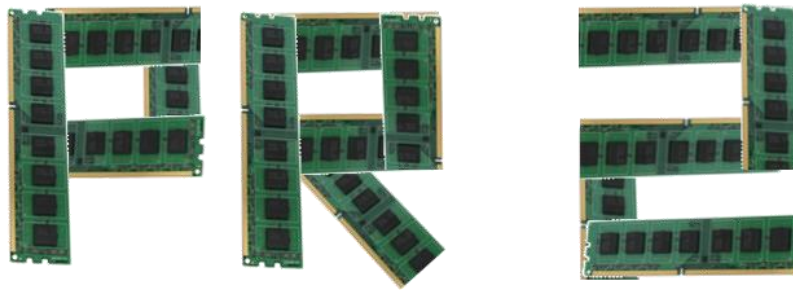


# Computación Paralela con Computadores de Memoria Distribuida usando MPI



**Arquitectura de sistemas y software de base**



4º ISA



Escuela Técnica Superior de  
Ingeniería Informática

**Paula Poley Ceballos**

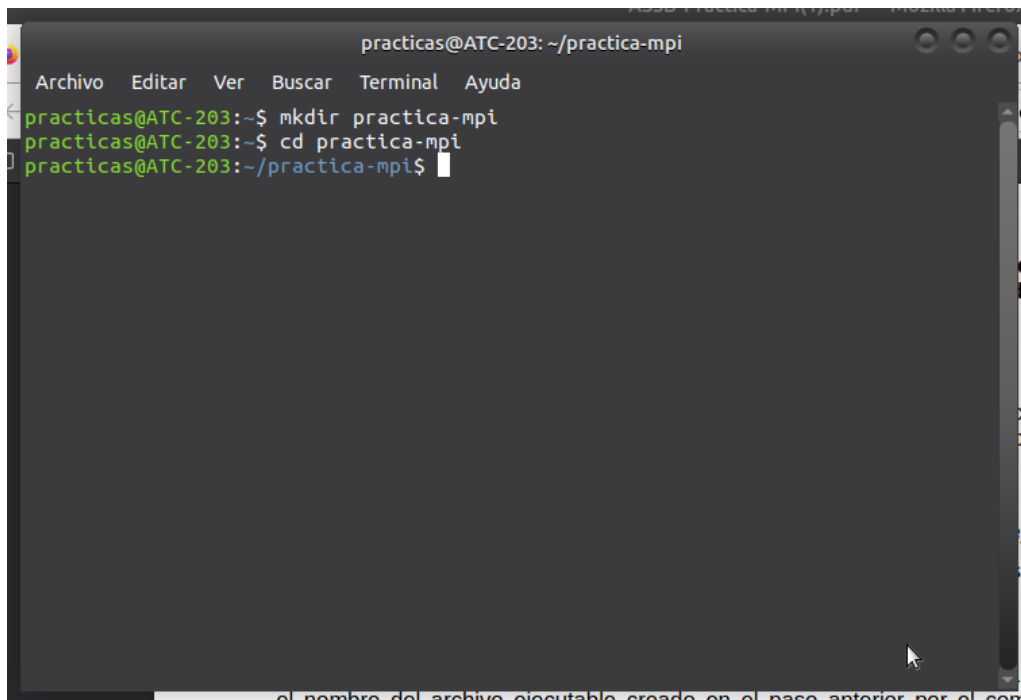
## ÍNDICE

<b>PRIMERA PARTE</b> .....	3
Ejercicio 1.1- Modificar el programa “hola_mundo_avanzado.c” .....	3
Ejercicio 1.2 – Modificar el programa añadiéndole una medición del tiempo de ejecución, usando la función MPI_Wtime. ....	9
a)Funcionamiento del programa con nº elementos = 6: .....	12
b) Líneas adicionales .....	13
<b>SEGUNDA PARTE</b> .....	14
Ejercicio 2.1 – Versión paralela del mismo programa utilizando MPI. Probar el programa obteniendo una estimación del valor de $\pi$ .....	14
Ejercicio 2.2 – Medición del tiempo de ejecución, usando la función MPI_Wtime y representar una gráfica del tiempo y de la aceleración respecto al número de procesos. ....	17
Ejercicio 2.3 – Lanzar la aplicación paralela desarrollada en la segunda parte sobre tres Pcs del aula.....	23

Para resolver todas las actividades indicadas en la primera y segunda parte de la guía de práctica 2 de la asignatura Arquitectura de Sistemas y Software de Base de 4º de Ingeniería de la Salud, antes voy a describir paso a paso lo que he hecho previamente.

Enciendo el ordenador del laboratorio, lo inicio en Linux y abro la consola de comandos. Para poder trabajar me creo una carpeta llamada “practica-mpi” (con los comandos `mkdir nombrecarpeta`) en la que voy a guardar los archivos de texto y todo aquello que sea relevante para dicha práctica.

Para trabajar en el directorio de mi carpeta se pone el comando `cd nombrecarpeta`, en este caso como vemos en la Figura 1, “`cd practica-mpi`.”

A screenshot of a Linux terminal window. The title bar at the top reads 'practicass@ATC-203: ~/practica-mpi'. Below the title bar is a menu bar with the options 'Archivo', 'Editar', 'Ver', 'Buscar', 'Terminal', and 'Ayuda'. The terminal shows three lines of commands and their outputs: 1. 'practicass@ATC-203:~\$ mkdir practica-mpi' followed by a new line. 2. 'practicass@ATC-203:~\$ cd practica-mpi' followed by a new line. 3. 'practicass@ATC-203:~/practica-mpi\$' followed by a cursor. The background of the terminal is dark grey.

el nombre del archivo ejecutable creado en el paso anterior por el comi (Figura 1)

Ya estamos en el directorio de trabajo creado para esta práctica. ¡Empecemos!.

## PRIMERA PARTE

### Ejercicio 1.1- Modificar el programa “hola\_mundo\_avanzado.c”

En el material publicado en la enseñanza virtual de esta práctica se encuentra un programa llamado “hola\_mundo\_avanzado.c” el cual he modificado para que en el mensaje de salida que muestra por pantalla cada proceso incluya al final del nombre del procesador en que está corriendo ese proceso.

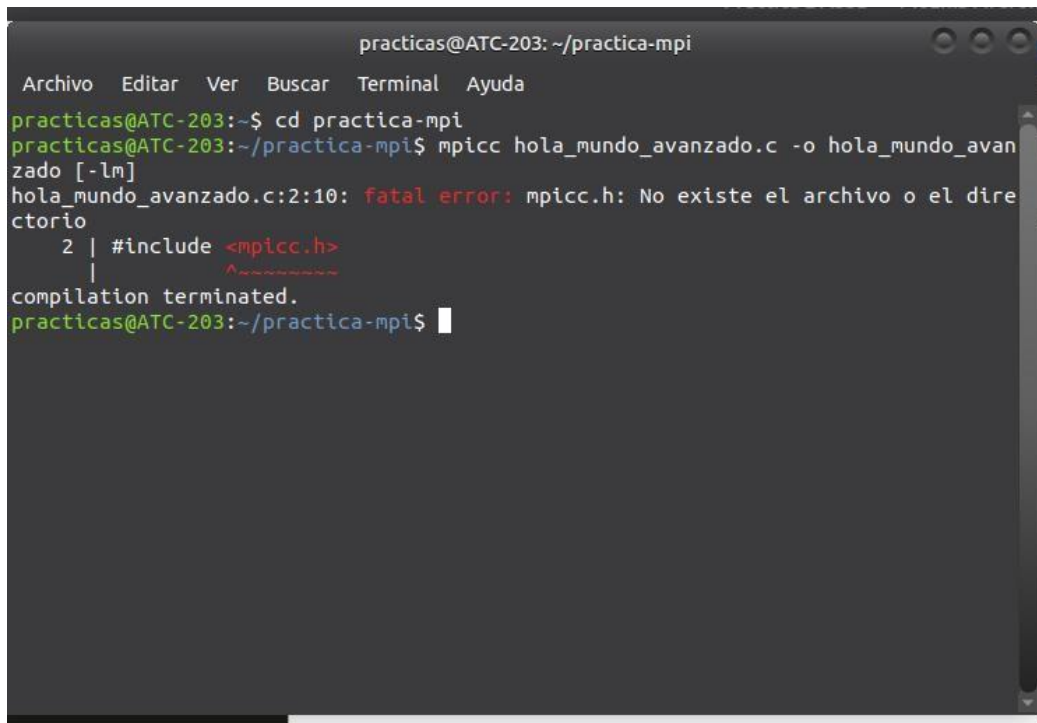
¡OJO! Aquí ya no se habla de hilos porque no comparten nada, ahora se les llama procesos. Cosa que no ocurría en la práctica 1 de esta misma asignatura.

Ya descargado el fichero se va a la consola de mandos y se escribe el comando para compilar el programa . El comando para hacerlo es:

**Compilación:** `mpicc nombre_archivo.c -o nombre_archivo [-lm]`

Siendo [-lm] opcional, este lo que hace es que enlaza el programa con las librerías matemáticas.

Al hacerlo sale un error como se puede ver en la Figura 2, esto es porque no se han instalado todos los paquetes necesarios para ello y por lo tanto habrá que meterlos para poder seguir.

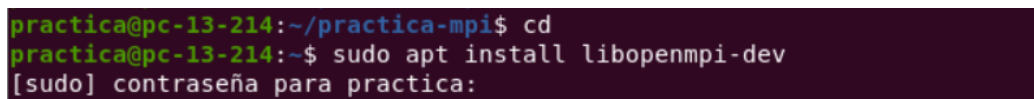
A terminal window titled 'practicass@ATC-203: ~/practica-mpi' with a menu bar (Archivo, Editar, Ver, Buscar, Terminal, Ayuda). The user enters 'cd practica-mpi' and then 'mpicc hola\_mundo\_avanzado.c -o hola\_mundo\_avanzado [-lm]'. The compiler outputs a 'fatal error: mpicc.h: No existe el archivo o el directorio' at line 2, column 10, pointing to an '#include <mpicc.h>' line. The compilation is terminated.

```
practicass@ATC-203: ~/practica-mpi
Archivo Editar Ver Buscar Terminal Ayuda
practicass@ATC-203:~$ cd practica-mpi
practicass@ATC-203:~/practica-mpi$ mpicc hola_mundo_avanzado.c -o hola_mundo_avanzado [-lm]
hola_mundo_avanzado.c:2:10: fatal error: mpicc.h: No existe el archivo o el directorio
   2 | #include <mpicc.h>
     |           ^~~~~~
compilation terminated.
practicass@ATC-203:~/practica-mpi$
```

(Figura 2)

Hay que escribir los siguientes comandos en el directorio raíz para ello escribimos el comando “cd” como podemos ver en la Figura 3.

- sudo apt install libopenmpi-dev
- sudo apt install openmpi-doc
- sudo apt install gcc-doc
- sudo apt install libgomp1

A terminal window showing the user changing to the root directory and installing the necessary MPI development packages.

```
practica@pc-13-214:~/practica-mpi$ cd
practica@pc-13-214:~$ sudo apt install libopenmpi-dev
[sudo] contraseña para practica:
```

(Figura 3)

```

practicass@ATC-203:~$ sudo apt install libopenmpi-dev
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias... Hecho
Leyendo la información de estado... Hecho
Se instalarán los siguientes paquetes adicionales:
  autoconf automake autotools-dev javascript-common libcaf-openmpi-3
  libcoarrays-dev libcoarrays-openmpi-dev libevent-2.1-7 libevent-dev
  libevent-extra-2.1-7 libevent-openssl-2.1-7 libhwloc-dev libhwloc15
  libibverbs-dev libjs-jquery libjs-jquery-ui libltdl-dev libnl-3-dev
  libnl-route-3-dev libnuma-dev libpmix-dev libsigsegv2 libtool m4
Paquetes sugeridos:
  autoconf-archive gnu-standards autoconf-doc apache2 | lighttpd | httpd
  libhwloc-contrib-plugins libjs-jquery-ui-docs libtool-doc openmpi-doc
  gcj-jdk m4-doc
Se instalarán los siguientes paquetes NUEVOS:
  autoconf automake autotools-dev javascript-common libcaf-openmpi-3
  libcoarrays-dev libcoarrays-openmpi-dev libevent-2.1-7 libevent-dev
  libevent-extra-2.1-7 libevent-openssl-2.1-7 libhwloc-dev libibverbs-dev
  libjs-jquery libjs-jquery-ui libltdl-dev libnl-3-dev libnl-route-3-dev
  libnuma-dev libopenmpi-dev libpmix-dev libsigsegv2 libtool m4
Se actualizarán los siguientes paquetes:
  libhwloc15
1 actualizados, 24 nuevos se instalarán, 0 para eliminar y 114 no actualizados.
Se necesita descargar 6.158 kB de archivos.
Se utilizarán 40,3 MB de espacio de disco adicional después de esta operación.
¿Desea continuar? [S/n] S
Des:1 http://es.archive.ubuntu.com/ubuntu jammy/main amd64 libsigsegv2 amd64 2.13-1ubu
ntu3 [14,6 kB]
Des:2 http://es.archive.ubuntu.com/ubuntu jammy/main amd64 m4 amd64 1.4.18-Subuntu2 [1
99 kB]

```

(Figura 4)

```

practicass@ATC-203:~$ sudo apt install openmpi-doc
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias... Hecho
Leyendo la información de estado... Hecho
Se instalarán los siguientes paquetes NUEVOS:
  openmpi-doc
0 actualizados, 1 nuevos se instalarán, 0 para eliminar y 114 no actualizados.
Se necesita descargar 1.072 kB de archivos.
Se utilizarán 4.268 kB de espacio de disco adicional después de esta operación.
Des:1 http://es.archive.ubuntu.com/ubuntu jammy/universe amd64 openmpi-doc all 4.1.2-2
ubuntu1 [1.072 kB]
Descargados 1.072 kB en 1s (1.153 kB/s)
Seleccionando el paquete openmpi-doc previamente no seleccionado.
(Leyendo la base de datos ... 297060 ficheros o directorios instalados actualmente.)
Preparando para desempaquetar .../openmpi-doc_4.1.2-2ubuntu1_all.deb ...
Desempaquetando openmpi-doc (4.1.2-2ubuntu1) ...
Configurando openmpi-doc (4.1.2-2ubuntu1) ...
Procesando disparadores para man-db (2.10.2-1) ...
practicass@ATC-203:~$ sudo apt install gcc-doc
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias... Hecho
Leyendo la información de estado... Hecho
Se instalarán los siguientes paquetes adicionales:
  gcc-11-doc
Se instalarán los siguientes paquetes NUEVOS:
  gcc-11-doc gcc-doc
0 actualizados, 2 nuevos se instalarán, 0 para eliminar y 114 no actualizados.
Se necesita descargar 2.697 kB de archivos.
Se utilizarán 11,7 MB de espacio de disco adicional después de esta operación.
¿Desea continuar? [S/n] S

```

(Figura 5)

Ya se ha instalado todos los paquetes. (Figura 4 y 5)

Repito los pasos anteriores para trabajar en el directorio de mi carpeta.

Compilo el programa y lo ejecuto. Ahora se ve que no hay ningún error y la respuesta que nos da .  
(Figura 6)

```

practicass@ATC-203:~/practica-mpi$ mpicc hola_mundo_avanzado.c -o hola_mundo_avanzado
practicass@ATC-203:~/practica-mpi$ mpirun -np 3 hola_mundo_avanzado
Yo soy el proceso 1 de 3: ¡Hola mundo!
Yo soy el proceso 2 de 3: ¡Hola mundo!
Soy el proceso 0 de 3: ¡Hola mundo!
practicass@ATC-203:~/practica-mpi$

```

(Figura 6)

Utilizando la función `MPI_Get_processor_name` modifiqué el programa dado por el profesor para que el mensaje de salida que muestra por pantalla sea el especificado en el enunciado. Subrayado de rojo en la Figura 7 son todos los cambios con respecto al programa anterior.

```

1 #include <stdio.h>
2 #include <mpi.h>
3
4 int main(int argc, char* argv[]) {
5
6     int mi_rango, tamanno, resultlen;
7     char name[MPI_MAX_PROCESSOR_NAME];
8
9
10    MPI_Init(&argc, &argv); /*Inicializa MPI*/
11    MPI_Comm_rank(MPI_COMM_WORLD, &mi_rango);
12    MPI_Comm_size(MPI_COMM_WORLD, &tamanno);
13    MPI_Get_processor_name(name, &resultlen);
14
15    if( mi_rango == 0)
16        printf("Soy el proceso %i de %i, corriendo en %s: ¡Hola mundo!\n", mi_rango, tamanno, name);
17    else
18        printf("Yo soy el proceso %i de %i, corriendo en %s.\n", mi_rango, tamanno, name);
19
20    MPI_Finalize();
21    return(0);
22
23 }

```

(Figura 7)

Repito los pasos anteriores, compilo con la función “`mpicc hola_mundo_avanzado.c -o hola_mundo_avanzado`” y ejecuto el programa “`mpirun -np 3 hola_mundo_avanzado`”. En esta última función he puesto un 3, es decir, le indico que el número de procesos que debe lanzar la aplicación son 3.

En la figura 8 se muestra el nuevo mensaje por pantalla.

```

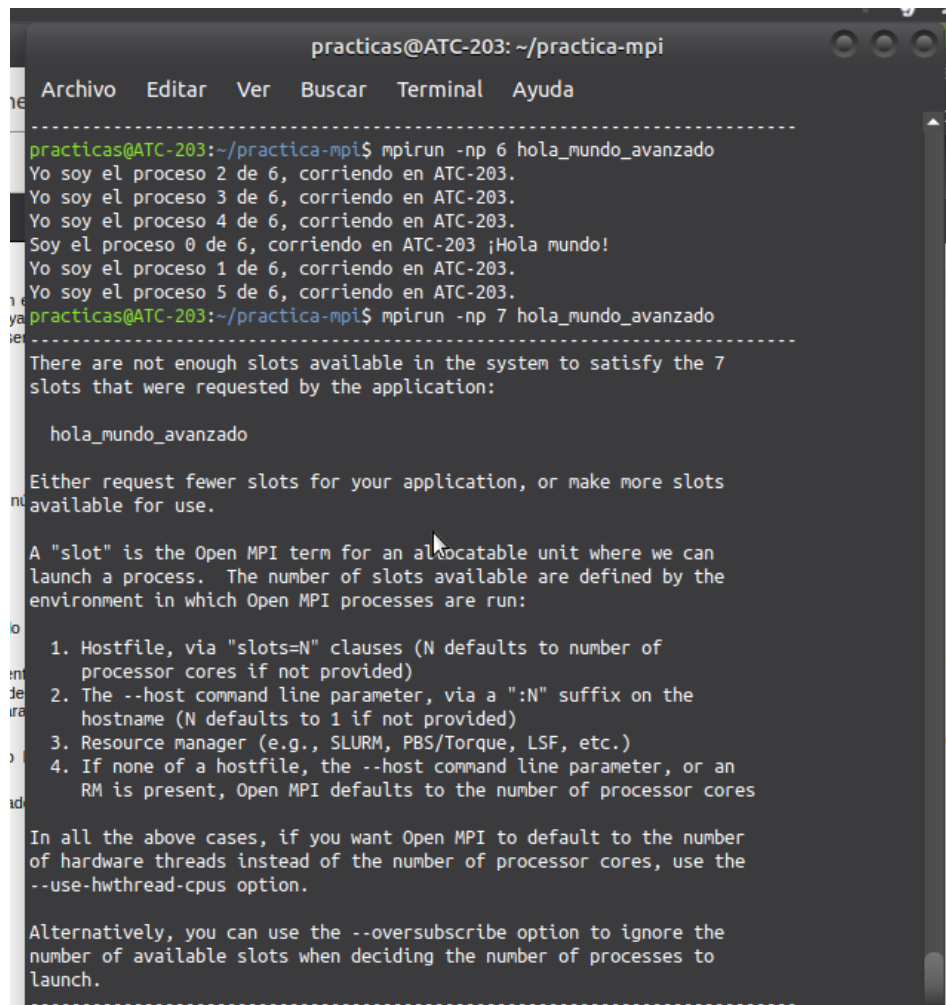
practicass@ATC-203:~/practica-mpi$ mpicc hola_mundo_avanzado.c -o hola_mundo_avanzado
practicass@ATC-203:~/practica-mpi$ mpirun -np 3 hola_mundo_avanzado
Yo soy el proceso 1 de 3, corriendo en ATC-203.
Yo soy el proceso 2 de 3, corriendo en ATC-203.
Soy el proceso 0 de 3, corriendo en ATC-203 ¡Hola mundo!
practicass@ATC-203:~/practica-mpi$

```

(Figura 8)

¿Qué sucede si se lanza un número de procesos que sea mayor que el número de núcleos del procesador utilizado?. Si en vez de 3 procesos como puse en la figura 8 pongo 7 procesos, dicho número de procesos excede del número de núcleos del procesador que en este caso tiene 6 cores físicos (12 cores lógicos) el PC utilizado. Lo que ocurre es que nos sale un mensaje diciendo “No hay suficientes espacios disponibles en el sistema para satisfacer los 7 espacios que solicitó la aplicación: hola\_mundo\_avanzado”.

Esto es porque en algunos sitios el mpi está configurado para que cuando pase el número de cores físicos que tenga la máquina, deja de ejecutarse, como se puede ver en la figura 9. Cuando pongo un número mayor a 6 procesos me sale dicho anuncio.

A screenshot of a terminal window titled 'practicass@ATC-203: ~/practica-mpi'. The window shows the execution of two MPI commands. The first command is 'mpirun -np 6 hola\_mundo\_avanzado', which runs successfully, displaying messages for processes 0 through 5. The second command is 'mpirun -np 7 hola\_mundo\_avanzado', which fails with an error message: 'There are not enough slots available in the system to satisfy the 7 slots that were requested by the application: hola\_mundo\_avanzado'. Below the error message, there is a detailed explanation of what a 'slot' is in the context of Open MPI and a list of four ways to configure the number of slots: 1. Hostfile via 'slots=N' clauses, 2. The '--host' command line parameter, 3. Resource manager (e.g., SLURM, PBS/Torque, LSF, etc.), and 4. If none of the above, Open MPI defaults to the number of processor cores. It also mentions the '--use-hwthread-cpus' option and the '--oversubscribe' option.

```
practicass@ATC-203: ~/practica-mpi
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
-----
practicass@ATC-203:~/practica-mpi$ mpirun -np 6 hola_mundo_avanzado
Yo soy el proceso 2 de 6, corriendo en ATC-203.
Yo soy el proceso 3 de 6, corriendo en ATC-203.
Yo soy el proceso 4 de 6, corriendo en ATC-203.
Soy el proceso 0 de 6, corriendo en ATC-203 ¡Hola mundo!
Yo soy el proceso 1 de 6, corriendo en ATC-203.
Yo soy el proceso 5 de 6, corriendo en ATC-203.
practicass@ATC-203:~/practica-mpi$ mpirun -np 7 hola_mundo_avanzado
-----
There are not enough slots available in the system to satisfy the 7
slots that were requested by the application:

  hola_mundo_avanzado

Either request fewer slots for your application, or make more slots
available for use.

A "slot" is the Open MPI term for an allocatable unit where we can
launch a process. The number of slots available are defined by the
environment in which Open MPI processes are run:

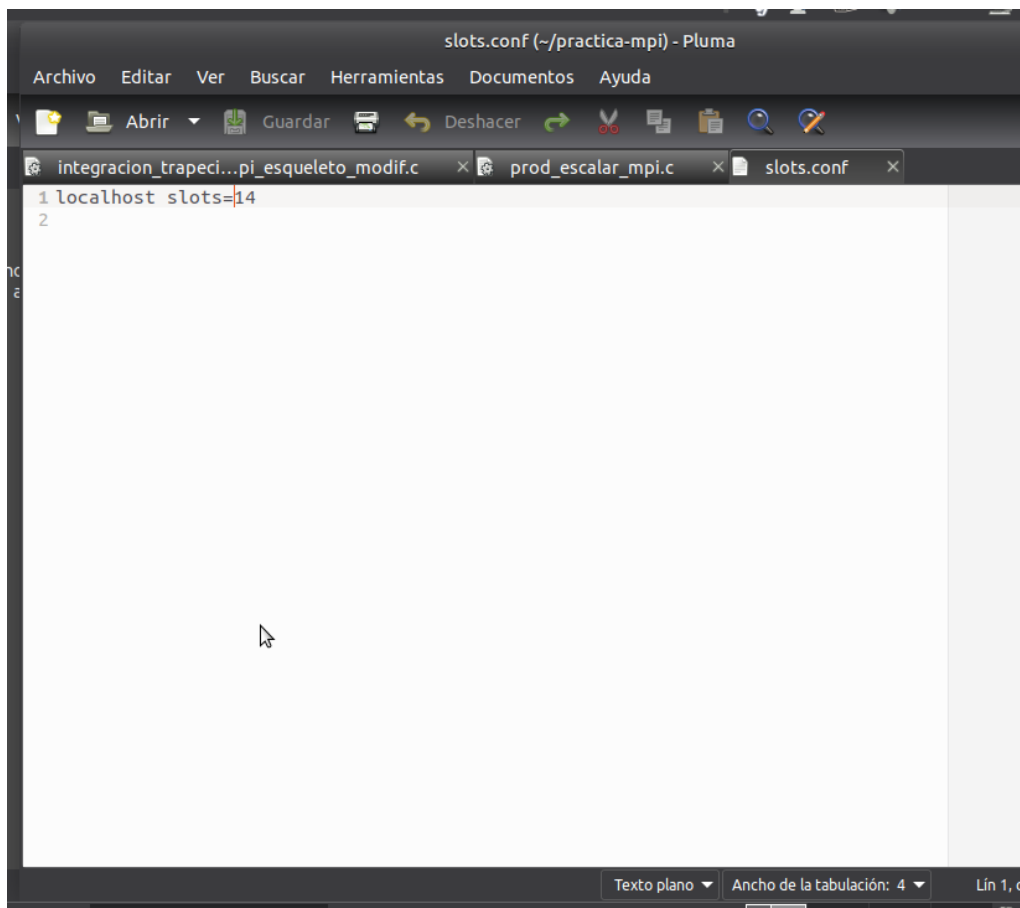
1. Hostfile, via "slots=N" clauses (N defaults to number of
   processor cores if not provided)
2. The --host command line parameter, via a ":N" suffix on the
   hostname (N defaults to 1 if not provided)
3. Resource manager (e.g., SLURM, PBS/Torque, LSF, etc.)
4. If none of a hostfile, the --host command line parameter, or an
   RM is present, Open MPI defaults to the number of processor cores

In all the above cases, if you want Open MPI to default to the number
of hardware threads instead of the number of processor cores, use the
--use-hwthread-cpus option.

Alternatively, you can use the --oversubscribe option to ignore the
number of available slots when deciding the number of processes to
launch.
-----
```

(Figura 9)

Esto lo soluciono creando un archivo llamado “slots.conf”. En el archivo pongo “localhost slots=14”, esto hace que el programa al ejecutarlo funcione hasta 14 procesos. A continuación, pongo en la consola de comandos “mpirun --hostfile slots.conf -np 7 hola\_mundo\_avanzado”. Y ya si me da la respuesta por pantalla, como se puede ver en la Figura 11.



(Figura 10)

```
practica@pc-13-185:~/practica-mpi$ mpirun -hostfile slots.conf -np 7 hola_mundo_avanzado
Yo soy el proceso 1 de 7, corriendo en pc-13-185.
Yo soy el proceso 4 de 7, corriendo en pc-13-185.
Yo soy el proceso 6 de 7, corriendo en pc-13-185.
Yo soy el proceso 2 de 7, corriendo en pc-13-185.
Yo soy el proceso 3 de 7, corriendo en pc-13-185.
Soy el proceso 0 de 7, corriendo en pc-13-185: ¡Hola mundo!
Yo soy el proceso 5 de 7, corriendo en pc-13-185.
practica@pc-13-185:~/practica-mpi$ mpirun -hostfile slots.conf -np 12 hola_mundo_avanzado
Yo soy el proceso 4 de 12, corriendo en pc-13-185.
Yo soy el proceso 2 de 12, corriendo en pc-13-185.
Yo soy el proceso 1 de 12, corriendo en pc-13-185.
Yo soy el proceso 5 de 12, corriendo en pc-13-185.
Yo soy el proceso 3 de 12, corriendo en pc-13-185.
Yo soy el proceso 6 de 12, corriendo en pc-13-185.
Yo soy el proceso 11 de 12, corriendo en pc-13-185.
Soy el proceso 0 de 12, corriendo en pc-13-185: ¡Hola mundo!
Yo soy el proceso 8 de 12, corriendo en pc-13-185.
Yo soy el proceso 9 de 12, corriendo en pc-13-185.
Yo soy el proceso 7 de 12, corriendo en pc-13-185.
Yo soy el proceso 10 de 12, corriendo en pc-13-185.
practica@pc-13-185:~/practica-mpi$
```

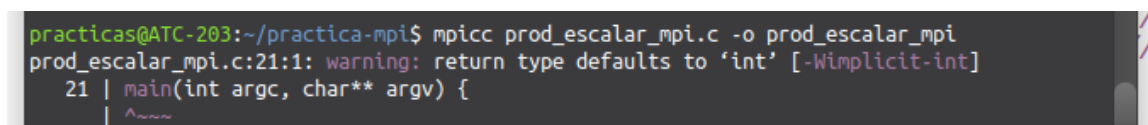
(Figura 11)



Líneas adicionales añadidas al programa o líneas que se han modificado:	char name[MPI_MAX_PROCESSOR_NAME];  MPI_Get_processor_name(name,&resultlen);  (mejor visto en las explicaciones anteriores)
¿Se puede lanzar un número de procesos mayor que el número de núcleos del procesador utilizado?	Explicado arriba (Figura 9)
¿Qué ocurre en ese caso?	Explicado arriba (Figura 10 y 11)

Ejercicio 1.2 – Modificar el programa añadiéndole una medición del tiempo de ejecución, usando la función MPI\_Wtime.

Guardo el archivo con el programa "prod\_escalar\_mpi.c", que es un ejemplo de cálculo simple en paralelo que realiza el producto escalar de dos vectores. Si se compila y ejecuta directamente sale un error (Figura 12) y es porque en la línea 21 del código falta un int de manera que al ponerlo quedaría: "int main(int argc, char\*\* argv){" que devuelve el resultado de la función, un número entero. (Figura 13).

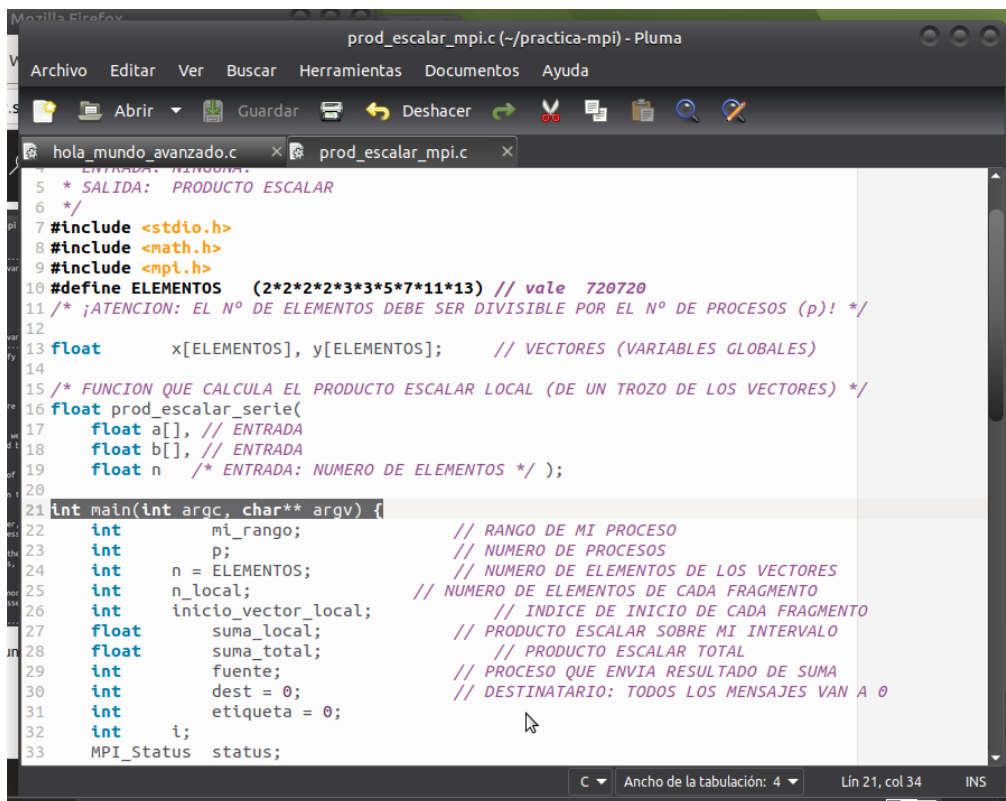


```

practicass@ATC-203:~/practica-mpi$ mpicc prod_escalar_mpi.c -o prod_escalar_mpi
prod_escalar_mpi.c:21:1: warning: return type defaults to 'int' [-Wimplicit-int]
 21 | main(int argc, char** argv) {
    |

```

(Figura 12)



```

prod_escalar_mpi.c (~/.practica-mpi) - Pluma
Archivo  Editar  Ver  Buscar  Herramientas  Documentos  Ayuda
Abrir  Guardar  Deshacer
hola_mundo_avanzado.c  prod_escalar_mpi.c
1  ENTRADA: NINGUNA.
2  * SALIDA: PRODUCTO ESCALAR
3  */
4
5  #include <stdio.h>
6
7  #include <math.h>
8
9  #include <mpi.h>
10 #define ELEMENTOS (2*2*2*3*3*5*7*11*13) // vale 720720
11 /* ¡ATENCIÓN: EL N° DE ELEMENTOS DEBE SER DIVISIBLE POR EL N° DE PROCESOS (p)! */
12
13 float x[ELEMENTOS], y[ELEMENTOS]; // VECTORES (VARIABLES GLOBALES)
14
15 /* FUNCIÓN QUE CALCULA EL PRODUCTO ESCALAR LOCAL (DE UN TROZO DE LOS VECTORES) */
16 float prod_escalar_serie(
17     float a[], // ENTRADA
18     float b[], // ENTRADA
19     float n // ENTRADA: NUMERO DE ELEMENTOS */ );
20
21 int main(int argc, char** argv) {
22     int mi_rango; // RANGO DE MI PROCESO
23     int p; // NUMERO DE PROCESOS
24     int n = ELEMENTOS; // NUMERO DE ELEMENTOS DE LOS VECTORES
25     int n_local; // NUMERO DE ELEMENTOS DE CADA FRAGMENTO
26     int inicio_vector_local; // INDICE DE INICIO DE CADA FRAGMENTO
27     float suma_local; // PRODUCTO ESCALAR SOBRE MI INTERVALO
28     float suma_total; // PRODUCTO ESCALAR TOTAL
29     int fuente; // PROCESO QUE ENVIA RESULTADO DE SUMA
30     int dest = 0; // DESTINATARIO: TODOS LOS MENSAJES VAN A 0
31     int etiqueta = 0;
32     int i;
33     MPI_Status status;

```

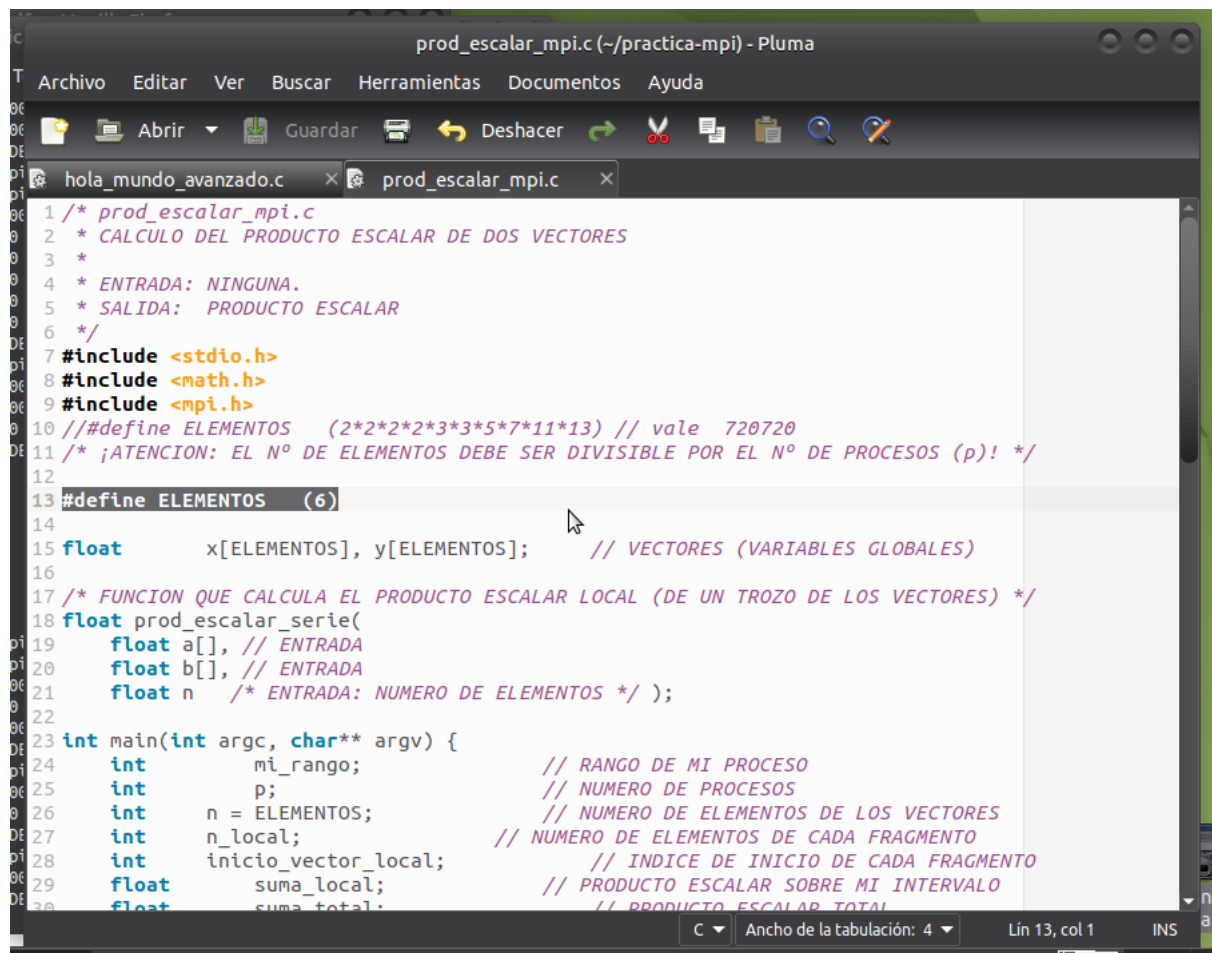
(Figura 13)

Ahora al compilarlo y ejecutarlo (Figura 14) ya no sale ningún error y muestra por pantalla la respuesta del programa de los 6 procesos que le he indicado.

```
practicass@ATC-203:~/practica-mpi$ mpicc prod_escalar_mpi.c -o prod_escalar_mpi
practicass@ATC-203:~/practica-mpi$ mpirun -np 6 prod_escalar_mpi
MI RANGO = 3 , SUMA LOCAL = 720720.000000
MI RANGO = 5 , SUMA LOCAL = 720720.000000
MI RANGO = 1 , SUMA LOCAL = 720720.000000
MI RANGO = 4 , SUMA LOCAL = 720720.000000
MI RANGO = 2 , SUMA LOCAL = 720720.000000
MI RANGO = 0 , SUMA LOCAL = 720720.000000
PRODUCTO ESCALAR USANDO p=6 TROZOS DE LOS VECTORES X E Y = 4324320.000000
practicass@ATC-203:~/practica-mpi$
```

(Figura 14)

Para ver cómo funciona el programa y que hace bien su funcionamiento cambio inicialmente el número de elementos de los vectores a un valor muy pequeño, 6. En la figura 15 subrayo en gris oscuro el cambio puesto.



```
1 /* prod_escalar_mpi.c
2 * CALCULO DEL PRODUCTO ESCALAR DE DOS VECTORES
3 *
4 * ENTRADA: NINGUNA.
5 * SALIDA: PRODUCTO ESCALAR
6 */
7 #include <stdio.h>
8 #include <math.h>
9 #include <mpi.h>
10 // #define ELEMENTOS (2*2*2*2*3*3*5*7*11*13) // vale 720720
11 /* ¡ATENCIÓN: EL Nº DE ELEMENTOS DEBE SER DIVISIBLE POR EL Nº DE PROCESOS (p)! */
12
13 #define ELEMENTOS (6)
14
15 float x[ELEMENTOS], y[ELEMENTOS]; // VECTORES (VARIABLES GLOBALES)
16
17 /* FUNCION QUE CALCULA EL PRODUCTO ESCALAR LOCAL (DE UN TROZO DE LOS VECTORES) */
18 float prod_escalar_serie(
19     float a[], // ENTRADA
20     float b[], // ENTRADA
21     float n /* ENTRADA: NUMERO DE ELEMENTOS */ );
22
23 int main(int argc, char** argv) {
24     int mi_rango; // RANGO DE MI PROCESO
25     int p; // NUMERO DE PROCESOS
26     int n = ELEMENTOS; // NUMERO DE ELEMENTOS DE LOS VECTORES
27     int n_local; // NUMERO DE ELEMENTOS DE CADA FRAGMENTO
28     int inicio_vector_local; // INDICE DE INICIO DE CADA FRAGMENTO
29     float suma_local; // PRODUCTO ESCALAR SOBRE MI INTERVALO
30     float suma_total; // PRODUCTO ESCALAR TOTAL
```

(Figura 15)

En la práctica me dice que ejecute el programa variando el número de procesos lanzados desde 1 hasta 3 y comprobar que los resultados son correctos. Simplemente al ejecutar el programa cambio el número como se puede ver en la figura 16.

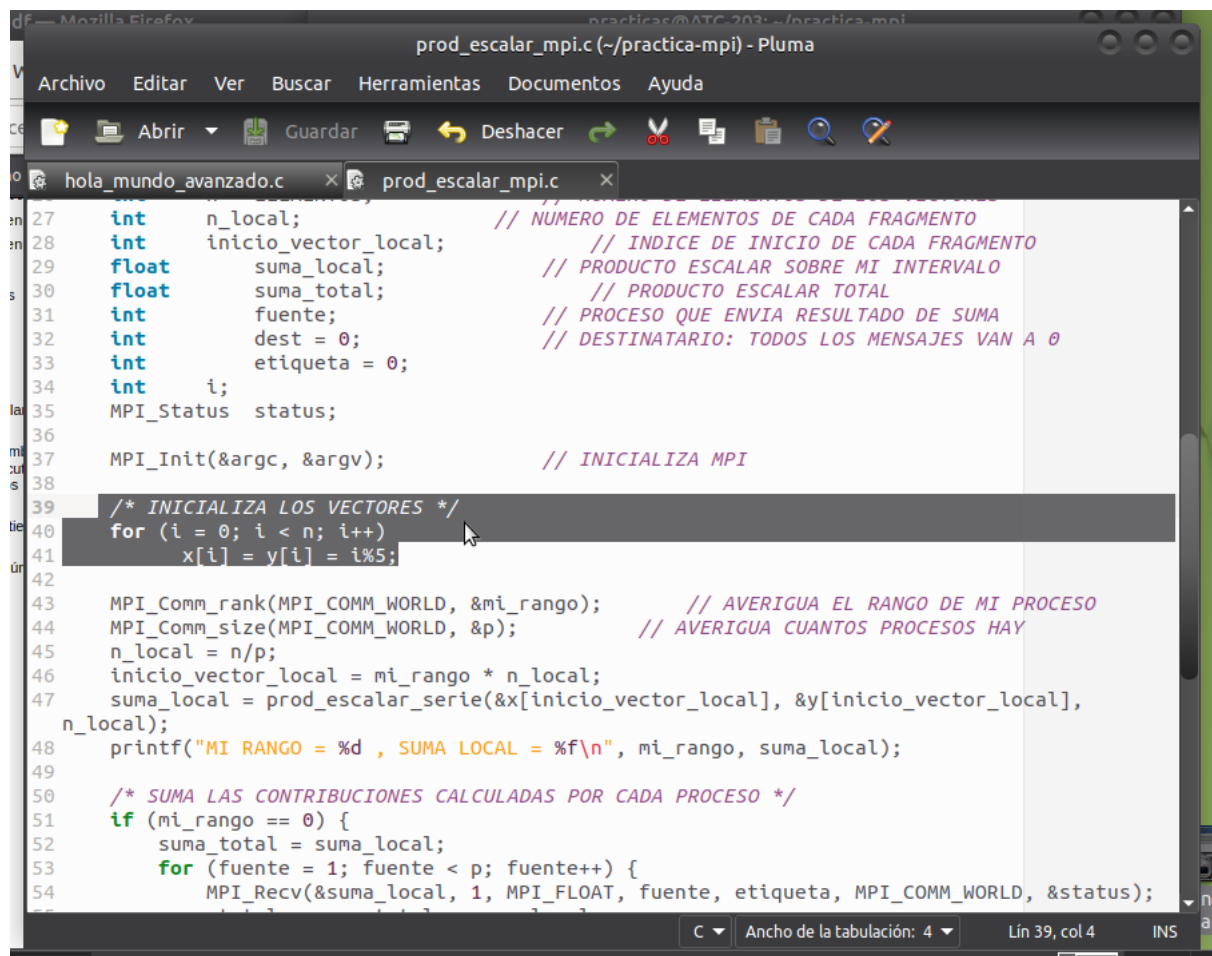
```
practicass@ATC-203:~/practica-mpi$ mpicc prod_escalar_mpi.c -o prod_escalar_mpi
practicass@ATC-203:~/practica-mpi$ mpirun -np 3 prod_escalar_mpi
MI RANGO = 2 , SUMA LOCAL = 16.000000
MI RANGO = 0 , SUMA LOCAL = 1.000000
MI RANGO = 1 , SUMA LOCAL = 13.000000
PRODUCTO ESCALAR USANDO p=3 TROZOS DE LOS VECTORES X E Y = 30.000000
practicass@ATC-203:~/practica-mpi$ mpirun -np 2 prod_escalar_mpi
MI RANGO = 1 , SUMA LOCAL = 25.000000
MI RANGO = 0 , SUMA LOCAL = 5.000000
PRODUCTO ESCALAR USANDO p=2 TROZOS DE LOS VECTORES X E Y = 30.000000
practicass@ATC-203:~/practica-mpi$ mpirun -np 1 prod_escalar_mpi
MI RANGO = 0 , SUMA LOCAL = 30.000000
PRODUCTO ESCALAR USANDO p=1 TROZOS DE LOS VECTORES X E Y = 30.000000
practicass@ATC-203:~/practica-mpi$
```

(Figura 16)

Efectivamente el cálculo del producto escalar del número de vector que le he puesto,6 es correcto, haciéndolo a mano da igual que el mostrado por pantalla.

$$(0*0) + (1*1) + (2*2) + (3*3) + (4*4) = 0 + 1 + 4 + 9 + 16 = 30$$

¿Y por qué no llega al 5? Porque el módulo de 5 es 0 entonces tendríamos [0,1,2,3,4,0] . Se ve en lo que he subrayado de gris en la figura 17.



```
27 int n_local; // NUMERO DE ELEMENTOS DE CADA FRAGMENTO
28 int inicio_vector_local; // INDICE DE INICIO DE CADA FRAGMENTO
29 float suma_local; // PRODUCTO ESCALAR SOBRE MI INTERVALO
30 float suma_total; // PRODUCTO ESCALAR TOTAL
31 int fuente; // PROCESO QUE ENVIA RESULTADO DE SUMA
32 int dest = 0; // DESTINATARIO: TODOS LOS MENSAJES VAN A 0
33 int etiqueta = 0;
34 int i;
35 MPI_Status status;
36
37 MPI_Init(&argc, &argv); // INICIALIZA MPI
38
39 /* INICIALIZA LOS VECTORES */
40 for (i = 0; i < n; i++)
41     x[i] = y[i] = i%5;
42
43 MPI_Comm_rank(MPI_COMM_WORLD, &mi_rango); // AVERIGUA EL RANGO DE MI PROCESO
44 MPI_Comm_size(MPI_COMM_WORLD, &p); // AVERIGUA CUANTOS PROCESOS HAY
45 n_local = n/p;
46 inicio_vector_local = mi_rango * n_local;
47 suma_local = prod_escalar_serie(&x[inicio_vector_local], &y[inicio_vector_local],
n_local);
48 printf("MI RANGO = %d , SUMA LOCAL = %f\n", mi_rango, suma_local);
49
50 /* SUMA LAS CONTRIBUCIONES CALCULADAS POR CADA PROCESO */
51 if (mi_rango == 0) {
52     suma_total = suma_local;
53     for (fuente = 1; fuente < p; fuente++) {
54         MPI_Recv(&suma_local, 1, MPI_FLOAT, fuente, etiqueta, MPI_COMM_WORLD, &status);
```

(Figura 17)

Modificar el programa añadiéndole una medición del tiempo de ejecución, usando para ello la función MPI\_Wtime. Medir el tiempo de ejecución del programa al variar el número de procesos con el que es lanzado desde 1 hasta 6 procesos. ¿Son lógicos los tiempos de ejecución que se observan?

a)Funcionamiento del programa con nº elementos = 6:

Nº de procesos: p= 1	Producto escalar = 30.000000
Nº de procesos: p =2	Producto escalar = 30.000000
Nº de procesos: p = 3	Producto escalar = 30.000000

Para medir el tiempo de ejecución del programa al variar el número de procesos con el que es lanzado desde 1 hasta 6 procesadores, modifiqué el programa añadiendo una medición del tiempo de ejecución usando la función MPI\_Wtime.

Empiezo inicializando dos variables de tipo double t1 (tiempo inicio) y t2 (tiempo final).

"t1=MPI\_Wtime();" arriba donde inicializa los vectores.

"t2=MPI\_Wtime();" debajo del printf donde se muestra el resultado por pantalla. Y en el printf siguiente "t2-t1". Como se puede ver en la figura 18.

```

12
13 #define ELEMENTOS    (6)
14
15 float      x[ELEMENTOS], y[ELEMENTOS];    // VECTORES (VARIABLES GLOBALES)
16
17 /* FUNCION QUE CALCULA EL PRODUCTO ESCALAR LOCAL (DE UN TROZO DE LOS VECTORES) */
18 float prod_escalar_serie(
19     float a[], // ENTRADA
20     float b[], // ENTRADA
21     float n    /* ENTRADA: NUMERO DE ELEMENTOS */ );
22
23 int main(int argc, char** argv) {
24     int      mi_rango;           // RANGO DE MI PROCESO
25     int      p;                 // NUMERO DE PROCESOS
26     int      n = ELEMENTOS;     // NUMERO DE ELEMENTOS DE LOS VECTORES
27     int      n_local;           // NUMERO DE ELEMENTOS DE CADA FRAGMENTO
28     int      inicio_vector_local; // INDICE DE INICIO DE CADA FRAGMENTO
29     float    suma_local;        // PRODUCTO ESCALAR SOBRE MI INTERVALO
30     float    suma_total;        // PRODUCTO ESCALAR TOTAL
31     int      fuente;             // PROCESO QUE ENVIA RESULTADO DE SUMA
32     int      dest = 0;          // DESTINATARIO: TODOS LOS MENSAJES VAN A 0
33     int      etiqueta = 0;
34     int      i;
35     double t1,t2;
36     MPI_Status status;
37
38     MPI_Init(&argc, &argv);      // INICIALIZA MPI
39
40     /* INICIALIZA LOS VECTORES */
41     t1=MPI_Wtime();
42     t2=MPI_Wtime();
43     printf("Tiempo de ejecución: %f\n", t2-t1);

```

```

38 MPI_Init(&argc, &argv); // INICIALIZA MPI
39
40 /* INICIALIZA LOS VECTORES */
41 t1=MPI_Wtime();
42 for (i = 0; i < n; i++)
43     x[i] = y[i] = i%5;
44
45 MPI_Comm_rank(MPI_COMM_WORLD, &mi_rango); // AVERIGUA EL RANGO DE MI PROCESO
46 MPI_Comm_size(MPI_COMM_WORLD, &p); // AVERIGUA CUANTOS PROCESOS HAY
47 n_local = n/p;
48 inicio_vector_local = mi_rango * n_local;
49 suma_local = prod_escalar_serie(&x[inicio_vector_local], &y[inicio_vector_local], n_local);
50 printf("MI RANGO = %d , SUMA LOCAL = %f\n", mi_rango, suma_local);
51
52 /* SUMA LAS CONTRIBUCIONES CALCULADAS POR CADA PROCESO */
53
54 if (mi_rango == 0) {
55     suma_total = suma_local;
56     for (fuente = 1; fuente < p; fuente++) {
57         MPI_Recv(&suma_local, 1, MPI_FLOAT, fuente, etiqueta, MPI_COMM_WORLD, &status);
58         suma_total = suma_total + suma_local;
59     }
60 } else {
61     MPI_Send(&suma_local, 1, MPI_FLOAT, dest, etiqueta, MPI_COMM_WORLD);
62 }
63
64 /* MUESTRA EL RESULTADO POR PANTALLA */
65 if (mi_rango == 0) {
66     printf("PRODUCTO ESCALAR USANDO p=%d TROZOS DE LOS VECTORES X E Y = %f\n", p, suma_total);
67     t2=MPI_Wtime();
68     printf("El tiempo de ejecución es de: %f\n", t2-t1);
69 }

```

(Figura 18)

#### b) Líneas adicionales

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

```

double t1,t2;
t1=MPI_Wtime(); y t2=MPI_Wtime();
printf("El tiempo de ejecución es de: %f\n", t2-t1);

```

Anteriormente puse en comentario los elementos que había en el programa para definir un número más pequeño y comprobar que todo funcionaba bien. Al ser así, lo quito como comentario y lo utilizo para calcular los tiempos de ejecución y aceleración.

Para ello ejecuto el programa "prod\_escalar\_mpi" desde 1 a 6 procesos . (Figura 19)

```

practic@ATC-203:~/practica-mpi$ mpirun -np 1 prod_escalar_mpi
MI RANGO = 0 , SUMA LOCAL = 4324320.000000
PRODUCTO ESCALAR USANDO p=1 TROZOS DE LOS VECTORES X E Y = 4324320.000000
El tiempo de ejecución es de: 0.004428
practic@ATC-203:~/practica-mpi$ mpirun -np 2 prod_escalar_mpi
MI RANGO = 0 , SUMA LOCAL = 2162160.000000
PRODUCTO ESCALAR USANDO p=2 TROZOS DE LOS VECTORES X E Y = 4324320.000000
El tiempo de ejecución es de: 0.003946
MI RANGO = 1 , SUMA LOCAL = 2162160.000000
practic@ATC-203:~/practica-mpi$ mpirun -np 3 prod_escalar_mpi
MI RANGO = 1 , SUMA LOCAL = 1441440.000000
MI RANGO = 0 , SUMA LOCAL = 1441440.000000
MI RANGO = 2 , SUMA LOCAL = 1441440.000000
PRODUCTO ESCALAR USANDO p=3 TROZOS DE LOS VECTORES X E Y = 4324320.000000
El tiempo de ejecución es de: 0.003785
practic@ATC-203:~/practica-mpi$ mpirun -np 4 prod_escalar_mpi
MI RANGO = 1 , SUMA LOCAL = 1081080.000000
MI RANGO = 2 , SUMA LOCAL = 1081080.000000
MI RANGO = 3 , SUMA LOCAL = 1081080.000000
MI RANGO = 0 , SUMA LOCAL = 1081080.000000
PRODUCTO ESCALAR USANDO p=4 TROZOS DE LOS VECTORES X E Y = 4324320.000000
El tiempo de ejecución es de: 0.005800
^[[practic@ATC-203:~/practica-mpi$ mpirun -np 5 prod_escalar_mpi
MI RANGO = 1 , SUMA LOCAL = 864861.000000
MI RANGO = 2 , SUMA LOCAL = 864866.000000
MI RANGO = 3 , SUMA LOCAL = 864869.000000
MI RANGO = 4 , SUMA LOCAL = 864870.000000
MI RANGO = 0 , SUMA LOCAL = 864854.000000
PRODUCTO ESCALAR USANDO p=5 TROZOS DE LOS VECTORES X E Y = 4324320.000000
El tiempo de ejecución es de: 0.005787

```

```

practicass@ATC-203:~/practica-mpi$ mpicc prod_escalar_mpi.c -o prod_escalar_mpi
practicass@ATC-203:~/practica-mpi$ mpirun -np 6 prod_escalar_mpi
MI RANGO = 4 , SUMA LOCAL = 720720.000000
MI RANGO = 5 , SUMA LOCAL = 720720.000000
MI RANGO = 1 , SUMA LOCAL = 720720.000000
MI RANGO = 3 , SUMA LOCAL = 720720.000000
MI RANGO = 0 , SUMA LOCAL = 720720.000000
MI RANGO = 2 , SUMA LOCAL = 720720.000000
PRODUCTO ESCALAR USANDO p=6 TROZOS DE LOS VECTORES X E Y = 4324320.000000
El tiempo de ejecución es de: 0.005815
practicass@ATC-203:~/practica-mpi$

```

(Figura 19)

Nº de procesos (p)	Tiempo de ejecución	Aceleración
1	0.004428	$0.004428/0.004428 = 1$
2	0.003946	$0.004428/0.003946 = 1,122149$
3	0.003785	$0.004428/0.003785 = 1,169881$
4	0.005800	$0.004428/0.005800 = 0,763448$
5	0.005787	$0.004428/0.005787 = 0,765163$
6	0.005815	$0.004428/0.005815 = 0,761478$

¿Son lógicos los tiempos de ejecución que se observan?

No son lógicos, no siguen ninguna forma. Al principio cuando aumenta el número de procesadores disminuye el tiempo de ejecución hasta que en  $p = 4$  aumenta el tiempo, siendo más grande que cuando ejecutaba con  $p=1$ , luego en el siguiente  $p=5$  baja un poco con respecto al anterior, pero al añadirle otro proceso vuelve a aumentar el tiempo de ejecución. Por consiguiente, la aceleración también se ve afectada y sin lógica con respecto al número de procesadores.

## SEGUNDA PARTE

En esta segunda parte nos encontramos con un programa en C que realiza una estimación numérica de la integral o área encerrada entre la gráfica de una función no negativa  $f(x)$ , dos líneas verticales situadas en posiciones  $x = a$  y  $x = b$  y el eje  $x$ , utilizando el “método de los trapecios”

Ejercicio 2.1 – Versión paralela del mismo programa utilizando MPI. Probar el programa obteniendo una estimación del valor de  $\pi$

En las siguientes figuras está el código del programa en versión paralela. En un rectángulo rojo he cogido todos los cambios que he hecho frente al programa dado. Figura 20



```

58  /* MUESTRA EL RESULTADO POR PANTALLA */
59  if (mi_rango == 0) {
60      printf("ESTIMACION USANDO n=%d TRAPECIOS,\n", n);
61      printf("DE LA INTEGRAL DESDE %f HASTA %f = %f\n", a, b, total);
62
63      printf("PI = %f\n", 2 * total);
64  }
65
66  MPI_Finalize(); // CIERRA EL UNIVERSO MPI
67 } /* MAIN */
68
69 /* FUNCION QUE CALCULA LA INTEGRAL LOCAL (AREA DEL TRAPECIO) */
70 float Trapecio(
71     float local_a /* entrada */,
72     float local_b /* entrada */,
73     int local_n /* entrada */,
74     float h /* entrada */) {
75
76     float integral; /* ALMACENA EL RESULTADO EN integral */
77     float x;
78     int i;
79
80     float f(float x); /* FUNCION QUE ESTAMOS INTEGRANDO */
81
82     // CALCULA EL VALOR DE integral DESDE local_a HASTA local_b
83     integral = (f(local_a) + f(local_b))/2.0;
84     x = local_a;
85     for (i = 1; i <= local_n-1; i++) {
86         x = x + h;
87         integral = integral + f(x);
88     }
89     integral = integral*h;
90     return integral;
91 } /* TRAPECIO */
92
93 /* FUNCION QUE ESTAMOS INTEGRANDO */
94 float f(float x) {
95     float return_val;
96     /* CALCULA f(x) Y DEVUELVE SU VALOR */
97     return_val = sqrt(fabs( 1 - x*x ));
98     return return_val;
99 } /* f */
100
101
102

```

(Figura 20)

Compilo el programa paralelo con el comando “mpicc integracion\_trapecios\_pi\_esqueleto.c -o integracion\_trapecios\_pi\_esqueleto -lm” y lo ejecuto poniendo “mpirun integracion\_trapecios\_pi\_esqueleto”

```

practicass@ATC-203: ~/practica-mpi
Archivo Editar Ver Buscar Terminal Ayuda
-----
practicass@ATC-203:~/practica-mpi$ mpicc integracion_trapecios_pi_esqueleto.c -o integracion_trapecios_pi_esqueleto -lm
practicass@ATC-203:~/practica-mpi$ mpirun -np 1 integracion_trapecios_pi_esqueleto
ESTIMACION USANDO n=10000 TRAPECIOS,
DE LA INTEGRAL DESDE -1.000000 HASTA 1.000000 = 1.570861
PI = 3.141722
practicass@ATC-203:~/practica-mpi$ mpirun -np 2 integracion_trapecios_pi_esqueleto
ESTIMACION USANDO n=10000 TRAPECIOS,
DE LA INTEGRAL DESDE -1.000000 HASTA 1.000000 = 1.570804
PI = 3.141608
practicass@ATC-203:~/practica-mpi$ mpirun -np 3 integracion_trapecios_pi_esqueleto
ESTIMACION USANDO n=10000 TRAPECIOS,
DE LA INTEGRAL DESDE -1.000000 HASTA 1.000000 = 1.570807
PI = 3.141615
practicass@ATC-203:~/practica-mpi$ mpirun -np 4 integracion_trapecios_pi_esqueleto
ESTIMACION USANDO n=10000 TRAPECIOS,
DE LA INTEGRAL DESDE -1.000000 HASTA 1.000000 = 1.570817
PI = 3.141635
practicass@ATC-203:~/practica-mpi$ mpirun -np 5 integracion_trapecios_pi_esqueleto
ESTIMACION USANDO n=10000 TRAPECIOS,
DE LA INTEGRAL DESDE -1.000000 HASTA 1.000000 = 1.570808
PI = 3.141616
practicass@ATC-203:~/practica-mpi$ mpirun -np 6 integracion_trapecios_pi_esqueleto
ESTIMACION USANDO n=10000 TRAPECIOS,
DE LA INTEGRAL DESDE -1.000000 HASTA 1.000000 = 1.570783
PI = 3.141566
practicass@ATC-203:~/practica-mpi$ mpirun -np 7 integracion_trapecios_pi_esqueleto
-----
There are not enough slots available in the system to satisfy the 7
slots that were requested by the application:

integracion_trapecios_pi_esqueleto

Either request fewer slots for your application, or make more slots
available for use.

```

Ocurre lo mismo que lo explicado en la primera parte , figura 9 y 10. Para solucionarlo hago lo mismo (creo un archivo slots.conf) y lo ejecuto. De manera que, al poner un número de procesos mayor, como es el caso de 8 no salga ningún error y se ejecute dicho programa. Figura 22.

```
-----
practicass@ATC-203:~/practica-mpi$ mpirun --hostfile slots.conf -np 8 integracion_trapecios_pi_esqueleto
ESTIMACION USANDO n=10000 TRAPECIOS,
DE LA INTEGRAL DESDE -1.000000 HASTA 1.000000 = 1.570803
PI = 3.141606
practicass@ATC-203:~/practica-mpi$
```

(Figura 22)

Estimación del valor de pi bastante acertada.

¿Qué ocurre si por ejemplo tanto en el programa “integracion\_trapecios\_pi\_esqueleto” y “hola\_mundo\_avanzado” ejecuto el programa en 20 procesos? Que me vuelve a dar el error de los slots como se puede ver en la figura 23 , porque cuando he creado el archivo de texto llamado “slots.conf” le he especificado que como máximo pueda hasta 14 procesos. Se puede ver en la figura 10.

```
practicass@ATC-203: ~/practica-mpi
Archivo Editar Ver Buscar Terminal Ayuda
la orden «mpirun» del paquete deb «slurm-wlm-torque (21.08.5-2ubuntu1)»
Pruebe con: sudo apt install <nombre del paquete deb>
practicass@ATC-203:~/practica-mpi$ mpirun -hostfile slots.conf -np 20 integracion_trapecios_pi_esqueleto
-----
There are not enough slots available in the system to satisfy the 20
slots that were requested by the application:

  integracion_trapecios_pi_esqueleto

Either request fewer slots for your application, or make more slots
available for use.

A "slot" is the Open MPI term for an allocatable unit where we can
launch a process. The number of slots available are defined by the
environment in which Open MPI processes are run:

  1. Hostfile, via "slots=N" clauses (N defaults to number of
     processor cores if not provided)
  2. The --host command line parameter, via a ":N" suffix on the
     hostname (N defaults to 1 if not provided)
  3. Resource manager (e.g., SLURM, PBS/Torque, LSF, etc.)
  4. If none of a hostfile, the --host command line parameter, or an
     RM is present, Open MPI defaults to the number of processor cores

In all the above cases, if you want Open MPI to default to the number
of hardware threads instead of the number of processor cores, use the
--use-hwthread-cpus option.

Alternatively, you can use the --oversubscribe option to ignore the
number of available slots when deciding the number of processes to
```

(Figura 23)

¿Cómo lo puedo resolver? Muy fácil, como ya lo he hecho anteriormente, en el archivo “slots.conf” en vez de 14 pongo 20 o un número más alto de lo que luego, yo vaya a poner en el código. Lo ejecuto y sale el resultado por pantalla sin ningún error. Figura 24

```
Abrir ▾ [icon] slots.conf
~/practica-mpi
1 localhost slots=20
```



```
practicass@ATC-203: ~/practica-mpi
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
Either request fewer slots for your application, or make more slots
available for use.

A "slot" is the Open MPI term for an allocatable unit where we can
launch a process. The number of slots available are defined by the
environment in which Open MPI processes are run:

1. Hostfile, via "slots=N" clauses (N defaults to number of
   processor cores if not provided)
2. The --host command line parameter, via a ":N" suffix on the
   hostname (N defaults to 1 if not provided)
3. Resource manager (e.g., SLURM, PBS/Torque, LSF, etc.)
4. If none of a hostfile, the --host command line parameter, or an
   RM is present, Open MPI defaults to the number of processor cores

In all the above cases, if you want Open MPI to default to the number
of hardware threads instead of the number of processor cores, use the
--use-hwthread-cpus option.

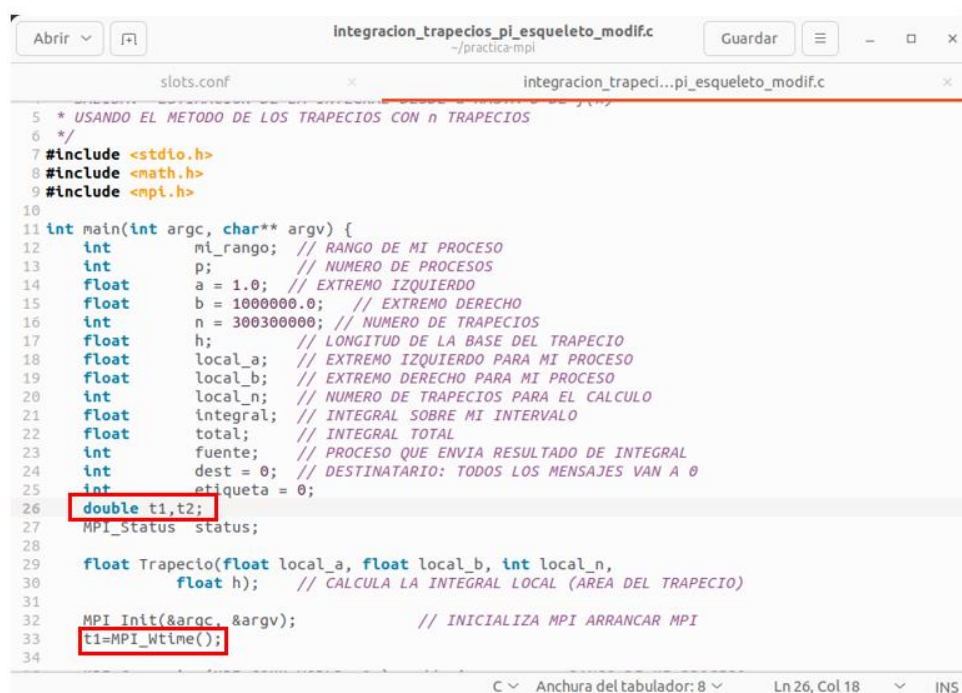
Alternatively, you can use the --oversubscribe option to ignore the
number of available slots when deciding the number of processes to
launch.

-----
practicass@ATC-203:~/practica-mpi$
practicass@ATC-203:~/practica-mpi$
practicass@ATC-203:~/practica-mpi$ mpirun -hostfile slots.conf -np 20 integracion_trapecios_pi_esqueleto
ESTIMACION USANDO n=10000 TRAPECIOS,
DE LA INTEGRAL DESDE -1.000000 HASTA 1.000000 = 1.570797
PI = 3.141593
practicass@ATC-203:~/practica-mpi$
```

(Figura 24)

Ejercicio 2.2 – Medición del tiempo de ejecución, usando la función `MPI_Wtime` y representar una gráfica del tiempo y de la aceleración respecto al número de procesos.

Para medir el tiempo de ejecución con la función `MPI_Wtime` en el código he puesto:



```
integracion_trapecios_pi_esqueleto_modif.c
~/practica-mpi
Guardar

slots.conf
integracion_trapeci...pi_esqueleto_modif.c

5 * USANDO EL METODO DE LOS TRAPECIOS CON n TRAPECIOS
6 */
7 #include <stdio.h>
8 #include <math.h>
9 #include <mpi.h>
10
11 int main(int argc, char** argv) {
12     int    mi_rango; // RANGO DE MI PROCESO
13     int    p;        // NUMERO DE PROCESOS
14     float  a = 1.0;  // EXTREMO IZQUIERDO
15     float  b = 1000000.0; // EXTREMO DERECHO
16     int    n = 300300000; // NUMERO DE TRAPECIOS
17     float  h;        // LONGITUD DE LA BASE DEL TRAPECIO
18     float  local_a;  // EXTREMO IZQUIERDO PARA MI PROCESO
19     float  local_b;  // EXTREMO DERECHO PARA MI PROCESO
20     int    local_n;  // NUMERO DE TRAPECIOS PARA EL CALCULO
21     float  integral; // INTEGRAL SOBRE MI INTERVALO
22     float  total;    // INTEGRAL TOTAL
23     int    fuente;   // PROCESO QUE ENVIA RESULTADO DE INTEGRAL
24     int    dest = 0; // DESTINATARIO: TODOS LOS MENSAJES VAN A 0
25     int    etiqueta = 0;
26     double t1, t2;
27     MPI_Status status;
28
29     float Trapecio(float local_a, float local_b, int local_n,
30                   float h); // CALCULA LA INTEGRAL LOCAL (AREA DEL TRAPECIO)
31
32     MPI_Init(&argc, &argv); // INICIALIZA MPI ARRANCAR MPI
33     t1=MPI_Wtime();
34 }
```

```

45
46     integral = Trapecio(local_a, local_b, local_n, h);
47
48     /* EL PROCESO 0 RECIBE Y SUMA LAS INTEGRALES CALCULADAS POR CADA PROCESO */
49     if (mi_rango == 0) {
50         /* Sumo los cálculos de cada proceso, que me envían los demás */
51         total = integral;
52         for (fuente = 1; fuente < p; fuente++) {
53             MPI_Recv(&integral, 1, MPI_FLOAT, fuente, etiqueta, MPI_COMM_WORLD, &status);
54             total = total + integral;
55         }
56     } else { /* Envío mi resultado al proceso 0 */
57         MPI_Send(&integral, 1, MPI_FLOAT, dest, etiqueta, MPI_COMM_WORLD);
58     }
59
60     /* MUESTRA EL RESULTADO POR PANTALLA */
61     if (mi_rango == 0) {
62         t2=MPI_Wtime();
63         printf("ESTIMACION USANDO n=%d TRAPECIOS,\n", n);
64         printf("DE LA INTEGRAL DESDE %f HASTA %f = %f\n", a, b, total);
65         printf("El tiempo de ejecución es de: %f\n", t2-t1);
66     }
67     printf("PI = %f\n", 2 * total);
68 }
69

```

(Figura 25)

Otras modificaciones por petición del enunciado son :

```

5  * USANDO EL METODO DE LOS TRAPECIOS CON n TRAPECIOS
6  */
7  #include <stdio.h>
8  #include <math.h>
9  #include <mpi.h>
10
11 int main(int argc, char** argv) {
12     int    mi_rango; // RANGO DE MI PROCESO
13     int    p;        // NUMERO DE PROCESOS
14     float  a = 1.0;  // EXTREMO IZQUIERDO
15     float  b = 1000000.0; // EXTREMO DERECHO
16     int    n = 300300000; // NUMERO DE TRAPECIOS
17     float  h;        // LONGITUD DE LA BASE DEL TRAPECIO
18     float  local_a;  // EXTREMO IZQUIERDO PARA MI PROCESO
19     float  local_b;  // EXTREMO DERECHO PARA MI PROCESO
20     int    local_n;  // NUMERO DE TRAPECIOS PARA EL CALCULO
21     float  integral; // INTEGRAL SOBRE MI INTERVALO
22     float  total;    // INTEGRAL TOTAL
23     int    fuente;   // PROCESO QUE ENVIA RESULTADO DE INTEGRAL
24     int    dest = 0; // DESTINATARIO: TODOS LOS MENSAJES VAN A 0
25     int    etiqueta = 0;
26     double t1,t2;
27     MPI_Status status;
28
29     float Trapecio(float local_a, float local_b, int local_n,
30                   float h); // CALCULA LA INTEGRAL LOCAL (AREA DEL TRAPECIO)
31
32     MPI_Init(&argc, &argv); // INICIALIZA MPI ARRANCAR MPI
33     t1=MPI_Wtime();
34
35     MPI_Comm_size(MPI_COMM_WORLD, &p); // mi_rango <-- RANGO DE MI PROCESO
36     MPI_Comm_rank(MPI_COMM_WORLD, &mi_rango);
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99 /* FUNCION QUE ESTAMOS INTEGRANDO */
100 float f(float x) {
101     float return_val;
102     /* CALCULA f(x) Y DEVUELVE SU VALOR */
103     return_val = log ( pow ( x, 10) );
104     return return_val;
105 } /* f */
106
107
108

```

(Figura 26)

Con estos datos, ejecuto el programa y anoto el tiempo de ejecución obtenido para un número de procesos desde 1 hasta 16 por ejemplo. Figura 27.

```
practicas@ATC-203: ~/practica-mpi
Archivo Editar Ver Buscar Terminal Ayuda
PI = 3.141635
practicas@ATC-203:~/practica-mpi$ mpirun -hostfile slots.conf -np 1 integracion_trapecios_pi_esqueleto_modif
ESTIMACION USANDO n=300300000 TRAPECIOS,
DE LA INTEGRAL DESDE 1.000000 HASTA 1000000.000000 = 7151120.000000
El tiempo de ejecución es de: 9.042610
PI = 14302241.000000
practicas@ATC-203:~/practica-mpi$ mpirun -hostfile slots.conf -np 2 integracion_trapecios_pi_esqueleto_modif
ESTIMACION USANDO n=300300000 TRAPECIOS,
DE LA INTEGRAL DESDE 1.000000 HASTA 1000000.000000 = 21453362.000000
El tiempo de ejecución es de: 4.634475
PI = 42906724.000000
practicas@ATC-203:~/practica-mpi$ mpirun -hostfile slots.conf -np 3 integracion_trapecios_pi_esqueleto_modif
ESTIMACION USANDO n=300300000 TRAPECIOS,
DE LA INTEGRAL DESDE 1.000000 HASTA 1000000.000000 = 28604482.000000
El tiempo de ejecución es de: 3.116115
PI = 57208964.000000
practicas@ATC-203:~/practica-mpi$ mpirun -hostfile slots.conf -np 4 integracion_trapecios_pi_esqueleto_modif
ESTIMACION USANDO n=300300000 TRAPECIOS,
DE LA INTEGRAL DESDE 1.000000 HASTA 1000000.000000 = 42906724.000000
El tiempo de ejecución es de: 2.424761
PI = 85813440.000000
practicas@ATC-203:~/practica-mpi$ mpirun -hostfile slots.conf -np 5 integracion_trapecios_pi_esqueleto_modif
ESTIMACION USANDO n=300300000 TRAPECIOS,
DE LA INTEGRAL DESDE 1.000000 HASTA 1000000.000000 = 57208964.000000
El tiempo de ejecución es de: 1.921256
PI = 114417928.000000
practicas@ATC-203:~/practica-mpi$ mpirun -hostfile slots.conf -np 6 integracion_trapecios_pi_esqueleto_modif
ESTIMACION USANDO n=300300000 TRAPECIOS,
DE LA INTEGRAL DESDE 1.000000 HASTA 1000000.000000 = 64300084.000000
El tiempo de ejecución es de: 1.575631
PI = 128720168.000000
practicas@ATC-203:~/practica-mpi$ mpirun -hostfile slots.conf -np 7 integracion_trapecios_pi_esqueleto_modif
ESTIMACION USANDO n=300300000 TRAPECIOS,
DE LA INTEGRAL DESDE 1.000000 HASTA 1000000.000000 = 78662328.000000
El tiempo de ejecución es de: 1.634440
PI = 157324656.000000
practicas@ATC-203:~/practica-mpi$ mpirun -hostfile slots.conf -np 8 integracion_trapecios_pi_esqueleto_modif
ESTIMACION USANDO n=300300000 TRAPECIOS,
DE LA INTEGRAL DESDE 1.000000 HASTA 1000000.000000 = 92964568.000000
El tiempo de ejecución es de: 1.435202
PI = 185929136.000000
practicas@ATC-203:~/practica-mpi$ mpirun -hostfile slots.conf -np 9 integracion_trapecios_pi_esqueleto_modif
ESTIMACION USANDO n=300300000 TRAPECIOS,
DE LA INTEGRAL DESDE 1.000000 HASTA 1000000.000000 = 100115688.000000
El tiempo de ejecución es de: 1.312120
PI = 208231376.000000
practicas@ATC-203:~/practica-mpi$ mpirun -hostfile slots.conf -np 10 integracion_trapecios_pi_esqueleto_modif
ESTIMACION USANDO n=300300000 TRAPECIOS,
DE LA INTEGRAL DESDE 1.000000 HASTA 1000000.000000 = 114417928.000000
El tiempo de ejecución es de: 1.186031
PI = 228835056.000000
practicas@ATC-203:~/practica-mpi$ mpirun -hostfile slots.conf -np 11 integracion_trapecios_pi_esqueleto_modif
ESTIMACION USANDO n=300300000 TRAPECIOS,
DE LA INTEGRAL DESDE 1.000000 HASTA 1000000.000000 = 128720168.000000
El tiempo de ejecución es de: 1.079849
PI = 257440336.000000
practicas@ATC-203:~/practica-mpi$ mpirun -hostfile slots.conf -np 12 integracion_trapecios_pi_esqueleto_modif
ESTIMACION USANDO n=300300000 TRAPECIOS,
DE LA INTEGRAL DESDE 1.000000 HASTA 1000000.000000 = 135428608.000000
El tiempo de ejecución es de: 0.995503
PI = 270857216.000000
practicas@ATC-203:~/practica-mpi$ mpirun -hostfile slots.conf -np 12 integracion_trapecios_pi_esqueleto_modif
ESTIMACION USANDO n=300300000 TRAPECIOS,
DE LA INTEGRAL DESDE 1.000000 HASTA 1000000.000000 = 135428608.000000
El tiempo de ejecución es de: 0.993377
PI = 270857216.000000
practicas@ATC-203:~/practica-mpi$ mpirun -hostfile slots.conf -np 13 integracion_trapecios_pi_esqueleto_modif
ESTIMACION USANDO n=300300000 TRAPECIOS,
DE LA INTEGRAL DESDE 1.000000 HASTA 1000000.000000 = 136560464.000000
El tiempo de ejecución es de: 1.081084
PI = 273120928.000000
practicas@ATC-203:~/practica-mpi$ mpirun -hostfile slots.conf -np 14 integracion_trapecios_pi_esqueleto_modif
ESTIMACION USANDO n=300300000 TRAPECIOS,
DE LA INTEGRAL DESDE 1.000000 HASTA 1000000.000000 = 132492224.000000
El tiempo de ejecución es de: 1.151604
PI = 264984448.000000
practicas@ATC-203:~/practica-mpi$ mpirun -hostfile slots.conf -np 15 integracion_trapecios_pi_esqueleto_modif
ESTIMACION USANDO n=300300000 TRAPECIOS,
DE LA INTEGRAL DESDE 1.000000 HASTA 1000000.000000 = 132674720.000000
El tiempo de ejecución es de: 1.072343
PI = 265349440.000000
practicas@ATC-203:~/practica-mpi$ mpirun -hostfile slots.conf -np 16 integracion_trapecios_pi_esqueleto_modif
ESTIMACION USANDO n=300300000 TRAPECIOS,
DE LA INTEGRAL DESDE 1.000000 HASTA 1000000.000000 = 132055104.000000
El tiempo de ejecución es de: 1.122465
PI = 264110208.000000
```

(Figura 27)

Anoto en un archivo de texto “tiempo.dat” todos los tiempos de ejecución que he obtenido en la figura de arriba en la columna de la derecha y el número de procesos a la izquierda.

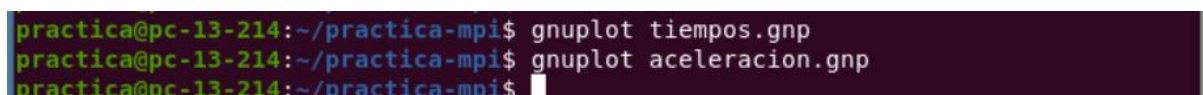
```
tiempo.dat: Bloc de notas
Archivo Editar Ver
1 9.042610
2 4.634475
3 3.116115
4 2.424761
5 1.921256
6 1.575631
7 1.634440
8 1.435202
9 1.312120
10 1.186031
11 1.079849
12 0.995503
13 1.081084
14 1.151604
15 1.072343
16 1.109197
Ln 1, Col 1 | 100% | Unix (LF) | UTF-8
```

Represento una gráfica del tiempo de ejecución respecto al número de procesos. Para ello tengo un archivo llamado “tiempos.gnp” el cual modifico como quiera para que luego al ejecutar “gnuplot tiempos.gnp” se me descargue una imagen con la gráfica.

A screenshot of a text editor window titled 'tiempos.gnp' with the path '~/.practica-mpi'. The editor contains gnuplot commands for plotting execution time against the number of processes. The commands are: 1 reset, 2 set terminal png, 3 set output 'tiempos.png', 4 set xlabel "Número de procesos", 5 set ylabel "Tiempo de ejecución", 6 set xtics 1, 7 set ytics 1, 8 set title "Tiempo de ejecución con respecto al número de procesos", 9, 10 #set key graph 0.87,0.95 box, 11, 12 plot 'tiempo.dat' notitle with linespoints ls 5. The status bar at the bottom shows 'Texto plano', 'Anchura del tabulador: 8', 'Ln 12, Col 48', and 'INS'.

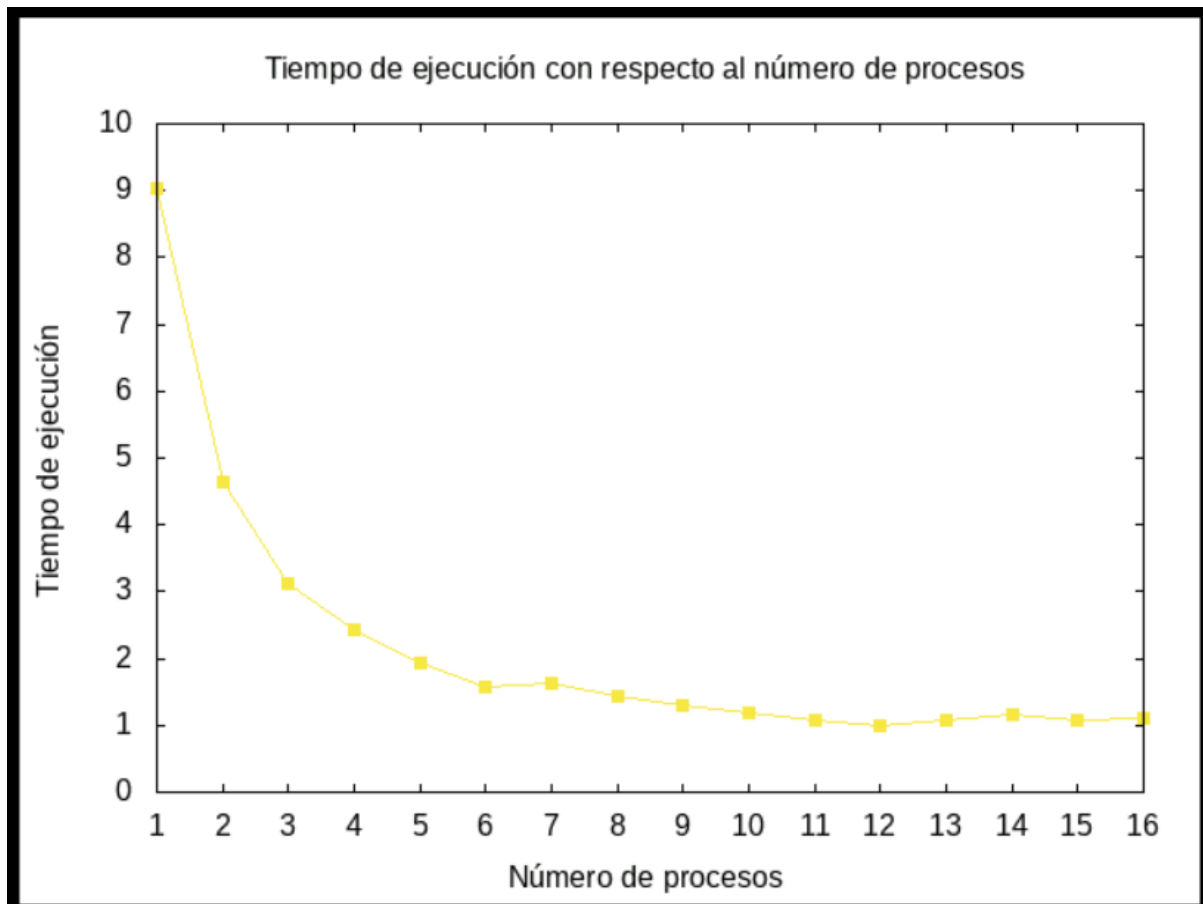
```
1 reset
2 set terminal png
3 set output 'tiempos.png'
4 set xlabel "Número de procesos"
5 set ylabel "Tiempo de ejecución"
6 set xtics 1
7 set ytics 1
8 set title "Tiempo de ejecución con respecto al número de procesos"
9
10 #set key graph 0.87,0.95 box
11
12 plot 'tiempo.dat' notitle with linespoints ls 5
```

En la consola de comandos escribo “gnuplot tiempos.gnp” el cual al ejecutarlo se crea directamente una imagen.

A screenshot of a terminal window showing the execution of gnuplot commands. The prompt is 'practica@pc-13-214:~/practica-mpi\$'. The first command is 'gnuplot tiempos.gnp' and the second is 'gnuplot aceleracion.gnp'.

```
practica@pc-13-214:~/practica-mpi$ gnuplot tiempos.gnp
practica@pc-13-214:~/practica-mpi$ gnuplot aceleracion.gnp
practica@pc-13-214:~/practica-mpi$
```

La imagen a continuación muestra que al aumentar el número de procesos disminuye el tiempo de ejecución. Este se hace notable en el cambio de 1 a 2 procesos. A partir de 2 procesos disminuye el tiempo, pero menos hasta que el número de procesos es 6 ya que el pc tiene 6 cores físicos, el siguiente (7) aumenta un poco y vuelve a disminuir hasta 12 que son los cores lógicos del pc. Es decir, se va alcanzando un valor asintótico, por mucho que aumente el número de procesos (procesadores) el tiempo no disminuye mucho. (Figura 28)



(Figura 28)

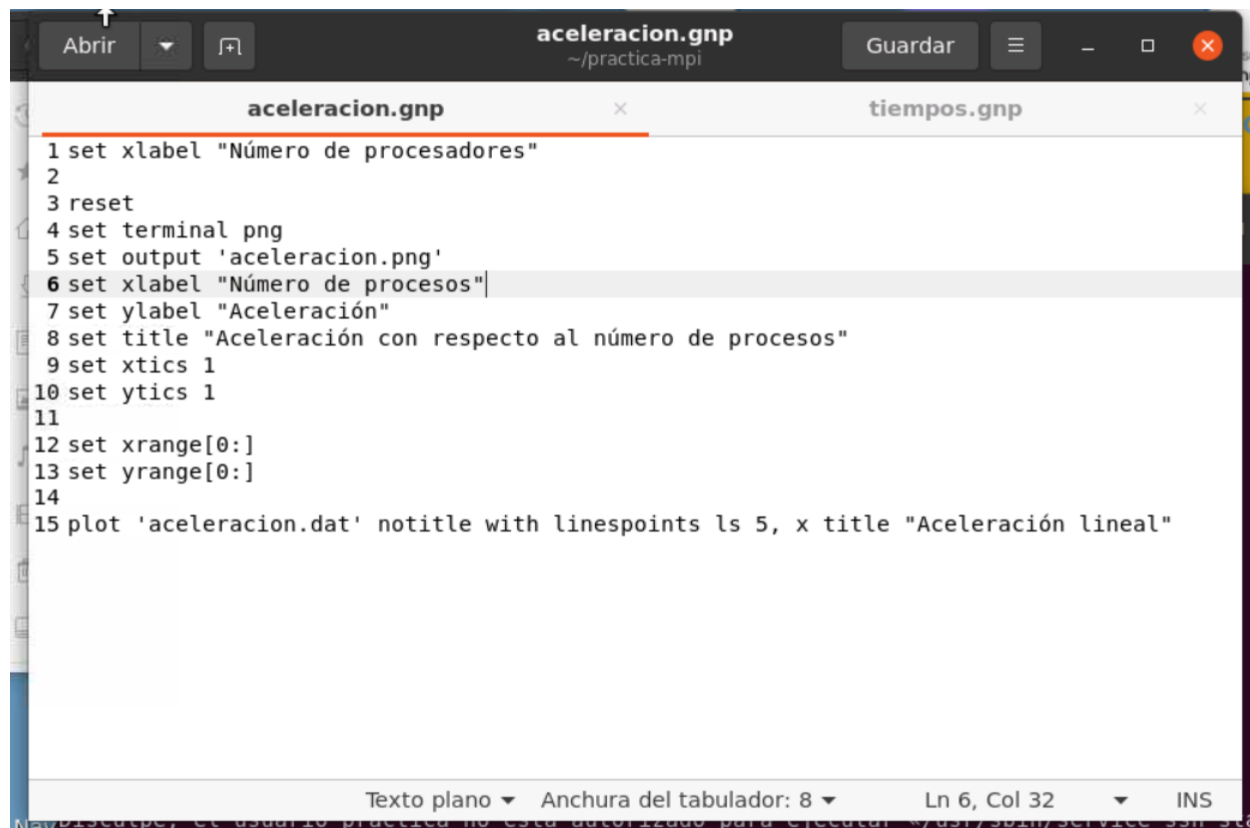
Ahora hago lo mismo, pero con la aceleración del programa paralelo. Divido el tiempo de ejecución de la aplicación paralela ejecutada con un único procesador entre el tiempo de ejecución de la aplicación paralela corriendo sobre p procesadores del mismo computador.

```
aceleracion.dat: Bloc de not...
Archivo  Editar  Ver  ⚙️

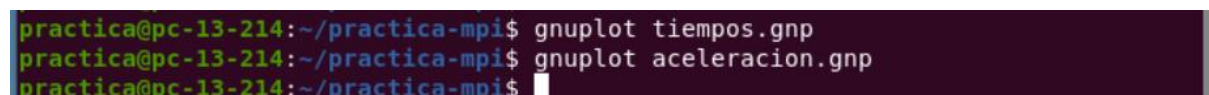
1 1
2 1.951161674
3 2.901885842
4 3.729278886
5 4.706613798
6 5.739040423
7 5.532543256
8 6.300583472
9 6.891602902
10 7.624261086
11 8.37395784
12 9.083458312
13 8.364391666
14 7.852187037
15 8.432572414
16 8.152393128

Ln 1, Col 1  100%  Unix (LF)  UTF-8
```

Represento una gráfica de la aceleración respecto al número de procesos Para ello tengo un archivo llamado “aceleracion.gnp” el cual modifiko como quiera para que luego al ejecutar “gnuplot “aceleracion.gnp” se me descargue una imagen con la gráfica.

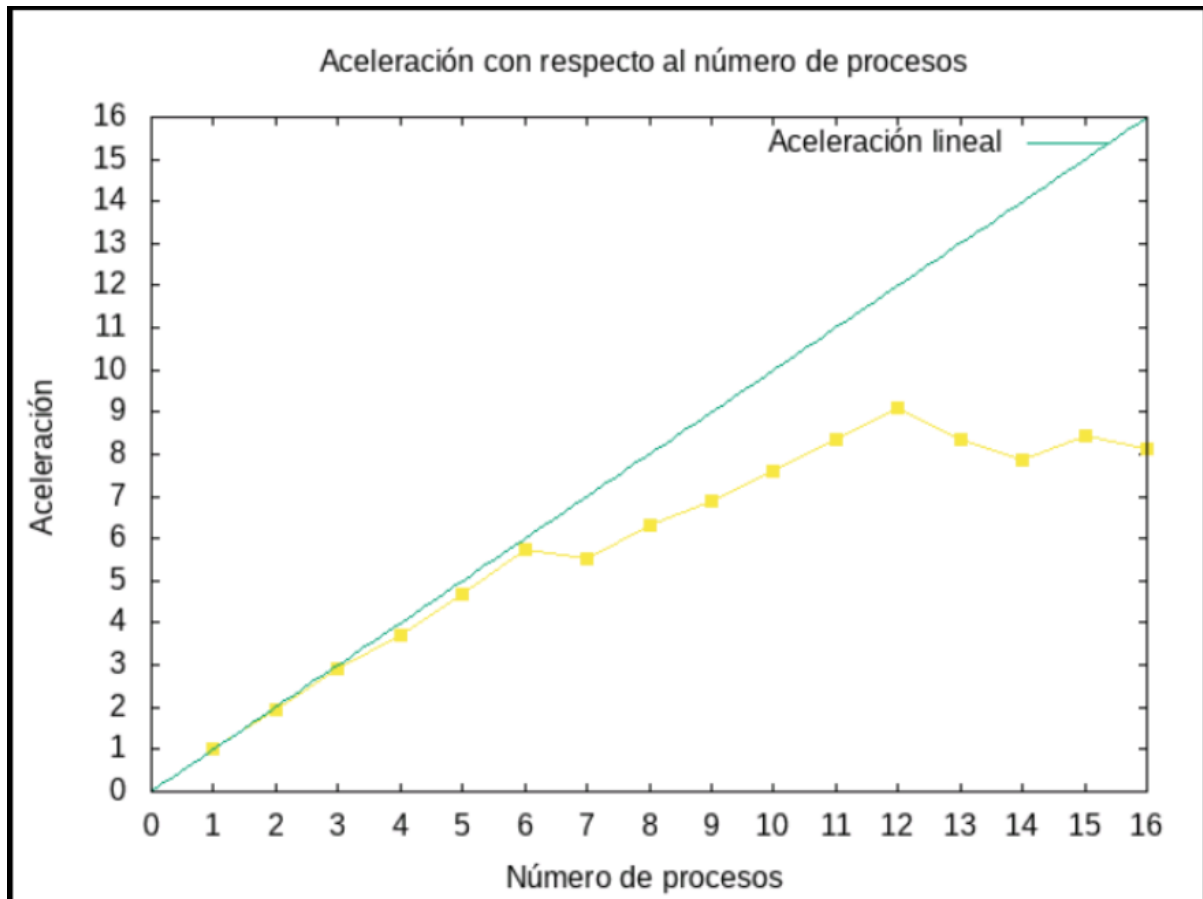
A screenshot of a text editor window titled 'aceleracion.gnp' with a subtitle '~/.practica-mpi'. The window contains 15 lines of gnuplot code. The code sets the x-axis label to 'Número de procesadores', resets the plot, sets the terminal to 'png', sets the output file to 'aceleracion.png', sets the x-axis label to 'Número de procesos', sets the y-axis label to 'Aceleración', sets the title to 'Aceleración con respecto al número de procesos', sets x and y ticks to 1, sets x and y ranges to [0:], and finally plots 'aceleracion.dat' with lines and points, using a line style of 5 and a title 'Aceleración lineal'. The status bar at the bottom indicates 'Texto plano', 'Anchura del tabulador: 8', 'Ln 6, Col 32', and 'INS'.

En la consola de comandos escribo “gnuplot tiempos.gnp” el cual al ejecutarlo se crea directamente una imagen.

A screenshot of a terminal window showing three commands being executed. The first command is 'gnuplot tiempos.gnp', the second is 'gnuplot aceleracion.gnp', and the third is an empty prompt. The terminal prompt is 'practica@pc-13-214:~/practica-mpi\$'.

En la gráfica siguiente ocurre lo mismo que con la gráfica de tiempo, pero aumentando los dos.

Al aumentar el número de procesos aumenta la aceleración. Este se hace notable en el cambio de 1 a 6 proceso ya que el pc tiene 6 cores físicos, el siguiente (7) disminuye un poco y vuelve a aumentar hasta 12 procesos que son los cores lógicos del pc. A partir de ahí se va alcanzando un valor asintótico , por mucho que aumente el número de procesos (procesadores) la aceleración no aumenta mucho o nada. (Figura 29)



(Figura 29)

Ejercicio 2.3 – Lanzar la aplicación paralela desarrollada en la segunda parte sobre tres Pcs del aula.

Este no se puede hacer porque cuando pongo el comando “sudo service ssh start” para activar el dominio SSH en todos los nodos adicionales donde se va a correr el programa me devuelve un error , al intentar hacerlo en el centro de cálculo por falta de tiempo los pcs de allí no tiene permiso para ejecutar dicho comando.

```
practica@pc-13-214:~/practica-mpi$ sudo service ssh start
[sudo] contraseña para practica:
Disculpe, el usuario practica no está autorizado para ejecutar «/usr/sbin/service ssh start»
como root en pc-13-214.etsii
practica@pc-13-214:~/practica-mpi$
```

Simplemente habría que seguir los pasos indicados en la práctica.

- Se activa el dominio SSH en todos los pcs donde se va a correr el programa
- Se genera la pareja de claves SSH privada y pública en el pc principal
- Se le envía la clave pública a los demás pcs donde se va a correr el programa
- Se crea un archivo de texto “hostfile” con una lista de los pcs donde se va a ejecutar los procesos, el número de slots de cada uno y su número de slots máximo.

- Se copia en los demás pcs el directorio donde está la aplicación a ejecutar
- Se ejecutan en los demás pcs indicando que se utilice el archivo hostfile y sus direcciones IP.

Para realizar las gráficas del tiempo de ejecución y de la aceleración se haría los mismos pasos que he seguido anteriormente en la actividad 2.2