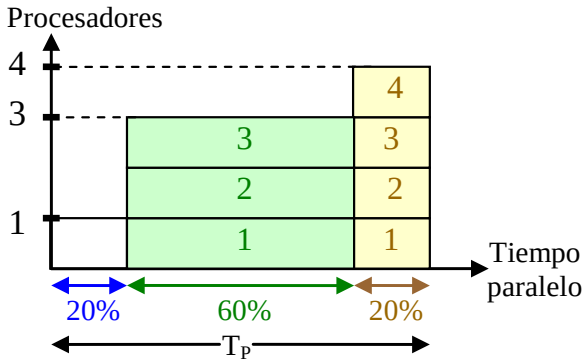
Aviso: *“Con motivo de la suspensión temporal de la actividad docente presencial en la UGR, se informa de las condiciones de uso de la aplicación de videoconferencia que a continuación se va a utilizar:*
1. La sesión podría ser grabada con el objeto de facilitar al estudiantado, con posterioridad, el contenido de la sesión docente.
2. Se recomienda a los asistentes que desactiven e inhabiliten la cámara de su dispositivo si no desean ser visualizados por el resto de participantes.
3. Queda prohibida la captación y/o grabación de la sesión, así como su reproducción o difusión, en todo o en parte, sea cual sea el medio o dispositivo utilizado. Cualquier actuación indebida comportará una vulneración de la normativa vigente, pudiendo derivarse las pertinentes responsabilidades legales.”

ARQUITECTURA DE COMPUTADORES
TEMA 2 Solución de problemas

Problema 1. Un programa tarda 40 s en ejecutarse en un multiprocesador. Durante un 20% de ese tiempo se ha ejecutado en cuatro procesadores (core); durante un 60%, en tres; y durante el 20% restante, en un procesador (consideramos que se ha distribuido la carga de trabajo por igual entre los procesadores que colaboran en la ejecución en cada momento, despreciamos sobrecarga). ¿Cuánto tiempo tardaría en ejecutarse el programa en un único procesador?¿Cuál es la ganancia en velocidad obtenida con respecto al tiempo de ejecución secuencial?¿Y la eficiencia?



El tiempo paralelo es T_P=40 segundos

El tiempo secuencial T_s=2.8*40=112 segundos

La ganancia de velocidad es 2.8 y la eficiencia 0.7

El eje horizontal de la figura corresponde al tiempo de ejecución del programa paralelo T_P.
El eje vertical corresponde al número de procesadores que están procesando durante el correspondiente tiempo paralelo.
Así, la figura muestra que durante el 20% de T_P (0.2T_P) se utiliza un procesador, durante un 60% (0.6T_P) trabajan tres procesadores, y durante el 20% restante (0.2T_P) trabajan 4.

Es decir T_P=(0.2*T_P)+(0.6*T_P)+(0.2*T_P)= (0.2+0.6+0.2)*T_P

El tiempo secuencial, T_s es el que tardaría un único procesador en realizar todo el trabajo. Es decir:

T_s=(0.2*T_P)*1+(0.6*T_P)*3+(0.2*T_P)*4= (0.2+1.8+0.8)*T_P = 2.8*T_P

La ganancia de velocidad será S=T_s/T_P=2.8*T_P/T_P= 2.8

La eficiencia es E=S/ N_procesadores =2.8/4 = 0.7

ARQUITECTURA DE COMPUTADORES
TEMA 2 Solución de problemas

Problema 2 Un programa tarda 20 s en ejecutarse en un procesador P1, y requiere 30 s en otro procesador P2. Si se dispone de los dos procesadores para la ejecución del programa (despreciamos sobrecarga):
¿Qué tiempo tarda en ejecutarse el programa si la carga de trabajo se distribuye por igual entre los procesadores P1 y P2?
¿Qué distribución de carga entre los dos procesadores P1 y P2 permite el menor tiempo de ejecución utilizando los dos procesadores en paralelo ?
¿Cuál es este tiempo?

La carga de trabajo correspondiente al programa es W y los tiempos para el procesador 1 y 2 son, respectivamente T1=20s. y T2=30s.

La velocidad del procesador 1 es V1= W/T1 = W/20s

La velocidad del procesador 2 es V2= W/T2 = W/30s

Si la carga de trabajo se distribuye por igual entre los dos procesadores, la carga del procesador 1 es W/2=0.5*W y la del procesador 2 es también W/2=W*0.5

Se considera que las velocidades de los procesadores 1 y 2 son las que tenían en la situación de partida, por lo que tendríamos que

V1 = W/20s = (W/2)/T1(0.5) donde T1(0.5) es el tiempo que tardaría el procesador 1 con la carga W*0.5 y

V2 = W/30s = (W/2)/T2(0.5) donde T2(0.5) es el tiempo del procesador 2 con la carga W*0.5

El tiempo que tardaría el procesador 1 sería T1(0.5) = (W/2)/V1 = (W/2)/(W/20s) = 20s/2=10s.

El tiempo que tardaría el procesador 2 sería T2(0.5) = (W/2)/V2 = (W/2)/(W/30s) = 30s/2=15s.
Dado que el procesador más lento tiene que esperar al más rápido, el tiempo paralelo sería T_P=max(T1(0.5), T2(0.5))=max(10s., 15 s.)=15s.

ARQUITECTURA DE COMPUTADORES
TEMA 2 Solución de problemas

Problema 2 (cont.) ¿Qué distribución de carga entre los dos procesadores P1 y P2 permite el menor tiempo de ejecución utilizando los dos procesadores en paralelo ? ¿Cuál es este tiempo?

Para minimizar el tiempo de procesamiento hay que distribuir la carga de forma que la mayor parte recaiga en el procesador más rápido para que los dos procesadores terminen al mismo tiempo: se conseguiría así una distribución equilibrada de la carga (load balancing)

La carga de trabajo para el procesador 1 sería a*W, siendo a un número entre 0 y 1, y la del procesador 2 sería el resto de la carga, es decir (1-a)*W

Igual que en antes, las velocidades de los procesadores 1 y 2 son las que tenían en la situación de partida:

La velocidad del procesador 1 es V1= W/T1 = W/20s

y la del procesador 2 es V2= W/T2 = W/30s por lo que tendríamos que

V1 = W/20s = (a*W)/T1(a) donde T1(a) es el tiempo que tardaría el procesador 1 con la carga a*W y

V2 = W/30s = ((1-a)*W)/T2(1-a) donde T2(1-a) es el tiempo del procesador 2 con la carga (1-a)*W

El tiempo que tardaría el procesador 1 sería T1(a) = (a*W)/V1 = (a*W)/(W/20s) = a*20s

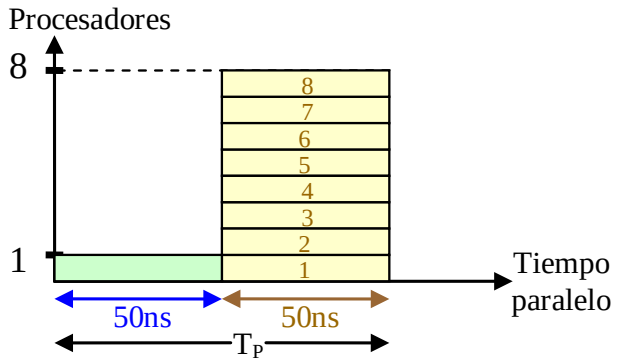
El tiempo que tardaría el procesador 2 sería T2(1-a) = ((1-a)*W)/V2 = ((1-a)*W)/(W/30s) = (1-a)*30s.

Como los dos procesadores han de terminar al mismo tiempo T1(a) = T2(1-a) → 20*a=30*(1-a) → 50*a=30 → a=3/5
Es decir, el procesador 1 recibiría 3/5 de la carga y el procesador 2, 2/5 de la carga.

Por tanto, el tiempo que tardaría el programa paralelo en ejecutarse sería T2(1-a) = T1(a) = a*20s=3/5*20 s. = 12s.

ARQUITECTURA DE COMPUTADORES
TEMA 2 Solución de problemas

Problema 3. ¿Cuál es fracción de código paralelo de un programa secuencial que, ejecutado en paralelo en 8 procesadores tarda un tiempo de 100 ns, durante 50ns utiliza un único procesador y durante otros 50 ns utiliza 8 procesadores (distribuyendo la carga de trabajo por igual entre los procesadores)?



El diagrama de tiempos y procesadores para el caso que se plantea se muestra en la figura

El tiempo secuencial del programa (tiempo que tardaría un solo procesador en ejecutar todo el programa) sería:

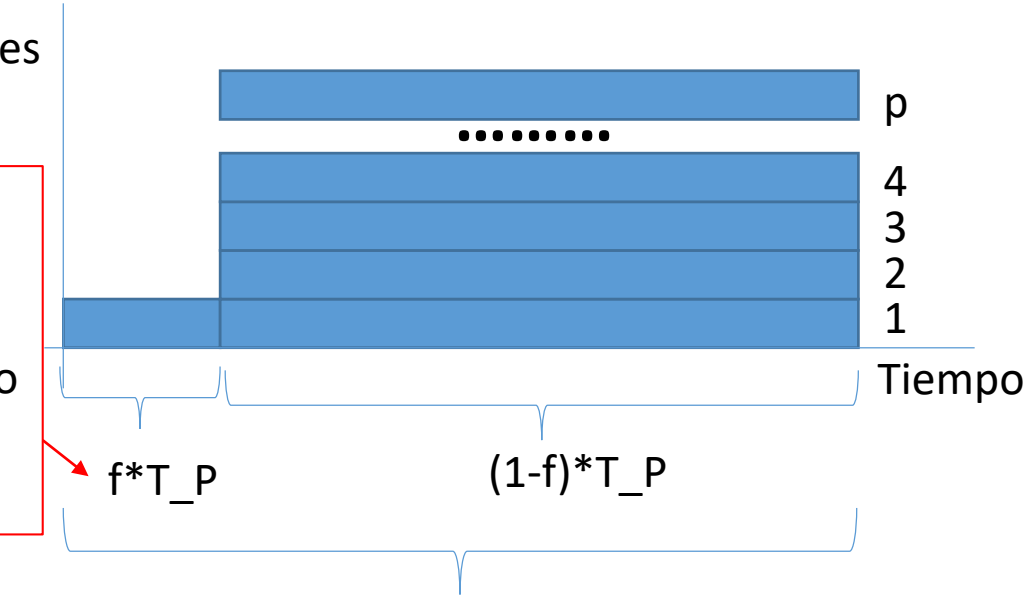
T_s = 50 ns (parte no paralelizable) * 1 + 50 ns (parte que se ha ejecutado en paralelo en 8 procesadores) * 8 = 450 ns

Por lo tanto la fracción de código NO paralelizable del código secuencial (f de la ley de Amdahl sería: 50 ns / 450 ns =1/9

La parte paralelizable del código secuencial que pide el problema es 1-f = 8/9

Por otro lado, la parte NO paralelizable del tiempo paralelo T_P (f en la ley de Gustafson) sería 50 ns / (50 ns + 50 ns) = 50ns /100ns = 1/2

Comparación Ley de Amdahl – Ley de Gustafson



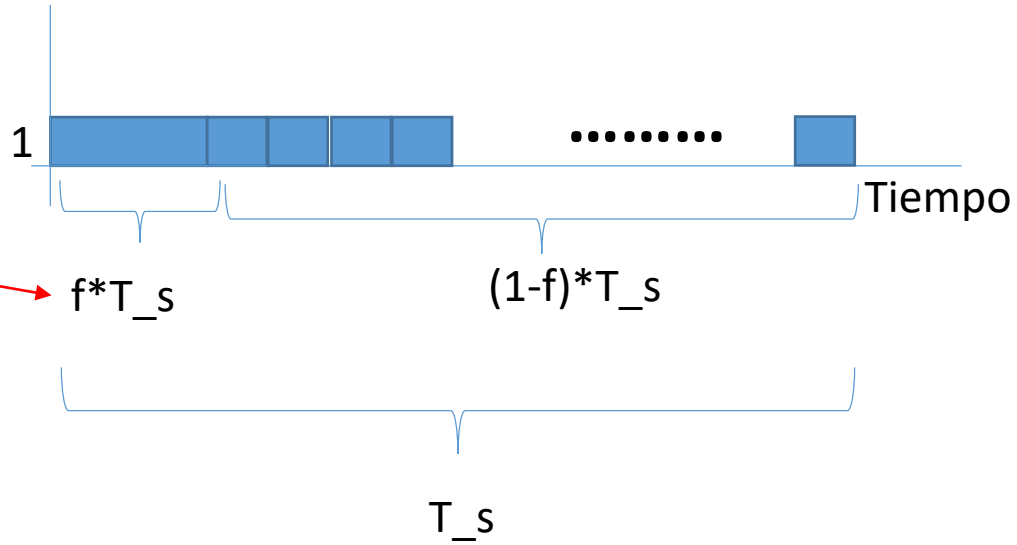
Parte no paralelizable del tiempo de ejecución paralelo con p procesadores

Tiempo que tarda un procesador en ejecutar el código paralelo de la izda.

$T_s = f*T_P + p*(1-f)*T_P$

$S = T_s/T_P =$
$= (f*T_P + p*(1-f)*T_P)/T_P = f+p*(1-f)$

Ley de Gustafson



Parte no paralelizable del tiempo de ejecución con 1 procesador

Ley de Amdahl

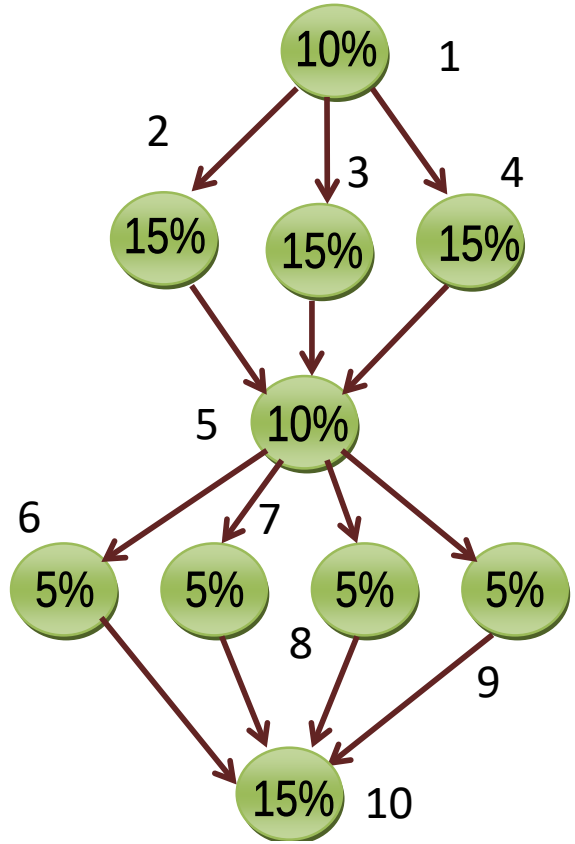
Tiempo que tardan p procesadores al paralelizar el código secuencial de la izda.

$T_P >= f*T_s + (1-f)*T_s/p$

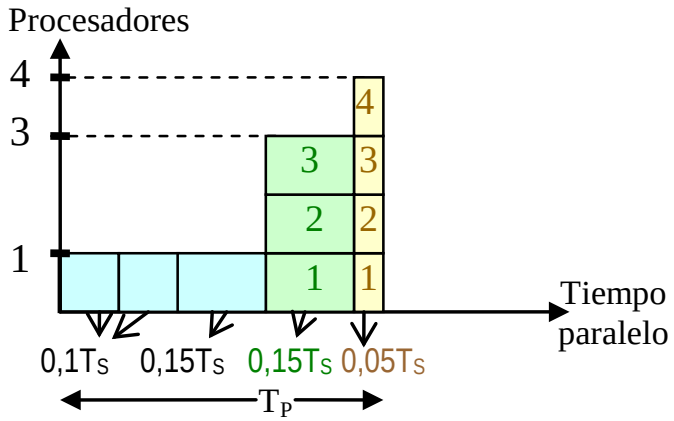
$S = T_s/T_P <=$
$<= T_s/(f*T_s + (1-f)*T_s/p) =$
$p/(p*f+(1-f) = p/(1+f*(p-1))$

ARQUITECTURA DE COMPUTADORES
TEMA 2 Solución de problemas

Problema 5 En la figura se presenta el grafo de dependencia entre tareas para una aplicación. La figura muestra la fracción del tiempo de ejecución secuencial que la aplicación tarda en ejecutar grupos de tareas del grafo. Suponiendo un tiempo de ejecución secuencial de 60 s, que las tareas no se pueden dividir en tareas de menor granularidad y que el tiempo de comunicación es despreciable, obtener el tiempo de ejecución en paralelo y la ganancia de velocidad en un computador con: (a) 4 procesadores; y (b) 2 procesadores



(a) Para 4 procesadores

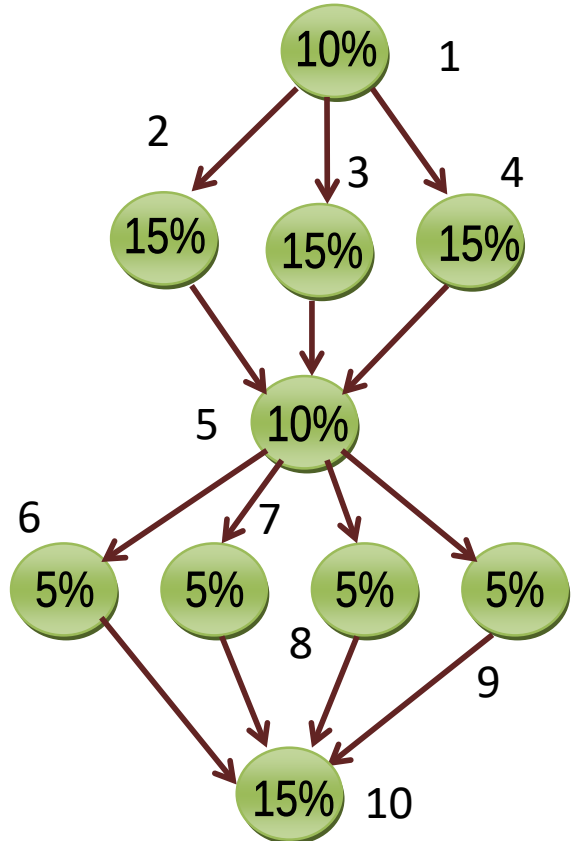


La asignación de procesadores a tareas debe hacerse teniendo en cuenta la disponibilidad de procesadores y la dependencia entre las tareas que muestra el diagrama de dependencias de la derecha donde aparecen numeradas las tareas (1, 2,…10).

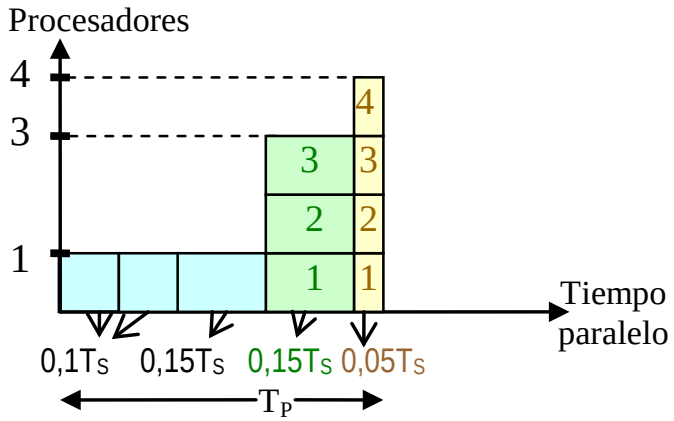
Asignación de tareas a procesadores: Para empezar, solo se puede ejecutar la tarea 1 que se asigna a un procesador y tardaría 0.1T_s. Hasta que no ha terminado 1, no pueden empezar las tareas 2, 3, y 4 que se asignarían a los procesadores P1, P2, P3 y juntas tardarían 0.15T_s dado que se ejecutan en paralelo. Después podría empezar la tarea 5 que se asignaría al procesador P1, por ejemplo dado que estaría libre y tardaría 0.1T_s. Hasta que no termina 5 no podrían empezar las siguientes 6, 7, 8, y 9 que se pueden asignar a los cuatro procesadores disponibles que las procesarían en paralelo y tardarían 0.05Ts en paralelo. Finalmente, cuando terminan estas tareas, podría empezar la tarea 10 en uno de los procesadores que quedarían libres y tardaría 0.15

ARQUITECTURA DE COMPUTADORES
TEMA 2 Solución de problemas

Problema 5 (cont.) En la figura se presenta el grafo de dependencia entre tareas para una aplicación. La figura muestra la fracción del tiempo de ejecución secuencial que la aplicación tarda en ejecutar grupos de tareas del grafo. Suponiendo un tiempo de ejecución secuencial de 60 s, que las tareas no se pueden dividir en tareas de menor granularidad y que el tiempo de comunicación es despreciable, obtener el tiempo de ejecución en paralelo y la ganancia de velocidad en un computador con: (a) 4 procesadores; y (b) 2 procesadores



(a) Para 4 procesadores



La asignación de procesadores a tareas debe hacerse teniendo en cuenta la disponibilidad de procesadores y la dependencia entre las tareas que muestra el diagrama de dependencias de la derecha donde aparecen numeradas las tareas (1, 2,…10).

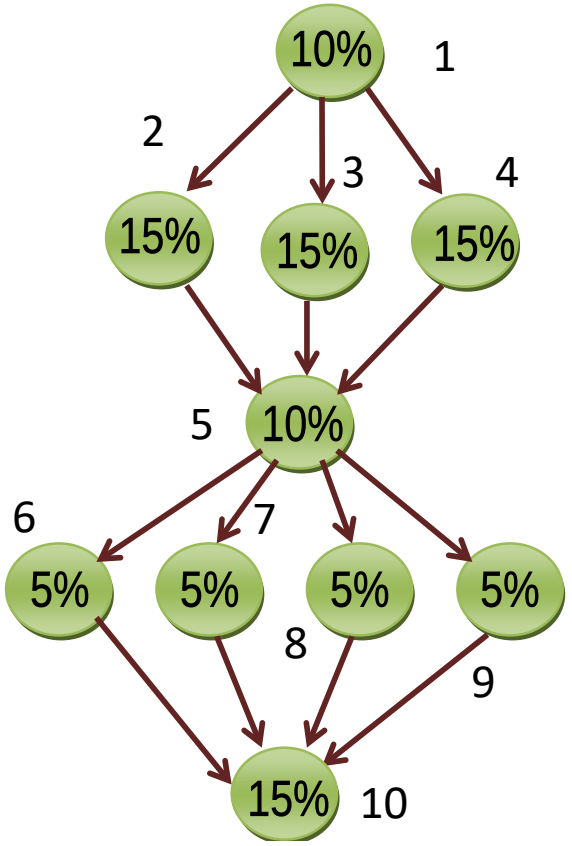
Por lo tanto, el tiempo paralelo sería (tal y como se muestra en la figura de la izquierda):

T_P= (0.1+0.1+0.15+0.15+0.05)*Ts = 0.55*Ts = 0.55*60= 33 s.

La ganancia de velocidad sería S(4)=Ts/T_P=Ts/0.55*Ts=1/0.55 = 1.82
La eficiencia es S(4)/4=1.82/4=0.455

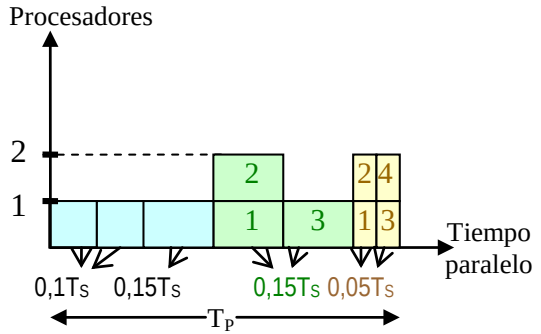
ARQUITECTURA DE COMPUTADORES
TEMA 2 Solución de problemas

Problema 5 (cont.) (b) Para 2 procesadores



Asignación de tareas a procesadores: Para empezar, solo se puede ejecutar la tarea 1 que se asigna a un procesador y tardaría 0.1Ts. Hasta que no ha terminado 1, no pueden empezar las tareas 2, 3, y 4. Pero solo dos de ellas se pueden ejecutar (por ejemplo 2 y 3) porque solo hay dos procesadores, y juntas tardarían 0.15Ts dado que se ejecutan en paralelo. Cuando terminan, a pesar de haber dos procesadores disponibles, solo podemos utilizar uno de ellos para la tarea 4 , porque hasta que esta no termine 4, no puede empezar la 5. Después podría realizarse la tarea 5 que se asignaría a uno de los procesadores que está libre, y tardaría 0.1Ts. Hasta que no termina 5 no podrían empezar las siguientes 6, 7, 8, y 9 que se pueden asignar, de dos en dos, a los dos procesadores disponibles que las procesarían en paralelo de dos en dos, y tardarían 2*0.05Ts. Finalmente, cuando terminan estas tareas, podría empezar la tarea 10 en uno de los procesadores que quedarían libres y tardaría 0.15

El tiempo paralelo sería (tal y como se muestra en la figura de la derecha):

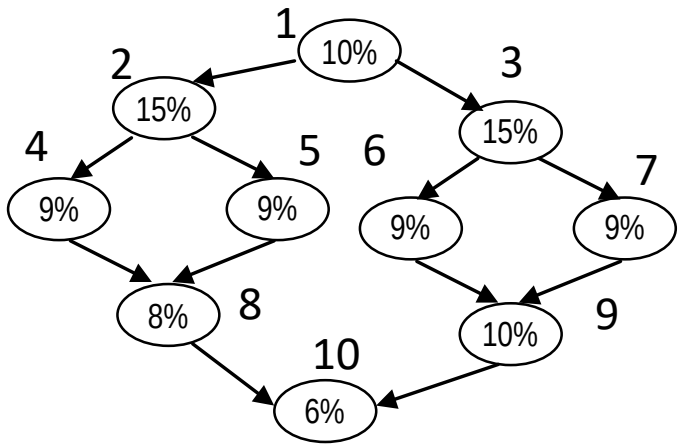


T_P= (0.1+0.1+0.15+0.15+0.15+0.05+0.05)*Ts = 0.75*Ts=0.75*60= 45 s.

La ganancia de velocidad sería S(2)=Ts/T_P=Ts/0.75*Ts=1/0.75 = 1.33
La eficiencia es S(2)/2=1.33/2=0.665. Es más eficiente utilizar 2 procesadores porque hay menos tiempo de procesador sin utilizar

ARQUITECTURA DE COMPUTADORES
TEMA 2 Solución de problemas

Problema 6. Un programa se ha conseguido dividir en 10 tareas. El orden de precedencia entre las tareas se muestra con el grafo dirigido siguiente. La ejecución de estas tareas en un procesador supone un tiempo de 2 sg. El 10% de ese tiempo es debido a la ejecución de la tarea 1; el 15% a la ejecución de la tarea 2; otro 15% a la ejecución de 3; cada tarea 4, 5, 6 o 7 supone el 9%; un 8% supone la tarea 8; la tarea 9 un 10%; por último, la tarea 10 supone un 6%. Se dispone de una arquitectura con 8 procesadores para ejecutar la aplicación. Consideramos que el tiempo de comunicación se puede despreciar. (a) ¿Qué tiempo tarda en ejecutarse el programa en paralelo? (b) ¿Qué ganancia en velocidad se obtiene con respecto a su ejecución secuencial?.

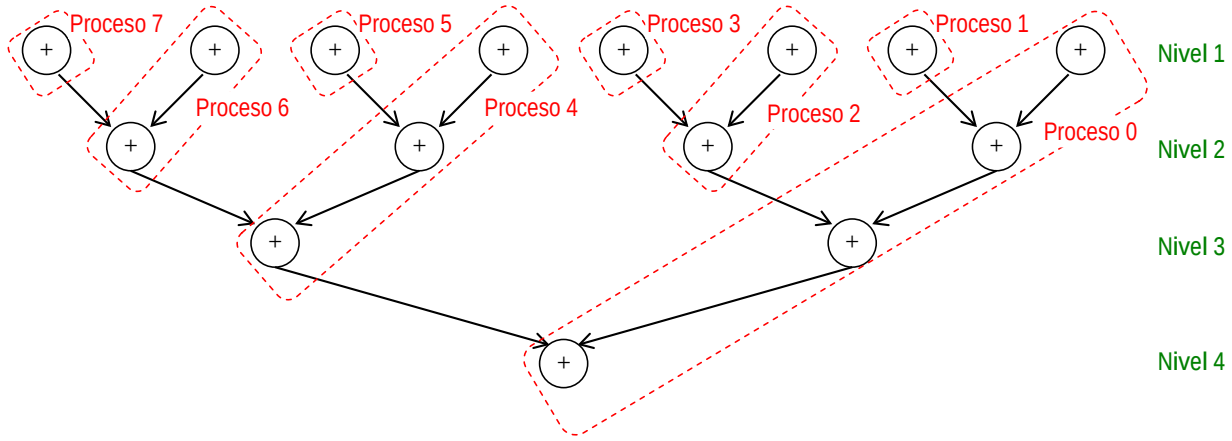


Asignación de Tareas a Procesadores: Partimos del grafo de dependencia de tareas de la izquierda y consideramos las dependencias y el tiempo de las tareas. Por otra parte tenemos ocho procesadores. Solo puede empezar a procesarse la tarea 1 que se asigna a un procesador. Después de 0.1Ts segundos, pueden empezar las tareas 2 y 3 que se pueden asignar a dos de los ocho procesadores que hay libres y se procesan en paralelo consumiendo 0.15Ts. Como terminan las dos tareas al mismo tiempo, se pueden empezar a ejecutar las tareas 4, 5, 6, y 7, que se pueden asignar a cuatro procesadores libres. Dado que se pueden ejecutar en paralelo y tardan lo mismo, las cuatro terminarían tras 0.09Ts segundos, y podrían empezar las tareas 8 y 9, que se podrían asignar a dos de los procesadores que estarían libres.
Las dos tareas, 8 y 9, se ejecutan en paralelo pero dado que la tarea 9 tarda más (0.1Ts frente a 0.08Ts), y la tarea 10 no podría empezar hasta que no terminen la 8 y la 9, el procesamiento paralelo de las tareas 8 y 9 tardaría, por tanto, 0.1Ts.
Una vez termine 9, se podrá empezar la tarea 10 asignando uno de los procesadores disponibles (los ocho estarían disponibles y tardaría 0.06Ts segundos.

Como se puede ver, no se podrían utilizar en paralelo más de cuatro de los ocho procesadores disponibles y el tiempo paralelo sería:
T_P=0.1+0.15+0.09+0.1+0.06)*Ts=0.5*Ts=0.5*2 s=1 s
S(8)=Ts/T_P=2/1=2 y la eficiencia sería S(8)/8=2/8=0.25 Es bastante baja pero hay que tener en cuenta que, como mucho, se utilizan 4 de los 8 procesadores.

ARQUITECTURA DE COMPUTADORES
TEMA 2 Solución de problemas

Problema 8 Supongamos que se va a ejecutar en paralelo la suma de n números en una arquitectura con p procesadores o cores (p y n potencias de dos) utilizando un grafo de dependencias en forma de árbol (divide y vencerás) para las tareas. Supongamos que se tarda una unidad de tiempo en realizar una suma.
(a) Dibujar el grafo de dependencias entre tareas para n 16 y p igual a 8. Hacer una asignación de tareas a procesos o hebras.
(b) Obtener el tiempo de cálculo paralelo para cualquier n y p con n>p.
(c) Obtener el tiempo comunicación del algoritmo suponiendo que las comunicaciones en un nivel del árbol se pueden realizar en paralelo en un número de unidades de tiempo igual al número de datos que recibe o envía un procesoen cada nivel del grafo de tareas (tenga en cuenta la asignación de tareas a procesos que ha considerado en el apartado a)).
(d) Suponiendo que el tiempo de sobrecarga es el tiempo de comunicación, obtener la ganancia en prestaciones.
(e) Obtener el número de procesadores para el que se obtiene la máxima ganancia con n números.



(a) Como hay 8 procesadores, inicialmente se distribuyen los n=16 números entre ellos. Corresponderían dos números por procesador, que cada uno sumaría, y se partiría de un número por procesador.
Como muestra la figura, tendríamos un número en cada proceso/hebra (o procesador)
Según aparece en la figura: en un primer paso, los procesos 1, 3, 5 y 7 envían sus números a los 0, 2, 4, y 6 en paralelo, y se suma cada número con el que hay en el procesador de destino.
Después el 2 y el 6 envían al 0 y al 4, su suma parcial que se suma con la disponible en el procesador de destino y, finalmente el procesador 4 manda su suma parcial al procesador 0, donde se suma con la suma parcial de este procesador. Se obtiene así la suma total en 3 pasos. Es decir, logaritmo en base 2 del número de procesadores, log2(p)

ARQUITECTURA DE COMPUTADORES
TEMA 2 Solución de problemas

Problema 8 (cont.)

(b) Como hay n números y p procesadores, inicialmente cada procesador recibe (n/p) números y realizará (n/p)-1 suma

Este paso se realiza en paralelo en los p procesadores y consumirá un tiempo T0=((n/p)-1)*t(suma), donde t(suma) es el tiempo que tarda un procesador en realizar una suma

Después, se produce cada uno de los pasos de comunicación y acumulación del número que se envía con la suma parcial que hay en el destino.

(c) El tiempo que tarda cada uno de estos pasos sería la suma de la comunicación más la acumulación. Es decir

Tcom_acc=t(com)+t(suma) y este paso se repite log2(p), por lo que la sincronización tardaría **log2(p)*(t(com)+t(sum))**

Por lo tanto, el tiempo total de procesamiento paralelo sería igual a la suma de todos estos tiempos: el segundo de estos tiempos sería el tiempo de sobrecarga u overhead.

T_P=T0+ log2(p)*(t(com)+t(sum))=((n/p)-1)*t(sum) + log2(p)*(t(com)+t(sum))

Si, para simplificar la expresión suponemos que **t(sum)=t(com)=t**
Entonces, **T_P=T0+ log2(p)*(t(com)+t(sum))=((n/p)-1)*t + 2*t*log2(p)=t*((n/p)-1)+2*log2(p)**

ARQUITECTURA DE COMPUTADORES
TEMA 2 Solución de problemas

Problema 8 (cont.)

(d) Partiendo de este tiempo de procesamiento paralelo

T_P=T0+ log2(p)*(t(com)+t(sum))=((n/p)-1)*t + 2*t*log2(p)=t*((n/p)-1)+2*log2(p))

y dado que el tiempo secuencial sería **T_s= (n-1)*t** (hay que hacer n-1 sumas (que hemos supuesto que tardan un tiempo t)

S(n,p)=T_s /T_P = (n-1)/(((n/p)-1)+2*log2(p))

Para valores suficientemente elevados de n (para los que tendría interés paralelizar la suma) podríamos suponer que 1 es despreciable frente a n y frente a n/p

Entonces:

S(n,p)=T_s /T_P = n/((n/p)+2*log2(p))

Igualando a 0 de derivada de S(n,p) con respecto a p se puede estimar el valor óptimo de p (el entero más próximo al que se obtiene al igualar a 0 la derivada y despejar p)
También se puede representar S(n,p) para distintos valores de p y obtener p gráficamente.

Apéndice para el problema 8 del Tema 2

Pseudocódigo SPMD para procedimiento de suma de números

Suma parcial de n/p números en cada procesador

Función de reducción ADD para obtener la suma total en el elemento pn=0 que imprimirá el valor obtenido

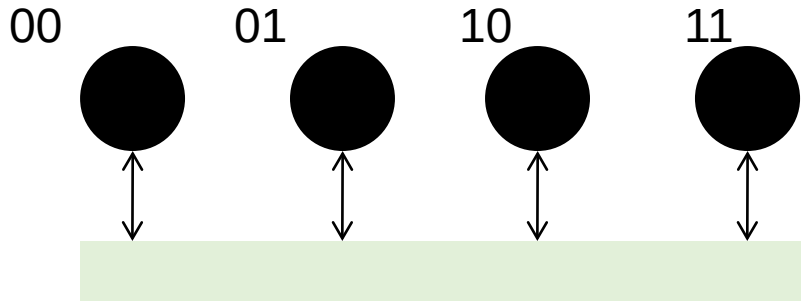
```
/* n   : elementos a sumar
/* p   : procesadores
/* pn  : identificador de procesador (idproc)

for (i=0; i<(n/p); ++i) S=S+V(i);

/* Reduccion
reduction(S,S,1,type,ADD,0,grupo);

if (pn==0) printf("Suma=%d",S);
```

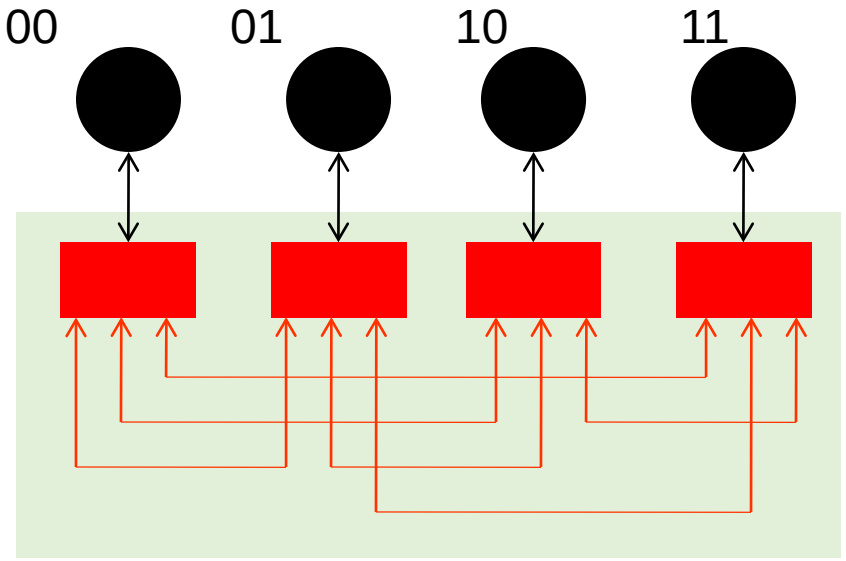
Pseudocódigo SPMD similar al anterior pero con la descripción de la función de reducción implementada con mensajes send y receive entre los procesadores

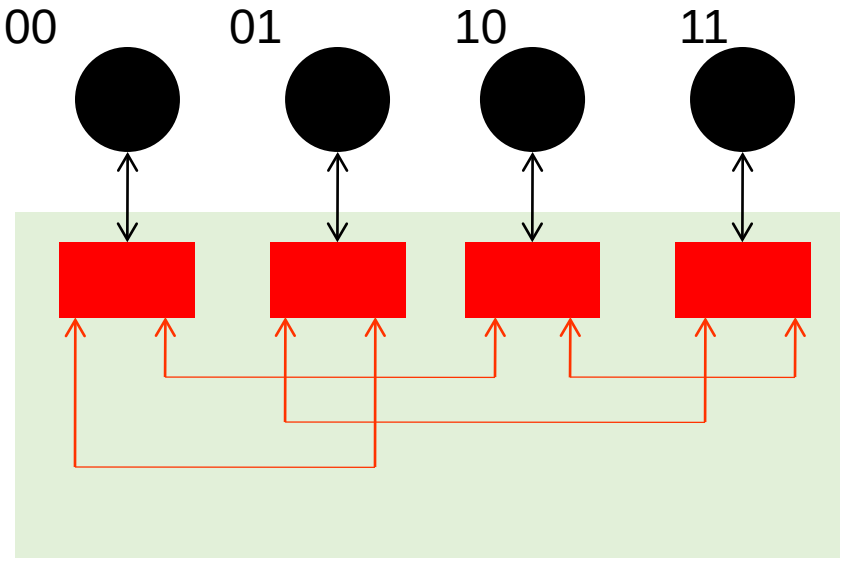
```
/* n   : elementos a sumar
/* p   : procesadores
/* pn  : identificador de procesador (idproc)

for (i=0; i<(n/p); ++i) S=S+V(i);

/* Reduccion
if (pn==0) for (i=1; i<p; i++) {
                        receive(SS,1,type,i,grupo);
                        S=S+SS;
}
else send(S,1,type,0,grupo);

if (pn==0) printf("Suma=%d",S);
```

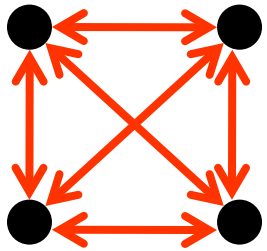
Función de Reducción: el procesador pn=0 recibe las sumas parciales de todos los demás y acumula

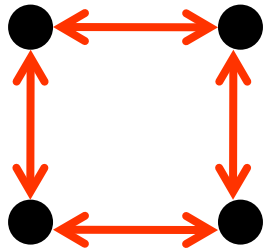
Función de Reducción: todos los procesadores excepto el pn=0 envían su suma parcial al pn=0


00
01
10
11


00
01
10
11


00
01
10
11





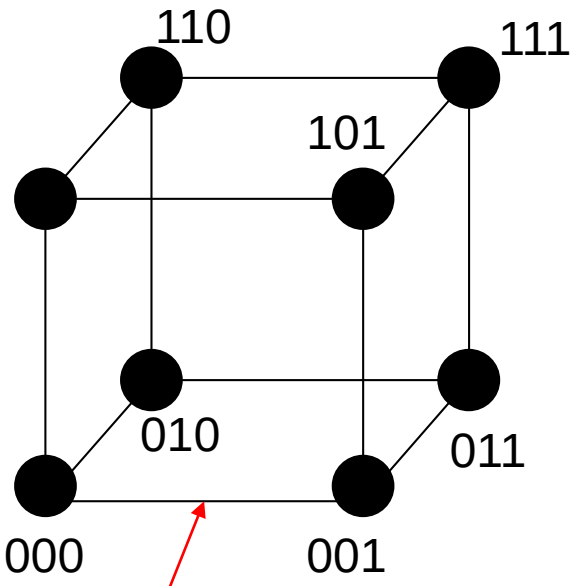
Obtención de sumas parciales en cada nodo

Reducción con comunicación de send y receive entre nodos conectados directamente por la red

```
/* n   : elementos a sumar
/* p   : procesadores (potencia de 2)
/* dim : p = power(2,dim)
/* pn  : identificador de procesador (idproc)

for (i=0; i<(n/p); ++i) S=S+V(i);

/* Comunicacion en cada una de las dimensiones
for (i=dim; i>0; --i) {

        if (pn<power(2,i)) {

                npn= pn ^ power(2,i-1);
                /* exor bit a bit pn y power(2,i-1)

                if (npn<pn) send(S,1,type,npn,group);
                else {
                        receive(SS,1,type,npn,group);
                        S=S+SS;
                }
        }
}
if (pn==0) printf("Suma=%d",S);
```

Send al procesador npn

receive del procesador npn



Procesadores conectados en un multicomputador hipercubo de dimensión 3

Pseudocódigo de programa SPMD con paso de mensajes para la suma de números en un multicomputador hipercubo (solo se comunican nodos conectados directamente)

dim=3

i=3; pn < power(2,3)=**8**

npn = pn ^ power(2,2)= pn ^ (100)

pn=0 ⟵ npn=4

pn=1 ⟵ npn=5

pn=2 ⟵ npn=6

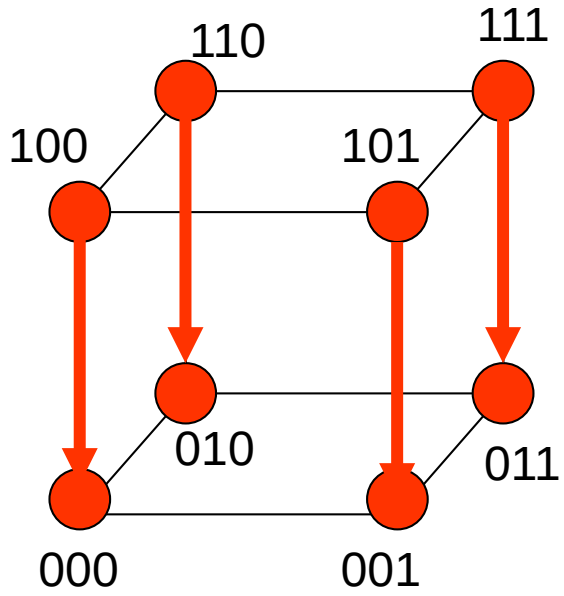
pn=3 ⟵ npn=7

pn=4 ⟶ npn=0

pn=5 ⟶ npn=1

pn=6 ⟶ npn=2

pn=7 ⟶ npn=3



Comunicaciones que se establecen en el hipercubo de dimensión 3 para i=3 en el bucle de comunicación del pseudocódigo de suma de números anterior

dim=3

i=2; pn < power(2,2)=**4**

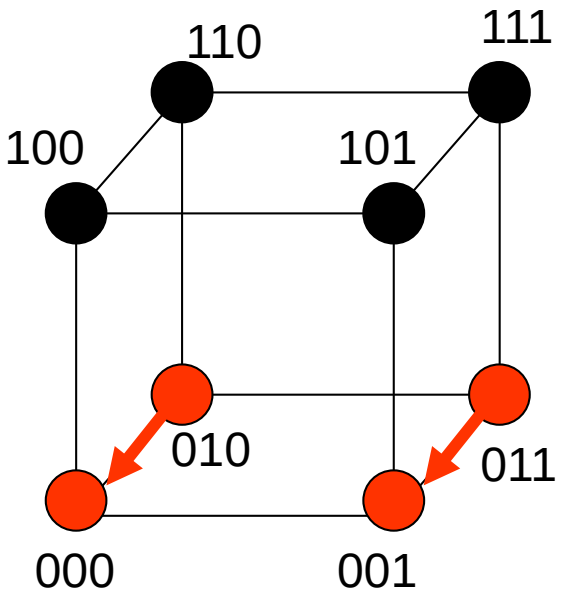
npn = pn ^ power(2,1)= pn ^ (010)

pn=0 ⟵ npn=2

pn=1 ⟵ npn=3

pn=2 ⟶ npn=0

pn=3 ⟶ npn=1



Comunicaciones que se establecen en el hipercubo de dimensión 3 para i=2 en el bucle de comunicación del pseudocódigo de suma de números anterior

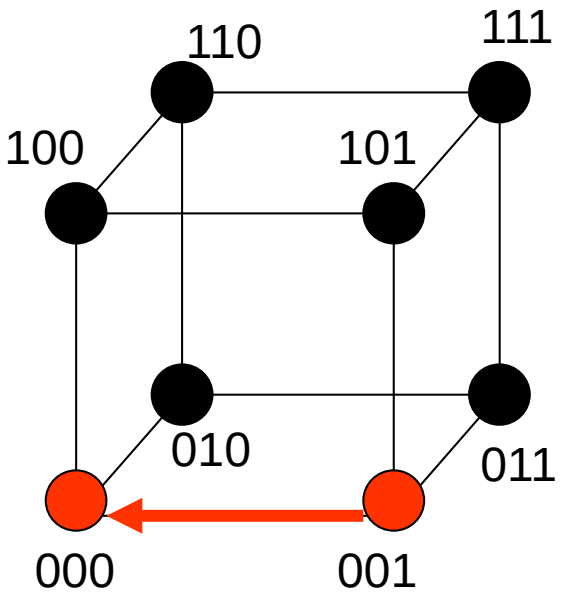
dim=3

i=1; pn < power(2,2)=**2**

npn = pn ^ power(2,0)= pn ^ (001)

pn=0 ⟵ npn=1

pn=1 ⟶ npn=0



Comunicaciones que se establecen en el hipercubo de dimensión 3 para i=1 en el bucle de comunicación del pseudocódigo de suma de números anterior

ARQUITECTURA DE COMPUTADORES
TEMA 2 Solución de problemas

Problema 10. Se quiere implementar un programa paralelo para un multicomputador que calcule la siguiente expresión para cualquier x (se trata del polinomio de interpolación de Lagrange)

$$L_i(x) = \frac{(x-a_0)\cdot\ldots(x-a_{i-1})\cdot(x-a_{i+1})\cdot\ldots(x-a_n)}{k_i} i = 0,1,\ldots,n$$

Inicialmente ki (=(a_i-a_0)·…(a_i-a_i-1) · (a_i-a_i+1) ·…(a_i-a_n))a_i y b_i se encuentra en el nodo i y x en todos los nodos. Sólo se van a usar funciones de comunicación colectivas. Indique cuál es el número mínimo de funciones colectivas que se pueden usar, cuáles serían y en qué orden se utilizarían y para qué se usan en cada caso.

Pr.	Inicialmente			
P0	a_0	x	k_0	b_0
P1	a_1	x	k_1	b_1
P2	a_2	x	k_2	b_2
P3	a_3	x	k_3	b_3

Resta paralela $A_i=(x-a_i)$
$(x-a_0)$
$(x-a_1)$
$(x-a_2)$
$(x-a_3)$

Todos reducen A_i con resultado en B_i
$(x-a_0).(x-a_1).(x-a_2).(x-a_3)$
$(x-a_0).(x-a_1).(x-a_2).(x-a_3)$
$(x-a_0).(x-a_1).(x-a_2).(x-a_3)$
$(x-a_0).(x-a_1).(x-a_2).(x-a_3)$

(1) Situación inicial de datos en un computador con cuatro procesadores

(2) Resta en paralelo en los procesadores

(3) Se aplica función de comunicación colectiva “Todos reducen” con el producto como función f y se obtiene en cada procesador lo que se indica en la tabla

ARQUITECTURA DE COMPUTADORES
TEMA 2 Solución de problemas

Problema 10 (cont.) Se quiere implementar un programa paralelo para un multicomputador que calcule la siguiente expresión para cualquier x (se trata del polinomio de interpolación de Lagrange)

$$L_i(x) = \frac{(x-a_0)\cdot\ldots(x-a_{i-1})\cdot(x-a_{i+1})\cdot\ldots(x-a_n)}{k_i} i = 0,1,\ldots,n$$

Inicialmente ki (=(a_i-a_0)·…(a_i-a_i-1) · (a_i-a_i+1) ·…(a_i-a_n))a_i y b_i se encuentra en el nodo i y x en todos los nodos. ……

Pr.	Cálculo de $L_i(x)$ en paralelo $C_i = B_i/(A_i \cdot k_i)$	Cálculo en paralelo $D_i = b_i \cdot C_i$
P0	$(x-a_1).(x-a_2).(x-a_3)/k_0$	$b_0 .(x-a_1).(x-a_2).(x-a_3)/k_0$
P1	$(x-a_0).(x-a_2).(x-a_3)/k_1$	$b_1 .(x-a_0).(x-a_2).(x-a_3)/k_1$
P2	$(x-a_0).(x-a_1).(x-a_3)/k_2$	$b_2 .(x-a_0).(x-a_1).(x-a_3)/k_2$
P3	$(x-a_0).(x-a_1).(x-a_2)/k_3$	$b_3 .(x-a_0).(x-a_1).(x-a_2)/k_3$

Reducción en P0 del contenido de D_i
Resultado = $\sum_{i=0}^{n}(D_i)$

(4) Cálculos en paralelo en cada procesador, que dispone de los datos necesarios en su memoria local

(5) Se aplica función de comunicación colectiva de “Re ducción” en el procesador 0 con la suma como función f y se obtiene en 0 lo que se indica en la tabla

ARQUITECTURA DE COMPUTADORES
TEMA 2 Solución de problemas

Problema 11. (a) Escriba un programa secuencial con notación algorítmica (podría escribirlo también en C) que determine si un número de entrada, x, es primo o no. El programa imprimirá si es o no primo. Tendrá almacenados en un vector, NP, los M números primos entre 1 y el máximo valor que puede tener un número en la entrada. ¿Cuántas operaciones de comparación se realizan?

Versión 1	Versión 2

b=0; i=0;
while ((b==0)&& (i<M)) do {
if (NP[i]==x) b=1;
i++;
}
if (b==1) printf("%d ES primo", x);
elseprintf("%d NO es primo", x);
	b=0;
	for(i=0;i<M;i++) {
	if (NP[i]==x) b=1;
	}
	if (b==1) printf("%d ES primo", x);
	elseprintf("%d NO es primo", x);

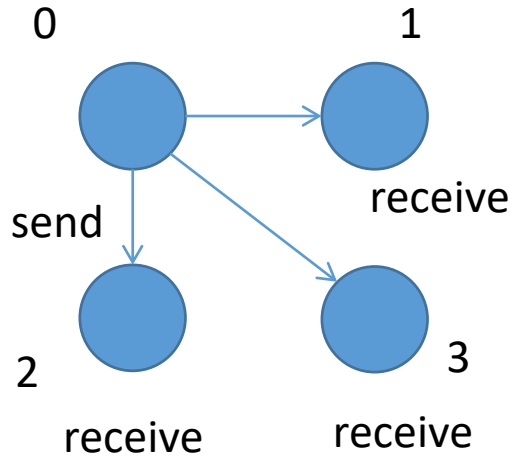
El número de comparaciones en la versión 1 depende de la posición del número si está en NP. Si no está hay que recorrer todo NP. En la versión 2 siempre se recorre todo NP

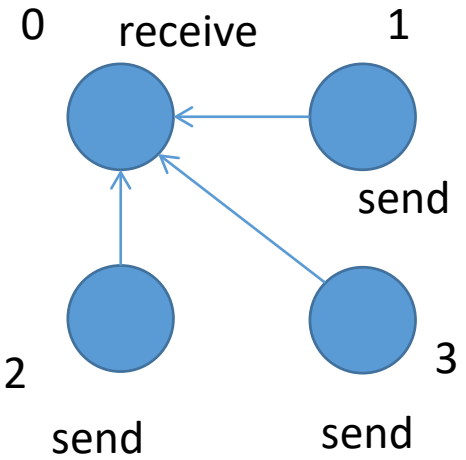
ARQUITECTURA DE COMPUTADORES

TEMA 2 Solución de problemas

Problema 11. (b) Escriba una versión paralela del programa anterior para un multicomputador usando paso de mensajes. El proceso 0 tiene inicialmente el número x y el vector NP en su memoria. El proceso 0 imprimirá en pantalla el resultado……

```
b=0; i=0;

//difusión del vector NP y de x
if (idproc==0)
 for (i=1; i<numprocesos) send(NP,M,tipo,i,grupo);
else receive(NP,M,tipo,0,grupo);
if (idproc==0)
 for (i=1; i<numprocesos) send(x,1,tipo,i,grupo);
else receive(x,1,tipo,0,grupo);
//Calculo
i=idproc;
while ((b==0)&&(i<M)) do {
if (NP[i]==x) b=1;
i=i+num_procesos;
}
// Recogida de resultados
if (idproc==0)
  for (i=1; i<num_procesos) {
          receive(baux,1,tipo,i,grupo);
          b=b &baux;
  }
else send(b,1,tipo,0,grupo);

if (idproc==0)
   if (b==1) printf("%d ES primo", x);
elseprintf("%d NO es primo", x);
```



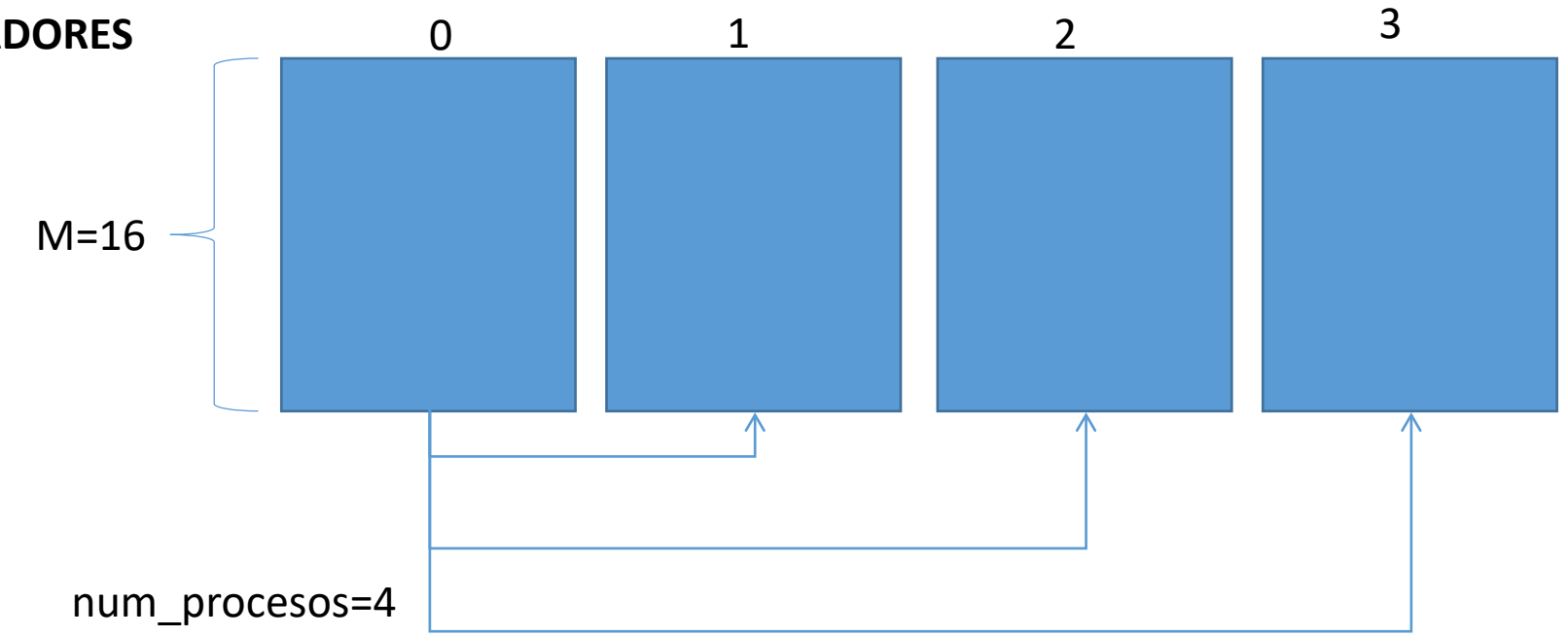


En este problema se considera que la herramienta de programación ofrece funciones send()/receive() para implementar comunicación uno-a-uno asíncrona, con la función send(buffer,count,datatype,idproc,group) no bloqueante y receive(buffer,count,datatype,idproc,group) bloqueante.

En primer lugar se difunden x y el vector NP, después se hace el cálculo (comparación y actualización) y finalmente se recoge el resultado en el procesador 0
En la figura se muestra para cuatro procesadores

ARQUITECTURA DE COMPUTADORES
TEMA 2 Problema 11(cont)

```
//Calculo
i=idproc;
while ((b==0)&&(i<M)) do {
if (NP[i]==x) b=1;
i=i+num_procesos;
}
```



El bucle de cálculo permite distribuir los datos de la matriz entre los procesadores según una estrategia de round-robin.

Como se ve en la figura (en rojo) elementos consecutivos en NP se asignan a procesadores con índices (iproc) consecutivos

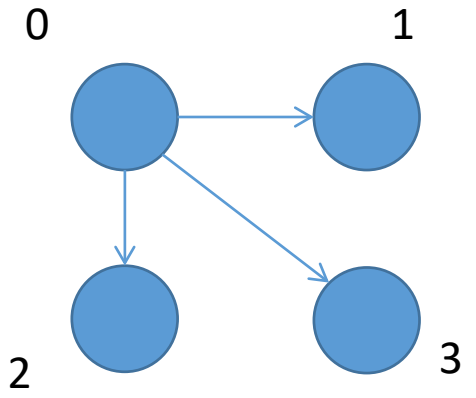
i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
idproc=0	**0**	1	2	3	**4**	5	6	7	**8**	9	10	11	**12**	13	14	15
idproc=1	0	**1**	2	3	4	**5**	6	7	8	**9**	10	11	12	**13**	14	15
idproc=2	0	1	**2**	3	4	5	**6**	7	8	9	**10**	11	12	13	**14**	15
idproc=3	0	1	2	**3**	4	5	6	**7**	8	9	10	**11**	12	13	14	**15**

El array NP se difunde a todos los procesadores (cada procesador solo utiliza parte del array)

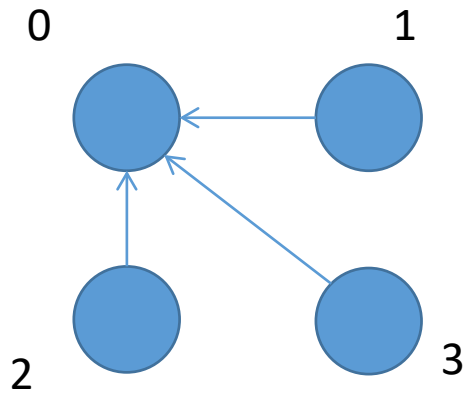
ARQUITECTURA DE COMPUTADORES
TEMA 2 Solución de problemas

Problema 12. (b) Escriba una versión paralela del programa secuencial del ejercicio 11 suponiendo que la herramienta de programación ofrece las funciones colectivas de difusión y reducción (escriba primero la versión secuencial). En la función de difusión, broadcast(buffer,count,datatype,idproc,group), se especifica …….

```
b=0; i=0;
//difusión del vector NP y de x
broadcast(NP,M,tipo,0,grupo);
broadcast(x,1,tipo,0,grupo);
//Calculo
i=idproc;
while ((b==0)&&(i<M)) do {
if (NP[i]==x) b=1;
i=i+num_procesos;
}
reduction(b,b,1,tipo,OR,0,grupo);
if (idproc==0)
    if (b==1) printf("%d ES primo", x);
elseprintf("%d NO es primo", x);
```



Broadcast desde 0



Reducción en 0

En este problema se considera que la herramienta de programanción ofrece la función de comunicación colectiva broadcast

Como en el caso anterior, en primer lugar se difunden x y el vector NP, y después se hace el cálculo (comparación y actualización), para finalmente recoger el resultado en el procesador 0

La existencia de la función broadcast simplifica el código. Se conseguirían buenas prestaciones si el broadcast se hubiera implementado considerando la topología de la red y las conexiones directas entre procesadores.

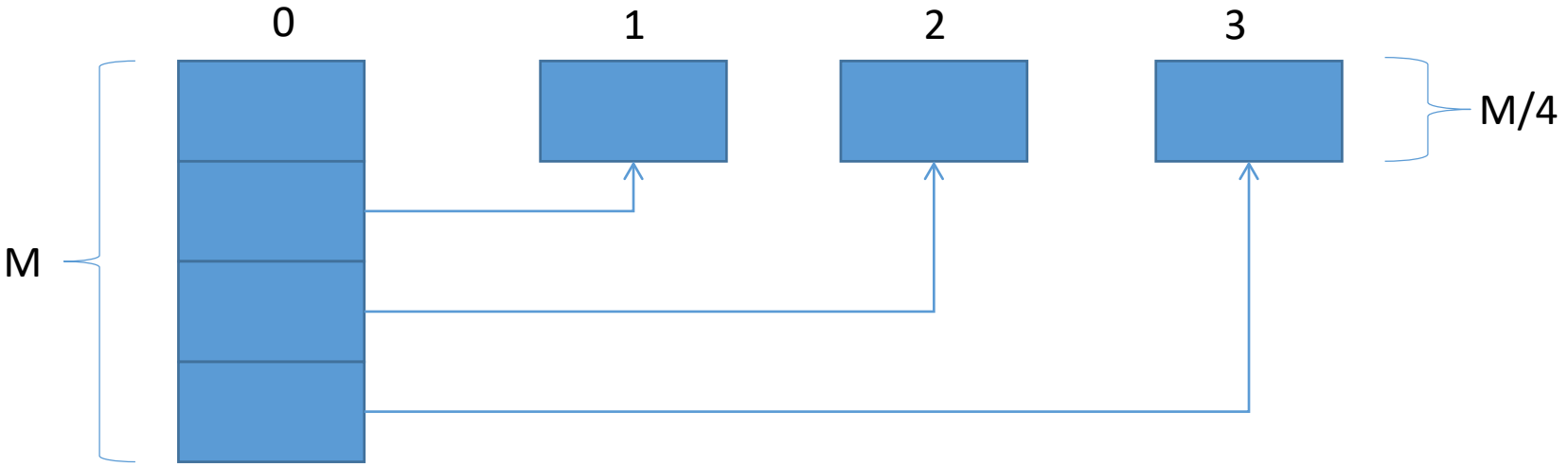
ARQUITECTURA DE COMPUTADORES
TEMA 2 Solución de problemas

Problema 13. Vuelva a escribir el programa paralelo del ejercicio anterior suponiendo que, además de las funciones colectivas anteriores, se dispone de dispersión y que M es divisible entre el número de procesos (escriba primero la versión secuencial). La función de comunicación colectiva es scatter(sendbuf,sendcnt,recvbuf,recvcnt,datatype,idproc,group)

```
b=0; i=0;
//difusión del vector NP y de x
scatter(NP,M,NP1, M/num_procesos,tipo,0,grupo);
broadcast(x,1,tipo,0,grupo);

//Cálculo
while ((b==0)&& (i<M/num_procesos)) do {
   if (NP1[i]==x) b=1;
  i=i+1;
}
reduction(b,b,1,tipo,OR,0,grupo);
if (idproc==0)
    if (b==1) printf("%d ES primo", x);
elseprintf("%d NO es primo", x);
```

Mediante la función scatter se consigue enviar una parte del array NP desde el procesador 0 (en este caso) a cada uno de los procesadores.

En la figura se consideran 4 procesadores y a cada uno de ellos se enviará una cuarta parte de NP que se almacenará en la matriz NP1 local en cada procesador, que solo explorará esa matriz



Problema 14

Versión 1

```
   b=0; i=0;

//solo un thread escribe en b
#pragma omp parallel\
private(idthread){
intilocal;
   idthread=omp_get_thread_num();
   nthreads=omp_get_num_threads();
   #pragma omp critical {
   ilocal=i; i++;
   }
   while ((b==0)&& (ilocal<M)) do {
      if (NP[ilocal]==x) b=1;
     #pragma ompcritical
     ilocal=i; i++;
     }
   }
}

   if (b==1) printf("%d ES primo", x);
   elseprintf("%d NO es primo", x);
```

Versión 2

```
   b=0;
#pragma omp parallel for \
reduction(b:|)
   for(i=0;i<M;i++)  {
      if (NP[i]==x) b=1;
   }

   if (b==1) printf("%d ES primo", x);
   elseprintf("%d NO es primo", x);
```