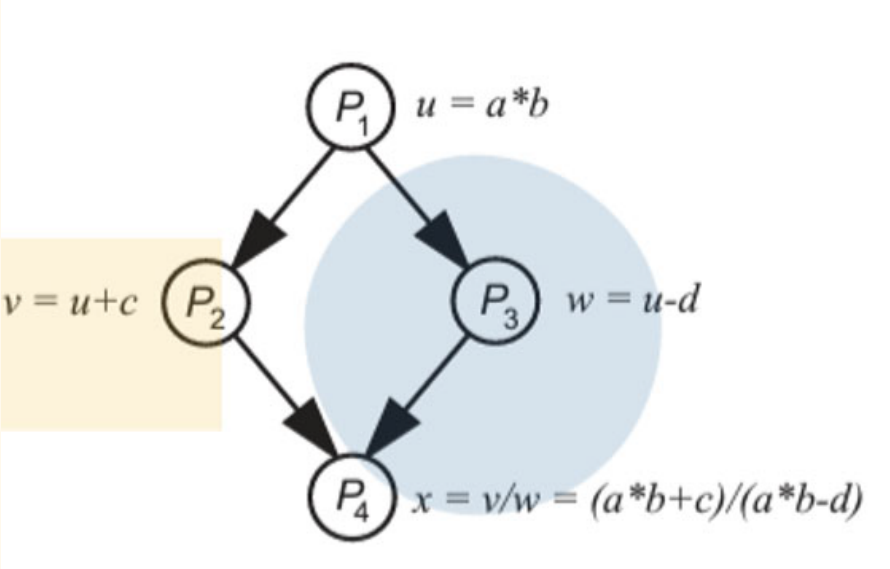
P1 = $u = a * b$

P2 = $v = u + c$

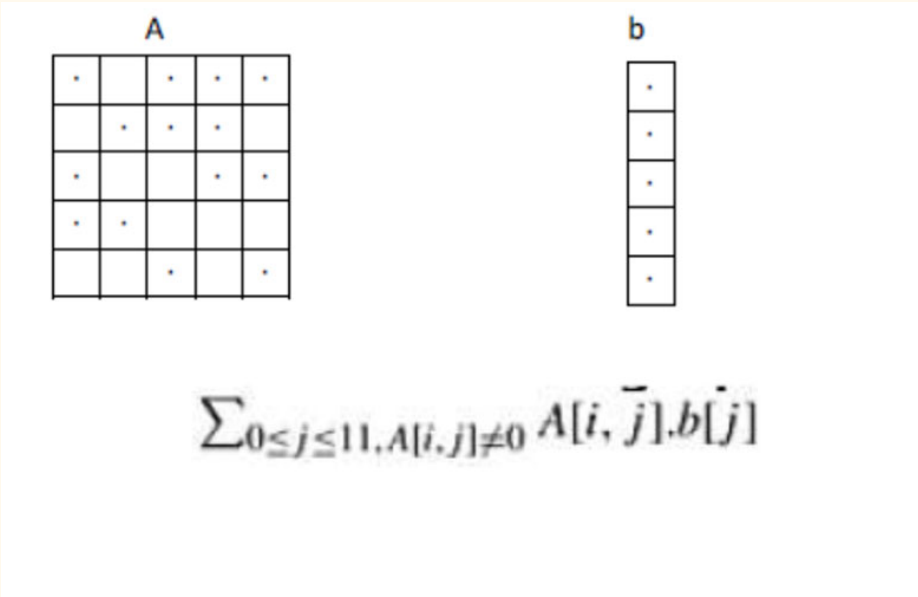
P3 = $w = u - d$

P4 = $\frac{v}{w}$

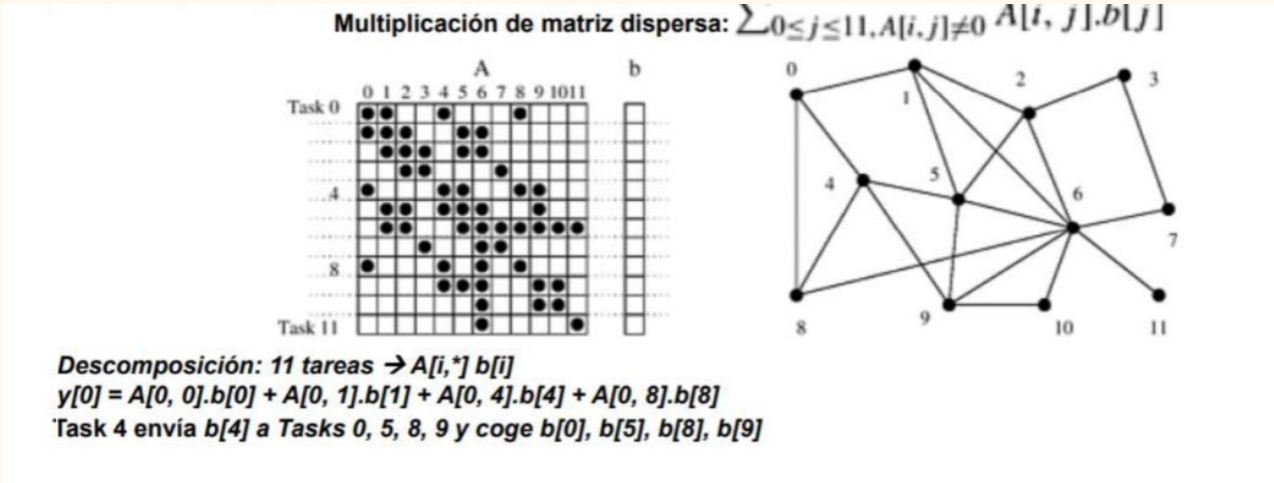
Considerando la dependencia de tareas tendríamos:



- Diseñar el grafo de dependencia e interacción para una matriz dispersa:



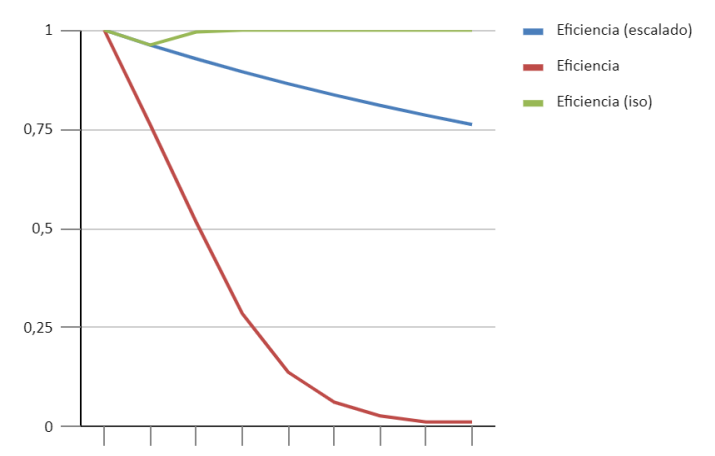
MIRAR el ejemplo de la Transparencia, es exactamente igual sólo cambian los datos que comparten las tareas, cada tarea procesa una fila y el correspondiente valor del vector b.



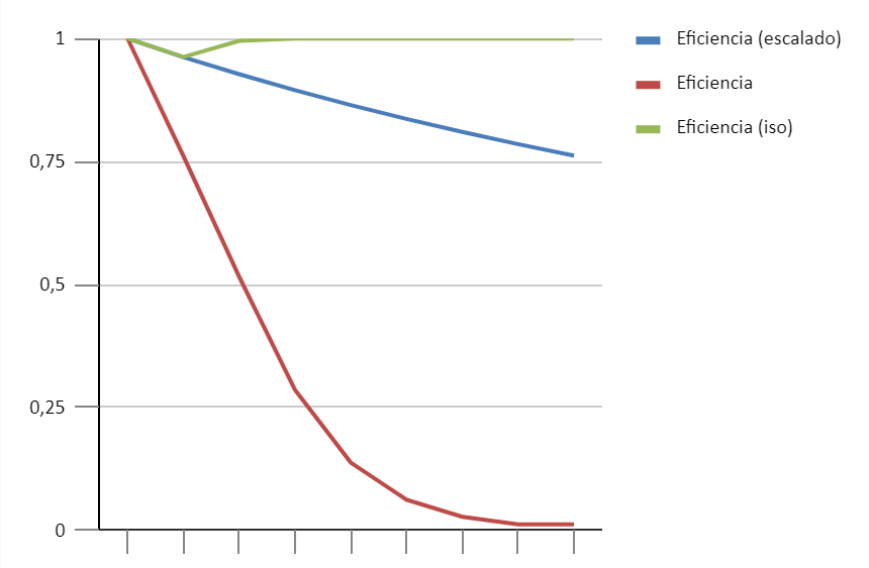
- El Speed-up escalado se define como el Speed-up obtenido cuando el tamaño del problema se incrementa linealmente con el número de procesadores.

    a. Para el problema de sumar n números en p procesadores, dibujar la curva del Speed-up escalado, asumiendo que el problema base para $p = 1$ consiste en sumar 256 números. Usar $p = 1$, 4, 16, 64 y 256. Asumir que se necesitan 10 unidades de tiempo para comunicar un número entre dos procesadores, y que la suma de dos números conlleva 1 unidad de tiempo.

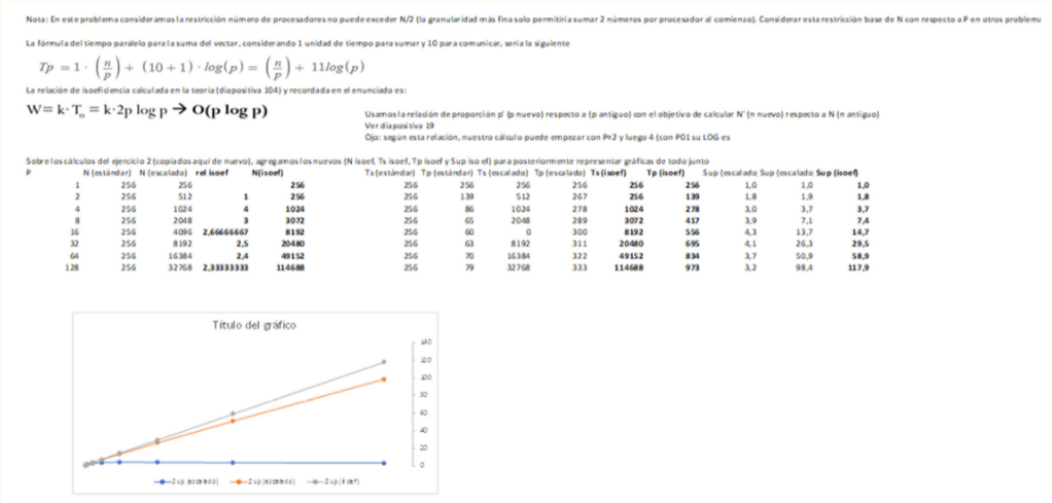
    b. Dibujar la curva del Speed-up estándar y compararla con la generada en el apartado anterior.



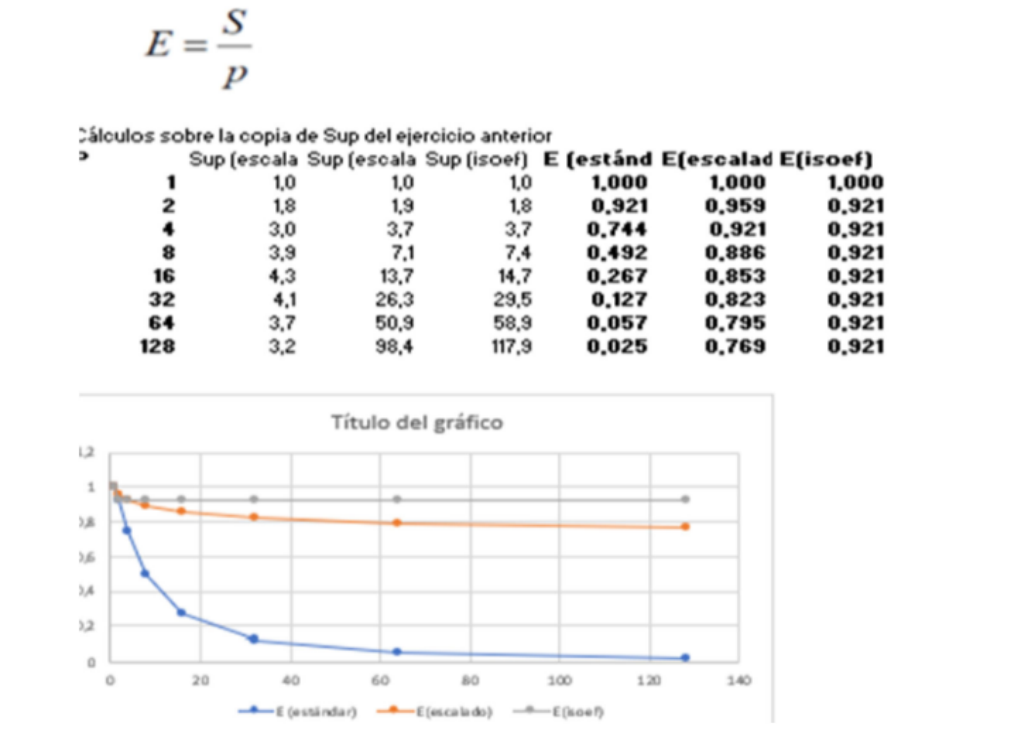
Procesadores	Eficiencia (escalado)	Eficiencia (std)	Eficiencia (iso)
1	1	1	1
2	0,962406015	0,7619047619	0,962406015
4	0,9275362319	0,5161290323	0,9951409135
8	0,8951048951	0,2857142857	0,9997965909
16	0,8648648649	0,1379310345	0,9999968211
32	0,8366013072	0,0625	0,9999999801
64	0,8101265823	0,02777777778	0,9999999999
128	0,7852760736	0,01234567901	1
256	0,7619047619	0,01234567901	1



- Partiendo del problema anterior, dibujar otra gráfica de Speed-up en el que el tamaño del problema aumente en concordancia con la función de isoeficiencia, que para este caso es *O(p·log p)*.



- Dado un problema cuyo $Tp = (n/p - 1) + 11 \cdot log\ p$, para valores de $p = 1$, 4, 16, 64, 256, 1024, 4096. ¿Cuál es el problema más grande que se puede solucionar si el tiempo total de ejecución no puede exceder las 512 unidades de tiempo? En general, ¿es posible solucionar un problema arbitrariamente grande en una cantidad fija de tiempo, suponiendo que se dispone de un número ilimitado de procesadores?



$$E = \frac{S}{p}$$

Cálculos sobre la copia de Sup del ejercicio anterior

	Sup (escala	Sup (escala	Sup (isoef)	E (estánd	E(escalad	E(isoef)
1	1,0	1,0	1,0	**1,000**	**1,000**	**1,000**
2	1,8	1,9	1,8	**0,921**	**0,959**	**0,921**
4	3,0	3,7	3,7	**0,744**	**0,921**	**0,921**
8	3,9	7,1	7,4	**0,492**	**0,886**	**0,921**
16	4,3	13,7	14,7	**0,267**	**0,853**	**0,921**
32	4,1	26,3	29,5	**0,127**	**0,823**	**0,921**
64	3,7	50,9	58,9	**0,057**	**0,795**	**0,921**
128	3,2	98,4	117,9	**0,025**	**0,769**	**0,921**

- El problema de sumar n números con p procesadores (subescalado), tiene una relación entre n y p, respecto a la Eficiencia, que se puede representar mediante la tabla:

n	$p = 1$	$p = 4$	$p = 8$	$p = 16$	$p = 32$
64	1.0	0.80	0.57	0.33	0.17
192	1.0	0.92	0.80	0.60	0.38
320	1.0	0.95	0.87	0.71	0.50

512	1.0	0.97	0.91	0.80	0.62

a. Calcular la función de isoeficiencia.

b. Calcular la expresión asociada a *To*.

$$Tp(n,p) = \left(\frac{n}{p} - 1\right) + 11 \cdot \log_2(p)$$

Para contestar a la primera pregunta, debemos relacionar Tp<=512 y calcular el umbral la "N" entera mayor tal que se cumpla la condición. Este cálculo se debe iterar para cada "p" instanciado a un valor del enunciado ("1", "4", ...)

$$512 \geq \left(\frac{n}{p} - 1\right) + 11 \cdot \log_2(p)$$

P=1 $\quad 512 \geq \left(\frac{n}{1} - 1\right) + 11 \cdot \log_2(1) \qquad n \leq 513$

p=4 $\quad 512 \geq \left(\frac{n}{4} - 1\right) + 11 \cdot \log_2(4) \qquad n \leq 1964$

...

Para la segunda pregunta, si consideramos una restricción hipótesis n <=p, el problema consistiría en ver el tiempo mínimo de ejecución

$$\frac{d}{dp}\left(\left(\frac{n}{p} - 1\right) + 11 \cdot \log_2(p) \right)$$

$$\frac{d}{dp}\left(\left(\frac{n}{p} - 1\right) + 11\log_2(p)\right) = \frac{11}{\ln(2)p} - \frac{n}{p^2}$$

$$\text{resolver para } p,\ \frac{11}{\ln(2)p} - \frac{n}{p^2} = 0$$

$$p = \frac{\ln(2)n}{11} \qquad n = \frac{11p}{\ln(2)}$$

$$Tp^{min} = \left(\frac{n}{\left(\frac{\ln(2)n}{11}\right)} - 1\right) + 11 \cdot \log_2\left(\frac{\ln(2)n}{11}\right). \quad \frac{11}{\ln(2)} - 1 + 11\left(\log_2(\ln(2)n) - \log_2(11)\right)$$

- El tiempo de ejecución paralelo de una implementación del algoritmo FFT (Transformada Rápida de Fourier) con p procesadores viene dado por $Tp = (n/p) \cdot \log n + 10(n/p) \cdot \log p$, para una secuencia de entrada de longitud n. El máximo número de procesadores que se pueden utilizar en este algoritmo es n. ¿Cuál es el tiempo mínimo en el que se puede ejecutar este algoritmo? ¿Cuál es el número de procesadores que producen este tiempo de ejecución?

- Suponer una plataforma paralela, que, para un cierto algoritmo, ofrece la

siguiente eficiencia:

$$E = \frac{1}{1+\frac{(p\times n^{1/2})+\alpha}{n}}$$

El tamaño del problema, W, se mide por el número de tareas, n, y es una constante.

a. Determinar el valor de la función To.

b. Determinar cuál debería ser el orden de crecimiento del tamaño del problema, en función del número de procesadores p, si se quiere mantener la Eficiencia constante.

—

Preguntas de Exámenes - Parcial 1 (21-22/22-23)

Motivaciones de la Computación Paralela y Aplicaciones Principales Actuales

Tenemos principalmente 3 motivaciones que dan lugar a la computación paralela; siendo estos:

- la ley de Moore, que nos dice que el número de transistores se duplicará cada 2 años;
- la escala de Dennard, que nos dice que mientras el tamaño de los transistores se reduce, su potencia se mantiene;
- y el inicio de la "era multicore", marcado por el 2005.

Por otro lado, las aplicaciones que tiene la computación paralela hoy en día son cosas como: la transición vectorial MPP, la transición Terascale-Petascale y la transición Petascale-Exascale; dando lugar esta última a los supercomputadores Petascale-Exascale.

Define: Exclusión Mútua, Interbloqueo y Semáforo, ¿cómo asiste el Sistema Operativo a la Computación de Alto Rendimiento?

- Exclusión Mútua → Actividad que realiza el SO para evitar que 2 o más procesos ingresen al mismo tiempo a la misma área de datos o recurso compartido.
- Interbloqueo → 2 procesos intentando acceder a la vez al proceso que tiene bloqueada su contraparte, no liberando el recurso en posesión hasta que el otro la libere.
- Semáforo → Estructura que cuenta con las operaciones wait() y signal() qué obliga a los procesos a esperar o actuar y redirige el flujo en el que los procesos acceden a los recursos.

El SO puede ayudar a la Computación de Alto Rendimiento mediante el uso de funciones y estructuras de datos a disposición del programador; siendo estos semáforos.

Define: Transformación Topológica, Congestión, Dilatación y Expansión

- Transformación Topológica → es una correspondencia 1 a 1 entre las partes de una figura con respecto a otra en la que la conexión entre dichas partes se mantienen. Un ejemplo sería convertir una taza en un *toro* (la figura geométrica no el animal).
- Congestión → es el máximo número de enlaces de la topología inicial mapeados en un único enlace de la topología final.
- Dilatación → es el máximo número de enlaces consecutivos de la topología final sobre los que se mapea 1 enlace de la topología inicial.

- Expansión → es la relación entre el número de redes a usar en ambas topologías.

Fases de la Generación de un Algoritmo Paralelo partiendo de uno Secuencial, ¿cómo se puede obtener un rendimiento mejorado?

En primer lugar, debemos hacer una descomposición de tareas; es decir, dividir un problema computacional en subproblemas para su posible ejecución concurrente. Esta descomposición tendrá asociada una granularidad (número y tamaño de las tareas obtenidas), ya sea esta fina (muchas tareas específicas) o gruesa (pocas tareas genéricas).

Para tener un mejor rendimiento (un mejor Speed-Up) debemos maximizar la concurrencia y minimizar el coste de comunicación (siendo la concurrencia el número medio de tareas a ejecutar simultáneamente). Después de esto, asignamos (o mapeamos) los procesos; para esto buscamos balancear el maximizar la concurrencia y minimizar la interacción entre procesos; preparando así los procesos para su ejecución.

Considérese un algoritmo que calcula el valor mínimo de una matriz tridimensional de longitud lado N y devuelve, además de dicho valor, la coordenada (x, y, z):

a. Plantear la descomposición de tareas para el problema.

b. Para una secuencia de N = 32 bits para P = 4 y P = 8. decir el Máximo grado de concurrencia y Grado medio de concurrencia alcanzables.

c. Justifique cuál podría ser una buena topología sobre la que hacer la asignación de esta descomposición y justifique la respuesta.

En el algoritmo que se muestra en el algoritmo siguiente, suponga una Descomposición tal que la ejecución de la Línea 7 es una tarea:

```
procedure FFT_like_pattern (A,n)
begin
    m := log2n;
    for j := 0 to m - 1 do
        k := 2^j;
        for i := 0 to n - 1 do
            A[ i ] := A[ i ] + A[ i XOR 2^j ];
    endfor
end FFT_like_pattern
```

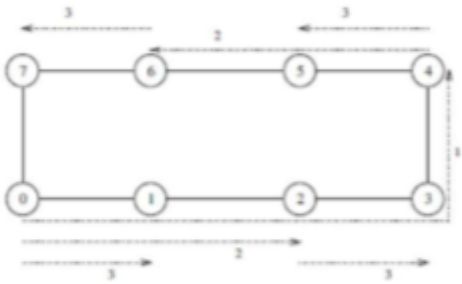
a. Dibujar el grafo de dependencia de tareas con su interacción
b. Si N = 16, idear un buen mapeo para 16 procesos
c. Si N = 8, idear un buen mapeo para 8 procesos

—

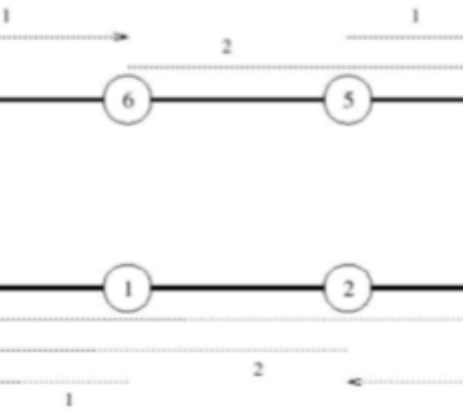
Preguntas de Exámenes - Parcial 2 (22-23)

Uso de Recursive Doubling para Red Anillo.

- Comunicación Secuencial:
  - **p-1** mensajes.
  - Cuello de botella nodo fuente
  - Desaprovechamiento de la red: una única conexión entre un par de nodos al mismo tiempo.
- **Recursive doubling:**
  - **log p** mensajes (etapas).
  - Los nodos destino en una etapa, se convierten en fuente en las siguientes.
  - Si el nodo 0 mandase mensaje al nodo 1 en el primer paso y después los nodos 0 y 1 intentaran mandar sendos mensajes a los nodos 2 y 3 respectivamente durante el segundo paso, el link entre el nodo 1 y 2 resultaría congestionado.



- El mismo esquema de comunicación pero a la inversa que en la difusión, para evitar congestión de los links.
  - 3 etapas.



Define dispersión y agrupamiento. Muestra la diferencia entre dispersión y difusión uno-a-todos.

- Dispersión: Varios datos almacenados en un único nodo, se reparten entre distintos procesadores (un dato por procesador).
- Agrupamiento: Varios datos almacenados en distintos procesadores, se almacenan simultáneamente en un único nodo (sin combinarse).

La diferencia entre la dispersión y la difusión uno-a-todos, es que en el caso de la segunda solo hay 1 nodo que hace de fuente y en la primera todos los nodos de la red hacen de fuente.

Define qué es el SpeedUp con tus propias palabras.

- **Speedup**: ganancia de rendimiento de la ejecución paralela, con respecto a la secuencial.

$$S = T_s / T_p \text{ (Límite teórico} = p)$$

¿Es posible obtener Speed-up mayor que *p*? (superlineales)

Dada la siguiente tabla:

Algorithm	A1	A2	A3	A4
p	n^2	$\log n$	n	$\sqrt{n}$
T_P	1	n	$\sqrt{n}$	$\sqrt{n}\log n$
S	$n\log n$	$\log n$	$\sqrt{n}\log n$	$\sqrt{n}$
E	$\frac{\log n}{n}$	1	$\frac{\log n}{\sqrt{n}}$	1
pT_P	n^2	$n\log n$	$n^{1.5}$	$n\log n$

Identificar el algoritmo (o algoritmo) con mejor rendimiento.

Análisis Asintótico:

Si la métrica es la velocidad, el algoritmo Al es el mejor, seguido de A3, A4 y A2 (en orden creciente de T_n).

En términos de eficiencia, A2 y A4 son los mejores, seguidos de A3 y Al.

En términos de costo, los algoritmos A2 y A4 tienen un costo óptimo, Al y A3 no lo son.

¡Es importante identificar los objetivos del análisis y utilizar métricas apropiadas!

Teniendo en cuenta todos estos datos, podemos decir que el algoritmo con mejor rendimiento es A4 debido al hecho de que usa una menor cantidad de procesadores así como la mejor eficiencia.

Definir que es un RTOS y compararlo con un SO Monoprocesador. Nombrar y definir los grados de cumplimiento de un Sistema de Tiempo Real.

asd

Tipos de Planificación con Prioridades en RTOS.

La asignación de mayor prioridad a las tareas de menor período (rate monotonic scheduling) es óptima para el modelo de tareas simple(tareas periódicas, independientes, con plazos iguales a los períodos):

- Condición de garantía de los plazos basada en la utilización mínima garantizada.
- Análisis del tiempo de respuesta.
- Cálculo de la interferencia máxima.
- Iteración lineal.
- Tiempo de cómputo.

Calcular, sabiendo que el tamaño $W = n$, el Tiempo Paralelo mínimo a partir de:

$$T_o = 2n + 10p\log_2 p$$

y

$$T_s = n$$

$T_o = pT_p - T_s \rightarrow (T_o + T_s) / p = T_p$

$p^{min} = T_p \, d/dp$

$T^{min}_p = T_p \, (p^{min})$

Dada una convolución con una imágen nxn y un kernel 3x3; la primera fase tarda en cumplirse $2(t_s + t_w n)$ y la segunda fase $9t_c n^2/p$. Entonces, calcular: Tp, SpeedUp y Eficacia.

$Tp = T1 + T2 = 2t_s + 2t_w n + (9t_c n2) / p = (2pt_s + 2pt_w n + 9t_c n^2) / p$

$Ts = n/1$

$S = T_s/T_p$

$E = S/p$