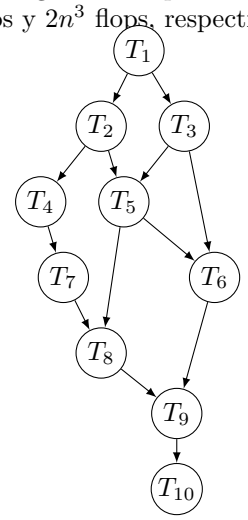
Cuestión 1 (1.2 puntos)

Se quiere paralelizar el siguiente fragmento de código mediante MPI con 2 procesos. Asumimos que `n` es una constante predefinida. En todas las llamadas, el segundo argumento (el que va a continuación de `n`) es de entrada-salida, lo cual induce las dependencias entre tareas. Se proporciona el grafo de dependencias asociado. El coste computacional de las funciones `f1`, `f2` y `f3` es de $\frac{1}{3}n^3$ flops, n^3 flops y $2n^3$ flops, respectivamente.

```
double A[n][n], B[n][n], C[n][n], D[n][n],
       E[n][n], F[n][n];

f1(n,A);      /* Tarea T1 */
f2(n,D,A);    /* Tarea T2 */
f2(n,F,A);    /* Tarea T3 */
f2(n,B,D);    /* Tarea T4 */
f3(n,E,F,D);  /* Tarea T5 */
f3(n,C,E,F);  /* Tarea T6 */
f1(n,B);      /* Tarea T7 */
f2(n,E,B);    /* Tarea T8 */
f3(n,C,E,E);  /* Tarea T9 */
f1(n,C);      /* Tarea T10 */
```



0.9 p.

(a) Implementa la versión paralela con MPI, utilizando únicamente primitivas de comunicación punto a punto. Se puede suponer que inicialmente todas las matrices contienen valores válidos iguales en todos los procesos. No es necesario que las matrices calculadas se recojan en ninguno de los procesos. La asignación escogida debe hacer, en primer lugar, que el cálculo esté repartido razonablemente entre los dos procesos y, en segundo lugar, que se minimicen las comunicaciones entre ellos.

Solución: Para repartir el cálculo entre los dos procesos, asignaremos las tareas T_2 y T_3 a procesos distintos. También asignaremos T_4 y T_7 a un proceso y T_5 a otro (dado que el coste de T_4 y T_7 juntas es menor que el de T_5). De la misma forma, asignaremos T_6 y T_8 a procesos distintos.

Para minimizar las comunicaciones, deberían asignarse las tareas T_2, T_4, T_7 y T_8 al mismo proceso, mientras el otro haría las tareas T_3, T_5 y T_6. Además, se replica la tarea T_1 para evitar el envío de un mensaje.

De acuerdo con esto, la asignación podría ser:

P_0: $T_1, T_2, T_4, T_7, T_8, T_9, T_{10}$.

P_1: T_1, T_3, T_5, T_6.

```
int rank;
MPI_Comm_rank(MPI_COMM_WORLD, &rank);

if (rank==0) {
  f1(n,A);     /* Tarea T1 */
  f2(n,D,A);   /* Tarea T2 */
  MPI_Send(D, n*n, MPI_DOUBLE, 1, 0, MPI_COMM_WORLD);
  f2(n,B,D);   /* Tarea T4 */
  f1(n,B);     /* Tarea T7 */
  MPI_Recv(E, n*n, MPI_DOUBLE, 1, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
  f2(n,E,B);   /* Tarea T8 */
  MPI_Recv(C, n*n, MPI_DOUBLE, 1, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
  f3(n,C,E,E); /* Tarea T9 */
  f1(n,C);     /* Tarea T10 */
} else if (rank==1) {
  f1(n,A);     /* Tarea T1 */
  f2(n,F,A);   /* Tarea T3 */
  MPI_Recv(D, n*n, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
```

```
    f3(n,E,F,D); /* Tarea T5 */
    MPI_Send(E, n*n, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD);
    f3(n,C,F,F); /* Tarea T6 */
    MPI_Send(C, n*n, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD);
  }
```

0.3 p. (b) Calcula el coste secuencial y el coste paralelo.

Solución: Coste secuencial:

$$t(n) = 3 \cdot \frac{1}{3}n^3 + 4 \cdot n^3 + 3 \cdot 2n^3 = 11n^3 \text{ flops}$$

Coste paralelo:

$t_a(n,2) = \frac{1}{3}n^3 + n^3 + 2n^3 + 2n^3 + 2n^3 + \frac{1}{3}n^3 = (7 + \frac{2}{3})n^3$ flops
$t_c(n,2) = 3(t_s + n^2 t_w)$
$t(n,2) = t_a(n,2) + t_c(n,2) = (7 + \frac{2}{3})n^3$ flops $+ 3(t_s + n^2 t_w)$

Cuestión 2 (1.2 puntos)

La siguiente función implementa la siguiente expresión $y = A * (A * x)$, donde A es una matriz cuadrada de dimensión N y x un vector columna de la misma dimensión.

```
void fun1(double A[N][N], double x[], double y[]) {
    int i,j;
    double z[N];

    for (i=0;i<N;i++) {
       z[i]=0.0;
       for (j=0;j<N;j++)
          z[i]+=A[i][j]*x[j];
    }
    for (i=0;i<N;i++) {
       y[i]=0.0;
       for (j=0;j<N;j++)
          y[i]+=A[i][j]*z[j];
    }
}
```

0.8 p. (a) Implementa una versión paralela mediante MPI, asumiendo que los datos de entrada se encuentran en el proceso 0 y que los resultados deben de encontrarse completos en dicho proceso al final de la ejecución. Se puede asumir que el tamaño del problema es un múltiplo del número de procesos.

Solución:

```
void fun1_par(double A[N][N], double x[], double y[]) {
    int p, np;
    int i,j;
    double zlcl[N];
    double Alcl[N][N];

    MPI_Comm_size(MPI_COMM_WORLD, &p);

    np = N/p;
    MPI_Scatter(A, np*N, MPI_DOUBLE, Alcl, np*N, MPI_DOUBLE, 0, MPI_COMM_WORLD);
    MPI_Bcast(x, N, MPI_DOUBLE, 0, MPI_COMM_WORLD);

    for (i=0;i<np;i++) {
       zlcl[i]=0.0;
       for (j=0;j<N;j++)
          zlcl[i]+=Alcl[i][j]*x[j];
    }
    MPI_Allgather(zlcl, np, MPI_DOUBLE, y, np, MPI_DOUBLE, MPI_COMM_WORLD);
    for (i=0;i<np;i++) {
       zlcl[i]=0.0;
```

```
            for (j=0;j<N;j++)
               zlcl[i]+=Alcl[i][j]*y[j];
         }
         MPI_Gather(zlcl, np, MPI_DOUBLE, y, np, MPI_DOUBLE, 0, MPI_COMM_WORLD);
      }
```

0.4 p.

(b) Calcula la expresión del tiempo paralelo, así como el Speed Up y la Eficiencia. Calcula también los valores de Speed Up y eficiencia cuando el tamaño del problema (N) tiende a infinito. Indicar por separado el coste de cada operación de comunicación realizada, así como el tiempo aritmético.

Solución: Se asume una implementación de Allgather en la que se realiza un Gather en un proceso y luego un Broadcast de longitud N.

$$t(N) = \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} 2 + \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} 2 = 2N^2 + 2N^2 = 4N^2$$

$$t(N,p) = t_{Scatter} + t_{Bcast} + t_{a1}(N,p) + t_{Allgather} + t_{Gather} + t_{a2}(N,p)$$

$$t_{Scatter} = (p-1)(t_s + \frac{N}{p}Nt_w)$$

$$t_{Bcast} = (p-1)(t_s + Nt_w)$$

$$t_{Allgather} = (p-1)(t_s + \frac{N}{p}t_w) + (p-1)(t_s + Nt_w)$$

$$t_{Gather} = (p-1)(t_s + \frac{N}{p}t_w)$$

$$t_a(N,p) = 2\sum_{i=0}^{\frac{N}{p}-1}\sum_{j=0}^{N-1} 2 = \frac{4N^2}{p}$$

$$t(N,p) = (p-1)(t_s + \frac{N}{p}Nt_w) + (p-1)(t_s + Nt_w) + \sum_{i=0}^{\frac{N}{p}-1}\sum_{j=0}^{N-1} 2 + (p-1)(t_s + \frac{N}{p}t_w) +$$

$$+(p-1)(t_s + Nt_w) + \sum_{i=0}^{\frac{N}{p}-1}\sum_{j=0}^{N-1} 2 + (p-1)(t_s + \frac{N}{p}t_w)$$

$$t(N,p) \approx 5pt_s + (N^2 + 2pN + 2N)t_w + 4\frac{N^2}{p} = 5pt_s + (N^2 + (2p+2)N)t_w + 4\frac{N^2}{p}$$

$$S(N,p) = \frac{t(N)}{t(N,p)} = \frac{4N^2}{5pt_s + (N^2 + (2p+2)N)t_w + 4\frac{N^2}{p}}$$

$$\lim_{N\to\infty} S(N,p) = \lim_{N\to\infty} \frac{t(n)}{t(n,p)} = \frac{4}{t_w + \frac{4}{p}}$$

$$E(N,p) = \frac{S(N,p)}{p} = \frac{4N^2}{p(5pt_s + (N^2 + (2p+2)N)t_w + 4\frac{N^2}{p})}$$

$$\lim_{N\to\infty} E(N,p) = \frac{4}{pt_w + 4}$$

Cuestión 3 (1.1 puntos)

Sea una matriz $A \in R^{n^2\times n^2}$ con bloques $A_i \in R^{n\times n}$, $i = 0, 1, \ldots, n-1$, situados a lo largo de la diagonal principal de la matriz A, tal como se muestra en la siguiente figura, siendo el resto de los elementos iguales a 0:

$$A = \begin{pmatrix} A_0 & 0_{n\times n} & \cdots & 0_{n\times n} \\ 0_{n\times n} & A_1 & \cdots & 0_{n\times n} \\ & & \ddots & \vdots \\ 0_{n\times n} & 0_{n\times n} & \cdots & A_{n-1} \end{pmatrix}.$$

0.9 p.

(a) Implementa mediante MPI la función `copia_bloques`, usando tipos derivados para reducir el número de mensajes enviados. Se supone que la matriz A se encuentra almacenada en el proceso P_0 y debe ser repartida

entre todos los procesos, de manera que la matriz local B del proceso P_i debe contener a la matriz A_i, $i = 0, 1, \ldots, n-1$. Suponed que el número de procesos es mayor o igual a n.

```
void copia_bloques(double A[n*n][n*n], double B[n][n]){
        ..........
}
```

Ejemplo de una matriz A de dimensión 9×9 con 3 procesos:

$$A = \begin{pmatrix} 0 & 1 & 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 3 & 4 & 5 & 0 & 0 & 0 & 0 & 0 & 0 \\ 6 & 7 & 8 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 9 & 10 & 11 & 0 & 0 & 0 \\ 0 & 0 & 0 & 12 & 13 & 14 & 0 & 0 & 0 \\ 0 & 0 & 0 & 15 & 16 & 17 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 18 & 19 & 20 \\ 0 & 0 & 0 & 0 & 0 & 0 & 21 & 22 & 23 \\ 0 & 0 & 0 & 0 & 0 & 0 & 24 & 25 & 26 \end{pmatrix}$$

$$B(P_0) = \begin{pmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \end{pmatrix}, B(P_1) = \begin{pmatrix} 9 & 10 & 11 \\ 12 & 13 & 14 \\ 15 & 16 & 17 \end{pmatrix}, B(P_2) = \begin{pmatrix} 18 & 19 & 20 \\ 21 & 22 & 23 \\ 24 & 25 & 26 \end{pmatrix}.$$

Solución:

```
void copia_bloques(double A[n*n][n*n], double B[n][n]){
  int i, rank;
  MPI_Status stat;
  MPI_Datatype ntype;
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Type_vector(n,n,n*n,MPI_DOUBLE,&ntype);
  MPI_Type_commit(&ntype);
  if (rank==0){
    for(i=1; i<n; i++)
      MPI_Send(&A[i*n][i*n], 1, ntype, i, 100, MPI_COMM_WORLD);
    MPI_Sendrecv(&A[0][0], 1, ntype, 0, 100, B, n*n, MPI_DOUBLE,
         0, 100, MPI_COMM_WORLD, &stat);
  }
  else if(rank<n)
    MPI_Recv(B, n*n, MPI_DOUBLE, 0, 100, MPI_COMM_WORLD, &stat);
  MPI_Type_free (&ntype);
}
```

0.1 p. (b) Calcula el tiempo de comunicaciones de la función `copia_bloques`.

Solución: Como el proceso P_0 tiene que enviar el bloque A_i al proceso P_i, $i = 1, \ldots, n-1$, el número total de mensajes será $n-1$, y como en cada mensaje se envían $n \times n$ datos, el tiempo de comunicaciones es

$$t_c = (n-1)(t_s + n^2 t_w).$$

0.1 p. (c) Calcula el tiempo de comunicaciones, suponiendo ahora que no se han usado tipos derivados en las comunicaciones punto a punto.

Solución: En este caso, el proceso P_0 tiene que enviar cada una de las n filas de A_i en mensajes separados al proceso P_i, $i = 1, \ldots, n-1$, luego el número total de mensajes será $(n-1)n$, y como en cada mensaje se envían n datos, el tiempo de comunicaciones es

$$t_c = (n-1)n(t_s + nt_w).$$