Nombre:	ALEJANDRO FERNÁNDEZ TRIGO
Fecha y horario del grupo:	09/05/22 12:30

ARQUITECTURA DE SISTEMAS DISTRIBUIDOS. 3º GII-TI. PRÁCTICA 7
PROGRAMACIÓN DE COMPUTADORES DE MEMORIA DISTRIBUIDA USANDO MPI.

RESULTADOS OBTENIDOS POR EL ALUMNO

Ejercicio 1:

Líneas adicionales añadidas al programa o líneas que se han modificado:	int mi_rango, ..., numero; char nombre[100]; MPI_Get_processor_name(nombre, &numero); printf("... corriendo en %s", nombre); ↳ en ambos printfs
¿Se puede lanzar un número de procesos mayor que el número de núcleos del procesador utilizado? ¿Qué ocurre en ese caso? 8 núcleos	No; causa un fallo y MPI informe de ello al no haber suficientes slots disponibles. ↓ "not enough slots available" Teóricamente, puede forzarse y el proceso se "repartiría" entre núcleos. (--oversubscribe)

Ejercicio 2:

¿Qué ocurre al ejecutar el programa "saludos" sólo para un proceso? ¿Por qué?	No hay ningún mensaje que transmitir ya que solo hay UN proceso. Es la misma razón por la que al ejecutar con mpirun -np 4 saludos, solo hay 3 mensajes. Solo un núcleo está enviando y ninguno recibiendo.

ALEJANDRO PDEZ. TRIGO

Ejercicio 3: con todos los cabeceros anteriores

Código fuente del programa:

```
dest = (mi_rango + 1 + P) % P;
sprintf(mensaje, "Saludos desde %d!", mi_rango);
MPI_Send(mensaje, strlen(mensaje)+1, MPI_CHAR, dest,
         etiqueta, MPI_COMM_WORLD);
MPI_Recv(mensaje, 100, MPI_CHAR, (mi_rango - 1 +P)%P,
         etiqueta, MPI_COMM_WORLD, &status);
Printf("%s\n", mensaje);
MPI_Finalize();
```

↳ mpirun -np 4 saludos_emilio

Saludos desde 3!
Saludos desde 0!
Saludos desde 1!
Saludos desde 2!

a) El proceso i, ¿debe enviar primero su mensaje al proceso i+1 y después recibir el mensaje del proceso i-1, o a la inversa (primero recibir y luego enviar)? ¿Importa el orden?	El orden importa ya que al estar en paralelo, no se puede pedir un mensaje que todavía no se ha enviado. No se puede esperar un mensaje que todavía no ha sido enviado.
b) ¿Qué ocurre cuando se ejecuta el programa sobre un único procesador (con un único proceso)?	El proceso origen se envía el mensaje a sí mismo (al 0) y la salida es "Saludos desde 0!".

Ejercicio 4:

a) Funcionamiento del programa con nº elementos = 6:

Nº de procesos: p =1	Sumas parciales= 30.000	Producto escalar = 30.000
Nº de procesos: p =2	Sumas parciales= 5... / 25...	Producto escalar = 30.000
Nº de procesos: p =3	Sumas parciales= 16... /13... /1...	Producto escalar = 30.000

b)

Líneas adicionales añadidas al programa o líneas que se han modificado:

```
int i;
double end, start;
:
MPI_Init(...);
start = MPI_Wtime();
:
:
MPI_Recv(...);
suma_total ...
}
end = MPI_Wtime();
:
printf("....", end-start);
```

→ la diferencia

Tiempo de ejecución del programa (con nº elementos = 2*2*2*2*3*3*5*7*11*13* 50 = 720720* 50) al variar el número de procesos (p) con el que es lanzado desde 1 hasta 6 procesos:

Nº de procesos (p)	Tiempo de ejecución	Aceleración
1	0,290090	1.00
2	0,146015	0,503
3	0,160893	0,908
4	0,088320	0,549
5	0,096839	0,912
6	0,070886	0,732

¿Son lógicos los tiempos de ejecución que se observan?

Claro; el tiempo total se reduce (pq. se está paralelizando la recv) pero al ser un vector grande se producen penalizaciones que van sumando tiempo.

¿Cuáles son los tiempos que intervienen?

El tiempo de procesar las operaciones (instrucciones), el de acceso a la memoria (debido a los fallos), el de comunicación de MPI y el de conmutación entre cada proceso.

ALEJANDRO FDEZ. TRIGO