



# Tutorial 1: *Nuestro primer programa MPI*

- Realizar vuestra primera aplicación paralela, que consistirá en un conjunto de tareas que únicamente visualizarán por pantalla el mensaje de “Hola Mundo Paralelo”, junto con nombre del ejecutable (`argv[0]`), su pid (llamada al sistema `getpid()`), su rango, la máquina en la cual se está ejecutando (`MPI_Get_processor_name`).

Solo el proceso raíz (`rango==0`) imprimirá el número de procesos arrancados por la aplicación paralela.

Acordaros de inicializar MPI antes de hacer cualquier llamada MPI y finalizar MPI antes de acabar el programa.

- Una vez implementada dicha aplicación, probarla con diferentes números de procesos.

# Tutorial 1: *Nuestro primer programa MPI*

- Se pide:


1. Ejecutar la aplicación con 4 procesos, todas ellas ejecutándose en una misma máquina (la asignada por el SGE).
2. Ejecutar la aplicación con n procesos ejecutándose tantos procesos como máquinas disponibles. Asignar únicamente un proceso a cada máquina.

- Entrega:

- Programas fuentes realizados, salida generada por cada uno de los trabajos, makefile y ficheros de configuración (machines) y los scripts de trabajo sge para lanzar el trabajo correspondiente a cada apartado.

- Llamadas involucradas:

- int *MPI\_Init*(int \*argc, char \*\*\*argv),
- *MPI\_Finalize*(),
- *MPI\_Comm\_size* (MPI\_Comm comm, int \*size)
- *MPI\_Comm\_rank* (MPI\_Comm comm, int \*rank)
- int *MPI\_Get\_processor\_name*( char \*name, int \*resultlen)



# Tutorial 2: *Comunicaciones punto a punto en MPI*

- Escribir un programa MPI con 2 tareas que repetidamente se envían N mensajes con M enteros la una a la otra en forma de ping-pong.
  - Utilizar en modo bloqueante de envío y recepción.
  - N = 200 mensajes
  - El tamaño M del mensaje se pasará como parámetro (M=128, por defecto)
- Calcular el tiempo de transmisión y el ancho de banda en función del tamaño del mensaje.
  - Para calcular el tiempo de transmisión de los mensajes, antes de cada transmisión y después de recibir la respuesta correspondiente se obtendrá el tiempo actual (utilizar la funciones de tiempo de MPI, MPI\_WTIME , para medir el tiempo requerido para transmitir un mensaje.).
  - La diferencia entre el tiempo de emisión y el tiempo de recepción dividido entre dos nos dará una aproximación al tiempo de transmisión del mensaje. Calcular el tiempo de transmisión promedio.



# Tutorial 2: *Comunicaciones punto a punto en MPI*

- Se pide:


1. Realizar la aplicación paralela descrita.
2. Analizar como varia el tiempo requerido en función del tamaño del mensaje transmitido (2, 8, 32, 128, 1024 y 8192 elementos).  
Utilizar los parámetros de la aplicación paralela para especificar el tamaño del mensaje a transmitir.
3. Realizar esta pruebas utilizando tanto dos procesos en el mismo nodo (comunicación por memoria compartida), como dos procesos en nodos diferentes (comunicación mediante memoria distribuida)

- Entrega:

- Programas fuentes realizados, script trabajos y makefile.
- Presentar (a ser posible de forma gráfica), comentar y analizar los resultados obtenidos en el apartado 2 y 3.

# Tutorial 3: *Comunicaciones bloqueantes, no bloqueantes, síncronas, con buffer*

- Ampliar el ejemplo anterior para tener  $T$  tareas que están conectadas en forma de anillo y se envían  $N$  mensajes con  $M$  enteros.
  - La tarea  $i$  envía su mensaje a la tarea  $((i+1)\%N)$ .
  - $N = 200$  mensajes
  - El tamaño  $M$  del mensaje se pasará como parámetro ( $M=128$ , por defecto)
- Utilizar los siguientes modos de envío (una versión del ejecutable para cada modo):
  - No Bloqueante.
  - Síncrono.
  - Con buffer.
- Opcionalmente, podéis calcular el tiempo de transmisión promedio y el ancho de banda de envío que se logra con cada método.



# Tutorial 3: *Comunicaciones bloqueantes, no bloqueantes, síncronas, con buffer*

## ■ Se pide:

1. Realizar la aplicación paralela descrita, utilizando comunicaciones no bloqueantes / síncronas y con buffer.
  - Un tipo de comunicación cada vez.
2. (Opcional) Realizar un estudio comparativo del ancho de banda que obtendremos en función del tipo de comunicaciones utilizadas.
  - Justificar las diferencias de rendimiento entre los diferentes tipos de comunicaciones.

## ■ Entrega:

- Programas fuentes realizados y makefile.
- (Opcional) Presentar (a ser posible de forma gráfica), comentar y analizar los resultados obtenidos en el apartado 2.



# Tutorial 4: *Comunicaciones Colectivas*

- Realizar un aplicación paralela que utilizando comunicaciones colectivas realice la suma de todos los elementos de una matriz de enteros de  $N \times M$ . Siendo  $N$  el número de procesos.
  - Asumid  $N=6$  y  $M=10$ .
  - Podéis definir el contenido de la matriz de forma estática en memoria.
- Realizar dos variantes de esta aplicación:
  - a) Utilizando un scatter para distribuir los datos entre las tareas y gather para recolectar los resultados parciales.
  - b) Utilizando una operación de reducción en dos pasos (reducción al distribuir los datos iniciales y reducción sobre los resultados parciales) para realizar las sumas:
    1. Una reducción de las sumas por filas, asignando  $X$  resultados a cada tareas ([MPI\\_REDUCE\\_SCATTER](#))
    2. Una reducción de los resultados parciales ([MPI\\_REDUCE](#)), asignando un resultado final al proceso raíz (rango = 0)



# Tutorial 4: *Comunicaciones Colectivas*

- Se pide:
  1. Realizar las dos versiones de la aplicación paralela descrita.
- Entrega:
  - Programas fuentes realizados y makefile.



# Tutorial 5: *Tipos Derivados*

- Realizar un aplicación paralela que permita dos tareas se envíen las siguientes estructura de datos, utilizando tipos derivados:
  - `struct Particula { float x; float y; float massa; char Nombre[50]; };`
  - submatriz de  $M \times M$  enteros, siendo  $M$  un valor dinámico que puede variar en función de los parámetros introducidos por el usuario.
- Realizar una aplicación que utilice los siguientes tipos derivados con alguna de las estructuras de datos anteriores (la que mejor se ajuste a la estructura de datos):
  - a) `MPI_Type_vector`
  - b) `MPI_Type_struct`
  - c) `Empaquetado / Desempaquetado.`



# Tutorial 5: *Tipos Derivados*

- Se pide:
  1. Realizar el la aplicación paralela anteriormente descrita.
- Entrega:
  - Programas fuentes realizados y makefile.