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:0.129539967
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 solo se usan 2, el 0 y el 1

(c) ¿Cuántos hilos (*threads*) se han creado en cada equipo y cuántos de esos hilos se han usado en la ejecución?
RESPUESTA: se han creado 32 hilos que se han usado al completo.

(d) ¿Qué número máximo de iteraciones se ha asignado a un hilo?
RESPUESTA: 2. Los threads 0,1,2 se ejecutan 2 veces.

(e) ¿Qué número mínimo de iteraciones se ha asignado a un equipo y cuál es ese equipo?
RESPUESTA: 32 iteraciones al "equipo 0" y 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 48 equipos y 1024 threads por defecto

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

```
[ac243@atcgrid coprocesadores]$ cat script.sh
#!/bin/bash
if [ $# != 0 ]
  then
    sbatch -pac4 -Aac --wrap "nvc -O2 -openmp -mp=gpu $1 -o $2"
else
    echo "Uso: ./scrip archivo.c archivo"
fi
[ac243@atcgrid coprocesadores]$ ./script.sh omp_offload2.c omp_offload2
Submitted batch job 147650
[ac243@atcgrid coprocesadores]$ srun -pac4 -Aac omp_offload2 35 3 32 > salida2.txt
[ac243@atcgrid coprocesadores]$ cat salida
cat: salida: No existe el fichero o el directorio
[ac243@atcgrid coprocesadores]$ cat salida2
cat: salida2: No existe el fichero o el directorio
[ac243@atcgrid coprocesadores]$ cat salida2.txt
Target device: 1
Target number: 0
Tiempo:0.126317024
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
```

(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: 48 equipos que representan los 48 streaming Multiprocessor(SM) de la Quadro RTX 5000 y 1024 hilos que es el máximo 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: empieza por el equipo 0 y le va asignando iteraciones a cada hilo de forma incremental (del 0 al max), cuando llega al max de hilos cambia de equipo y vuelve a

asignar hilos a iteraciones.
Se asigna el equipo 0 e hilo 2 para la iteración 2 y se asigna el equipo 1 e hilo 1 para la iteración 1025.

```
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 1023, en thread 1023/1024 del team 0/48
Iteracción 1024, en thread 0/1024 del team 1/48
Iteracción 1025, en thread 1/1024 del team 1/48
```

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, solo se crean un máximo de 1024 threads por equipo. Al pasarse de 1024 threads, vuelve a empezar por el 0. Sin embargo, al asignar más de 1055 threads, el programa da "core"

CAPTURAS (que justifiquen la respuesta)

```
[ac243@atcgrid coprocesadores]$ srun -pac4 -Aac omp_offload 1056 1 1056 > salida.txt
catsrun: error: atcgrid4: task 0: Aborted (core dumped)
[ac243@atcgrid coprocesadores]$ cat salida.txt
Target device: 1
Target number: 0
Fatal error: Could not launch CUDA kernel on device 0, error 1
[ac243@atcgrid coprocesadores]$
```

```
Iteracción 1017, en thread 1017/1024 del team 0/1
Iteracción 1018, en thread 1018/1024 del team 0/1
Iteracción 1019, en thread 1019/1024 del team 0/1
Iteracción 1020, en thread 1020/1024 del team 0/1
Iteracción 1021, en thread 1021/1024 del team 0/1
Iteracción 1022, en thread 1022/1024 del team 0/1
Iteracción 1023, en thread 1023/1024 del team 0/1
Iteracción 1024, en thread 0/1024 del team 0/1
Iteracción 1025, en thread 1/1024 del team 0/1
Iteracción 1026, en thread 2/1024 del team 0/1
Iteracción 1027, en thread 3/1024 del team 0/1
Iteracción 1028, en thread 4/1024 del team 0/1
```

(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: la variable de control teams-thread-limit-var controla el número máximo de threads por team que es 1024 en nuestro coprocesador.

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 un equipo con 1024 hilos (el equipo completo).

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

RESPUESTA: la Quadro RTX 5000 tiene 3072 núcleos de procesamiento paralelo y estamos usando 1024 thread. Asumiendo que los núcleos no usan multithreading, se usa 1024/3072 = ⅓ del procesamiento.

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):

```
[ac243@atcgrid coprocesadores]$ sbatch -pac4 -Aac --wrap "nvc -O2 -openmp -mp=gpu daxpbyz32_ompoff.c -o daxpbyz32_o
mpoff_GPU"
Submitted batch job 147718
[ac243@atcgrid coprocesadores]$ sbatch -pac4 -Aac --wrap "nvc -O2 -openmp -mp=multicore daxpbyz32_ompoff.c -o daxpb
yz32_ompoff_CPU"
Submitted batch job 147719
[ac243@atcgrid coprocesadores]$ srun -pac4 -Aac daxpbyz32_ompoff_
daxpbyz32_ompoff_CPU  daxpbyz32_ompoff_GPU
[ac243@atcgrid coprocesadores]$ srun -pac4 -Aac daxpbyz32_ompoff_
daxpbyz32_ompoff_CPU  daxpbyz32_ompoff_GPU
[ac243@atcgrid coprocesadores]$ srun -pac4 -Aac daxpbyz32_ompoff_CPU 49152 2 3
Target device: 0
 *      Tiempo: ((Reserva+inicialización) host 0.000152111) + (target enter data 0.000000000) + (target1 0.00213098
5) + (host actualiza 0.000205994) + (target data update 0.000000000) + (target2 0.000072956) + (target exit data 0.
000000000)= 0.002562046  / Tamaño Vectores:49152        / alpha*x[0]+beta*y[0]=z[0](2.000000*4915.200195+3.000000*4
915.200195=24576.000000) / / alpha*x[49151]+beta*y[49151]=z[49151](2.000000*9830.299805+3.000000*0.100000=19660.900
391) /
[ac243@atcgrid coprocesadores]$ srun -pac4 -Aac daxpbyz32_ompoff_GPU 49152 2 3
Target device: 1
 *      Tiempo: ((Reserva+inicialización) host 0.000221968) + (target enter data 0.141943932) + (target1 0.00050497
1) + (host actualiza 0.000122070) + (target data update 0.000070095) + (target2 0.000038862) + (target exit data 0.
000157118)= 0.143059015  / Tamaño Vectores:49152        / alpha*x[0]+beta*y[0]=z[0](2.000000*4915.200195+3.000000*4
915.200195=24576.000000) / / alpha*x[49151]+beta*y[49151]=z[49151](2.000000*9830.299805+3.000000*0.100000=19660.900
391) /
[ac243@atcgrid coprocesadores]$

mpoff_GPU"
```

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

RESPUESTA: la directiva target enter data ya que tiene que reservar memoria dinámicamente dando una sobrecarga que se ve reflejada en el tiempo 0.1419 secs.

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

RESPUESTA: la directiva mencionada antes, la directiva del segundo target team distribute parallel for y la exit data.

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`

```c
#include <omp.h>
/**
 * @file  pi.c
 * @brief PI paralelo con integración numérica, área de rectángulos
 * @author Yeray Lopez Ramirez
 *
 * **Compilación**
 * @code
 *  sbatch -pac4 -Aac --wrap "nvc -O2 -openmp -mp=gpu pi_ompoff.c -o pi_ompoff_GPU"
 *  sbatch -pac4 -Aac --wrap "nvc -O2 -openmp -mp=multicore pi_ompoff.c -o pi_ompoff_CPU"
 * @endcode
 *
 *  **Ejecución**
 * ~~~~~~~~~~~~~~~~~~~~~~~~
 * srun -pac4 -Aac pi_ompoff 0
 * ~~~~~~~~~~~~~~~~~~~~~~~~
 *
*/
int main(int argc, char **argv)
{
  register double width;
  double sum;
  double PI25DT = 3.14159265358979323846264338327950288419716939937510;
  double t0,t1,t2;
  register long intervals,i;

  //Los procesos calculan PI en paralelo
  if (argc<2) {printf("Falta número de intevalos");exit(-1);}
  intervals=atol(argv[1]);
  if (intervals<1) {intervals=10000000; printf("Intervalos=%d.\n",intervals);}
  width = 1.0 / intervals;
  sum = 0;

  printf("Hay %i dispositivos de GPU.\n", omp_get_num_devices());
  t0 = omp_get_wtime();
  #pragma omp target enter data map(to: intervals) //transferencia de datos a la gpu
  t1 = 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);
  }
  sum *= width;

  t2 = omp_get_wtime();

  printf("Tiempo de transferencia de datos a la GPU: %8.6fs.\n", t1-t0);
  printf("Intervalos:%d \tPI:%26.24f. \tError:%26.24f.\n", intervals, sum, fabs(sum - PI25DT));
  printf("Tiempo de cálculo: %8.6fs.\n", t2-t1);
  printf("Tiempo total: %8.6fs\n", t2-t0);
  return(0);
}
```

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

```
[ac243@atcgrid coprocesadores]$ srun -pac4 -Aac pi_ompoff_CPU 10000000
Hay 0 dispositivos de GPU.
Tiempo de transferencia de datos hacia la GPU: 0.000000s.
Intervalos:10000000     PI:3.14159265358992279004724 0.  Error:0.000000000000129674049276.
Tiempo de cálculo: 0.025884s.
Tiempo de transferencia de datos desde la GPU: 0.025884s.
Tiempo total: 0.025884s
[ac243@atcgrid coprocesadores]$ srun -pac4 -Aac pi_ompoff_GPU 10000000
Hay 1 dispositivos de GPU.
Tiempo de transferencia de datos hacia la GPU: 0.119475s.
Intervalos:10000000     PI:3.14159265358979400417638 3.  Error:0.000000000000000888178420.
Tiempo de cálculo: 0.001236s.
Tiempo de transferencia de datos desde la GPU: 0.001236s.
Tiempo total: 0.120715s
[ac243@atcgrid coprocesadores]$
```

La CPU gana a la GPU en tiempo total debido a la sobrecarga de cargar y descargar los datos sin embargo la GPU le gana en tiempo de cómputo. Pero al aumentar las iteraciones pasa lo siguiente

```
[ac243@atcgrid coprocesadores]$ srun -pac4 -Aac pi_ompoff_CPU 100000000
Hay 0 dispositivos de GPU.
Tiempo de transferencia de datos hacia la GPU: 0.000000s.
Intervalos:100000000    PI:3.14159265358990991146015 4.  Error:0.000000000000116795462191.
Tiempo de cálculo: 0.193537s.
Tiempo de transferencia de datos desde la GPU: 0.193537s.
Tiempo total: 0.193537s
[ac243@atcgrid coprocesadores]$ srun -pac4 -Aac pi_ompoff_GPU 100000000
Hay 1 dispositivos de GPU.
Tiempo de transferencia de datos hacia la GPU: 0.136821s.
Intervalos:100000000    PI:3.14159265358979400417 6383.  Error:0.000000000000000888178420.
Tiempo de cálculo: 0.008672s.
Tiempo de transferencia de datos desde la GPU: 0.008672s.
Tiempo total: 0.145497s
[ac243@atcgrid coprocesadores]$
```

RESPUESTA: La GPU destroza a la CPU en cálculo pero sin embargo los tiempos son similares debido a la sobrecarga por la transferencia de datos hacia y desde la GPU. Eso se debe a que la GPU debe reservar memoria y cargar los datos en sus núcleos y luego devolverlos a la CPU, tardando bastante en el proceso.