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))**

```
[MarioLindezMartinez ac419@atcgrid:~/Actividad_OpenMP] 2023-04-15 sábado
$ srun -pac4 -Aac nvc -O2 -openmp -mp=gpu omp_offload.c -o omp_offload_GPU
[MarioLindezMartinez ac419@atcgrid:~/Actividad_OpenMP] 2023-04-15 sábado
$ srun -pac4 -Aac omp_offload_GPU 35 3 32
Target device: 1
Tiempo:0.113904953
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
```

Amarillo: (b)
Verde: (c)
Rojo: (d)
Rosa: (e)

Contestar las siguientes preguntas:

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

RESPUESTA: Se han creado tres equipos aunque en la ejecución se han llegado a usar únicamente dos.

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

RESPUESTA: En cada equipo se han creado 32 hilos. Mientras que del equipo 0 sí se han usado todos los threads, del equipo 1 tan solo se han usado 3 de los 32 hilos creados.

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

RESPUESTA: Se ha asignado un máximo de una iteración por hilo, ya que observamos que cada thread realiza una única iteración.

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

RESPUESTA: Se ha establecido un número mínimo de 3 iteraciones al team 1.

2. Eliminar en `omp_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: Por defecto se utilizan 48 equipos y se usan 1024 hilos por equipo

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

```
[MarioLindezMartinez ac419@atcgrid:~/Actividad_OpenMP] 2023-04-15 sábado
$ srun -pac4 -Aac omp_offload_GPU2 35
Target device: 1
Tiempo:0.126107931
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
```

(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: Sí, podemos relacionar el numero de equipos con el número de Streaming Multiprocessor (SM) que posee el coprocesador, siendo 1024 el número de threads por SM.

(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: Por defecto se asignarán en orden, es decir, al equipo 0 se le asignarán 1024 iteraciones, a una iteración por hilo. La iteración 2 es asignada al hilo 2 del equipo 0, mientras que la iteración 25 se le asignará al hilo 1 del equipo 1.

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: No, pues si ejecutamos el programa con un número bajo de hilos como mínimo se crearán 32 hilos. Por otra parte cuando ejecutamos el programa con un número alto de hilos se produce una violación de segmento, siendo 1055 hilos el número máximo que le podemos indicar. Sin embargo, observamos tras varias ejecuciones que siempre se crearán hilos en múltiplos de 32, siendo 32 hilos el mínimo que podremos crear y 1024 los hilos máximos.

CAPTURAS (que justifiquen la respuesta)

```
[MarioLindezMartinez ac419@atcgrid:~/Actividad_OpenMP] 2023-04-15 sábado
$ srun -pac4 -Aac omp_offload_GPU 20 4 10
Target device: 1
Tiempo:0.126178026
Iteracción 0, en thread 0/32 del team 0/4
Iteracción 1, en thread 1/32 del team 0/4
Iteracción 2, en thread 2/32 del team 0/4
Iteracción 3, en thread 3/32 del team 0/4
Iteracción 4, en thread 4/32 del team 0/4
Iteracción 5, en thread 5/32 del team 0/4
Iteracción 6, en thread 6/32 del team 0/4
Iteracción 7, en thread 7/32 del team 0/4
Iteracción 8, en thread 8/32 del team 0/4
Iteracción 9, en thread 9/32 del team 0/4
Iteracción 10, en thread 10/32 del team 0/4
Iteracción 11, en thread 11/32 del team 0/4
Iteracción 12, en thread 12/32 del team 0/4
Iteracción 13, en thread 13/32 del team 0/4
Iteracción 14, en thread 14/32 del team 0/4
Iteracción 15, en thread 15/32 del team 0/4
Iteracción 16, en thread 16/32 del team 0/4
Iteracción 17, en thread 17/32 del team 0/4
Iteracción 18, en thread 18/32 del team 0/4
Iteracción 19, en thread 19/32 del team 0/4
```

```
[MarioLindezMartinez ac419@atcgrid:~/Actividad_OpenMP] 2023-04-15 sábado
$ srun -pac4 -Aac omp_offload_GPU 20 48 1055 > output.txt
[MarioLindezMartinez ac419@atcgrid:~/Actividad_OpenMP] 2023-04-15 sábado
$ srun -pac4 -Aac omp_offload_GPU 20 48 1056 > output.txt
srun: error: atcgrid4: task 0: Aborted (core dumped)
```

```
[MarioLindezMartinez ac419@atcgrid:~/Actividad_OpenMP] 2023-04-15 sábado
$ srun -pac4 -Aac omp_offload_GPU 20 48 900
Target device: 1
Tiempo:0.141798019
Iteracción 0, en thread 0/896 del team 0/48
Iteracción 1, en thread 1/896 del team 0/48
Iteracción 2, en thread 2/896 del team 0/48
Iteracción 3, en thread 3/896 del team 0/48
Iteracción 4, en thread 4/896 del team 0/48
Iteracción 5, en thread 5/896 del team 0/48
Iteracción 6, en thread 6/896 del team 0/48
Iteracción 7, en thread 7/896 del team 0/48
Iteracción 8, en thread 8/896 del team 0/48
Iteracción 9, en thread 9/896 del team 0/48
Iteracción 10, en thread 10/896 del team 0/48
Iteracción 11, en thread 11/896 del team 0/48
Iteracción 12, en thread 12/896 del team 0/48
Iteracción 13, en thread 13/896 del team 0/48
Iteracción 14, en thread 14/896 del team 0/48
Iteracción 15, en thread 15/896 del team 0/48
Iteracción 16, en thread 16/896 del team 0/48
Iteracción 17, en thread 17/896 del team 0/48
Iteracción 18, en thread 18/896 del team 0/48
Iteracción 19, en thread 19/896 del team 0/48
[MarioLindezMartinez ac419@atcgrid:~/Actividad_OpenMP] 2023-04-15 sábado
$ srun -pac4 -Aac omp_offload_GPU 20 48 1055
Target device: 1
Tiempo:0.111403942
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
```

896/32 = 28

1024/32 = 32

(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, podemos relacionar el número mínimo de hilos que podemos crear con que número de threads en un warp, y el número máximo con el máximo número de threads por 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: Por defecto se usan 1 equipo y 1024 hilos.

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

RESPUESTA: Sabemos que la GPU posee 48 streaming multiprocessors (SM) donde cada uno tiene múltiples núcleos CUDA. Como en total hay 3072 núcleos de procesamiento, podremos calcular cuántos núcleos CUDA posee cada SM. $\frac{3072}{48}=64$ núcleos CUDA.

Por tanto, al estar usando un solo equipo (un único SM) y por haber más hilos que procesadores, estaremos usando todos ellos. Por tanto utilizamos 64 núcleos de 3072 totales por lo que $\frac{64}{3072}\cdot 100=2'083\,\%$ del total de núcleos.

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

`z = a · x + b · 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];
```

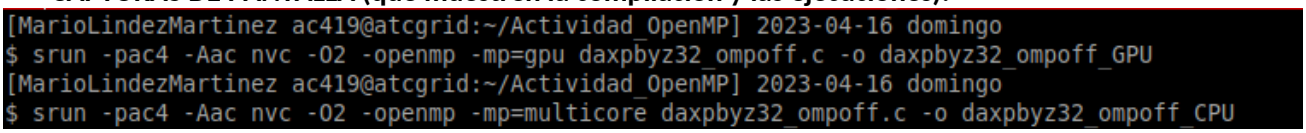
5) t6-t7 es el tiempo que supone `target exit data`, que transfiere los resultados de las variables con `from` y libera el espacio ocupado en la memoria del coprocesador.

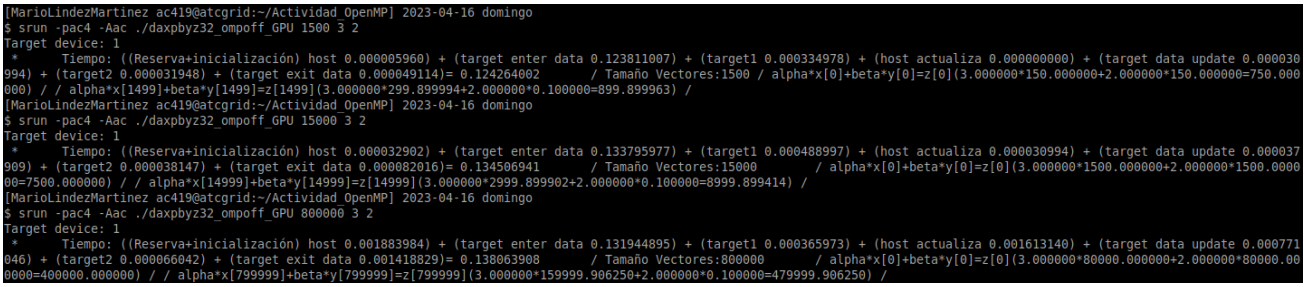
Compilar `daxpbyz32_off.c` para la GPU y para las CPUs de atctrid4 usando:

```
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 1.000 y 100.000 y contestar a las siguientes preguntas.

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





```
[MarioLindezMartinez ac419@atcgrid:~/Actividad_OpenMP] 2023-04-16 domingo
$ srun -pac4 -Aac ./daxpbyz32_ompoff_CPU 1500 3 2
Target device: 0
 *      Tiempo: ((Reserva+inicialización) host 0.000009060) + (target enter data 0.000000000) + (target1 0.002039909) + (host actualiza 0.000006199) + (target data update 0.000000000) + (target2 0.000005960) + (target exit data 0.000000000)= 0.002061129      / Tamaño Vectores:1500 / alpha*x[0]+beta*y[0]=z[0](3.000000*150.000000+2.000000*150.000000=750.000000) / / alpha*x[1499]+beta*y[1499]=z[1499](3.000000*299.899994+2.000000*0.100000=899.899963) /
[MarioLindezMartinez ac419@atcgrid:~/Actividad_OpenMP] 2023-04-16 domingo
$ srun -pac4 -Aac ./daxpbyz32_ompoff_CPU 15000 3 2
Target device: 0
 *      Tiempo: ((Reserva+inicialización) host 0.000061035) + (target enter data 0.000000000) + (target1 0.002173901) + (host actualiza 0.000135899) + (target data update 0.000000000) + (target2 0.000034094) + (target exit data 0.000000000)= 0.002404928        / Tamaño Vectores:15000       / alpha*x[0]+beta*y[0]=z[0](3.000000*1500.000000+2.000000*1500.000000=7500.000000) / / alpha*x[14999]+beta*y[14999]=z[14999](3.000000*2999.899902+2.000000*0.100000=8999.899414) /
[MarioLindezMartinez ac419@atcgrid:~/Actividad_OpenMP] 2023-04-16 domingo
$ srun -pac4 -Aac ./daxpbyz32_ompoff_CPU 800000 3 2
Target device: 0
 *      Tiempo: ((Reserva+inicialización) host 0.001661062) + (target enter data 0.000000000) + (target1 0.004249096) + (host actualiza 0.003136873) + (target data update 0.000000000) + (target2 0.001018047) + (target exit data 0.000000000)= 0.010065079        / Tamaño Vectores:800000      / alpha*x[0]+beta*y[0]=z[0](3.000000*80000.000000+2.000000*80000.000000=400000.000000) / / alpha*x[799999]+beta*y[799999]=z[799999](3.000000*159999.906250+2.000000*0.100000=479999.906250) /
```

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

RESPUESTA: En la GPU supone mucho más tiempo el primer target enter data (t_2 -t_1). Esto se debe a que se deben de mapear las variables a un ámbito de datos de coprocesador, es decir, reservar espacio de variables en el coprocesador.

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

RESPUESTA: Suponen más tiempo en la GPU que en la CPU las directivas “target enter data”, “target data update” y “target exit data”. Esto se debe a que estas directivas mapea, actualiza y desmapea variables en un ámbito de datos del coprocesador lo cual conlleva un tiempo determinado en comparación a cuando se usan en la CPU, la cual usa la memoria del host que es la propia CPU por lo que estas directivas son instantáneas.

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`

```c
int main(int argc, char **argv)
{
  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;

  double t1 = omp_get_wtime();

  #pragma omp target enter data map(to: width, intervals, sum)

  double 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);
          }

  double t3 = omp_get_wtime();
```

```
  #pragma omp target exit data map(delete: intervals, width) map (from: sum)
  sum *= width;

  double t4 = omp_get_wtime();

  printf ("Iteraciones:\t%d\t\\ PI:\t%26.24f\t\\ Tiempo:\t%8.6f\n", intervals, sum, t4-t2);
  printf ("Error:\t%8.6f\n", fabs(M_PI-sum));

  printf (" *\t(Target enter data): %11.9f\n", t2-t1);
  printf (" *\t(Target teams distribute parallel for): %11.9f\n", t3-t2);
  printf (" *\t(Target exit data): %11.9f\n", t4-t3);
  printf (" *\t(TOTAL): %11.9f\n", t4-t1);

  return(0);
}
```

CAPTURAS DE PANTALLA (mostrar la compilación y la ejecución para 10.000.000 intervalos de integración en atcgrid4 – envío(s) a la cola):

```
[MarioLindezMartinez ac419@atcgrid:~/Actividad_OpenMP] 2023-04-16 domingo
$ srun -pac4 -Aac nvc -O2 -openmp -mp=gpu pi-ompoff.c -o pi-ompoff_GPU
[MarioLindezMartinez ac419@atcgrid:~/Actividad_OpenMP] 2023-04-16 domingo
$ srun -pac4 -Aac ./pi-ompoff_GPU 10000000
Iteraciones:    10000000        \ PI:   3.14159265358794004176383      \ Tiempo:       0.001422
Error:  0.000000
 *      (Target enter data): 1.296418905
 *      (Target teams distribute parallel for): 0.001376152
 *      (Target exit data): 0.000046015
 *      (TOTAL): 1.297841072
```

```
[MarioLindezMartinez ac419@atcgrid:~/Actividad_OpenMP] 2023-04-16 domingo
$ srun -pac4 -Aac nvc -O2 -openmp -mp=multicore pi-ompoff.c -o pi-ompoff_CPU
[MarioLindezMartinez ac419@atcgrid:~/Actividad_OpenMP] 2023-04-16 domingo
$ srun -pac4 -Aac ./pi-ompoff_CPU 10000000
Iteraciones:    10000000        \ PI:   3.14159265358782901946137      \ Tiempo:       0.012825
Error:  0.000000
 *      (Target enter data): 0.000000000
 *      (Target teams distribute parallel for): 0.012825012
 *      (Target exit data): 0.000000000
 *      (TOTAL): 0.012825012
```

RESPUESTA:

Para el caso en el que usamos la CPU observamos, como ya hemos visto en anteriores ejercicios, que no se produce tiempo de comunicación, por lo que el tiempo total del programa será el de cálculo. Este hecho permite que sea más rápida que la ejecución de este programa usando la GPU, ya que el tiempo de comunicaciones (enter y exit data) hacen el programa más lento.

Sin embargo, viendo ambas ejecuciones podemos ver claramente como al usar la GPU se mejora significativamente el tiempo de cálculo, por lo que podemos suponer que para programas con cálculos de mucho mayor tamaño podrá merecer la pena ese tiempo extra en comunicaciones.