1. Suponga que el comunicador MPI_COMM_WORLD está compuesto por tres procesos 0, 1, y 2 y sobre ellos se ejecuta el siguiente código en MPI:

```
int x, y, z;
Int rank;
MPI_Comm_rank(MPI_COMM_WORLD, &mi_rank);
switch (mi_rank) {
        case 0:
                x = 0; y = 1; z = 2;
                MPI_Bcast(&x, 1, MPI_INT, 0, MPI_COMM_WORLD);
                (*) MPI_Send(&y, 1, MPI_INT, 2, 43, MPI_COMM_WORLD);
                MPI_Bcast(&z, 1, MPI_INT, 1, MPI_COMM_WORLD);
                (...)
                .....
                break;
        case 1:
                x = 3; y = 4; z = 5;
                MPI_Bcast(&x, 1, MPI_INT, 0, MPI_COMM_WORLD);
                MPI_Bcast(&y, 1, MPI_INT, 1, MPI_COMM_WORLD);
                break;
        case 2:
                x = 6; y = 7; z = 8;
                MPI_Bcast(&z, 1, MPI_INT, 0, MPI_COMM_WORLD);
                MPI_Recv(&x, 1, MPI_INT, 0, 43, MPI_COMM_WORLD);
                MPI_Bcast(&y, 1, MPI_INT, 1, MPI_COMM_WORLD);
                break;
}
printf (“ x = %d y = %d z = %d\n”,x,y,z);1.
```

a) ¿Cuáles son los valores de x, y, z para cada proceso después de cada llamada? Indícalo de forma justificada con una tabla que muestre los valores de las variables en cada proceso después de cada sentencia .[1.5]

	Proceso 0	Proceso 1	Proceso 2
Sentencia 1	X=0, y=1, z=2 Ya que este proceso es raiz del broadcast, no cambian ninguno de los valores	X=0, y=4, z=5 El broadcast recibe en x el valor de x del proceso 1, que es 0	X=6, y=7, z=0 El broadcast recibe en z el valor de x del proceso 0, que es 0
Sentencia 2	X=0, y=1, z=2 Un send no altera los valores	X=0, y=4, z=5 Al ser raiz del broadcast no cambia ningun valor	X=1, y=7, z=0 El recv recibe en x el valor de y del proceso 0 que es 1
Sentencia 3	X=0, y=1, z=4 El broadcast recibe en z el valor de y del proceso 1, que es 4	---	X=1, y=4, z=0 El broadcast recibe en y el valor de y del proceso 1, que es 4

b) Se desea realizar una modificación del código anterior que consiste en trasladar la línea marcada por (*) y situarla sobre la línea punteada. ¿Es aún posible la ejecución del código? ¿Se obtendrían los mismos resultados? Justifique su respuesta. [0.5]

El código puede ejecutarse sin problemas y los resultados no varían ya que aunque MPI_Send es bloqueante, MPI_Bcast no lo es, por lo que se evita así el interbloqueo

2. Dada la malla de procesos 4x4 que se muestra a continuación, en la que cada proceso aparece con su rango en el comunicador MPI_COMM_WORLD. Escribir una porción de código C basado en el uso de la función:

*int MPI_Comm_split(MPI_Comm comm, int color, int key, MPI_Comm *newcomm)*

que permita obtener un nuevo comunicador (new_comm) que incluya a los procesos que aparecen con fondo gris en la figura pero cuyos rangos se ordenen de forma inversamente proporcional al rango que tenían en comunicador original. Utilizar el menor número de órdenes posible. [1.6]

a)

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

b)

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

a)

```
//...
MPI_Comm nuevocom;
int rank;
MPI_Comm_rank(MPI_COMM_WORLD,&rank);
int color = (rank/4)%2
MPI_Comm_split(MPI_COMM_WORLD, color, -rank, &nuevocom);
//...
```

b)

```
//...
MPI_Comm nuevocom;
int rank;
MPI_Comm_rank(MPI_COMM_WORLD,&rank);
int color = ((rank/4) + (rank % 4)) % 2
MPI_Comm_split(MPI_COMM_WORLD, color, -rank, &nuevocom);
//...
```

3. Considérese una matriz bidimensional cuadrada con 300 x 300 enteros que se reparte entre 9 procesadores siguiendo una distribución cíclica por bloques. Especifica los 4 parámetros (Px, Py, mbx, mby) que definen la distribución particular de la matriz entre los 9 procesadores en cada uno de estos casos [2]:

a) Distribución cíclica por columnas.

$n=300 \quad m=300 \quad p=9$

Usamos una malla de procesos $1\times p$, por lo que:

$p_x=1 \quad p_y=9$
$m_{bx}=\lceil n/p_x \rceil=\lceil 300/1 \rceil=300 \quad m_{by}=\lceil m/p_y \rceil=\lceil 300/9 \rceil=34$

b) Distribución cíclica por filas.

$n=300 \quad m=300 \quad p=9$

Usamos una malla de procesos $p\times 1$, por lo que:

$p_x=9 \quad p_y=1$
$m_{bx}=\lceil n/p_x \rceil=\lceil 300/9 \rceil=34 \quad m_{by}=\lceil m/p_y \rceil=\lceil 300/1 \rceil=300$

c) Distribución por bloques estándar.

$n=300 \quad m=300 \quad p=9$

Usamos una malla de procesos $\sqrt{p}\times\sqrt{p}$, por lo que:

$p_x=3 \quad p_y=3$
$m_{bx}=\lceil n/p_x \rceil=\lceil 300/3 \rceil=100$
$m_{by}=\lceil n/p_y \rceil=\lceil 300/3 \rceil=100$

d) Distribución por bloques de 3 filas.

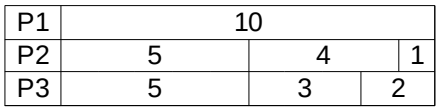
$n=300 \quad m=300 \quad p=9$

Usamos una malla de procesos $p\times 1$, por lo que:

$p_x=9 \quad p_y=1$
$m_{bx}=3 \quad m_{by}=\lceil m/p_y \rceil=\lceil 300/1 \rceil=300$

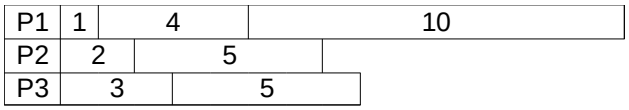
4. Disponemos de 7 tareas que deben ser ejecutadas sobre 3 procesadores idénticos con tiempos de ejecución 1, 2, 3, 4, 5, 5 y 10 unidades, respectivamente. Asumiendo que encargar la realización de una tarea a un procesador no conlleva ningún coste, calcular el speedup que se obtendría en el peor caso y en el mejor caso, cuando se utiliza un esquema de asignación dinámica para planificar la ejecución de las tareas en los tres procesadores

Mejor caso



$$Speedup = \frac{T_{sec}}{T_{par}} = \frac{30}{10} = 3$$

Peor caso



$$Speedup = \frac{T_{sec}}{T_{par}} = \frac{30}{15} = 2$$

5. Responda V (Verdadero) o F (Falso) en el recuadro a la izquierda de cada pregunta.

F	Al distribuir por bloques una malla de diferencias finitas 2D entre varios procesos, los requerimientos de comunicación de cada proceso son generalmente proporcionales a la superficie de bloque de malla asignado a dicho proceso.
V	Al diseñar la estructura de comunicación, interesa que varias operaciones de computación y comunicación se puedan realizar concurrentemente.
V	El grado medio de concurrencia de una descomposición en tareas suele aumentar cuando se hace más fina la granularidad
V	Uno de los objetivos de la etapa de descomposición en tareas es describir el paralelismo potencial que presenta un cálculo.
V	Las distribuciones por bloques de elementos contiguos para matrices se suelen usar en problemas que exhiben una alta localidad espacial y en los que el costo computacional de cada elemento de la matriz es similar
F	Para poder paralelizar un bucle en un programa C que usa OpenMP, el bucle puede expresarse con cualquier formato válido en C
F	Replicar parte de los datos de un programa paralelo reduce el aprovechamiento de los procesadores.
F	Un buen reparto de la carga computacional entre los procesos no garantiza una asignación de tareas a procesos cercana a la solución óptima.
V	Una buena regla general para realizar la descomposición del dominio es centrarse inicialmente en las estructuras de datos de mayor tamaño o en las que se usan más frecuentemente.

6. Se pretende paralelizar un algoritmo iterativo que actúa sobre una matriz cuadrada A=(a ij) con 1000 × 1000 enteros. Inicialmente se asume que la matriz A tiene un valor inicial $A^{(0)}$ (sería el valor de A para la iteración inicial, t=0) y se desea realizar la actualización de A en cada iteración (es decir, pasar del valor en la iteración t al valor en t+1). La actualización de cada elemento a ij de A en la iteración t+1 se lleva acabo a partir de los valores de A en la iteración anterior t, usando la siguiente expresión:

$$a\,ij^{(t+1)} = a\,ij^{(t)} - 2\,ai{+}1\,j^{(t)} - a\,ij{+}1^{(t)} - aij - 1^{(t)} + 2\,ai - 1\,j^{(t)}$$

La fórmula de diferencias finitas se aplica a cada elemento a ij de A mientras a ij cumpla una condición que tiene un costo computacional constante.
Se asume que en la fase de descomposición se asigna una tarea a la gestión de cada elemento de A.

a) Establecer la distribución más apropiada sobre 4 procesos idénticos cuando se sabe que el número de iteraciones que requiere cada elemento de A no es el mismo para todos los elementos de A, sino que es proporcional a la suma del número de fila y el número de columna que ocupa. Justificar la respuesta con concisión.

Ya que la carga computacional no es constante, la submatriz de cada proceso será de un tamaño distinto. Como el numero de iteraciones de un elemento es proporcional a la fila y a la columna de este, entonces la carga computacional de una submatriz será proporcional a la suma de la fila y la columna de todos sus elementos.

Siguiendo este principio he diseñado un programa que parte de una matriz en la que cada elemento es la suma de su fila y su columna y va dividiéndola en todas las 4 submatrices posibles buscando la division que cause que las sumas de cada submatriz difieran lo menor posible.

Tras ejecutar este programa he encontrado que la division mas optima seria la siguiente:

- Una primera matriz de 618 x 618 que agrupa los elementos con menos carga computacional.
- Dos matrices 618 x 382 y 382 x 618 que agrupan los elementos con una carga de computación media.
- Una ultima matriz 382 x 382 que agrupa los elementos con mas carga computacional.

Aquí una representación a menor escala de la división realizada:

0	1	2	3	4	5	6	7	8	9
1	2	3	4	5	6	7	8	9	10
2	3	4	5	6	7	8	9	10	11
3	4	5	6	7	8	9	10	11	12
4	5	6	7	8	9	10	11	12	13
5	6	7	8	9	10	11	12	13	14
6	7	8	9	10	11	12	13	14	15
7	8	9	10	11	12	13	14	15	16
8	9	10	11	12	13	14	15	16	17
9	10	11	12	13	14	15	16	17	18

b) Establecer la distribución más apropiada sobre 4 procesos idénticos cuando se sabe que el número de iteraciones que requiere cada elemento de A no es el mismo para todos los elementos de A, sino que es

proporcional al número de fila que ocupa. Justificar la respuesta con concisión.

Ya que la carga depende del numero de filas, se podría realizar una división por bloques de columnas y la carga quedaría muy bien equilibrada, pero esto aumentaría las comunicaciones. Por eso la solución mas optima consistiría en dos submatrices superiores de gran tamaño que acumulen la mayoría de posiciones con baja carga computacional y dos submatrices inferiores de menor tamaño pero con los elementos con mas carga computacional.

Para calcular la posición mas optima he vuelto a realizar otro programa y esta vez el resultado obtenido ha sido que las dos matrices superiores deben ser de tamaño 707 x 500 y las inferiores de tamaño 293 x 500

Aquí una representación en menor escala de la solución:

0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3	3	3
4	4	4	4	4	4	4	4	4	4
5	5	5	5	5	5	5	5	5	5
6	6	6	6	6	6	6	6	6	6
7	7	7	7	7	7	7	7	7	7
8	8	8	8	8	8	8	8	8	8
9	9	9	9	9	9	9	9	9	9