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))**

```
[ac410@atcgrid Extra]$ cat salida.txt
Target device: 1
Tiempo:0.187662125
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 se han usado solo 2 (Team 0 y Team 1)

(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 quedado 32 hebras. En el primero se han usado las 32 y en el segundo equipo se han usado 3.

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

RESPUESTA: 1 iteración

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

RESPUESTA: El número mínimo es 0 y se ha asignado al equipo 2.

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: Número de hilos 1024 (Máximo threads por SM (o CUDA Block)) y números de equipos es 48 (SIMT)

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

```
[ac410@atcgrid Extra]$ srun -pac4 -Aac omp_offload_2 35 3 > salida2.txt
[ac410@atcgrid Extra]$ cat salida2.txt
Target device: 1
Tiempo:0.138327122
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
[ac410@atcgrid Extra]$
```

(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: Respondido en la pregunta anterior

(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 una iteración a cada hebra de cada equipo por round-robin. La iteración 2 se asigna la thread 2/1024 del team 0/48. La iteración número 1025 se ejecutaría en el thread 0/1024 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: El máximo valor posible por equipo es 1024 aunque se especifique otro en la entrada del programa. Esto se debe a que es el máximo número de threads por SM que tiene la GPU. De tomar valores muy grandes, provocamos un core dumped en el grid.

CAPTURAS (que justifiquen la respuesta)

```
[ac410@atcgrid Extra]$ srun -pac4 -Aac omp_offload_GPU 35 3 1050 > salida.txt
[ac410@atcgrid Extra]$ cat salida
cat: salida: No such file or directory
[ac410@atcgrid Extra]$ cat salida.txt
Target device: 1
Tiempo:0.122583866
Iteracción 0, en thread 0/1024 del team 0/3
Iteracción 1, en thread 1/1024 del team 0/3
Iteracción 2, en thread 2/1024 del team 0/3
Iteracción 3, en thread 3/1024 del team 0/3
Iteracción 4, en thread 4/1024 del team 0/3
Iteracción 5, en thread 5/1024 del team 0/3
Iteracción 6, en thread 6/1024 del team 0/3
Iteracción 7, en thread 7/1024 del team 0/3
Iteracción 8, en thread 8/1024 del team 0/3
Iteracción 9, en thread 9/1024 del team 0/3
Iteracción 10, en thread 10/1024 del team 0/3
Iteracción 11, en thread 11/1024 del team 0/3
Iteracción 12, en thread 12/1024 del team 0/3
Iteracción 13, en thread 13/1024 del team 0/3
Iteracción 14, en thread 14/1024 del team 0/3
Iteracción 15, en thread 15/1024 del team 0/3
Iteracción 16, en thread 16/1024 del team 0/3
Iteracción 17, en thread 17/1024 del team 0/3
Iteracción 18, en thread 18/1024 del team 0/3
Iteracción 19, en thread 19/1024 del team 0/3
Iteracción 20, en thread 20/1024 del team 0/3
Iteracción 21, en thread 21/1024 del team 0/3
Iteracción 22, en thread 22/1024 del team 0/3
Iteracción 23, en thread 23/1024 del team 0/3
Iteracción 24, en thread 24/1024 del team 0/3
Iteracción 25, en thread 25/1024 del team 0/3
Iteracción 26, en thread 26/1024 del team 0/3
Iteracción 27, en thread 27/1024 del team 0/3
Iteracción 28, en thread 28/1024 del team 0/3
Iteracción 29, en thread 29/1024 del team 0/3
Iteracción 30, en thread 30/1024 del team 0/3
Iteracción 31, en thread 31/1024 del team 0/3
Iteracción 32, en thread 32/1024 del team 0/3
Iteracción 33, en thread 33/1024 del team 0/3
Iteracción 34, en thread 34/1024 del team 0/3
```

```
[ac410@atcgrid Extra]$ srun -pac4 -Aac omp_offload_GPU 35 3 3000 > salida.txt
srun: error: atcgrid4: task 0: Aborted (core dumped)
[ac410@atcgrid Extra]$
```

(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: Se relaciona con el número máximo de threads por SM para el número máximo de hebras por equipo.

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 solo se crea un equipo, aunque el número de hebras se mantiene con 1024. En caso de ejecutar más iteraciones de 1024, se asigna el mayor número posible a cada hebra por orden creciente. Por ejemplo con 1025 iteraciones, la iteración 0 y la 1 la hacen la hebra 0, y el resto de hebras una iteración. Con 1026 la 0 y la 1 la hebra 0, la 2 y la 3 la hebra 1 y el resto una iteración. Así sucesivamente.

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

RESPUESTA: Un 33%. 1024 núcleos se usan y se tienen 3072 núcleos de procesamiento paralelo.

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];
```

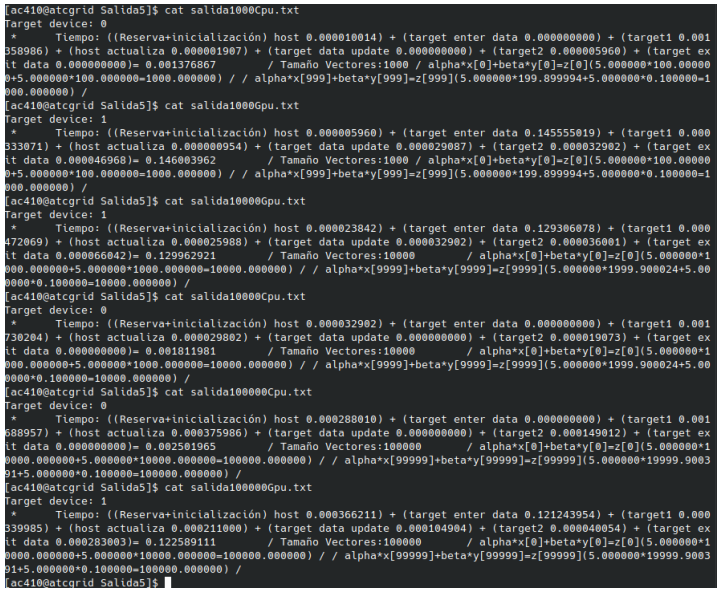
5) t6-t7 es el tiempo que supone `target exit data`, que transfiere los resultados de las variables con `from` y libera el espacio ocupado en la memoria del coprocesador.

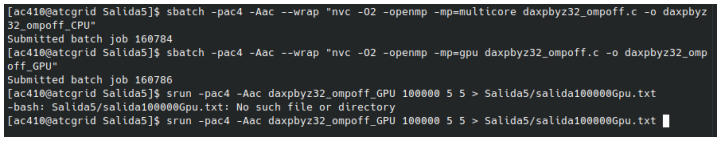
Compilar `daxpbyz32_off.c` para la GPU y para las CPUs de atctrid4 usando:

```
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):





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

RESPUESTA: La directiva target enter data porque tiene que mapear todo el vector x del host al target. Al ser un movimiento de datos de memoria, consume una gran cantidad de tiempo.

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

RESPUESTA: El target enter data por lo explicado anteriormente (de hecho en el de CPU no consume tiempo al no tener que mapear nada). El target data update tenemos el mismo caso en el que hay que mover cosas en memoria y se ralentiza el tiempo. Por último ocurre lo mismo en el target exit data, pues hay que pasar los datos del coprocesador a la memoria usual.

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
int main(int argc, char **argv)
{
  double width;
  double sum;
  register int intervals, i;
  double t1 = 0, t2 = 0, t3 = 0, t4 = 0; //para tiempo

  //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: 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(from: sum)

t4 = omp_get_wtime();

sum *= width;

  printf("Iteraciones:\t%d\t. PI:\t%26.24f\t. Error de cálculo:\t%8.6f\n",
  printf("Tiempo de transferencia hacia GPU; \t%8.6f\n", t2-t1);
  printf("Tiempo de cálculo de PI:\t%8.6f\n", t3-t2);
  printf("Tiempo de transferencia desde GPU:\t%8.6f\n", 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):

```
[ac410@atcgrid Extra]$ sbatch -pac4 -Aac --wrap "nvc -O2 -openmp -mp=gpu pi.c -o pi_GPU"
Submitted batch job 161362
[ac410@atcgrid Extra]$ srun -pac4 -Aac pi_GPU 1000000
Iteraciones:    1000000 . PI:   3.14159265358987704885862 5      . Error de cálculo:     0.000000
Tiempo de transferencia hacia GPU;      0.126463
Tiempo de cálculo de PI:        0.034645
Tiempo de transferencia desde GPU:      0.000062
[ac410@atcgrid Extra]$ sbatch -pac4 -Aac --wrap "nvc -O2 -openmp -mp=multicore pi.c -o pi_CPU"
Submitted batch job 161382
[ac410@atcgrid Extra]$ srun -pac4 -Aac pi_CPU 1000000
Iteraciones:    1000000 . PI:   3.14159265358986239391 4700      . Error de cálculo:     0.000000
Tiempo de transferencia hacia GPU;      0.000001
Tiempo de cálculo de PI:        0.002913
Tiempo de transferencia desde GPU:      0.000000
[ac410@atcgrid Extra]$
```

RESPUESTA: Los menores tiempos en la CPU se deben a la no necesidad de estar transfiriendo datos entre memorias.