Antonio Javier Rodríguez Romero

OpenMP coprocesadores

1. Consideraciones previas

- Se usará el compilador nvc de Nvidia, en particular, se utilizará la versión 21.2 que está instalado en el nodo atcgrid4 (se debe tener en cuenta que distintas versiones de nvc podrían generar distinto código ejecutable).
- El objetivo de estos ejercicios es habituarse a la organización de la GPU y al compilador, y entender la sobrecarga que introduce el uso del coprocesador (GPU, en este caso).
- El compilador nvc espera que el código termine con un salto de línea
- Entregar este fichero en pdf (AC_OpenMPCoprocesadores_ApellidosNombre.pdf) en un zip junto con los códigos implementados (AC_OpenMPCoprocesadores_ApellidosNombre.zip).
- Los códigos debe mantenerlos en su directorio de atcgrid hasta final del curso.
- El tamaño de la letra en las capturas debe ser similar al tamaño de la letra en este documento.
- En las capturas de pantalla se debe ver el nombre del usuario, fecha y hora.
  - Debe modificar el prompt en los computadores que utilice para que aparezca su nombre y apellidos, su usuario (`\u`), el computador (`\h`), el directorio de trabajo del bloque práctico (`\w`), la fecha (`\D`) completa (`%F`) y el día (`%A`). Para modificar el prompt utilice lo siguiente (si es necesario, use `export` delante):

    ```
    PS1="[NombreApellidos \u@\h:\w] \D{%F %A}\n$"
    ```
    (.bash_profile)

    donde NombreApellidos es su nombre seguido de sus apellidos, por ejemplo: JuanOrtuñoVilariño

Ejercicios basados en los ejemplos del seminario

1. **(a)** Compilar el ejemplo `omp_offload.c` del seminario en el nodo atcgrid4::

```
srun –pac4 –Aac nvc -O2 -openmp -mp=gpu omp_offload.c -o omp_offload_GPU
```

(`-openmp` para que tenga en cuenta las directivas OpenMP y `-mp=gpu` para que el código delimitado con `target` se genere para un dispositivo gpu)

Ejecutar `omp_offload_GPU` usando:

```
srun –pac4 –Aac omp_offload_GPU 35 3 32
```

CONTENIDO FICHERO: `salida.txt` (**destaque en el resultado de la ejecución con colores las respuestas a las preguntas (b)-(e))**

```
Target device: 1
Tiempo:0.146371126
Iteracción 0, en thread 0/32 del team 0/3
Iteracción 1, en thread 1/32 del team 0/3
Iteracción 2, en thread 2/32 del team 0/3
Iteracción 3, en thread 3/32 del team 0/3
Iteracción 4, en thread 4/32 del team 0/3
Iteracción 5, en thread 5/32 del team 0/3
Iteracción 6, en thread 6/32 del team 0/3
```

```
Iteracción 7, en thread 7/32 del team 0/3
Iteracción 8, en thread 8/32 del team 0/3
Iteracción 9, en thread 9/32 del team 0/3
Iteracción 10, en thread 10/32 del team 0/3
Iteracción 11, en thread 11/32 del team 0/3
Iteracción 12, en thread 12/32 del team 0/3
Iteracción 13, en thread 13/32 del team 0/3
Iteracción 14, en thread 14/32 del team 0/3
Iteracción 15, en thread 15/32 del team 0/3
Iteracción 16, en thread 16/32 del team 0/3
Iteracción 17, en thread 17/32 del team 0/3
Iteracción 18, en thread 18/32 del team 0/3
Iteracción 19, en thread 19/32 del team 0/3
Iteracción 20, en thread 20/32 del team 0/3
Iteracción 21, en thread 21/32 del team 0/3
Iteracción 22, en thread 22/32 del team 0/3
Iteracción 23, en thread 23/32 del team 0/3
Iteracción 24, en thread 24/32 del team 0/3
Iteracción 25, en thread 25/32 del team 0/3
Iteracción 26, en thread 26/32 del team 0/3
Iteracción 27, en thread 27/32 del team 0/3
Iteracción 28, en thread 28/32 del team 0/3
Iteracción 29, en thread 29/32 del team 0/3
Iteracción 30, en thread 30/32 del team 0/3
Iteracción 31, en thread 31/32 del team 0/3
Iteracción 32, en thread 0/32 del team 1/3
Iteracción 33, en thread 1/32 del team 1/3
Iteracción 34, en thread 2/32 del team 1/3
```

Contestar las siguientes preguntas:

(a) ¿Cuántos equipos (*teams*) se han creado y cuántos se han usado realmente en la ejecución?

RESPUESTA: Se han creado 3 y utilizado 2

(b) ¿Cuántos hilos (*theads*) se han creado en cada equipo y cuántos de esos hilos se han usado en la ejecución?

RESPUESTA: Se han creado 32 hilos por equipo, de los cuales se han utilizado los 32 del equipo 0 y los 3 primeros del equipo 1.

(c) ¿Qué número máximo de iteraciones se ha asignado a un hilo?

RESPUESTA: Una iteración por cada hilo

(d) ¿Qué número mínimo de iteraciones se ha asignado a un equipo y cuál es ese equipo?

RESPUESTA: 3 iteraciones al equipo 1

2. Eliminar en `opp_offload.c` `num_teams(nteams)` y `thread_limit(mthreads)` y la entrada como parámetros de `nteams` y `mthreads`. Llamar al código resultante `opp_offload2.c`. Compilar y ejecutar el código para poder contestar a las siguientes preguntas:

(a) ¿Qué número de equipos y de hilos por equipo se usan por defecto?

RESPUESTA: Se usan por defecto 48 equipos y 1024 hilos por equipo

CAPTURA (que muestre el envío a la cola y el resultado de la ejecución)

```
[AntonioJavierRodríguezRomero ac441@atcgrid:~] 2023-04-18 martes
$srun -pac4 -Aac omp_offload2_GPU 35
Target device: 1
Tiempo:0.136998892
Iteracción 0, en thread 0/1024 del team 0/48
Iteracción 1, en thread 1/1024 del team 0/48
Iteracción 2, en thread 2/1024 del team 0/48
Iteracción 3, en thread 3/1024 del team 0/48
Iteracción 4, en thread 4/1024 del team 0/48
Iteracción 5, en thread 5/1024 del team 0/48
Iteracción 6, en thread 6/1024 del team 0/48
Iteracción 7, en thread 7/1024 del team 0/48
Iteracción 8, en thread 8/1024 del team 0/48
Iteracción 9, en thread 9/1024 del team 0/48
Iteracción 10, en thread 10/1024 del team 0/48
Iteracción 11, en thread 11/1024 del team 0/48
Iteracción 12, en thread 12/1024 del team 0/48
Iteracción 13, en thread 13/1024 del team 0/48
Iteracción 14, en thread 14/1024 del team 0/48
Iteracción 15, en thread 15/1024 del team 0/48
Iteracción 16, en thread 16/1024 del team 0/48
Iteracción 17, en thread 17/1024 del team 0/48
Iteracción 18, en thread 18/1024 del team 0/48
Iteracción 19, en thread 19/1024 del team 0/48
Iteracción 20, en thread 20/1024 del team 0/48
Iteracción 21, en thread 21/1024 del team 0/48
Iteracción 22, en thread 22/1024 del team 0/48
Iteracción 23, en thread 23/1024 del team 0/48
Iteracción 24, en thread 24/1024 del team 0/48
Iteracción 25, en thread 25/1024 del team 0/48
Iteracción 26, en thread 26/1024 del team 0/48
Iteracción 27, en thread 27/1024 del team 0/48
Iteracción 28, en thread 28/1024 del team 0/48
Iteracción 29, en thread 29/1024 del team 0/48
Iteracción 30, en thread 30/1024 del team 0/48
Iteracción 31, en thread 31/1024 del team 0/48
Iteracción 32, en thread 32/1024 del team 0/48
Iteracción 33, en thread 33/1024 del team 0/48
Iteracción 34, en thread 34/1024 del team 0/48
[AntonioJavierRodríguezRomero ac441@atcgrid:~] 2023-04-18 martes
```

(b) ¿Es posible relacionar estos números con alguno de los parámetros, comentados en el seminario, que caracterizan al coprocesador que estamos usando? ¿Con cuáles?

RESPUESTA:Se relaciona con los valores del número de filas del componente "*Streaming Multiprocessor(SM) SIMT*", **48**, y el máximo de threads por SM, **1024**.

(c) ¿De qué forma se asignan por defecto las iteraciones del bucle a los equipos y a los hilos dentro de un equipo? Contestar además las siguientes preguntas: ¿a qué equipo y a qué hilo de ese equipo se asigna la iteración 2? Y ¿a qué equipo y a qué hilo de ese equipo se asigna la iteración 1025, si la hubiera? (realizar las ejecuciones que se consideren necesarias para contestar a esta pregunta, en particular, alguna ejecución con un número de iteraciones de al menos 1025)

RESPUESTA: Se le asignará por defecto una iteración a cada hebram comenzando por la hebra con identificador 0 del equipo 0, así completando todas las hebras de este antes de pasar al siguiente. De esta forma, la iteración 2 se asignaría al equipo 0, en concreto a la hebra 2. La iteración 1025, si hubiera, se llevaría acabo en el equipo 1 y en la hebra 1 de este.

3. Ejecutar la versión original, `omp_offload`, con varios valores de entrada hasta que se pueda contestar a las siguientes cuestiones:

(a) ¿Se crean cualquier número de hilos (*threads*) por equipo que se ponga en la entrada al programa? (probar también con algún valor mayor que 3000) En caso negativo, ¿qué número de hilos por equipo son posibles?

RESPUESTA: En el caso de que se especifiquen menos de 32 *threads*, se tomarán

32 siempre debido a que estos se agrupan en *warps* que comprenden este mismo número de hilos. De esta forma, al menos siempre se utilizará una unidad de estos. Si se especifican más de 1024, se tomarán 1024 siempre y cuando no se llegue a intentar utilizar un *warp* más, es decir, a partir de 1056. Una vez alcanzado este número, se dará un *segmentation fault.*

CAPTURAS (que justifiquen la respuesta)

```
[AntonioJavierRodríguezRomero ac441@atcgrid:~] 2023-04-18 martes
$srun -pac4 -Aac omp_offload_GPU 35 3 16
Target device: 1
Tiempo:0.141628981
Iteracción 0, en thread 0/32 del team 0/3
Iteracción 1, en thread 1/32 del team 0/3
Iteracción 2, en thread 2/32 del team 0/3
Iteracción 3, en thread 3/32 del team 0/3
Iteracción 4, en thread 4/32 del team 0/3
Iteracción 5, en thread 5/32 del team 0/3
Iteracción 6, en thread 6/32 del team 0/3
Iteracción 7, en thread 7/32 del team 0/3
Iteracción 8, en thread 8/32 del team 0/3
Iteracción 9, en thread 9/32 del team 0/3
Iteracción 10, en thread 10/32 del team 0/3
Iteracción 11, en thread 11/32 del team 0/3
Iteracción 12, en thread 12/32 del team 0/3
Iteracción 13, en thread 13/32 del team 0/3
Iteracción 14, en thread 14/32 del team 0/3
Iteracción 15, en thread 15/32 del team 0/3
Iteracción 16, en thread 16/32 del team 0/3
Iteracción 17, en thread 17/32 del team 0/3
Iteracción 18, en thread 18/32 del team 0/3
Iteracción 19, en thread 19/32 del team 0/3
Iteracción 20, en thread 20/32 del team 0/3
Iteracción 21, en thread 21/32 del team 0/3
Iteracción 22, en thread 22/32 del team 0/3
Iteracción 23, en thread 23/32 del team 0/3
Iteracción 24, en thread 24/32 del team 0/3
Iteracción 25, en thread 25/32 del team 0/3
Iteracción 26, en thread 26/32 del team 0/3
Iteracción 27, en thread 27/32 del team 0/3
Iteracción 28, en thread 28/32 del team 0/3
Iteracción 29, en thread 29/32 del team 0/3
Iteracción 30, en thread 30/32 del team 0/3
Iteracción 31, en thread 31/32 del team 0/3
Iteracción 32, en thread 0/32 del team 1/3
Iteracción 33, en thread 1/32 del team 1/3
Iteracción 34, en thread 2/32 del team 1/3
[AntonioJavierRodríguezRomero ac441@atcgrid:~] 2023-04-18 martes
```

```
[AntonioJavierRodríguezRomero ac441@atcgrid:~] 2023-04-18 martes
$srun -pac4 -Aac omp_offload_GPU 35 3 1056
Target device: 1
Fatal error: Could not launch CUDA kernel on device 0, error 1
srun: error: atcgrid4: task 0: Aborted (core dumped)
[AntonioJavierRodríguezRomero ac441@atcgrid:~] 2023-04-18 martes
```

```
[AntonioJavierRodríguezRomero ac441@atcgrid:~] 2023-04-18 martes
$srun -pac4 -Aac omp_offload_GPU 35 3 3000
Target device: 1
Fatal error: Could not launch CUDA kernel on device 0, error 1
```

(b) ¿Es posible relacionar el número de hilos por equipo posibles con alguno o algunos de los parámetros, comentados en el seminario, que caracterizan al coprocesador que se está usando? Indicar cuáles e indicar la relación.

RESPUESTA: Sí, se relaciona con el número máximo de *threads por SM* o *CUDA Block*.

4. Eliminar las directivas `teams` y `distribute` en `omp_offload2.c`, llamar al código resultante `opp_offload3.c`. Compilar y ejecutar este código para poder contestar a las siguientes preguntas:

(a) ¿Qué número de equipos y de hilos por equipo se usan por defecto?

RESPUESTA: Se usa por defecto un equipo con 1024 hilos

(b) ¿Qué tanto por ciento del número de núcleos de procesamiento paralelo de la GPU se están utilizando? Justificar respuesta.

RESPUESTA: Por las trasparencias sabemos que el coprocesador GPU consta de 38 *SM*, cada uno de ellos compuesto por núcleos *CUDA.* Como en total contamos con 3072 núcleos, sabemos que existirán 3072/48 = 64 núcleos por *SM.*

En este caso, solo utilizamos un equipo, un *SM*, y como tomamos un mayor número de *threads* que núcleos hay en este, todos estos estarán trabajando. De esta forma obtenemos que 64núcleos trabajando/3072núcleos total= 0.2083=2.083%

5. En el código `daxpbyz32_ompoff.c` se calcula (`a` y `b` son escalares, `x`, `y` y `z` son vectores):

$$z = a \cdot x + b \cdot y$$

Se han introducido funciones `omp_get_wtime()` para obtener el tiempo de ejecución de las diferentes construcciones/directivas `target` utilizadas en el código.

1) t2-t1 es el tiempo de `target enter data`, que reserva de espacio en el dispositivo coprocesador para `x`, `y`, `z`, `N` y `p`, y transfiere del host al coprocesador de aquellas que se mapean con `to` (`x`, `N` y `p`).
2) t3-t2 es el tiempo del primer `target teams distribute parallel for` del código, que se ejecuta en paralelo en el coprocesador del bucle:

```
for (int i = 0; i < N; i++) z[i] = p * x[i];
```

3) t4-t3 es el tiempo de `target update`, que transfiere del host al coprocesador `p` e `y`.
4) t5-t4 es el tiempo del segundo target `teams distribute parallel for` del código, que ejecuta en paralelo en el coprocesador del bucle:

```
for (int i = 0; i < N; i++) z[i] = z[i] + p * y[i];
```

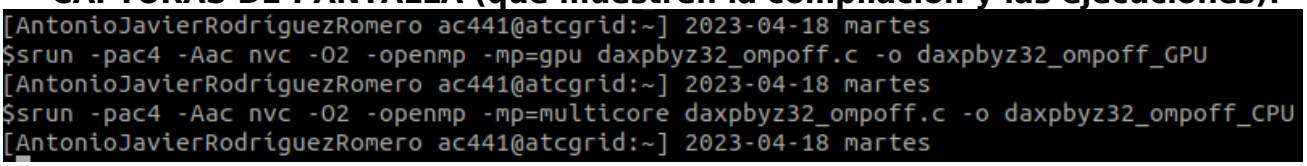
5) t6-t7 es el tiempo que supone `target exit data`, que transfiere los resultados de las variables con `from` y libera el espacio ocupado en la memoria del coprocesador.

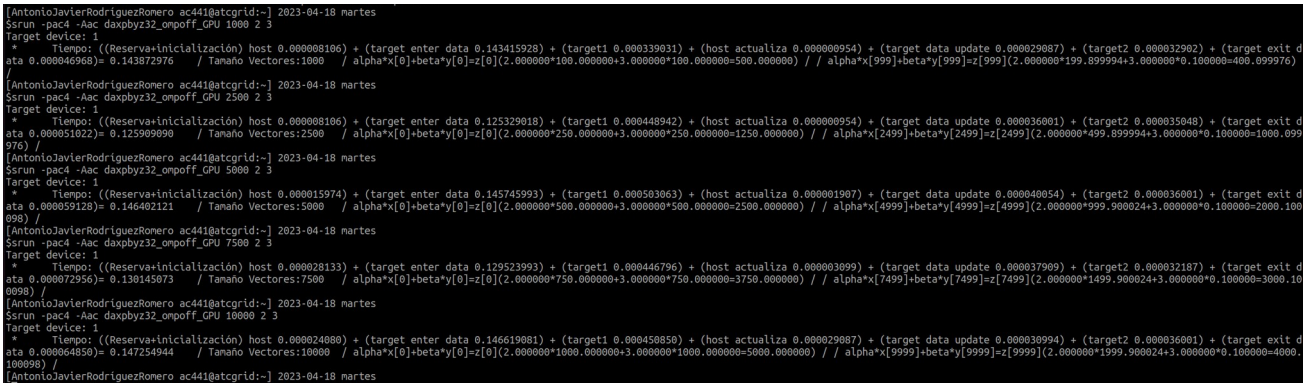
Compilar `daxpbyz32_off.c` para la GPU y para las CPUs de atctrid4 usando:

```
srun –pac4 –Aac nvc -O2 -openmp -mp=gpu daxpbyz32_ompoff.c -o daxpbyz32_ompoff_GPU
srun –pac4 –Aac nvc -O2 -openmp -mp=multicore daxpbyz32_ompoff.c -o daxpbyz32_ompoff_CPU
```

En `daxpbyz32_off_GPU` el coprocesador será la GPU del nodo y, en `daxpbyz32_off_CPU`, será el propio host. En ambos casos la ejecución aprovecha el paralelismo a nivel de flujo de instrucciones del coprocesador. Ejecutar ambos para varios valores de entrada usando un número de componentes `N` para los vectores entre 1000 y 100000 y contestar a las siguientes preguntas.

CAPTURAS DE PANTALLA (que muestren la compilación y las ejecuciones):





```
[AntonioJavierRodriguezRomero ac441@atcgrid:~] 2023-04-18 martes
$srun -pac4 -Aac daxpbyz32_ompoff_CPU 1000 2 3
Target device: 0
 *      Tiempo: ((Reserva+inicialización) host 0.000007868) + (target enter data 0.000000000) + (target1 0.002027035) + (host actualiza 0.000001907) + (target data update 0.000000000) + (target2 0.000006199) + (target exit d
ata 0.000000000)= 0.002043009    / Tamaño Vectores:1000 / alpha*x[0]+beta*y[0]=z[0](2.000000*100.000000+3.000000*100.000000=500.000000) / / alpha*x[999]+beta*y[999]=z[999](2.000000*199.899994+3.000000*0.100000=400.099976) /
[AntonioJavierRodriguezRomero ac441@atcgrid:~] 2023-04-18 martes
$srun -pac4 -Aac daxpbyz32_ompoff_CPU 2500 2 3
Target device: 0
 *      Tiempo: ((Reserva+inicialización) host 0.000019073) + (target enter data 0.000000000) + (target1 0.001767874) + (host actualiza 0.000006199) + (target data update 0.000000000) + (target2 0.000010967) + (target exit d
ata 0.000000000)= 0.001804113    / Tamaño Vectores:2500 / alpha*x[0]+beta*y[0]=z[0](2.000000*250.000000+3.000000*250.000000=1250.000000) / / alpha*x[2499]+beta*y[2499]=z[2499](2.000000*499.899994+3.000000*0.100000=1000.09997
6) /
[AntonioJavierRodriguezRomero ac441@atcgrid:~] 2023-04-18 martes
$srun -pac4 -Aac daxpbyz32_ompoff_CPU 5000 2 3
Target device: 0
 *      Tiempo: ((Reserva+inicialización) host 0.000020027) + (target enter data 0.000000000) + (target1 0.001553059) + (host actualiza 0.000014067) + (target data update 0.000000000) + (target2 0.000011921) + (target exit d
ata 0.000000000)= 0.001599073    / Tamaño Vectores:5000 / alpha*x[0]+beta*y[0]=z[0](2.000000*500.000000+3.000000*500.000000=2500.000000) / / alpha*x[4999]+beta*y[4999]=z[4999](2.000000*999.900024+3.000000*0.100000=2000.10009
8) /
[AntonioJavierRodriguezRomero ac441@atcgrid:~] 2023-04-18 martes
$srun -pac4 -Aac daxpbyz32_ompoff_CPU 7500 2 3
Target device: 0
 *      Tiempo: ((Reserva+inicialización) host 0.000026941) + (target enter data 0.000000000) + (target1 0.001579046) + (host actualiza 0.000018835) + (target data update 0.000000000) + (target2 0.000015020) + (target exit d
ata 0.000000000)= 0.001639843    / Tamaño Vectores:7500 / alpha*x[0]+beta*y[0]=z[0](2.000000*750.000000+3.000000*750.000000=3750.000000) / / alpha*x[7499]+beta*y[7499]=z[7499](2.000000*1499.900024+3.000000*0.100000=3000.1000
98) /
[AntonioJavierRodriguezRomero ac441@atcgrid:~] 2023-04-18 martes
$srun -pac4 -Aac daxpbyz32_ompoff_CPU 10000 2 3
Target device: 0
 *      Tiempo: ((Reserva+inicialización) host 0.000035048) + (target enter data 0.000000000) + (target1 0.001991987) + (host actualiza 0.000038862) + (target data update 0.000000000) + (target2 0.000020027) + (target exit d
ata 0.000001192)= 0.002087116    / Tamaño Vectores:10000        / alpha*x[0]+beta*y[0]=z[0](2.000000*1000.000000+3.000000*1000.000000=5000.000000) / / alpha*x[9999]+beta*y[9999]=z[9999](2.000000*1999.900024+3.000000*0.100000
=4000.100098) /
[AntonioJavierRodriguezRomero ac441@atcgrid:~] 2023-04-18 martes
```

(a) ¿Qué construcción o directiva target supone más tiempo en la GPU?, ¿a qué se debe?

RESPUESTA: la directiva *target enter data*, ya que lleva acabo una copia en la memoria de los coprocesadores de las variables especificadas. Al tener que r*ealizar* procesos como mapearlas en memoria, consume más tiempo

(b) ¿Qué construcciones o directivas target suponen más tiempo en la GPU que en la CPU?, ¿a qué se debe?

RESPUESTA: estas son *enter data, data update y exit data*, debido a que en la CPU utiliza la memoria del *host,* de forma que el tiempo de creación, actualización y borrado de las variables es mínimo.

2. Resto de ejercicios

6. A partir del código secuencial que calcula PI, obtener un código paralelo basado en las construcciones/directivas OpenMP para ejecutar código en coprocesadores. El código debe usar como entrada el número de intervalos de integración y debe imprimir el valor de PI calculado, el error cometido y los tiempos (1) del cálculo de pi y (2) de la trasferencia hacia y desde el coprocesador. Generar dos ejecutables, uno que use como coprocesador la CPU y otro que use la GPU. Comparar los tiempos de ejecución total, cálculo y comunicación obtenidos en atcgrid4 con la CPU y la GPU, indicar cuáles son mayores y razonar los motivos.

CAPTURA CÓDIGO FUENTE: `pi-ompoff.c`

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <omp.h>
#include <math.h>

int main(int argc, char **argv)
{

  double t1 = 0.0, t2 = 0.0, t3 = 0.0, t4 = 0.0;
  register double width;
  double sum;
  register int intervals, i;

  //Los procesos calculan PI en paralelo

  if (argc<2) {printf("Falta número de intevalos");exit(-1);}
  intervals=atoi(argv[1]);

  if (intervals<1) {intervals=1E6; printf("Intervalos=%d",intervals);}

  width = 1.0 / intervals;
```

```
  sum = 0;
  t1 = omp_get_wtime();
  #pragma omp target enter data map(to: width, intervals, sum)
  t2 = omp_get_wtime();
  #pragma omp target teams distribute parallel for reduction(+:sum)
  for (i=0; i<intervals; i++) {
    register double x = (i + 0.5) * width;
    sum += 4.0 / (1.0 + x * x);
  }
  t3 = omp_get_wtime();
  #pragma omp target exit data map(delete: intervals, width) map(from: sum)
  sum *= width;
  t4 = omp_get_wtime();
  printf("Iteraciones:\t%d\t. PI:\t%26.24f\t. Tiempo:\t%8.6f\n", intervals,sum,t3-t1);

  printf("Error:\t%8.6f\n", fabs(M_PI-sum));
  printf("(Target enter data): \t%8.6f\n (Target teams distribute parallel for): \t%8.6f\n (Target exit data): \t%8.6f\n",t2-
t1,t3-t2,t4-t3);
  return(0);
}
```

CAPTURAS DE PANTALLA (mostrar la compilación y la ejecución para 10000000 intervalos de integración en atcgrid4 – envío(s) a la cola):

```
[AntonioJavierRodriguezRomero ac441@atcgrid:~] 2023-04-18 martes
$srun -pac4 -Aac nvc -O2 -openmp -mp=gpu pi-ompoff.c -o pi-ompoff_GPU
[AntonioJavierRodriguezRomero ac441@atcgrid:~] 2023-04-18 martes
$srun -pac4 -Aac nvc -O2 -openmp -mp=multicore pi-ompoff.c -o pi-ompoff_CPU
[AntonioJavierRodriguezRomero ac441@atcgrid:~] 2023-04-18 martes
$srun -pac4 -Aac pi-ompoff_GPU 10000000
Iteraciones:    10000000        . PI:   3.14159265358979448265593      . Tiempo:       1.345605
Error:  0.000000
(Target enter data):    1.344140
 (Target teams distribute parallel for):        0.001465
 (Target exit data):    0.000052
[AntonioJavierRodriguezRomero ac441@atcgrid:~] 2023-04-18 martes
$srun -pac4 -Aac pi-ompoff_CPU 10000000
Iteraciones:    10000000        . PI:   3.14159265358978290194613 7      . Tiempo:       0.013116
Error:  0.000000
(Target enter data):    0.000001
 (Target teams distribute parallel for):        0.013115
 (Target exit data):    0.000000
[AntonioJavierRodriguezRomero ac441@atcgrid:~] 2023-04-18 martes
```

RESPUESTA: Vemos como el tiempo de ejecución es bastante mayor en la versión de GPU que en la de CPU. Esto se debe a que, como he mencionado en el apartado anterior, mientras que en el caso de la CPU los tiempos de creación y actualización de ámbitos de variables es mínimo, en el caso de GPU es bastante más lento. Por otro lado, al hacer uso el código de la cláusula *reduction*, obliga en el caso de la GPU a una comunicación entre núcleos que puede ralentizar aún más la ejecución.