OpenMP coprocesadores

1. Consideraciones previas

- Se usará el compilador nvc de Nvidia, que se puede descargar de su página web. En atcgrid está instalado en el nodo atcgrid4.
- 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

Ejercicios basados en los ejemplos del seminario

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

```
sbatch –pac4 –Aac --wrap “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 > salida.txt
```

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

```
Target device: 1
Tiempo:0.142221928
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:

(b) ¿Cuántos equipos (*teams*) se han creado y cuántos se han usado realmente en la ejecución?
RESPUESTA: Se han creado 3 equipos, pero realmente se han usado 2.

(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, de los cuales se han usado todos en el equipo 0 y 3 en el equipo 1.

(d) ¿Qué número máximo de iteraciones se ha asignado a un hilo?
RESPUESTA: Una iteración.

(e) ¿Qué número mínimo de iteraciones se ha asignado a un equipo y cuál es ese equipo?
RESPUESTA: 3 iteraciones, en este caso 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 `omp_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 usan 48 equipos con 1024 hilos por cada equipo.

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

```
[ac412@atcgrid ~]$ sbatch -pac4 -Aac --wrap "nvc -O2 -openmp -mp=gpu omp_offload2.c -o omp_offload2_GPU"
Submitted batch job 153412
[ac412@atcgrid ~]$ srun -pac4 -Aac omp_offload2_GPU 35 > salida.txt
[ac412@atcgrid ~]$ cat salida.txt
Target device: 1
Tiempo:0.129277945
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 este número con alguno de los parámetros, comentados en el seminario, que caracterizan al coprocesador que estamos usando? ¿Con cuáles?

RESPUESTA: En la diapositiva 9 (Nvidia Quadro RTX 5000) aparecen los valores con los que se relacionan, concretamente con las filas “Streaming Multiprocessor (SM) SIMT 48” y “Máximo de threads por SM (o CUDA Block) 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: Por defecto, se le asigna una iteración a cada hebra dentro del equipo. La iteración 2 se le asigna al primer equipo (el equipo 0), y concretamente se le asigna a la hebra 1 del equipo 0. En cuanto a la iteración 1025, se le asignaría al equipo 1, concretamente a su hebra 0 (esto en caso de numerar las iteraciones empezando por el 1, no por el 0, en caso de empezar a numerar por iteración 0, entonces la itración 1025 le tocaría a la hebra 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, en caso de indicar un número de hilos inferior a 32, se crean 32 por cada equipo. Para 3000, obtenemos una violación de segmento. Concretamente, para cualquier valor mayor o igual a 1056. Tras varios intentos, he visto que el máximo de hilos que se crean para un equipo es 1024. Lo de los 32 hilos se debe a que los threads se envían a ejecución agrupados en unidades llamadas *warp*, que se componen de 32 threads. Siempre que se da un valor, los hilos que se crean son el múltiplo de 32 inmediatamente inferior al número introducido, esto se debe a que los últimos hilos no se pueden agrupar en un *warp* (unidad de 32 threads).

CAPTURAS (que justifiquen la respuesta)

```
[ac412@atcgrid ~]$ srun -pac4 -Aac omp_offload_GPU 35 4 14 > salida.txt
[ac412@atcgrid ~]$ cat salida.txt
Target device: 1
Tiempo:0.113854885
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
Iteracción 20, en thread 20/32 del team 0/4
Iteracción 21, en thread 21/32 del team 0/4
Iteracción 22, en thread 22/32 del team 0/4
Iteracción 23, en thread 23/32 del team 0/4
Iteracción 24, en thread 24/32 del team 0/4
Iteracción 25, en thread 25/32 del team 0/4
Iteracción 26, en thread 26/32 del team 0/4
Iteracción 27, en thread 27/32 del team 0/4
Iteracción 28, en thread 28/32 del team 0/4
Iteracción 29, en thread 29/32 del team 0/4
Iteracción 30, en thread 30/32 del team 0/4
Iteracción 31, en thread 31/32 del team 0/4
Iteracción 32, en thread 0/32 del team 1/4
Iteracción 33, en thread 1/32 del team 1/4
Iteracción 34, en thread 2/32 del team 1/4
```

```
[ac412@atcgrid ~]$ srun -pac4 -Aac omp_offload_GPU 35 4 1000 > salida.txt
[ac412@atcgrid ~]$ cat salida.txt
Target device: 1
Tiempo:0.131039143
Iteracción 0, en thread 0/992 del team 0/4
Iteracción 1, en thread 1/992 del team 0/4
Iteracción 2, en thread 2/992 del team 0/4
Iteracción 3, en thread 3/992 del team 0/4
Iteracción 4, en thread 4/992 del team 0/4
Iteracción 5, en thread 5/992 del team 0/4
Iteracción 6, en thread 6/992 del team 0/4
Iteracción 7, en thread 7/992 del team 0/4
Iteracción 8, en thread 8/992 del team 0/4
Iteracción 9, en thread 9/992 del team 0/4
Iteracción 10, en thread 10/992 del team 0/4
Iteracción 11, en thread 11/992 del team 0/4
Iteracción 12, en thread 12/992 del team 0/4
Iteracción 13, en thread 13/992 del team 0/4
Iteracción 14, en thread 14/992 del team 0/4
Iteracción 15, en thread 15/992 del team 0/4
Iteracción 16, en thread 16/992 del team 0/4
Iteracción 17, en thread 17/992 del team 0/4
Iteracción 18, en thread 18/992 del team 0/4
Iteracción 19, en thread 19/992 del team 0/4
Iteracción 20, en thread 20/992 del team 0/4
Iteracción 21, en thread 21/992 del team 0/4
Iteracción 22, en thread 22/992 del team 0/4
Iteracción 23, en thread 23/992 del team 0/4
Iteracción 24, en thread 24/992 del team 0/4
Iteracción 25, en thread 25/992 del team 0/4
Iteracción 26, en thread 26/992 del team 0/4
Iteracción 27, en thread 27/992 del team 0/4
Iteracción 28, en thread 28/992 del team 0/4
Iteracción 29, en thread 29/992 del team 0/4
Iteracción 30, en thread 30/992 del team 0/4
Iteracción 31, en thread 31/992 del team 0/4
Iteracción 32, en thread 32/992 del team 0/4
Iteracción 33, en thread 33/992 del team 0/4
Iteracción 34, en thread 34/992 del team 0/4
```

```
[ac412@atcgrid ~]$ srun -pac4 -Aac omp_offload_GPU 35 4 1055 > salida.txt
[ac412@atcgrid ~]$ cat salida.txt
Target device: 1
Tiempo:0.134536982
Iteracción 0, en thread 0/1024 del team 0/4
Iteracción 1, en thread 1/1024 del team 0/4
Iteracción 2, en thread 2/1024 del team 0/4
Iteracción 3, en thread 3/1024 del team 0/4
Iteracción 4, en thread 4/1024 del team 0/4
Iteracción 5, en thread 5/1024 del team 0/4
Iteracción 6, en thread 6/1024 del team 0/4
Iteracción 7, en thread 7/1024 del team 0/4
Iteracción 8, en thread 8/1024 del team 0/4
Iteracción 9, en thread 9/1024 del team 0/4
Iteracción 10, en thread 10/1024 del team 0/4
Iteracción 11, en thread 11/1024 del team 0/4
Iteracción 12, en thread 12/1024 del team 0/4
Iteracción 13, en thread 13/1024 del team 0/4
Iteracción 14, en thread 14/1024 del team 0/4
Iteracción 15, en thread 15/1024 del team 0/4
Iteracción 16, en thread 16/1024 del team 0/4
Iteracción 17, en thread 17/1024 del team 0/4
Iteracción 18, en thread 18/1024 del team 0/4
Iteracción 19, en thread 19/1024 del team 0/4
Iteracción 20, en thread 20/1024 del team 0/4
Iteracción 21, en thread 21/1024 del team 0/4
Iteracción 22, en thread 22/1024 del team 0/4
Iteracción 23, en thread 23/1024 del team 0/4
Iteracción 24, en thread 24/1024 del team 0/4
Iteracción 25, en thread 25/1024 del team 0/4
Iteracción 26, en thread 26/1024 del team 0/4
Iteracción 27, en thread 27/1024 del team 0/4
Iteracción 28, en thread 28/1024 del team 0/4
Iteracción 29, en thread 29/1024 del team 0/4
Iteracción 30, en thread 30/1024 del team 0/4
Iteracción 31, en thread 31/1024 del team 0/4
Iteracción 32, en thread 32/1024 del team 0/4
Iteracción 33, en thread 33/1024 del team 0/4
Iteracción 34, en thread 34/1024 del team 0/4
```

```
[ac412@atcgrid ~]$ srun -pac4 -Aac omp_offload_GPU 35 4 1056 > salida.txt
srun: error: atcgrid4: task 0: Aborted (core dumped)
```

```
[ac412@atcgrid ~]$ srun -pac4 -Aac omp_offload_GPU 35 4 3000 > salida.txt
srun: error: atcgrid4: task 0: Aborted (core dumped)
```

(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 un dato de la diapositiva 9. Concretamente, con el número máximo de

threads por SM (o CUDA Block), lo cual tiene sentido, porque es el máximo número de hilos que se han podido llegar a crear por equipo en el anterior apartado.

4. Eliminar las directivas `teams y distribute` en `omp_offload2.c`, llamar al código resultante `omp_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 usan por defecto un 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: En las transparencias hemos visto que el coprocesador GPU se compone de 48 *streaming multiprocessors*, cada uno con múltiples núcleos *CUDA*. Como en total hay 3072 núcleos de procesamiento, cada SM tiene 3072/48 = 64 núcleos CUDA (SP).

Al estar usando un solo equipo (SM), y por haber más hilos (1024) que procesadores (SP), estaremos utilizando todos los procesadores. En resumen, estamos utilizando 64 núcleos de 3072, luego 64/3072 x 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):

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

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:

```
sbatch –pac4 –Aac --wrap “nvc -O2 -openmp -mp=gpu daxpbyz32_ompoff.c -o daxpbyz32_ompoff_GPU”
sbatch –pac4 –Aac --wrap “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):

```
[ac412@atcgrid ~]$ sbatch -pac4 -Aac --wrap "nvc -O2 -openmp -mp=gpu daxpbyz32_ompoff.c -o daxpbyz32_ompoff_GPU"
Submitted batch job 153712
[ac412@atcgrid ~]$ sbatch -pac4 -Aac --wrap "nvc -O2 -openmp -mp=multicore daxpbyz32_ompoff.c -o daxpbyz32_ompoff_CPU"
Submitted batch job 153715
```

```
[ac412@atcgrid ~]$ srun -pac4 -Aac daxpbyz32_ompoff_GPU 1000 2 3
Target device: 1
 *      Tiempo: ((Reserva+inicialización) host 0.000005007) + (target enter data 0.131062984) + (target1 0.000368834) + (host actualiza 0.000000954) + (target data update 0.000030041) + (target2 0.000035
048) + (target exit data 0.000054121)= 0.131556988      / 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) /
[ac412@atcgrid ~]$ srun -pac4 -Aac daxpbyz32_ompoff_GPU 5000 2 3
Target device: 1
 *      Tiempo: ((Reserva+inicialización) host 0.000012159) + (target enter data 0.119108915) + (target1 0.000457048) + (host actualiza 0.000001907) + (target data update 0.000039101) + (target2 0.000034
809) + (target exit data 0.000050068)= 0.119704008       / 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](2.0
00000*999.900024+3.000000*0.100000=2000.100098) /
[ac412@atcgrid ~]$ srun -pac4 -Aac daxpbyz32_ompoff_GPU 10000 2 3
Target device: 1
 *      Tiempo: ((Reserva+inicialización) host 0.000029087) + (target enter data 0.132200003) + (target1 0.000463963) + (host actualiza 0.000030994) + (target data update 0.000030994) + (target2 0.000036
955) + (target exit data 0.000065088)= 0.132857084       / 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) /
```

```
[ac412@atcgrid ~]$ srun -pac4 -Aac daxpbyz32_ompoff_CPU 1000 2 3
Target device: 0
 *      Tiempo: ((Reserva+inicialización) host 0.000008106) + (target enter data 0.000000000) + (target1 0.002006054) + (host actualiza 0.000001907) + (target data update 0.000000000) + (target2 0.000005
007) + (target exit data 0.000000000)= 0.002021074       / 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](2.00000
0*199.899994+3.000000*0.100000=400.099976) /
[ac412@atcgrid ~]$ srun -pac4 -Aac daxpbyz32_ompoff_CPU 5000 2 3
Target device: 0
 *      Tiempo: ((Reserva+inicialización) host 0.000019789) + (target enter data 0.000000000) + (target1 0.001799107) + (host actualiza 0.000019073) + (target data update 0.000000000) + (target2 0.000010
967) + (target exit data 0.000000000)= 0.001848936       / 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](2.0
00000*999.900024+3.000000*0.100000=2000.100098) /
[ac412@atcgrid ~]$ srun -pac4 -Aac daxpbyz32_ompoff_CPU 10000 2 3
Target device: 0
 *      Tiempo: ((Reserva+inicialización) host 0.000039101) + (target enter data 0.000000954) + (target1 0.001709938) + (host actualiza 0.000030041) + (target data update 0.000000000) + (target2 0.000019
073) + (target exit data 0.000000000)= 0.001799107       / 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) /
```

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

RESPUESTA: La directiva target enter data supone más tiempo en la GPU. Esto se debe a que se tienen que copiar en la memoria de los coprocesadores las variables indicadas, por lo que debe mapearlas, crear las tablas de paginación, etc.

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

RESPUESTA: Las directivas target que suponen más tiempo en la GPU que en la CPU son *target enter data*, *target data update* y *target exit data*. Esto se debe a que en la CPU no se usa la memoria de la GPU, sino la del host, que es la de la CPU, por lo que no tarda nada en crear un nuevo ámbito de variables en la CPU, al igual que al actualizarlas y borrarlas (ver diapositiva 11).

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 la GPU. Generar dos ejecutables, uno que use como coprocesador la CPU y otro que use la GPU. Comparar los tiempos de ejecución 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 <omp.h>
```

```
#include <time.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):

```
[ac412@atcgrid ~]$  sbatch -pac4 -Aac --wrap "nvc -O2 -openmp -mp=gpu pi.c -o pi_GPU"
Submitted batch job 162828
[ac412@atcgrid ~]$  sbatch -pac4 -Aac --wrap "nvc -O2 -openmp -mp=multicore pi.c -o pi_CPU"
Submitted batch job 162831
[ac412@atcgrid ~]$ srun -pac4 -Aac pi_GPU 10000000
Iteraciones:    10000000        . PI:   3.14159265358979448265593       . Tiempo:       0.242226
Error:  0.000000000000001332267630
(Target enter data):    0.240773
 (Target teams distribute parallel for):        0.001403
 (Target exit data):    0.000050
[ac412@atcgrid ~]$ srun -pac4 -Aac pi_CPU 10000000
Iteraciones:    10000000        . PI:   3.14159265358978290194613       . Tiempo:       0.008539
Error:  0.000000000000010214051827
(Target enter data):    0.000000
 (Target teams distribute parallel for):        0.008539
 (Target exit data):    0.000000
```

RESPUESTA: He de decir que hemos debatido entre varios compañeros sobre cómo delimitar la zona de los tiempos para el cálculo de pi, al final he decidido que sea desde que se hace el target enter data hasta después de sum *= width. Vemos que el tiempo de la GPU es mayor, esto se debe a por lo que habíamos visto en el ejercicio anterior, pues en la CPU (host) las variables ya están en memoria pero cuando se usa la GPU no, por lo que se consume tiempo en la creación de un nuevo ámbito de variables. Además, en la GPU se hace uso de un mayor número de núcleos que la CPU, y la cláusula reduction

implica comunicación, que también añade retardo (se vio en clases de teoria), y cuesta más comunicar un gran número de núcleos que una cantidad menor.