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 36 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:0.113121986
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
Iteracción 35, en thread 3/32 del team 1/3
```

Contestar las siguientes preguntas **justificando la respuesta usando el contenido del fichero salida.txt**:

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

RESPUESTA:

Claramente, al final de cada linea hay una fracción que nos indica el equipo que ha ejecutado la iteración. Aparece "*/3", es decir, se han creado tres equipos. Realmente, en la ejecución solo se han usado dos de ellos; esto es así pues el número de iteraciones a asignar a cada equipo es 32 pues el número de participantes de cada equipo es dicho número.

(c) ¿Cuántos hilos (*threads*) 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 threads, de los cuales se han usado todos los del primer equipo, solo las cuatro primeras del segundo y ninguna de tercero. Esto es así pues no se han realizado tantas iteraciones como hebras había en total.

(d) ¿Qué número de iteraciones se ha asignado a `cada` hilo?

RESPUESTA:

A cada hilo se le ha asignado la hebra correspondiente con su identificador; es decir, a la hebra n-ésima se le ha asignado la iteración n-ésima. No obstante, al haber mas de 32 iteraciones, esta asignación se ha continuado por *round-robin;* por tanto, cada hebra ha realizado aquellas iteraciones cuyo número es múltiplo de su identificador de hebra.

(e) ¿Qué número de iteraciones se ha asignado a `cada` equipo?

RESPUESTA:

A cada equipo se le han asignado tantas iteraciones como hebras tiene, es decir cada hebra ejecutará una iteración; y así ha sido. Por tanto, a cada equipo se le han asignado 32 iteraciones.

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:

```
Target device: 1
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
Iteracción 35, en thread 35/1024 del team 0/48
```

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

RESPUESTA:

Como se puede ver en el cuadro de arriba, el número de equipos establecido por defecto es 48 dentro de los cuales hay 1024 hebras por defecto.

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

```
[ac418@atcgrid Coprocesadores]$ srun -pac4 -Aac ejecutables/omp_offload_GPU2 36
> salida2.txt
[ac418@atcgrid Coprocesadores]$ cat salida2.txt
```

Solo se ha proporcionado el comando de ejecución en la cola pues la salida se encuentra en el cuadro de arriba.

(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:

La respuesta es afirmativa, pues si nos fijamos en la diapositiva 9 con título: “Nvidia Quadro RTX 5000” aparece una tabla con las características. Estas características dicen que el número de SIMT del cuadro es de 48 SM, lo que concuerda con el número de equipos creados. De la misma manera, fijándonos en el número máximo de hebras por SM obtenemos que dicho número es 1024, lo que concuerda con el número de hebras creadas para cada equipo.

(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 1026, si la hubiera? (realizar las ejecuciones que se consideren necesarias para contestar a esta pregunta, en particular, ejecuciones con un número de iteraciones superior a 1026)

RESPUESTA:

Siguiendo el cuadro donde aparece la salida de la ejecución del código vemos que, por defecto, se asigna un total de 1024 iteraciones a cada equipo, es decir, una iteración para cada una de las hebras. Además, cada una de esas hebras ya asignadas a un ekipo, recibira aquella iteración cuyo número de iteración sea múltiplo de su identificador de hebra. Por tanto, la hebra i-ésima del equipo k-ésimo recibirá la iteración 1024*k+i.

Cabe recordar que la numeración en estos ámbitos, si tenemos n elementos, comenzará en 0 y terminará en n-1.

La iteración 2 se asgina a la hebra 2 del equipo 0.
La iteración 1026 se asigna a la hebra 2 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:

Tal y como se ve en las capturas, esto no es así pues vemos que para números de hilos menos que 32, el propio módulo toma por defecto 32 hilos. En general, tal y como se ve en la 2ª captura, se toma un número de hilos que sea múltiplo de 32, en ese caso toma 992 que es 32*31; dicho múltimplo es el inmediatamente inferior al número de hilos proporcionado; esto es así porque los últimos hilos, es decir, el resto de dividir el número de hilos proporcionado entre 32, no se pueden asignar en un “bloque” o *wrap* de 32.

Pasando al caso interesante, cuando tomamos un valor de hilos muy alto, la ejecución del programa ocasiona un *core dumpe*, esto es así hasta el valor 1056; es decir, si tomamoes valores de hilos menores que 1056 aplicará lo ya explicado, en otro caso tomará un *core dumped*. Esto se ha obtenido comenzando a ejecutar el programa con 3000 hilos y realizando búsqueda binaria en el rango [1,3000].

Además, tal y como se vió en el ejercicio anterior; el número máximo de hilos que se pueden crear es de 1024 para cada equipo.

CAPTURAS (que justifiquen la respuesta)

```
[ac418@atcgrid Coprocesadores]$ srun -pac4 -Aac ejecutables/omp_offload_GPU 33 3
 30 > salida.txt
[ac418@atcgrid Coprocesadores]$ cat salida.txt
Target device: 1
Tiempo:0.143882036
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
```

```
[ac418@atcgrid Coprocesadores]$ srun -pac4 -Aac ejecutables/omp_offload_GPU 33 3
 1056 > salida.txt
srun: error: atcgrid4: task 0: Aborted (core dumped)
[ac418@atcgrid Coprocesadores]$ srun -pac4 -Aac ejecutables/omp_offload_GPU 33 3
 1055 > salida.txt
[ac418@atcgrid Coprocesadores]$ srun -pac4 -Aac ejecutables/omp_offload_GPU 33 3
 3000 > salida.txt
srun: error: atcgrid4: task 0: Aborted (core dumped)
[ac418@atcgrid Coprocesadores]$
```

```
[ac418@atcgrid Coprocesadores]$ srun -pac4 -Aac ejecutables/omp_offload_GPU 33 3
 999 > salida.txt
[ac418@atcgrid Coprocesadores]$ cat salida.txt
Target device: 1
Tiempo:0.125753880
Iteracción 0, en thread 0/992 del team 0/3
Iteracción 1, en thread 1/992 del team 0/3
Iteracción 2, en thread 2/992 del team 0/3
Iteracción 3, en thread 3/992 del team 0/3
Iteracción 4, en thread 4/992 del team 0/3
Iteracción 5, en thread 5/992 del team 0/3
Iteracción 6, en thread 6/992 del team 0/3
Iteracción 7, en thread 7/992 del team 0/3
Iteracción 8, en thread 8/992 del team 0/3
Iteracción 9, en thread 9/992 del team 0/3
Iteracción 10, en thread 10/992 del team 0/3
Iteracción 11, en thread 11/992 del team 0/3
Iteracción 12, en thread 12/992 del team 0/3
Iteracción 13, en thread 13/992 del team 0/3
Iteracción 14, en thread 14/992 del team 0/3
Iteracción 15, en thread 15/992 del team 0/3
Iteracción 16, en thread 16/992 del team 0/3
Iteracción 17, en thread 17/992 del team 0/3
Iteracción 18, en thread 18/992 del team 0/3
Iteracción 19, en thread 19/992 del team 0/3
Iteracción 20, en thread 20/992 del team 0/3
Iteracción 21, en thread 21/992 del team 0/3
Iteracción 22, en thread 22/992 del team 0/3
Iteracción 23, en thread 23/992 del team 0/3
Iteracción 24, en thread 24/992 del team 0/3
Iteracción 25, en thread 25/992 del team 0/3
Iteracción 26, en thread 26/992 del team 0/3
Iteracción 27, en thread 27/992 del team 0/3
Iteracción 28, en thread 28/992 del team 0/3
Iteracción 29, en thread 29/992 del team 0/3
Iteracción 30, en thread 30/992 del team 0/3
Iteracción 31, en thread 31/992 del team 0/3
Iteracción 32, en thread 32/992 del team 0/3
[ac418@atcgrid Coprocesadores]$
```

(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 respuesta es afirmativa pues, tal y como se comentaba por encima en el apartado anterior, el número de hilos asignados a cada equipo es múltiplo de 32. En la misma diapositiva que en el ejercicio 2 se encuentra el número

máximo de hebras en un *wrap*, es decir, en un bloque de hilos a ejecutarse. Dicho bloque se manda a ejecución de forma simultánea.

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:

```
[ac418@atcgrid Coprocesadores]$ cat salida.txt
Target device: 1
Tiempo:0.129302025
Iteracción 0, en thread 0/1024 del team 0/1
Iteracción 1, en thread 1/1024 del team 0/1
Iteracción 2, en thread 2/1024 del team 0/1
Iteracción 3, en thread 3/1024 del team 0/1
Iteracción 4, en thread 4/1024 del team 0/1
Iteracción 5, en thread 5/1024 del team 0/1
Iteracción 6, en thread 6/1024 del team 0/1
Iteracción 7, en thread 7/1024 del team 0/1
Iteracción 8, en thread 8/1024 del team 0/1
Iteracción 9, en thread 9/1024 del team 0/1
Iteracción 10, en thread 10/1024 del team 0/1
Iteracción 11, en thread 11/1024 del team 0/1
Iteracción 12, en thread 12/1024 del team 0/1
Iteracción 13, en thread 13/1024 del team 0/1
Iteracción 14, en thread 14/1024 del team 0/1
Iteracción 15, en thread 15/1024 del team 0/1
Iteracción 16, en thread 16/1024 del team 0/1
Iteracción 17, en thread 17/1024 del team 0/1
Iteracción 18, en thread 18/1024 del team 0/1
Iteracción 19, en thread 19/1024 del team 0/1
Iteracción 20, en thread 20/1024 del team 0/1
Iteracción 21, en thread 21/1024 del team 0/1
Iteracción 22, en thread 22/1024 del team 0/1
Iteracción 23, en thread 23/1024 del team 0/1
Iteracción 24, en thread 24/1024 del team 0/1
Iteracción 25, en thread 25/1024 del team 0/1
Iteracción 26, en thread 26/1024 del team 0/1
Iteracción 27, en thread 27/1024 del team 0/1
Iteracción 28, en thread 28/1024 del team 0/1
Iteracción 29, en thread 29/1024 del team 0/1
Iteracción 30, en thread 30/1024 del team 0/1
Iteracción 31, en thread 31/1024 del team 0/1
Iteracción 32, en thread 32/1024 del team 0/1
```

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

RESPUESTA:

Tal y como se ve en las salidas de la ejecución del programa, se crea un solo equipo con 1024 hebras por defecto. Esto es así pues la directiva *teams* hace que se cree una liga de equipos. Probando con tamaños superiores a 1024 se ve que no se crean más equipos arreglo al número de iteraciones que se tengan que hacer. Por tanto, podemos concluir que la directiva *teams* se encarga de crear la liga de equipos formada por más de un equipo, si no se dice nada se crean 48; y la directiva *distribute* se encarga de distribuir las itearaciones por cada uno de los equipos, por defecto, se asignan 1024 iteraciones a cada equipo antes de cambiar del mismo.

(b) ¿Qué tanto por ciento del número de SM se están utilizando? Justificar respuesta.

RESPUESTA:

Según pone en las diapositivas, el coprocesador GPU dispone de 48 SM, cada uno de los cuales con 64 núcleos CUDA, pues en total, disponemos de 3072 núcleos CUDA; luego (3072/48)=64.

Como usamos solamente un equipo, es decir, estamos usando un solo SM, y como hay más hilos que procesadores, estaremos usando todos los procesadores del SM.

Por tanto, (64/3072)*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.

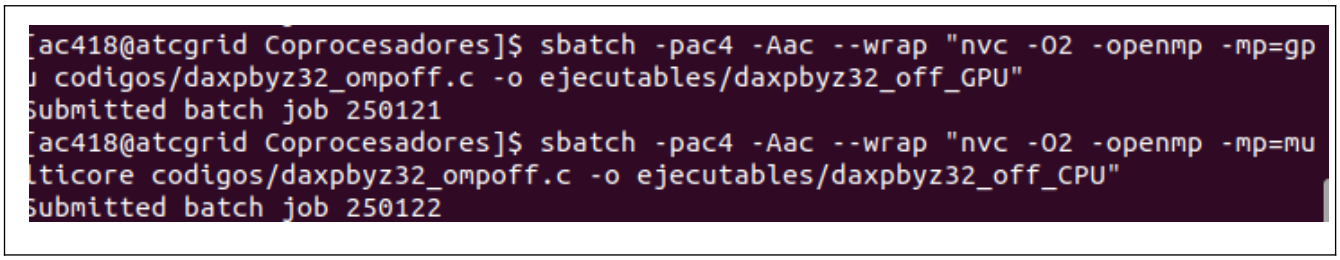
1) t2-t1 es el tiempo de `target enter data`, que reserva espacio en el dispositivo coprocesador para `x`, `y`, `z`, `N` y `p`, y transfiere del host al coprocesador aquellas que se mapean con `to` (`x`, `N` y `p`).
2) t3-t2 es el tiempo del primer `target teams distribute parallel for` del código, que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



Las salidas se encuentran en los archivos salidaCPU.txt y salidaGPU.txt. No se han añadido las capturas pues no se pueden ver con claridad.

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

RESPUESTA:

Tal y como se ve en los archivos, en el caso GPU, es la directiva target enter data la que tarda más pues engloba al tiempo que requiere copiar en la memoria de los coprocesadores las variables indicadas en la directiva. Estas variables deben ser mapeadas, aparecer en las tablas de paginación,…

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

RESPUESTA:

Dichas directivas son *target enter data, host actualiza, target data update, target2 y target exit data.* El motivo de que ocurra esto es que la CPU no usa memoria de la GPU, sino del *host*, lo que provoca ese mínimo tiempo en crear un nuevo ámbito de variables de la CPU. Esto mismo ocurre al actualizarlas (*target data update)* y al borrarlas(*target exit data*).

En conclusión, llegamos a que la CPU es un recurso bastante más rápido que la GPU. Tal y como se ve en los archivo, el tiempo de ejecución de la CPU es 100 veces menor que el de la GPU.

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 **la precisión** del resultado y los **tiempos de ejecución total, cálculo y comunicación** obtenidos en atcgrid4 con la CPU y la GPU, indicar cuál arquitectura son mejores y razonar los motivos.

CAPTURA CÓDIGO FUENTE: `pi-ompoff.c`

```
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;

//Transmision de datos a memoria (mapeado)//
tr_t1 = omp_get_wtime();

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

tr_t2 = omp_get_wtime();
//Fin Transmision de datos a memoria (mapeado)//

tr_tt = tr_t2 - tr_t1;  //Tiempo mapeado

//CALCULO DE PI//
pi_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+=(double)(4/(1+x*x));
  }

#pragma omp target exit data map(delete:intervals,width)map(from:sum)
sum *= width;
pi_t2 = omp_get_wtime();
//FIN CALCULO PI//

pi_tt= pi_t2-pi_t1;     //Tiempo calculos
printf("Iteraciones:\t%d\t. PI:\t%26.24f\t. Tiempos:\n\tTransmision:%8.6f\n\tCalc

return(0);
```

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

```
[ac418@atcgrid Coprocesadores]$ sbatch -pac4 -Aac --wrap "nvc -O2 -openmp -mp=mu
lticore ./codigos/pi-ompoff.c -o ejecutables/pi-ompoff_CPU"
Submitted batch job 251896
[ac418@atcgrid Coprocesadores]$ cat slurm-251895.out
cat: slurm-251895.out: No such file or directory
[ac418@atcgrid Coprocesadores]$ cat slurm-251896.out
/opt/rh/devtoolset-11/root/usr/bin/ld: warning: /opt/nvidia/hpc_sdk21.2/Linux_x8
6_64/21.2/compilers/lib/nvhpc.ld contains output sections; did you forget -T?
[ac418@atcgrid Coprocesadores]$ sbatch -pac4 -Aac --wrap "nvc -O2 -openmp -mp=gp
u ./codigos/pi-ompoff.c -o ejecutables/pi-ompoff_GPU"
Submitted batch job 251899
[ac418@atcgrid Coprocesadores]$ cat slurm-251899.out
/opt/rh/devtoolset-11/root/usr/bin/ld: warning: /opt/nvidia/hpc_sdk21.2/Linux_x8
6_64/21.2/compilers/lib/nvhpc.ld contains output sections; did you forget -T?
[ac418@atcgrid Coprocesadores]$ srun -pac4 -Aac ejecutables/pi-ompoff_CPU
Falta número de intevalossrun: error: atcgrid4: task 0: Exited with exit code 25
5
```

```
        Transmision:1.390728
        Calculos:       0.000664
[ac418@atcgrid Coprocesadores]$ srun -pac4 -Aac ejecutables/pi-ompoff_CPU 100000
00
Iteraciones:    10000000        . PI:   3.14159265358978290194613 7      . Tiempo
s:
        Transmision:0.000000
        Calculos:       0.013714
[ac418@atcgrid Coprocesadores]$ srun -pac4 -Aac ejecutables/pi-ompoff_GPU 100000
00
Iteraciones:    10000000        . PI:   3.14159265358979267190875 4      . Tiempo
s:
        Transmision:1.269622
        Calculos:       0.001468
[ac418@atcgrid Coprocesadores]$
```

RESPUESTA:

Según fuentes externas, las 25 primeras cifras del número pi son:

3.14159265358979323846264 3

Luego, si comparamos con los resultados obtenidos, vemos que la ejecución en la GPU es mucho más clara en este sentido, obteniendo una diferencia de una unidad |93-92| frente al *host* que ha obtenido una diferencia de 10 unidades |93-82|.

No obstante, si realizamos la comparativa de tiempos, el tiempo de transmisión del *host* es nulo pues dichas variables ya se encuentran en memoria; dicho tema se ha tradato en ejercicios anteriores. Además, el tiempo de mapeo de la GPU es positivo pues requiere que se pasen los datos a su memoria.

Por otra parte, pese a que en general la GPU es más tardía, los cálculos en la GPU son mayoritariamente más rápidos obteniendo una superioridad de algún orden de magnitud.

Por tanto, llegamos a la conclusión de que podemos perder unos segundos de ejecución a cambio de ganar en procisión, es decir, conviene usar la GPU.

Nota: Tal y como aparece en el código, la sección de transimisión se ha oocupado simplemente del mapeo de varibles y la región de cálculos se ha ocupado del cálculo de pi y de su eliminación de la memoria en la cual se ha mapeado. Ha sido decidido así por el autor de este archivo.