Problema: Siga un algoritme paral·lel que compleix la Llei de Amdahl per a l'acceleració on el paràmetre **α** (part no paral·lelitzable de l'algoritme) segueix la següent expressió: $\alpha(p,n)= p/n^2$, on p és el nombre de processadors i n és la grandària del problema.

a) (1) Calculeu l'expressió de l'acceleració S(n,p) per a este algoritme i explica com l'has obtingut.
b) (.5) Calculeu els límits de la eficiència $E(n,p)$ de l'algoritme quan ($n\rightarrow\infty$) i quan ($p\rightarrow\infty$).
c) (1) S(n,p) és una funció lineal respecte de p, o pot tindre algun màxim? Calculeu-lo per a qualsevol p.

a) Com $\alpha(p,n)= p/n^2$ i que la llei de Amdahl per a l'acceleració té la següent expressió: $S(n,p)= \frac{T(n,1)}{T(n,p)} = \frac{T(n,1)}{\alpha T(n,1)+\frac{(1-\alpha)T(n,1)}{p}}$, es sustitueix el valor de α pel de la expressió i queda

$$S(n,p)= \frac{1}{\alpha+\frac{(1-\alpha)}{p}} = \frac{p}{p\alpha+(1-\alpha)} = \frac{p}{p\left(\frac{p}{n^2}\right)+1-\left(\frac{p}{n^2}\right)} = \frac{\boldsymbol{p\,n^2}}{\boldsymbol{n^2+p(p-1)}}$$

b) Com que l'eficiència ve donada per l'expressió $E(n,p)=S(n,p)/p$, tenim que $E(n,p) = \frac{p\,n^2}{p(n^2+p(p-1))} = \frac{n^2}{n^2+p(p-1)}$. Per tant els límits que ens demanen seran:

$$\lim_{n\to\infty} E(n,p) = \frac{n^2}{n^2+p(p-1)} = \frac{n^2}{n^2} = 1$$
$$\lim_{p\to\infty} E(n,p) = \frac{n^2}{n^2+p(p-1)} = \frac{n^2}{p^2} = 0$$

c) S(n,p) no és una funció lineal respecte de p, ja que apareix un terme lineal però també un terme inversament quadràtic. Per tant, pot tindre un màxim que podem calcular obtenint la primera derivada de S(n,p):

$S'(n,p)= \frac{d}{dp}\left(\frac{p\,n^2}{n^2+p(p-1)}\right) = \frac{n^2}{n^2+p(p-1)} - \frac{p\,n^2(2p-1)}{(n^2+p(p-1))^2}$,

i igualant-la a cero, d'on ens queda que:

$\frac{n^2}{n^2+p(p-1)} - \frac{p\,n^2(2p-1)}{(n^2+p(p-1))^2} = 0 \rightarrow \frac{n^2}{n^2+p(p-1)} = \frac{p\,n^2(2p-1)}{(n^2+p(p-1))^2} \rightarrow$

$1 = \frac{p(2p-1)}{n^2+p(p-1)} \rightarrow n^2+p(p-1) = p(2p-1) \rightarrow n^2+p^2 = 2p^2 \rightarrow \boldsymbol{n^2 = p^2}$

Ja que p es distint de cero i que sols té sentit una solució positiva, hi haurà un màxim quan es complisca que $p = n$. No és necessària la segona derivada per vore que es tracta d'un màxim, ja que en l'expressió de S(n,p) el terme lineal multiplica n^2, mentre que el terme inversament quadràtic està sumant a n^2, i ja que $n >> 1$, el terme numerador serà major que el denominador fins que p tinga un valor equiparable a n. En eixe cas tenim que

$$S(n,p)= \frac{n\,n^2}{n^2+n(n-1)} = \frac{\boldsymbol{n^2}}{\boldsymbol{2n-1}}$$

Problema: Es vol paral·lelitzar, per a un sistema multiprocessador el següent segment de codi:

```
for (i = 0; i < n; i++){
  ...  //codi per a cada iteració
}
```

L'execució de cadascuna de las iteracions del bucle, sense dependència de dades, suposa un temps d'execució $2\cdot(i+1)\cdot t_c$, on t_c és el temps d'execució d'una instrucció. En l'execució paral·lela, la inicialització, comunicació i sincronització de p processos suposa un temps $(1+\log p)\cdot t_c$. Es considera que $n>>p$.

a) (1) Obteniu l'expressió de la funció d'isoeficiència. Comenta cóm s'ha obtingut.
b) (1) Calculeu la relació R/C per a este algoritme i obteniu el seu valor asimptòtic per a n.
c) (1) Donat l'execució d'este algoritme amb p processadors per a una mida n, indica quin hauria de ser el nombre de processadors p' perquè es mantingués igual la eficiència al executar-lo amb la mida del problema $n'= 4n$. Descriviu cóm s'ha obtingut.

a) Com que cadascuna de les n iteracions del bucle tenen una cost $2\cdot(i+1)\cdot tc$, on i és el nombre de la iteració, el cost seqüencial de l'algoritme serà:

$$W(n,1)=\sum_{i=0}^{n-1}2(i+1)t_c=\sum_{i=1}^{n}2it_c=2\left(\frac{n(n+1)}{2}\right)t_c=n(n+1)t_c$$

Llavors $S(n,p)$ serà:

$$S(n,p)=\frac{n(n+1)t_c}{\frac{n(n+1)}{p}t_c+(1+\log_2 p)\,t_c}$$

I la eficiència

$$E(n,p)=S(n,p)/p=\frac{n(n+1)t_c}{n(n+1)t_c+p(1+\log_2 p)t_c}=\frac{n(n+1)t_c}{n(n+1)t_c+p\log_2(2p))t_c}$$

Si substituïm per $W(n,1)$ y l'aïllem, obtenim la funció d'Isoeficiència:

$$\boldsymbol{W(n,1)=\left(\frac{E}{1-E}\right)\cdot p\log_2(2p)\,t_c.}$$

b) La relació R/C per a este algoritme és la relació entre el cost computacional del algoritme paral·lel i el sobrecost per la inicialització, comunicació i sincronització dels p processos:

$$\boldsymbol{R/C=\frac{\frac{n(n+1)}{p}}{(1+\log_2 p)}=\frac{n(n+1)}{p(1+\log_2 p)}}$$

El seu valor asimptòtic per a n serà de: **O(n²)**

c) Ens demanen el valor de p' perquè es mantinga l'eficiència quan n'= 4n .

Com $W(n,1)=n(n+1)\,t_c$, Per al cas n' tenim que $W(n',1)=n'(n'+1)\,t_c=4n(4n+1)\,t_c$.

La funció d'isoeficiència te l'expressió $W(n,1)=\left(\frac{E}{1-E}\right)\cdot p\log_2(2p)t_c$, i donat que $n\gg p\geq 1$ tenim que:

$$\frac{W(n\prime,1)}{W(n,1)}=\frac{p\prime\log_2(2p\prime)}{p\log_2(2p)}.\rightarrow\frac{n\prime(n'+1)}{n(n+1)}\cong\frac{n'^2}{n^2}=\frac{16n^2}{n^2}=\mathbf{16}=\boldsymbol{\frac{p\prime\log_2(2p\prime)}{p\log_2(2p)}}.$$

No hi ha solució exacta sempre per a aquesta equació, però podem vore alguns resultats per a alguns valors de p i les aproximacions que tindríem:

p= 2 → $p'\log_2(2p')=16p\log_2(2p)=16*2*2=64$ → p' = 13,46

p= 4 → $p'\log_2(2p')=16p\log_2(2p)=16*4*3=192$ → p' = 32

p= 8 → $p'\log_2(2p')=16p\log_2(2p)=16*2*4=512$ → p' = 71,5

Per a la propera sessió, i per que repasseu el que hem vist a classe, us he pujat un fitxer amb un problema de examen resolt i un de proposat. El problema proposat heu de lliurar-lo al començar la classe. No l'agafaré després.

Problema resolt

Se quiere paralelizar para un sistema multiprocesador el siguiente segmento de código:

```
for(i=0; i<n; i++){
   Código para iteración i;}
```

La ejecución de cada una de las iteraciones del bucle, sin dependencia de datos, supone un tiempo de ejecución $3n \cdot t_c$, donde t_c es el tiempo de ejecución de una instrucción.

En la ejecución paralela, la inicialización, comunicación y sincronización de *p* procesos supone un tiempo $k \cdot log(p) \cdot t_c$ (siendo *k* constante). Se considera que $n >> p$.

a) (0,5) Obtener las expresiones para el tiempo de ejecución secuencial (T_s) y el tiempo paralelo del segmento de código con p procesadores (T_p). Comenta cómo se ha obtenido.

b) (1,5) Dado la ejecución de este algoritmo con *p* procesadores para una talla *n*, indica cual debería ser la talla del problema *n'* para que se mantuviera igual la eficiencia al ejecutarlo con p'= p^2 procesadores.

a) El tiempo de ejecución secuencial de este segmento de código viene determinado por el número de iteraciones del bucle for (n) y el coste de cada una de ellas ($3nt_c$), por lo que:

$$T_s = n(3nt_c) = 3n^2 t_c$$

Para el tiempo paralelo tenemos que tener en cuenta que el bucle es paralelizable y que hay un tiempo extra debido a la ejecución paralela (inicialización, comunicación y sincronización de los p procesos). Como además n>>p el tiempo paralelo será:

$$T_p = 3\frac{n^2}{p} t_c + k \log(p)\, t_c$$

b) Para poder hacer este cálculo de forma fácil, calculamos primero la función de isoeficiencia. Comenzaremos identificando el coste secuencial del código en función de n a partir de Ts:

$$Ts = 3n^2 t_c = W(n,1) t_c$$

La expresión de la eficiencia viene dada por la expresión:

$$E(n,p) = E = \frac{S(n,p)}{p} = \frac{3n^2 t_c}{p\left(3\frac{n^2}{p} + k\log(p)\right) t_c} = \frac{W(n,1)}{W(n,1) + kp\log(p)}$$

Operando y despejando W(n,1) tenemos:

$$W(n,1) = \left(\frac{E}{1-E}\right) kp \log(p)$$

Queremos saber cuánto debe valer n' para que al pasar de p a p'=p^2 procesadores tengamos la misma eficiencia. Esto supone que:

$$W(n',1) = \left(\frac{E}{1-E}\right) kp' \log(p') = \left(\frac{E}{1-E}\right) kp^2 \log(p^2) = \left(\frac{E}{1-E}\right) 2kp^2 \log(p)$$

Si dividimos las igualdades anteriores tenemos que

$$\frac{W(n',1)}{W(n,1)} = \frac{3n'^2}{3n^2} = \frac{2kp^2\log(p)}{kp\log(p)} = 2p$$

Despejando n' obtenemos que:

$$n' = n\sqrt{2p}$$

Problema proposat:

Es vol paral·lelitzar, per a un sistema multiprocessador el següent segment de codi:

```
for(i=0; i<n; i++){
   Codi per a la iteració i;}
```

L'execució de cadascuna de las iteracions del bucle, sense dependència de dades, suposa un temps d'execució $5n^2 \cdot t_c$, on t_c és el temps d'execució d'una instrucció.

En l'execució paral·lela, la inicialització, comunicació i sincronització de p processos suposa un temps $(k_1+k_2 log_2 p)t_c$ (sent k_1 *i* k_2 constants). Es considera que $n>>p$.

c) (1) Obtindre l'expressió de la funció d'isoeficiència. Comenta cóm s'ha obtingut.

d) (1) Donat l'execució d'este algoritme amb p processadors per a una mida n, indica quin hauria de ser la mida del problema n' per a que es mantingués igual la eficiència al executar-lo amb p'= 2p processadors. Comenta cóm s'ha obtingut.

```
/*****************************************
 E8: SOLUCIÓN A LOS DOS PROBLEMAS PROPUESTOS
*****************************************/

//-----------------
// PROBLEMA 1:
//-----------------

// a) Rutina
// He quitado includes y demás, solo dejo lo imprescindible para el problema
// ...

int main( int argc, char *argv[])
{
  double x[k]; // como esta repartido, son suficientes k elementos por proceso, no hace falta
n.
  double suma_local,suma_total;

  MPI_Init(&argc,&argv);

  suma_local=0;
  for (i=0; i<k; i++) {
    suma_local+=x[i];
  }

MPI_Allreduce(&suma_local,&suma_total,1,MPI_DOUBLE,MPI_SUM,MPI_COMM_WORLD
);
  // Otra solución consiste en utilizar Reduce seguido de Broadcast:
  //
MPI_Reduce(&suma_local,&suma_total,1,MPI_DOUBLE,MPI_SUM,0,MPI_COMM_WORLD
);
  // MPI_Bcast(&suma_total,1,MPI_DOUBLE,0,MPI_COMM_WORLD);

  MPI_Finalize();
  return 0;
}

/*
b) Modelo temporal

ta -> Op float,
tcom=tin +m*tc
m -> bytes que se transmiten
tin=10tc
tc -> Comunicar 1 byte
```

```
Vamos a suponer que tanto la opción Allreduce como Reduce seguido de Bcast tienen el
mismo coste
(en un sistema real, Allreduce podría estar optimizado, pero no hay elementos en el
problema que nos
permmitan saberlo).

Reduce -> tcom = log2(p)*(tin+m*tc)
          tcal = log2(p)*op*ta
  m en este caso es 8 bytes (un double)
  op en este caso es 1 pues solo se realiza una operación (una suma pues solo hay un
elemento)
  Por lo tanto:
    tcom = log2(p)*(10tc+8tc) = 18log2(p)tc
    tcal = log2(p)*ta

Bcast -> tcom = log2(p)*(tin+m*tc)
  m es 8 pues solo enviamos un double
  Por lo tanto:
    tcom = log2(p)*(10tc+8tc) = 18log2(p)tc

Calculos en el bucle
    tcal = k*ta

Sumando por tipo de contribución:
  tcom = 18log2(p)tc + 18log2(p)tc = 36log2(p)tc
  tcal = ( log2(p) + k )ta
  Ttotal = tcom + tcal =  36log2(p)tc + (log2(p) + k)ta
*/



//-----------------
// PROBLEMA 2:
//-----------------

// a) Rutina
// He quitado includes y demás, solo dejo lo imprescindible para el problema
// ...

int main( int argc, char *argv[])
{
  double xl[n][n];
  double x[n][n]; // bastaria con que esta matriz estuviera solo en 0 que es el unico que la
utiliza

  MPI_Init(&argc,&argv);
```

```
  // si suponemos que los elementos de la matriz están consecutivos en memoria,
  // basta con la siguiente instruccion:
  MPI_Reduce(&xl[0][0],&x[0][0],n*n,MPI_DOUBLE,MPI_SUM,0,MPI_COMM_WORLD);

  // si la matriz no se ha definido de forma estática, o no estamos seguros de cómo
  // se ha creado, deberíamos proceder así:
  // for (i=0; i<n; i++) {
  //   MPI_Reduce(xl[i],x[i],n,MPI_DOUBLE,MPI_SUM,0,MPI_COMM_WORLD);
  // }
  // en este segundo caso, los costes temporales aumentan un poco por el tiempo inicial de
cada reduce

  MPI_Finalize();
  return 0;
}

/*
b) Modelo temporal

ta -> Op float,
tcom=m*tc   // en este caso no tenemos tin o tin=0
m -> bytes que se transmiten
tc -> Comunicar 1 byte

Reduce -> tcom = log2(p)*(m*tc)
          tcal = log2(p)*op*ta

En este caso se transmiten n*n doubles (m=8*n*n bytes) y se hacen n*n sumas (op=n*n)
Por lo tanto:
  tcom = log2(p)*8*n*n*tc
  tcal = log2(p)*n*n*ta
Sumamos términos:
  Ttotal= tcom + tcal = log2(p)*n*n*(ta+8tc)

Si se optó por la opción de un reduce por fila, el resultado sería:
  tcom = n*log2(p)*8*n*tc
  tcal = n*log2(p)*n*ta  // se queda igual
Sumamos términos:
  Ttotal= tcom + tcal = log2(p)*n*n*(ta+8tc)

  En ambos casos el resultado coincide, pero si hubiéramos tenido tin, entonces
  el caso con el bucle hubiera tardado más.
 */
```

```
{
  // Cálculo de PI
  int n,i,rank,size,bsize,remain,beg,end;
  double x;

  MPI_Init ( &argc, &argv );
  MPI_Comm_rank ( MPI_COMM_WORLD, &rank );
  MPI_Comm_size ( MPI_COMM_WORLD, &size );

  if ( rank == 0 )
    {
      printf("Introduce la precisión del cálculo (n > 0): ");
      fflush(stdout);
      scanf("%d",&n);
    }
  MPI_Bcast ( &n, 1, MPI_INT, 0, MPI_COMM_WORLD );
  bsize= n/size;
  remain= n%size;
  if ( rank == 0 )
    {
      beg= 0;
      end= bsize + remain;
    }
  else
    {
      beg= rank*bsize + remain;
      end= beg + bsize;
    }

  double PI25D = 3.14159265358979323846264338;
  double h = 1.0 / (double) n;
-UUU:----F1  ej3.c          12% L23    (C/l Abbrev)----------------------------------------------------
```

```
  if ( rank == 0 )
    {
      beg= 0;
      end= bsize + remain;
    }
  else
    {
      beg= rank*bsize + remain;
      end= beg + bsize;
    }

  double PI25D = 3.14159265358979323846264338;
  double h = 1.0 / (double) n;
  double sum = 0.0;
  for (i = beg; i < end; i++) {
    x = h * ((double)i + 0.5);
    sum += (4.0 / (1.0 + x*x));
  }
  double res;
  MPI_Reduce ( &sum, &res, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD );
  if ( rank == 0 )
    {
      double pi = res * h;
      printf ("El valor aproximado de PI es: %f, con un error de %f\n",pi,fabs(pi - PI25D));
    }

  MPI_Finalize ();

  return 0;
}
```

Problema: Donat el següent programa paral·lel en MPI que calcula la integral d'una funció amb una precisió *n* i el resultat s'emmagatzema en tots els processos: (*Dado el siguiente programa paralelo en MPI que calcula la integral de una función con una precisión n y el resultado se almacena en todos los procesos:*)

```
int main(void) {
   int rank, sz, local_n, n;
   double a = 0.0, b = 5.0, h, local_a, local_b;
   double local, total;
   int source;
   MPI_Init(NULL, NULL);
   MPI_Comm_rank(MPI_COMM_WORLD, &rank);
   MPI_Comm_size(MPI_COMM_WORLD, &sz);
   if(rank==0){
     printf("Introduce la precisión del cálculo (n > 0): ");
     scanf("%d",&n);}
   MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
   h = (b-a)/n; /* h es el mismo para todos los procesos */ 2 FLOP
   local_n = n/sz; /* es el número de trapezoides */
   local_a = a + rank*local_n*h; 3 FLOP
   local_b = local_a + local_n*h; 2 FLOP
   local = Trap(local_a,local_b,local_n, h);//Coste= 5*local_n+2 FLOP
   if (rank == 0) {
      total = local;
      for (source = 1; source < sz; source++) {
         MPI_Recv(&local, 1, MPI_DOUBLE, source, 0, MPI_COMM_WORLD,
         MPI_STATUS_IGNORE);
         total += local; 1 FLOP
       }
     } else {
      MPI_Send(&local, 1, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD);
    }
   if (rank == 0) //para el coste no se tiene en cuenta la E/S
      printf("La integral de %f a %f = %.15e\n", a, b, total);
   MPI_Bcast(&total, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
   MPI_Finalize();
   return 0;
} /* main */
```

Problema: Donat el següent programa paral·lel en MPI que calcula la integral d'una funció amb una precisió *n* i el resultat s'emmagatzema en tots els processos: (*Dado el siguiente programa paralelo en MPI que calcula la integral de una función con una precisión n y el resultado se almacena en todos los procesos:*)

```
int main(void) {
  int rank, sz, local_n, n;
  double a = 0.0, b = 5.0, h, local_a, local_b;
  double local, total;
  int source;
  MPI_Init(NULL, NULL);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Comm_size(MPI_COMM_WORLD, &sz);
  if(rank==0){
    printf("Introduce la precisión del cálculo (n > 0): ");
    scanf("%d",&n);}
  MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
  h = (b-a)/n; /* h es el mismo para todos los procesos */ 2 FLOP
  local_n = n/sz; /* es el número de trapezoides */
  local_a = a + rank*local_n*h; 3 FLOP
  local_b = local_a + local_n*h; 2 FLOP
  local = Trap(local_a,local_b,local_n, h);//Coste= 5*local_n+2 FLOP
  if (rank == 0) {
     total = local;
     for (source = 1; source < sz; source++) {
        MPI_Recv(&local, 1, MPI_DOUBLE, source, 0, MPI_COMM_WORLD,
        MPI_STATUS_IGNORE);
        total += local; 1 FLOP
      }
    } else {
     MPI_Send(&local, 1, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD);
   }
  if (rank == 0) //para el coste no se tiene en cuenta la E/S
     printf("La integral de %f a %f = %.15e\n", a, b, total);
  MPI_Bcast(&total, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
  MPI_Finalize();
  return 0;
} /* main */
```

a) Calcula el model de cost temporal aproximat d'este programa considerant per a la part computacional sols les operacions en coma flotant (cada operació costa t_a) i que per a les comunicacions es segueix el model $t_{com}(m) = t_{in} + m \cdot t_c$, on *m* està en bytes i $t_{in} = 50 \cdot t_c$.(*Calcula el modelo de coste temporal aproximado de este programa considerando para la parte computacional sólo las operaciones en coma flotante (cada operación cuesta t_a) y que para las comunicaciones se sigue el modelo $t_{com}(m) = t_{in} + m \cdot t_c$, donde m está en bytes y $t_{in} = 50 \cdot t_c$.)*

El coste temporal del programa viene dado por la suma de los costes de las operaciones en coma flotante y por las comunicaciones.

Coste de operaciones en coma flotante: Las 5 operaciones iniciales, más el cálculo de la función más la suma de los resultados parciales.

$(7+(5(n/p)+2+(p-1))*t_a = (9+5(n/p)+(p-1))*t_a$, donde n es el tamaño del problema y p es el número de procesos.

Coste de comunicaciones: Considero las dos operaciones colectivas más las (p-1) comunicaciones punto a punto,

MPI Bcast de n → $(t_{in}+4t_c)\ log_2 p$

MPI_Send/Recv → $(t_{in}+8t_c)\ (p-1)$

MPI_Bcast de local → $(t_{in}+8t_c)\ log_2 p$

La suma de lo cual nos da:

$$(54t_c+58t_c)\ log_2p\ +58t_c\ (p-1) = ((112)\ log_2p\ +58(p-1))\ t_c$$

Coste total

$$(9+5(n/p) + (p-1))*t_a\ +((112)\ log_2p\ +58(p-1))\ t_c$$

b) Calcula el rendiment en MFLOPS d'este programa a partir del model anterior, per a n=1024, si s'executa amb p=8 (processos=processadors) que funcionen a F= 4 GHz (t_a=1/F) i la xarxa d'interconnexió té un ample de banda màxim de BW= 32 Gb/s (velocitat efectiva, t_c=1/BW). ¿Quin serà el rendiment quan n →∞? (*Calcula el rendimiento en MFLOPS de este programa a partir del modelo anterior, para n=1024, si se ejecuta con p=8 (procesos=procesadores) que funcionan a F= 4 GHz (t_a=1/F) y la red de interconexión tiene un ancho de banda máximo de BW= 32 Gb/s (velocidad efectiva, t_c=1/BW). ¿Cuál será el rendimiento cuando n →∞?)*

El Rendimiento en MFLOPS se define como la relación entre el coste de operaciones en coma flotante (FLOP) del algoritmo secuencial dividido por el coste temporal del algoritmo paralelo en microsegundos (10^{-3} segundos). Si se divide por nanosegundos serían GFLOPS, etc.:

Coste secuencial del algoritmo en FLOP= (5n + 4) FLOP (no se tiene en cuenta el cálculo de las variables local_a y local_b, pues no se requieren en la versión secuencial)

Coste temporal del algoritmo paralelo:

$$(9+5(n/p) + (p-1))*t_a\ +((112)\ log_2p\ +58(p-1))\ t_c$$

De los datos se deduce que:

ta= 1/4GHz= 0.25 ns

tc= 1B/32Gb/s= 8b/32 Gb/s= 0.25 ns

Luego para n=1024

$$Rendimiento=\frac{5n+4}{(9+5(n/p)+(p-1))*t_a\ +((112)\ \log_2 p\ +58(p-1))\ t_c} = \frac{5*1024+4}{\left(\left(9+5\left(\frac{1024}{8}\right)+(8-1)\right)+((112)\log_2 8\ +58(8-1))\right)*0.25ns} =$$

$$\frac{5*1024+4}{\left((9+5*128+7)+((112)*3\ +58*7)\right)*0.25ns} = \frac{5124}{(656+742)*0.25ns} = \frac{5124}{(1398)*0.25ns} = \mathbf{14.66\ GFLOPS}$$

Cuando n→∞ :

$$Rendimiento= \frac{5n+4}{(9+5(n/p)+(p-1))*t_a\ +((112)\ \log_2 p\ +58(p-1))\ t_c} = \frac{5n}{5\left(\frac{n}{p}\right)*0.25ns} = \mathbf{4p\ GFLOPS}$$

c) Proposa una modificació del codi per a millorar la relació R/C del programa. Indica amb claredat què part del codi es substitueix, presenta el codi alternatiu i justifica la millora de la relació R/C del nou codi respecte de l'anterior. (*Propón una modificación del código para mejorar la relación R/C del programa. Indica con claridad qué parte del código se sustituye, presenta el código alternativo y justifica la mejora de la relación R/C del nuevo código respecto del anterior.)*

Se propone una modificación en la que las comunicaciones punto a punto se sustituyen por una operación colectiva MPI_Reduce():

```
MPI_Reduce(&local, &total, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
```

Como después se realiza un MPI_Bcast de la variable total, se podría unir en MPI_Reduce anterior con el MPI_Bcast y realizar un MPI_AllReduce:

```
MPI_AllReduce(&local, &total, 1, MPI_DOUBLE, MPI_SUM, MPI_COMM_WORLD);
```

La relación R/C del algoritmo paralelo inicial viene dada por la expresión:

$$R/C= \frac{(9+5(\mathbf{n/p})+(p-1))*t_a}{((\mathbf{112})\ \mathbf{log_2 p}+58(p-1))\ t_c} = \frac{(\mathbf{A}+(p-1))t_a}{(\mathbf{B}+58(p-1))\ t_c}$$

Donde las variables A y B sustituyen a las expresiones en negrita. Si asumimos que n>>p, se cumple que en general R/C>0.

En cualquier caso, la nueva relación R/C tras introducir Reduce vendría dada por la expresión:

$$R'/c' = \frac{(9+5(n/p)+\log_2 p)*t_a}{((112)\log 2p + 58\log_2 p)\,t_c} = \frac{(A+\log_2 p)t_a}{(B+58\log_2 p)\,t_c}$$

Para comprobar que se produce una mejora se debe cumplir que R'/c'> R/c:

$$\frac{A+\log_2 p}{B+58\log_2 p} > \frac{A+(p-1)}{B+58(p-1)}$$

Operando tenemos que

$(A+\log_2 p)(B+58(p-1)) > (A+(p-1))(B+58\log_2 p) \rightarrow$

$AB+58A(p-1)+B\log_2 p+58(p-1)\log_2 p > AB+58A\log_2 p+(p-1)B+58(p-1)\log_2 p \rightarrow$

$58A(p-1)+B\log_2 p > 58A\log_2 p+(p-1)B \rightarrow$

$58A((p-1)-\log_2 p) > B((p-1)-\log_2 p) \rightarrow$

Dado que para p>2 la expresión $(p-1)-\log_2 p > 0$ podemos simplificar la ecuación, quedándonos que $58A > B$, y por la asunción inicial suponiendo que n>>p, se debe cumplir que $58(9+5(n/p))>112\log_2 p$ o lo que es lo mismo que para que haya una mejora se debe cumplir que p>2 y n debe cumplir la desigualdad $n> ((112/58)\log_2 p - 9)p/5 = (1.94\log_2 p - 9)p/5$

Nota: Indica explícitament les assumpcions que prengues. Es pot utilitzar qualsevol de les funcions MPI que s'indiquen en la taula adjunta. La puntuació dependrà de la correcció del programa i l'optimització de les comunicacions.
Nota: Indica explícitamente las asunciones que tomes. Se puede utilizar cualquiera de las funciones MPI que se indican en la tabla adjunta. La puntuación dependerá de la corrección del programa y la optimización de las comunicaciones.

EJERCICIOS DE ARQUITECTURA DE COMPUTADORES

Se proponía hacer el ejercicio 51 del tema 3 de los apuntes:

51. Dentro de las redes estáticas las topologías n-cubo k-arias son unas de las que poseen mejores propiedades para construir multicomputadores escalables.
    a) Indica cuales con las características de Grado, Diámetro, Número de enlaces (bidireccionales), Bisección y Simetría de este tipo de redes.
    b) Dibuja una red 3-cubo 2-aria y una red 2-cubo 3-aria, numerando de forma dimensional sus nodos.
    c) Dibuja sobre una red 3-cubo 2-aria los caminos que seguirían los mensajes en una implementación basada en árbol de expansión mínimo para una función Broadcast que parta del nodo 0.

Solución

Las explicaciones de cada resultado las podemos ver en el vídeo 41 de Claver, salvo para el número de enlaces totales que en el vídeo solo se cuentan los de una de las dimensiones. El vídeo se encuentra en https://www.youtube.com/watch?v=5K9eUKNfSIM&feature=youtu.be

Parte a)

Grado: es el número de enlaces que salen o entran en cada nodo (salvo que sean distintos los que entran y salen, en cuyo caso hay que distinguir). Como son bidireccionales, hay dos enlaces que salen en cada dimensión en cada nodo, por lo tanto: **Grado=2*n* (*k*>2)**. Si k es 2, entonces solo hay un canal hacia cada dimensión, pues no se cuentan los extremos, por lo que el Grado en ese caso es *n* (la mitad de lo anterior).

Diámetro: Camino más largo en la red suponiendo siempre caminos mínimos. Una forma de recorrer un camino de forma mínima es ir recorriendo cada dimensión por orden, por ejemplo, se puede ir por X y cuando se acaba, seguir por Y, y así sucesivamente hasta llegar al destino. Como es un torus, hay canales de vuelta, por lo que el camino más largo que se puede hacer en cada dimensión es $\lfloor k/2 \rfloor$ como para llegar al destino se deben recorrer las *n* dimensiones: **Diámetro=$n\lfloor k/2 \rfloor$**.

Enlaces bidireccionales: Son todos los enlaces bidireccionales que hay, esto es, agrupación de enlace de entrada y de salida. Hemos visto que cada nodo tiene 2*n* enlaces que salen, y tenemos k^n nodos, por lo que hay $2nk^n$ enlaces en total, pero como nos piden los bidireccionales, agrupamos de dos en dos, por lo que los bidireccionales son la mitad de los totales, por tanto: **Bidireccionales=nk^n (*k*>2)**. Si k es 2 (hipercubo) no contamos el canal de vuelta por los extremos y entonces el número de canales es la mitad que lo anterior.

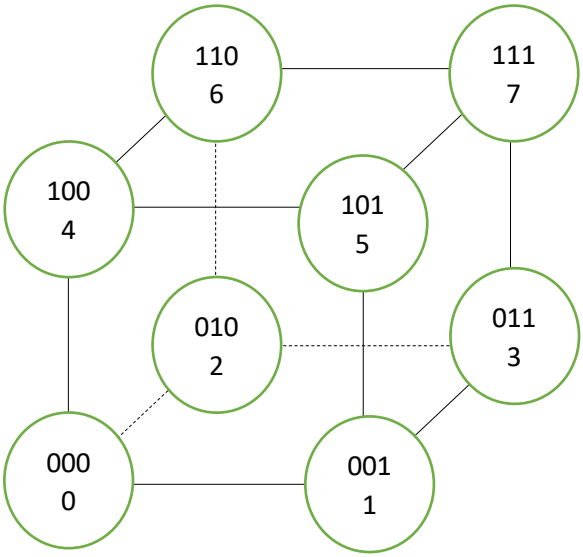
Bisección: Es el número de canales que atraviesan un plano que corta la red en dos partes iguales. Si pensamos en una red malla de dos dimensiones (2D) vemos que los canales bidireccionales que atraviesan un corte longitudinal es justo *k*, si la red es torus, como en nuestro caso, es el doble por los canales de vuelta en los extremos. Si vamos a 3D será *k*k* y 2**k*k* respectivamente, es decir, añadimos un factor *k* por cada dimensión nueva que añadimos, por lo tanto: **Bisección=$2k^{n-1}$ (k>2)**. Si *k* es 2 (hipercubo) entonces la bisección es la mitad, es decir: k^{n-1}.

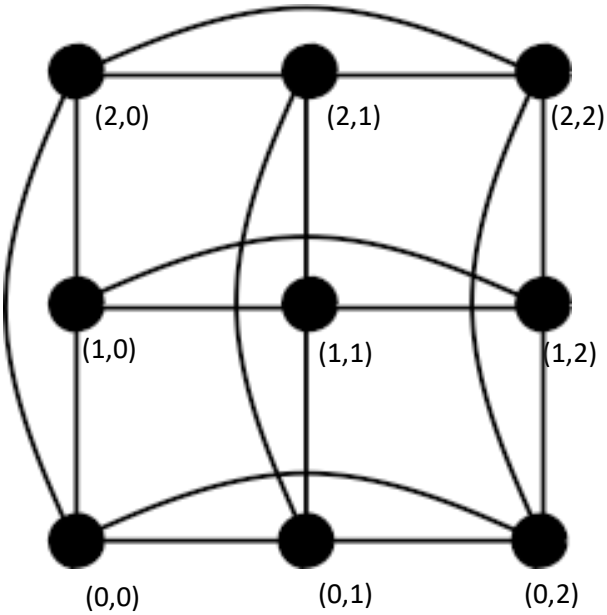
Simetría: Una red es simétrica si se ve igual desde todos sus nodos. **Una red n-cubo k-aria es simétrica**, pues dado un nodo cualquiera, la red se ve siempre igual topológicamente. Podemos coger, por ejemplo, el nodo en una de las esquinas de la red, la red se ve desde este nodo de una determinada manera. A continuación, podemos coger otro nodo y llevarlo, mediante rotaciones del dibujo en cada dimensión a la posición de la esquina que estábamos viendo antes. Podemos observar que el diseño de la red es exactamente el mismo y el nuevo

nodo de la esquina ve la red exactamente igual que el nodo que antes ocupaba esa posición. Esto, por ejemplo, no se podría hacer con la red malla, pues no se puede “rotar” la red.

Parte b)

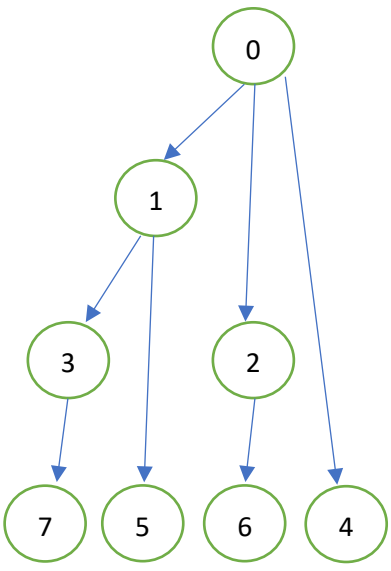
A continuación se presentan las topologías que se pedían con su numeración basada en coordenadas. En el caso del hipercubo (cubo tridimensional binario) al ser las coordenadas 0,1 el resultado es un número binario que puede ponerse también en decimal si suponemos una codificación binaria en potencias de 2. El orden de las dimensiones en las coordenadas puede ser cualquiera, pero debe seguirse el mismo para la misma red. Aquí se ha usado ZYX para el hipercubo y (Y,X) para el cubo bidimensional ternario(2-cubo 3-ario).

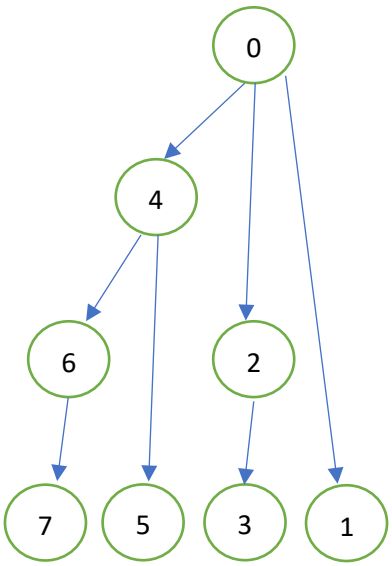




Parte c)

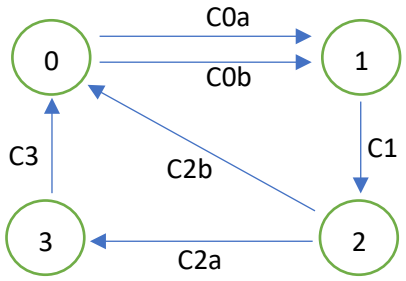
Aquí hay muchas posibilidades, pero sean cuales sean los caminos elegidos, se debe cumplir que debe existir un enlace directo entre los nodos que comuniquemos. Por ejemplo, dado el hipercubo anterior, no es posible enviar directamente un mensaje de 0 a 7, o de 1 a 2, etc. Por ejemplo, podemos seguir el árbol que ya se vio al analizar el coste de la operación Broadcast en MPI, donde cada nodo envía su mensaje al vecino con coordenadas más próximas, o se puede hacer como en el vídeo 41 de Claver donde se envía primero al nodo con coordenadas más lejanas. Estos son los árboles (el de la derecha se corresponde con el vídeo):





ARQUITECTURA DE COMPUTADORES

Se dispone de la siguiente red estática:



1) Dado el siguiente algoritmo de encaminamiento para la anterior red:

```
Procedure alg1(actual, destino) {
      If (actual==destino) canal=CPU;
      Else if (actual==0) canal=selecciona(C0a,C0b);
      Else if (actual==2)
         If (destino==3) canal=C2a;
         Else canal=C2b;
      Else Canal=Cactual;
}
```

Responde a las siguientes cuestiones:

a) ¿Es determinista o adaptativo? Justifica la respuesta.
b) ¿Busca siempre el camino mínimo (minimal)? Justifica la respuesta.
c) Dibuja el Grafo de Dependencia de Canales y señala los ciclos que observes, si los hay.
d) ¿Dirías que el algoritmo está libre de interbloqueos? Justifica la respuesta.

2) Diseña y muestra, en pseudocódigo como el anterior, un algoritmo de encaminamiento **minimal**, **determinista** y **libre de interbloqueos** para dicha red. A partir del algoritmo diseñado:
   a) Justifica por qué el algoritmo es determinista y minimal.
   b) Dibuja el Grafo de Dependencia de Canales y señala los ciclos que observes, si los hay.
   c) Justifica por qué el algoritmo que has diseñado está libre de interbloqueos.

El ejercicio se hará sobre papel, a mano, y se pasará a pdf para subirlo a Aula Virtual.

SOLUCIÓN

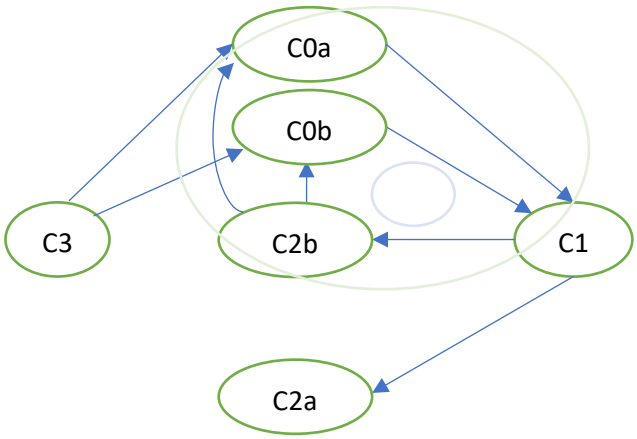
1.a) No es determinista, pues dado un origen y un destino, no siempre se va por el mismo camino (por ejemplo, para ir de 0 a cualquier otro nodo, podemos elegir entre dos caminos). Es adaptativo, pues puedo elegir entre varios caminos. De hecho, si suponemos caminos mínimos,

es completamente adaptativo, pues puedo elegir el camino que quiera de entre todos los posibles. Si no restringiéramos a caminos mínimos, entonces sería parcialmente adaptativo pues, por ejemplo, para ir de 2 a 0, solo me permite un camino de los dos posibles.

1.b) Sí que busca siempre el camino mínimo posible entre dos nodos. La única opción de que fuera por un camino más largo es que se permitiera pasar por 3 cuando va de 2 a 0, pero eso el algoritmo no lo contempla.

1.c) Grafo de dependencias entre canales:

Se observan dos ciclos marcados con óvalos de diferentes colores.



1.d) Aunque el grafo presenta varios ciclos, como es adaptativo, no podemos concluir que va a tener interbloqueos (bloqueos mortales) ni que está libre de ellos. Si fuera determinista, sí que podríamos decir que el algoritmo no está libre de interbloqueos, pero no es el caso.

No obstante, si encontramos una **configuración de interbloqueo**, es decir, una situación donde se produce un interbloqueo, entonces podemos concluir que el algoritmo no está libre de interbloqueos. En este caso es posible imaginar una situación de interbloqueo de la siguiente manera: 1 envía a 0, 2 envía a 1, 0 envía 2 y 3 envía a 2. Con el envío de 1 a 0, ocupamos C1 y luego quiero el C2b, con el envío de 2 a 1, ocupo el canal C2b (bloqueando al paquete que está en C1). Al mismo tiempo, 0 envía a 2 con lo que ocupa C0a (por ejemplo) pero se para pues C1 está ocupado. También al mismo tiempo, el paquete que había salido ya de 3, había llegado a 0 y ahora sale hacia 2, ocupando C0b, pero se bloquea pues C1 está ocupado. En ese instante se produce una configuración de interbloqueo y ningún paquete puede continuar. Por lo tanto, **el algoritmo no está libre de interbloqueos.**

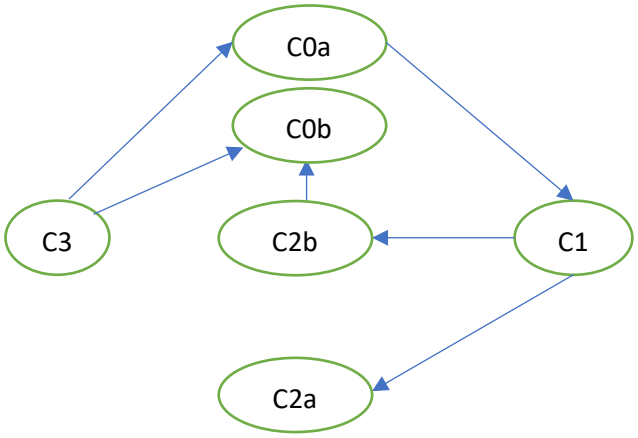
2) Hay más de una solución para obtener un algoritmo libre de interbloqueos dada la red propuesta. Lo más sencillo es crear un algoritmo cuyo grafo de dependencias no tenga ciclos. Para más sencillez se puede optar por un algoritmo determinista que solo nos permita una opción en cada caso. Para romper los dos ciclos del grafo anterior, podemos restringir si ir por C0a o C0b dependiendo del destino. Por ejemplo, si estando en 0, solo permito ir por C0b cuando el destino es 1, desaparece directamente la dependencia entre C0b y C1 rompiéndose el ciclo más pequeño (azul). Pero además, como cualquier paquete que viaje por C2b tiene su destino en 0 o en 1, nunca tomará el enlace C0a, por lo que también desaparece la dependencia entre C2a y C0a, rompiéndose el ciclo externo (verde). El algoritmo sería entonces:

```
Procedure alg2(actual, destino) {
      If (actual==destino) canal=CPU;
      Else if (actual==0)
         If (destino==1) canal=C0b;
         Else canal=C0a;
      Else if (actual==2)
         If (destino==3) canal=C2a;
         Else canal=C2b;
      Else Canal=Cactual;
}
```

2.a) El algoritmo es determinista pues dado un origen y un destino siempre elige el mismo canal de salida, no permite elegir entre varios. Es minimal, pues al igual que antes siempre elige caminos mínimos.

2.b) Grafo de dependencias entre canales. Es como el de antes, pero tal como se comentaba, desaparece la dependencia entre C0b y C1 (si un paquete va por C0b es porque su destino es 1 y no puede seguir por C1). También desaparece la dependencia entre C2b y C0a, pues si un paquete va por C2b es porque, o bien su destino es 0, en cuyo caso no sigue ni por C0a ni por C0b, o bien porque su destino es 1, en cuyo caso va por C0b, nunca por C0a.

El grafo resultante no tiene ciclos:



2.c) Como el grafo resultante no tiene ciclos, podemos concluir que el algoritmo de encaminamiento correspondiente está libre de interbloqueos.

Se proponía hacer el ejercicio 49 del tema 3 de los apuntes:

49. Define analíticamente (mediante una fórmula) la latencia de transporte entre dos nodos separados por una distancia D en una red que utiliza conmutación vermiforme (wormhole), con buffers de almacenamiento para cada una de las salidas de los conmutadores y en las entradas. Asume que la cabecera del paquete consta de 2 phits, mientras que el resto del paquete ocupa 100 phits más; denota el tiempo de encaminamiento por paquete como t_r, el tiempo de conmutación de un phit como $t_s = t_w$, y el ancho de banda como $BW=w/t_w$, donde w es el ancho del canal en bits.
    a) Explica cada uno de los términos en la expresión, utilizando letras para cualquier término que aparezca en la expresión y que no haya sido definido en el enunciado. Además, especifica el tipo de unidades (por ejemplo, segundos) de cada término utilizado.
    b) Indica cómo se modificaría dicha expresión si $t_s = 2 \cdot t_w$

Solución

La solución comentada de este problema la podemos ver en el vídeo 42 de Claver, que se encuentra en https://www.youtube.com/watch?v=nxolOdXqX_k&feature=youtu.be

Parte a)

En la solución vamos a suponer que un flit y un phit ocupan lo mismo. En teoría hemos visto que el envío de un paquete suponiendo lombriz se realiza de forma segmentada, de manera que el coste de envío de un conjunto de datos será el tiempo de envío del primer dato más el resto de datos multiplicado por la etapa más pequeña del cauce.

Tal como vimos en teoría, el envío del primer phit o flit será el tiempo de atravesar el canal (tw) más el tiempo de encaminar o rutar (tr) más el tiempo de atravesar el conmutador o switch (ts) y todo esto multiplicado por el número de enlaces/encaminadores que atraviesa el paquete que es D. De esta manera, el tiempo del primer flit será Tprimer=D(tw+tr+ts).

Sin embargo, la cabecera del paquete de este problema tiene 2 flits en lugar de uno, por lo que, hasta que el router no tiene los dos flits de cabecera no puede enviarla al siguiente. Esto significa que tenemos que tratar la cabecera como un bloque en la segmentación y no podemos segmentar estos dos flits entre sí. Por lo tanto, estos dos flits los tenemos que enviar uno tras otro sin segmentar, por lo que tenemos que sumar los tiempos tw y ts de uno y del otro (el tr no, pues es común a ambos y solo se cuenta una vez). Por lo tanto, el tiempo de la cabecera, en el caso particular de este problema, será Tcab=D(tr+2(tw+ts)).

El resto de flits se envían de forma segmentada y no importa la longitud D del camino, sino la etapa más lenta que será max{tw,ts} que multiplicaremos por el número de flits que quedan por enviar que será $\lceil L/W \rceil$, si suponemos que L es la cantidad de bits de los datos a enviar y W los bits que tiene un flit (aquí tr no influye pues el camino ya está establecido). Con todo esto, la latencia o tiempo total de envío del paquete, para este problema en particular, será:

$$Tlatencia = D\big(tr + 2(tw + ts)\big) + \max\{tw, ts\}\lceil L/W \rceil \qquad (1)$$

En el problema nos dicen que ts=tw por lo que podemos sustituirlo en todas partes y ponerlo todo en función de tw. También nos dicen que los flits que se envían son 100, por lo que L=100W. Sustituimos todo y la expresión de la latencia queda:

$$Tlatencia = D(tr + 4tw) + 100tw$$

O bien,

$$Tlatencia = Dtr + (4D + 100)tw$$

Además de esto, debíamos explicar cada término y dar sus unidades:

Tiempo de encaminamiento o rutado (tr). Es el tiempo que tarda el encaminador (su unidad de control y arbitraje) en establecer el camino entre el puerto de entrada y el de salida. Por un lado, necesita tiempo en tomar la decisión y, por otro, tarda un poco en configurar el conmutador para conectar el puerto de entrada con el de salida.

Tiempo de enlace o canal (tw). Es el tiempo que se gasta en transferir un flit desde un encaminador a otro a través del canal. Tiene relación directa con el ancho de banda del canal BW utilizándose ambos de forma indistinta, dependiendo de cómo se defina la transmisión entre un nodo y otro, ancho del canal, etc.

Tiempo de conmutador (s). Es el tiempo que le cuesta a un flit atravesar el encaminador del buffer de entrada al de salida.

Longitud del camino (D). Es el número de enlaces que atraviesa un paquete entre un origen y un destino. Es adimensional.

La unidad para medir el tiempo es el segundo, pero dado que los retrasos implicados en la conmutación son del orden de nanosegundos, se suele utilizar el nanosegundo como unidad.

El ancho de banda se mide en bits por segundo o bytes por segundo. También por lo anterior, es bastante frecuente utilizar Gigas o Megas (bits o bytes) por segundo.

Parte b)

En este caso se nos pide obtener la expresión de la latencia suponiendo que ts=2tw (el tiempo de atravesar el conmutador es el doble de lento que el del enlace). Para ello partimos de la expresión inicial de la latencia (1) y sustituimos ts, con lo que nos queda:

$$Tlatencia = D\big(tr + 2(tw + 2tw)\big) + \max\{tw, 2tw\}\lceil L/W \rceil$$

Sustituyendo L por su valor obtendremos:

$$Tlatencia = D(tr + 6tw) + 200tw$$

O bien,

$$Tlatencia = Dtr + (200 + 6D)tw$$