2º curso / 2º cuatr.
Grados en
Ing. Informática

Arquitectura de Computadores

Coprocesadores en OpenMP

Material elaborado por Mancia Anguita
Profesores: Mancia Anguita, Maribel García y Christian Morillas

UNIVERSIDAD
DE GRANADA

Departamento de
Ingeniería de Computadores,
Automática y Robótica
UNIVERSIDAD DE GRANADA

Bibliografía

- Especificaciones OpenMP para coprocesadores:
  - Sec. 2.14 (“*Device Directives*”):
    - https://www.openmp.org/specifications/
- Compilador nvc de Nvidia (instalado en atcgrid4, implementa parte de OpenMP 5.0):
  - https://docs.nvidia.com/hpc-sdk/compilers/hpc-compilers-user-guide/index.html#openmp-use
  - https://docs.nvidia.com/hpc-sdk/compilers/hpc-compilers-user-guide/index.html#openmp-subset

Contenidos

- **OpenMP 5**
- Coprocesador en atcgrid
- Arquitectura CPU + coprocesador
- Construcciones/Directivas para ejecutar código en coprocesadores
- Construcciones/Directivas target, teams, distribute, parallel y for
  - Combinar construcciones/directivas
- Construcción/Directiva target data
- Construcciones/Directivas target enter data y target exit data
- Cláusulas
- Variables de control y funciones

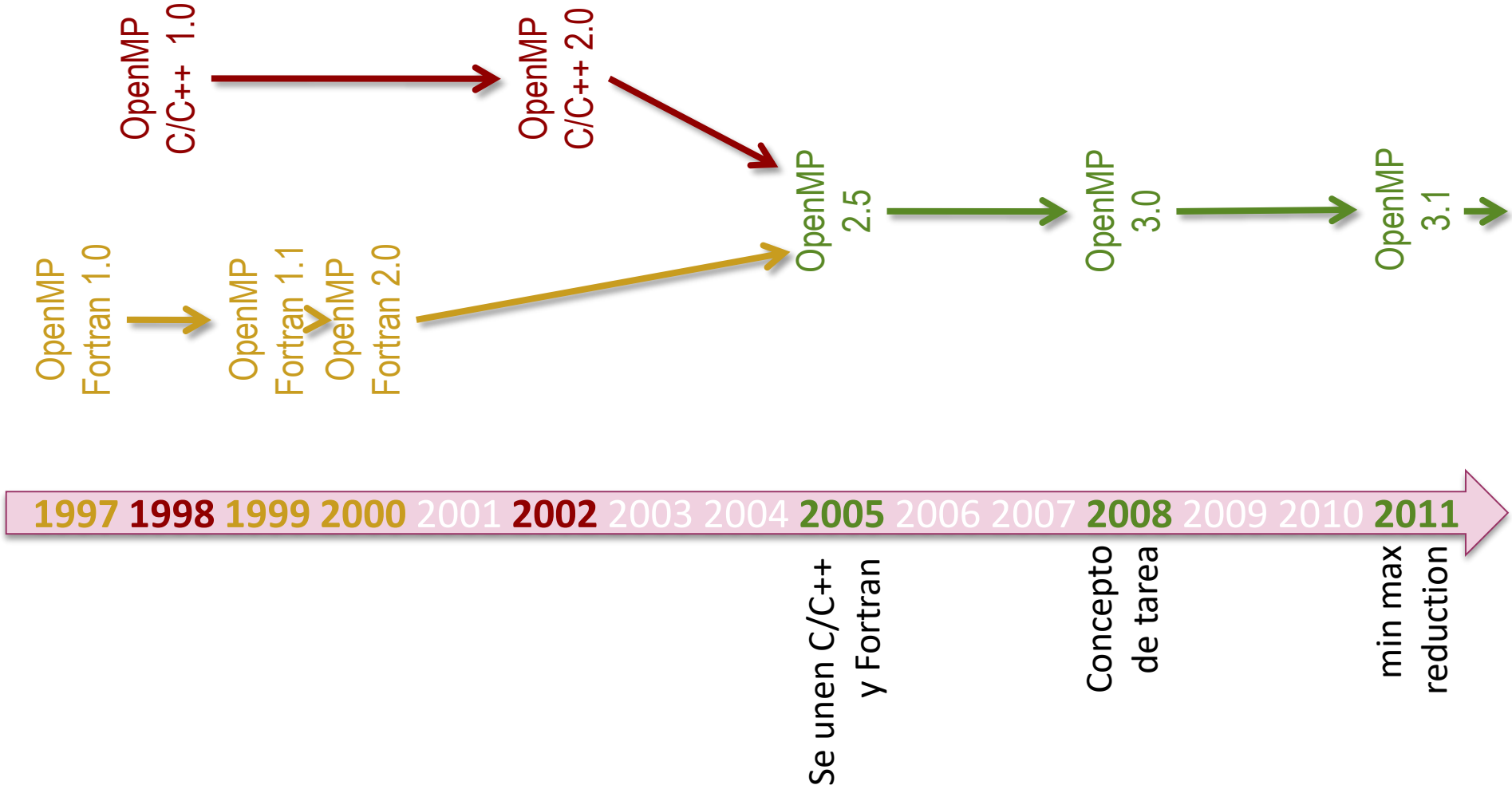
¿Qué es OpenMP 5?

- Es una API para C/C++ y Fortran para escribir código paralelo, usando directivas y funciones, con el paradigma/estilo de programación de
  - variables compartidas para ejecutar aplicaciones en paralelo en varios threads en una CPU
  - desde 2013, también con el estilo de programación de paralelismo de datos para ejecutar código paralelo en aceleradores (coprocesadores)
- API (*Application Programming Interface*):
  - **Capa de abstracción** que permite al programador acceder cómodamente a través de una interfaz a un conjunto de funcionalidades.
- La API OpenMP define/comprende:
  - **Directivas** del compilador, **funciones** de biblioteca, y **variables** de entorno.

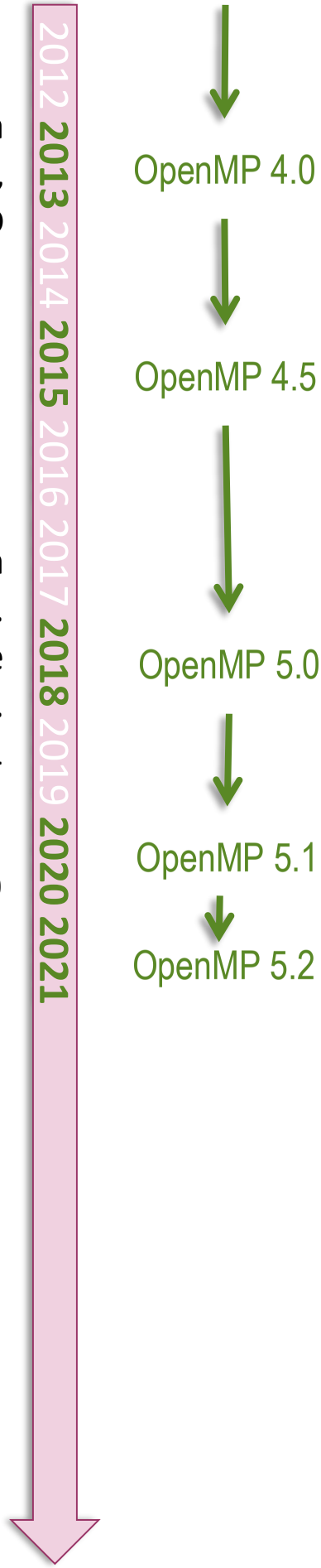
¿Qué es OpenMP 5?

- Es una herramienta para programación paralela:
  - No automática (no extrae paralelismo implícito)
  - Con un modelo de programación:
    - Basado el paradigma/estilo de variables compartidas y en **paralelismo de datos** (Lección 4/Tema 2)
      - Ofrece al programador los medios para distribuir la ejecución de un programa entre diferentes recursos computacionales, llamados dispositivos (***devices***) en las especificaciones de OpenMP (CPUs, GPUs, DSPs, …), en un sistema de cómputo heterogéneo.
    - Multithread
      - Los threads de un dispositivo no pueden migrar a otro.
      - La ejecución comienza en el dispositivo anfitrión (***host***).
      - Los threads tienen los datos en memorias. Los dispositivos pueden o no compartir memoria.
    - Basada en directivas del compilador y funciones (Lección 4/Tema2):
      - El código paralelo OpenMP es código escrito con un lenguaje secuencial (C, C++ o Fortran) + directivas y funciones de la interfaz OpenMP
  - Portable.

Evolución de OpenMP



Evolución de OpenMP



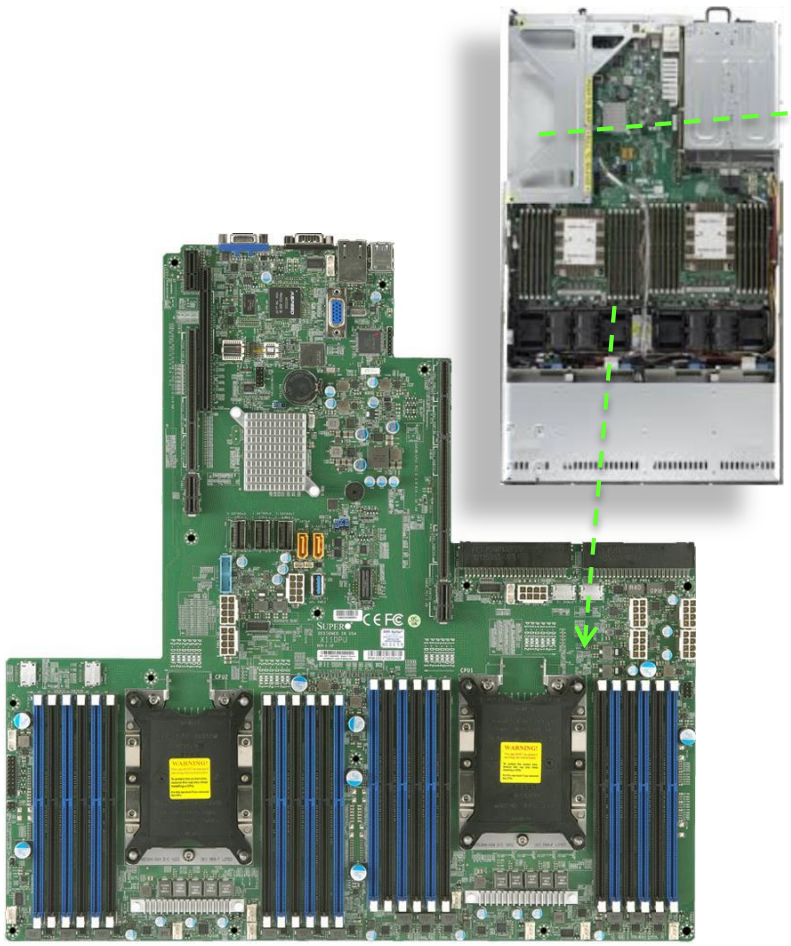
Descarga de código a coprocesadores, SIMD

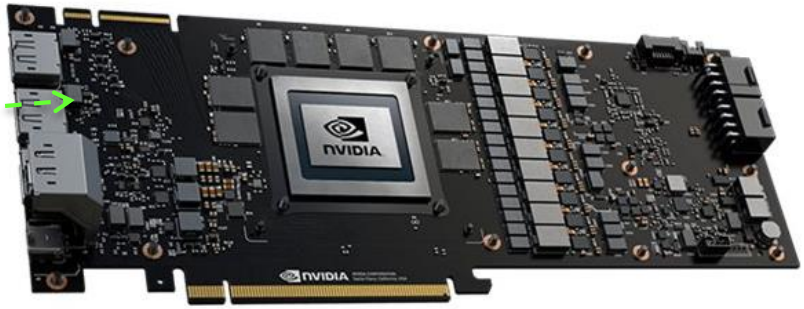
Ampliación descarga a código en coprocesadores

Ampliación descarga a coprocesadores. Modelo del sistema de memoria. Soporte inicial para C11

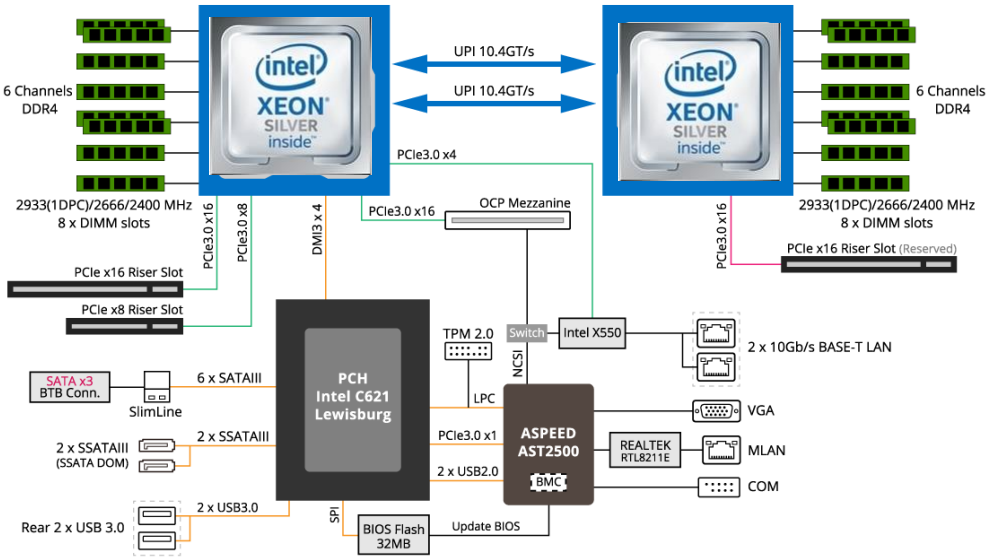
Soporte completo para C11, C++11, C++14, C++20

Nodo con coprocesador en atcgrid: atcgrid4





Nvidia Quadro RTX 5000 (3,072 núcleos)

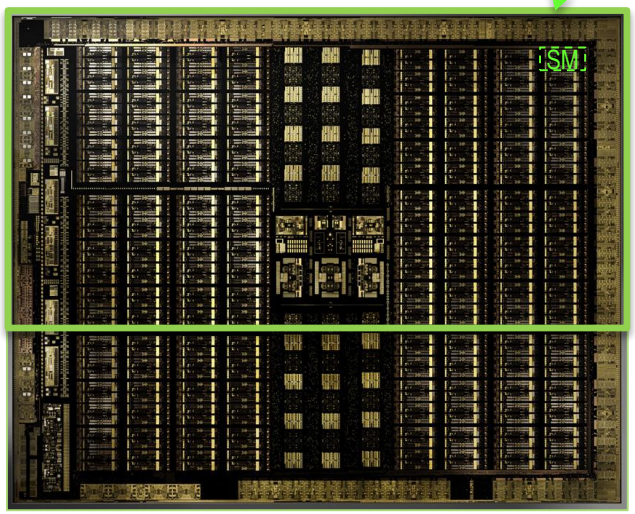


Intel® C621 Chipset

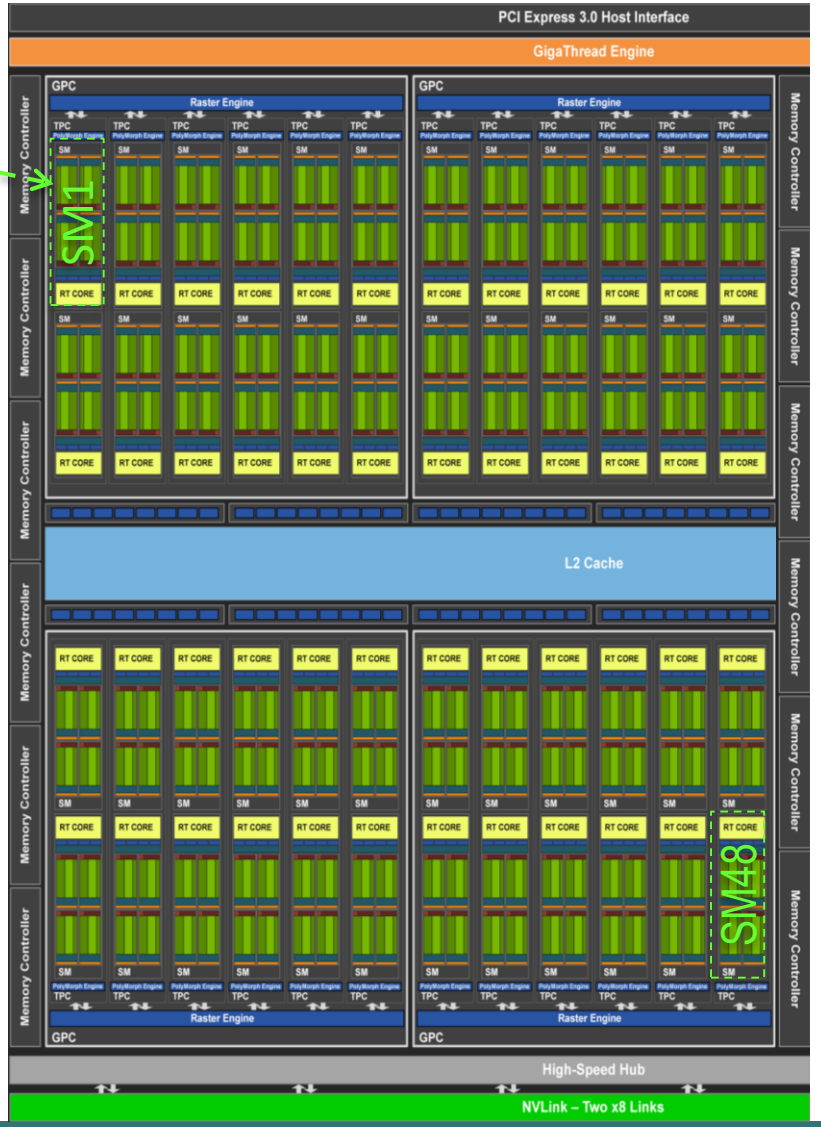
Supermicro SYS-6019U-TR4 1U
https://www.supermicro.com/en/products/motherboard/X11DPU

Nvidia Quadro RTX 5000

Características	Quadro RTX 5000
Núcleos de procesamiento paralelo	3072
Streaming Multiprocessor (SM) SIMT	48
Máximo de *threads* por SM (o CUDA Block)	1024
Nº de threads en un warp (los threads se envían a ejecución agrupados en unidades llamadas *warp* en CUDA)	32
Memoria de la GPU en la tarjeta	GDDR6 de 16 GB
Bus para conexión con host	PCI Express 3.0 x 16



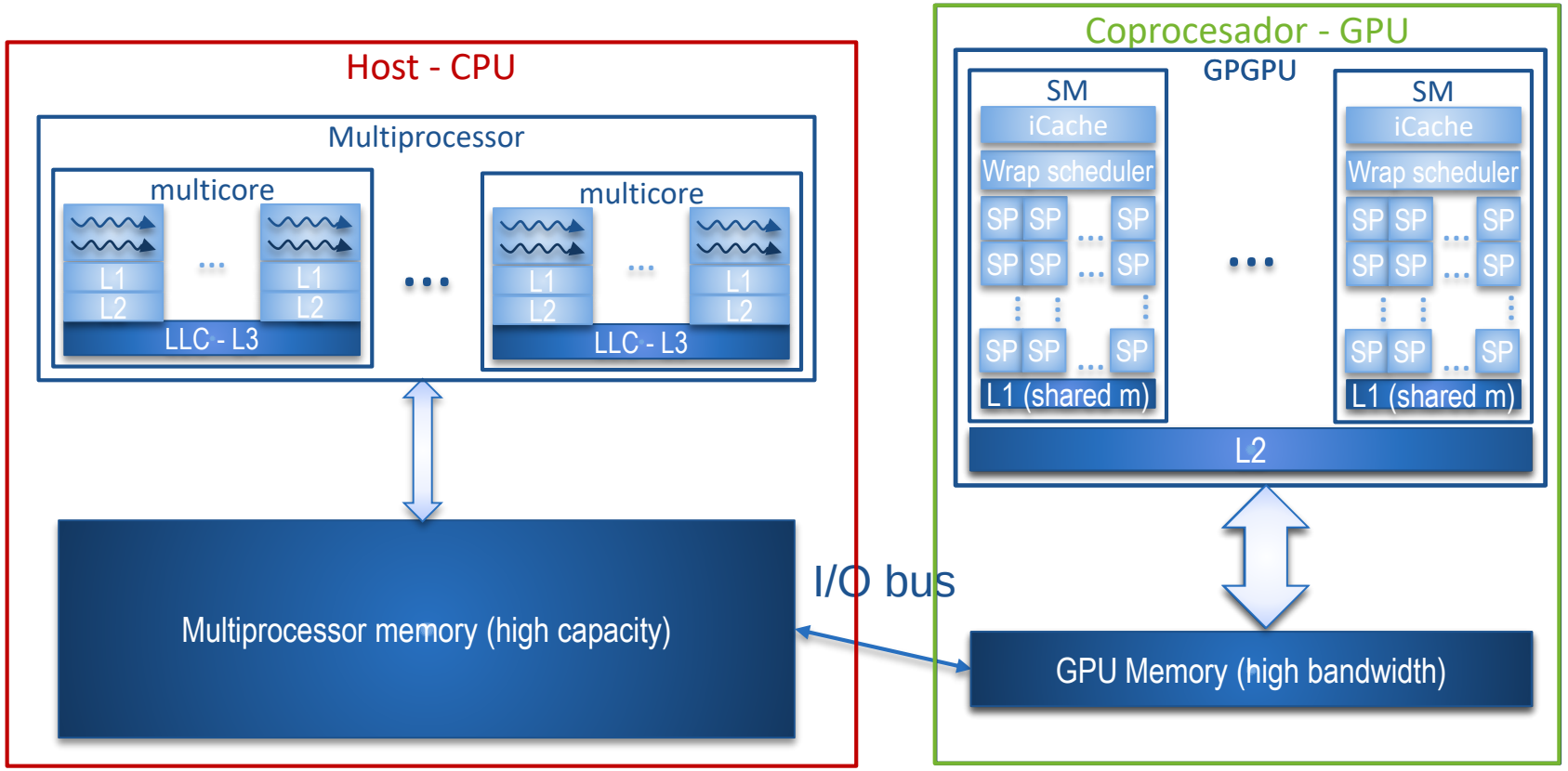
Nvidia Quadro RTX 5000 tiene 2/3 de núcleos que el chip mostrado (48 SM en lugar de 72)



Contenidos

- OpenMP 5
- Coprocesador en atcgrid
- **Arquitectura CPU + coprocesador**
- Construcciones/Directivas para ejecutar código en coprocesadores
- Construcciones/Directivas target, teams, distribute, parallel y for
  - Combinar construcciones/directivas
- Construcción/Directiva target data
- Construcciones/Directivas target enter data y target exit data
- Cláusulas
- Variables de control y funciones

Arquitectura CPU + coprocesador GPU



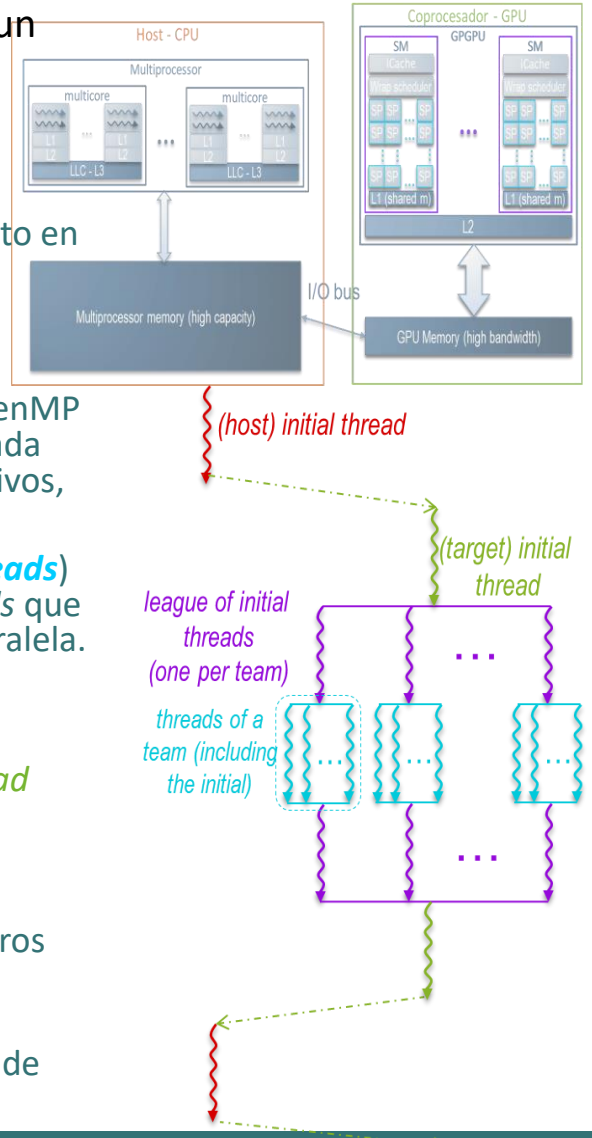
Los núcleos (*cores*) de una CPU pueden ejecutar dos hilos (= 1 hilo)

SP (*Streaming Processor*) o núcleo CUDA
SM (*Streaming Multiprocessor*)
GPGPU (*General purpose* GPU)

Se usa un bus de E/S (*Input/Output*) para la comunicación entre el host y la GPU (transferencias entre memorias)

Uso de coprocesadores en OpenMP: terminología y definiciones

- OpenMP, desde la versión 4.0, permite la distribución de la ejecución de un código entre los diferentes dispositivos de cómputo disponibles en una plataforma.
- Ejecución de programas:
  - Para OpenMP un **dispositivo** o ***device*** (CPU, GPU, DSP, …) es un recurso de cómputo en el que se puede ejecutar una **región** de código
  - El dispositivo anfitrión o ***host device*** es aquel en el que comienza la ejecución del programa OpenMP. Si hay un total de N dispositivos, el host se identifica por N.
  - Los dispositivos usados como coprocesadores/aceleradores se denominan en OpenMP dispositivo destino o ***target devices***. Desde el host se enviará trabajo a estos. A cada uno se le asigna un identificador, de 0 a N-1, siendo N el número total de dispositivos, incluido el host.
  - Cada dispositivo tendrá un conjunto de flujo de instrucciones o hilos/hebras (***threads***) propio que no pueden migrar a otro dispositivo. Hay flujos llamados *initial threads* que crearán a otros flujos que junto a él colaborarán en la ejecución de una región paralela. Estos flujos se pueden organizar en equipos, cada uno tendrá un *initial threads.*
  - El primer flujo que se ejecuta en el host es el *initial thread* del host.
  - Cuando se quiere ejecutar código en un dispositivo destino se crea un *initial thread* asociado a este.
- Gestión de datos:
  - La memoria a la que acceden los dispositivos puede o no estar compartida con otros dispositivos.
  - OpenMP usa mapeo o ***map*** para denominar la forma en la que las variables se comparten entre dispositivos. El mapeo no implica necesariamente transferencia de datos entre dispositivos.



Contenidos

- OpenMP 5
- Coprocesador en atcgrid
- Arquitectura CPU + coprocesador
- **Construcciones/Directivas para ejecutar código en coprocesadores**
- Construcciones/Directivas target, teams, distribute, parallel y for
  - Combinar construcciones/directivas
- Construcción/Directiva target data
- Construcciones/Directivas target enter data y target exit data
- Cláusulas
- Variables de control y funciones

Directivas para usar coprocesadores (4.5)

TYPES		Executable				Declarative
Devices		target, target data			target enter data/ target exit data, target update	declare target/end declare target, declare target
Teams		teams	distribute			
Threads	Parallel	parallel				
	Work-sharing	single	for			
		sections				
	Synchroni-zation	critical*		atomic	barrier, flush*	

TYPES	With structure block	With an associated loop	With a statement	Stand-alone	With declaration-definition of variables, functions

- * solo soportada para cpu en nvc, error para gpu
- *Stand-alone directives*: directivas ejecutables que no tienen código de usuario asociado
- Se han destacado en color las directivas añadidas en v4 para el uso de coprocesadores

Contenidos

- OpenMP 5
- Coprocesador en atcgrid
- Arquitectura CPU + coprocesador
- Construcciones/Directivas para ejecutar código en coprocesadores
- **Construcciones/Directivas target, teams, distribute, parallel y for**
  - Combinar construcciones/directivas
- Construcción/Directiva target data
- Construcciones/Directivas target enter data y target exit data
- Cláusulas
- Variables de control y funciones

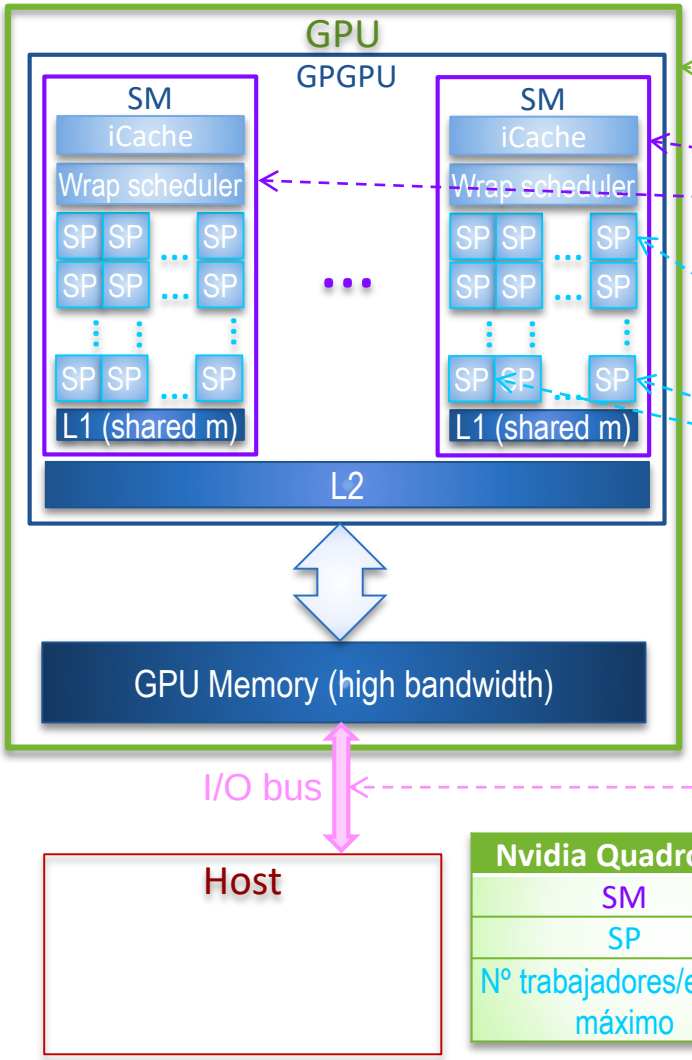
Directivas target, teams, distribute, parallel y for en coprocesadores

TYPES		Executable				Declarative
Devices		**target**, target data			target enter data/ target exit data, target update	declare target/end declare target, declare target
Teams		**teams**	**distribute**			
Threads	Parallel	**parallel**				
	Work-sharing	single	**for**			
		sections				
	Synchroni-zation	critical*		atomic	barrier, flush*	

TYPES	With structure block	With an associated loop	With a statement	Stand-alone	With declaration-definition of variables, functions

- **#pragma omp target** delimita el código a ejecutar en un dispositivo coprocesador. El entorno de datos queda definido por el mapeo implícito y explícito de variables
- **#pragma omp teams** crea una liga de equipos de trabajo, en particular crea los hilos iniciales (*initial threads*) de cada equipo.
- **#pragma omp distribute** distribuye las iteraciones de un bucle entre los equipos (*teams*)
- **#pragma omp parallel** cada hilo inicial que encuentra esta construcción crea el resto de hilos que conforma el equipo.
- **#pragma omp for** distribuye las iteraciones de un bucle entre los hilos de un equipo.

Construcciones/directivas y hardware



Nvidia Quadro RTX 5000	
SM	48
SP	64
Nº trabajadores/equipo máximo	1024

daxpyz_off.c

$z_i = a \times x_i + y_i$

```
…
init(x,y);
…
#pragma omp target \ //delimita trabajo para coprocesador
 map(to:N,alpha,x[:N],y[:N]) map(from:z[:N])
{
  #pragma omp teams \ //crea una liga de equipos de trabajo
   num_teams(nteams) thread_limit(mthreads)
  {
    int s_block = N /  omp_get_num_teams();
    #pragma omp distribute //reparte trabajo entre equipos
    for (i = 0; i < N; i += s_block) {

      #pragma omp parallel//crea los trabajadores de los equipos
      {
        #pragma omp for //reparte trabajo entre trab./equipo
        for (int j=i; j<i+s_block; j++) {
          z[j] = alpha * x[j] + y[j];
        }
      }
    }
  }
}
print_results(N,z);
…
#pragma omp target\ //delimita trabajo para coprocesador
 map(to:N,x[:N],y[:N]) map(tofrom:z[:N])
{…}
```

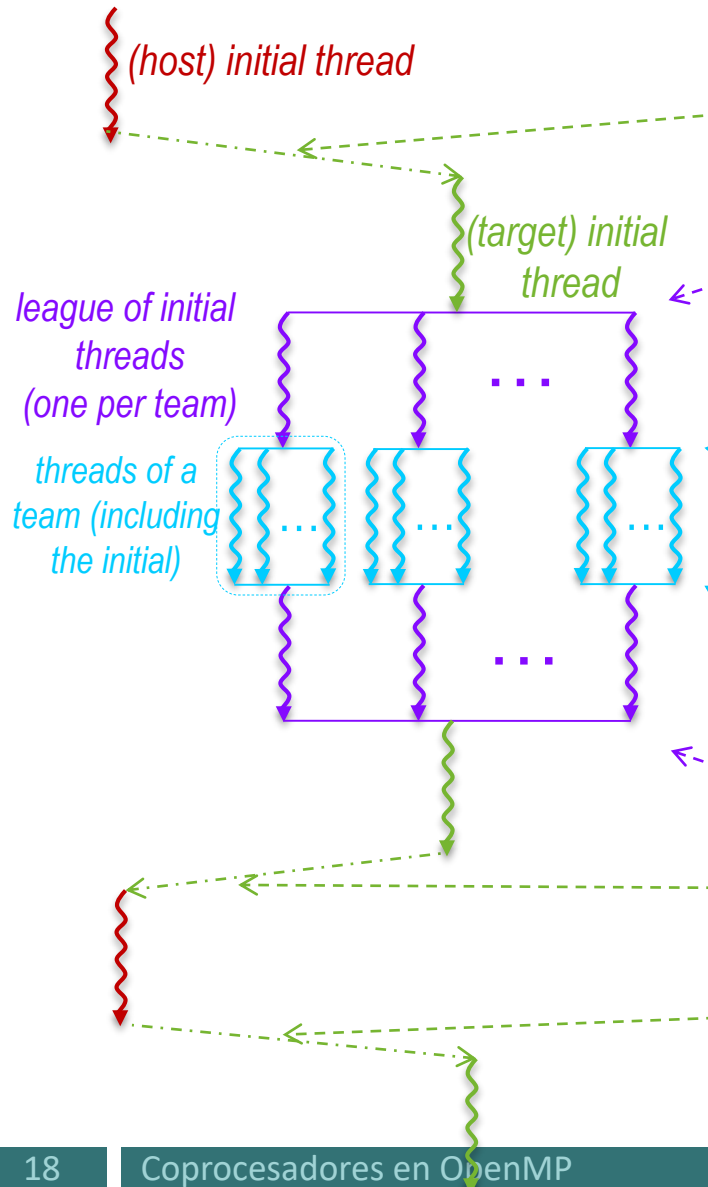
Elemento	OpenMP
coprocesador	*target device*
trabajo coprocesador	*target task*
liga de equipos	*league*
equipo de trabajadores	*team*
trabajador	*thread*

host

coprocesador

host coproc.

Directivas y flujos de instrucciones



daxpyz_off.c

$z_i = a \times x_i + y_i$

```
…
init(x,y);
…
#pragma omp target \ //delimita trabajo para coprocesador
 map(to:N,alpha,x[:N],y[:N]) map(from:z[:N])
{
  #pragma omp teams \ //crea una liga de equipos de trabajo
   num_teams(nteams) thread_limit(mthreads)
  {
    int s_block = N / omp_get_num_teams();
    #pragma omp distribute //reparte trabajo entre equipos
    for (i = 0; i < N; i += s_block) {

      #pragma omp parallel //crea trabajadores/equipo
      {
        #pragma omp for //reparte trabajo entre trab./equipo
        for (int j=i; j<i+s_block; j++) {
          z[j] = alpha * x[j] + y[j];
        }
      }
    }
  }
}
print_results(N,z);
…
#pragma omp target\ //delimita trabajo para coprocesador
 map(to:N,x[:N],y[:N]) map(tofrom:z[:N])
{…}
```

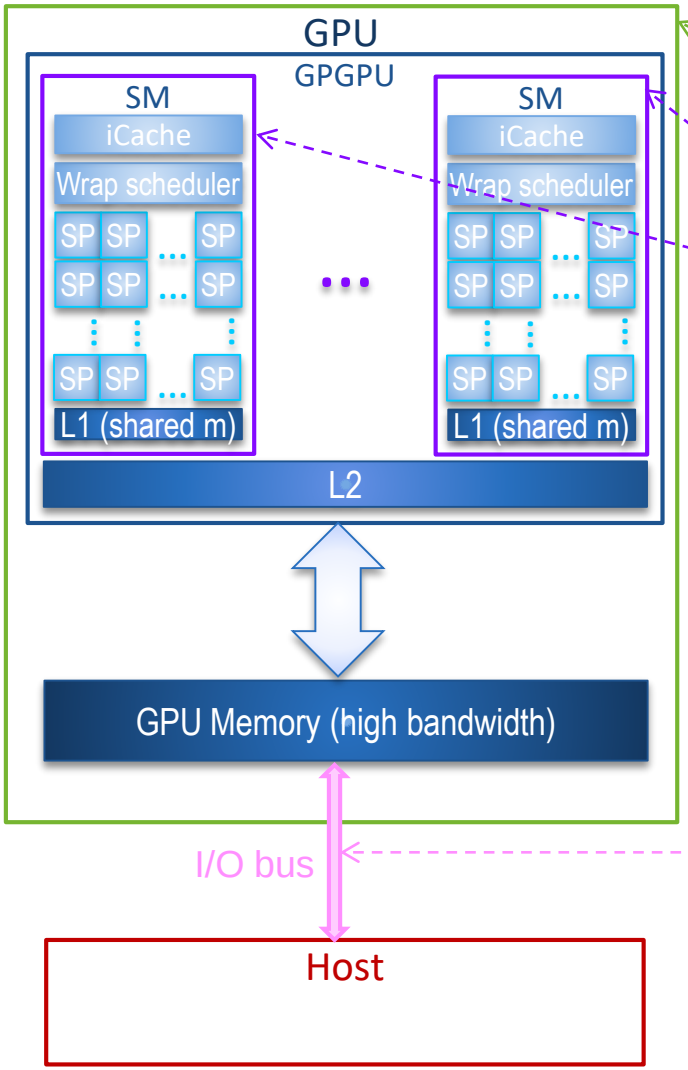
Elemento	OpenMP
coprocesador	*target device*
trabajo coprocesador	*target task*
liga de equipos	*league*
equipo de trabajadores	*team*
trabajador	*thread*

host

coprocesador

host coproc.

Combinar directivas



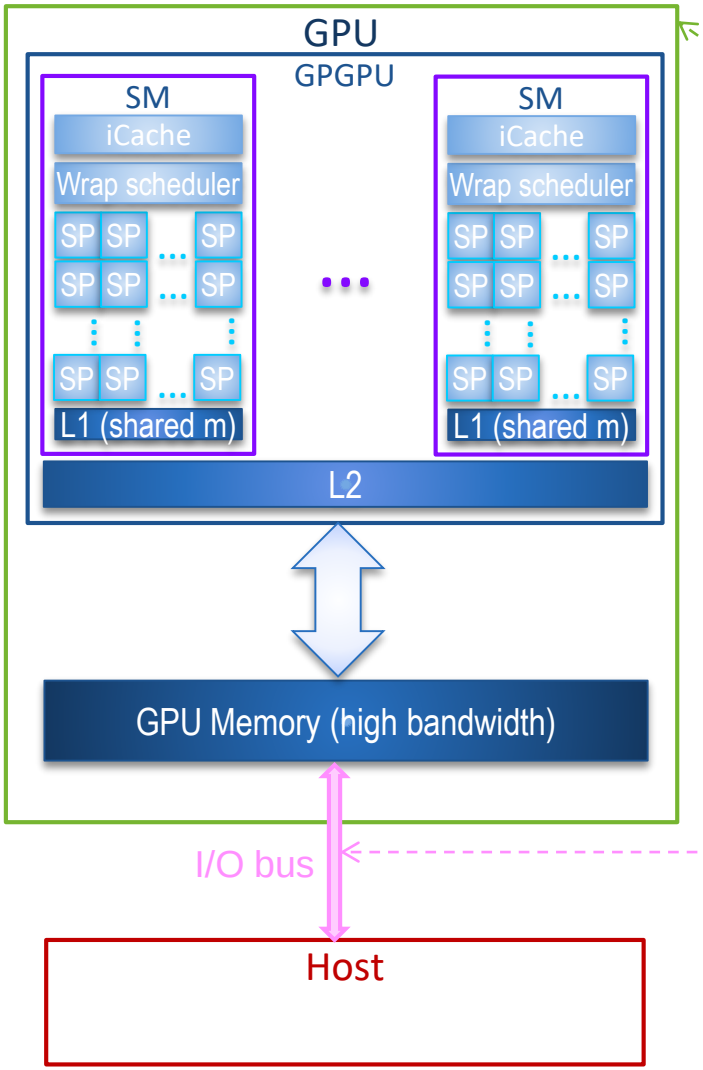
daxpyz_off2.c

```
…
init(x,y);
…
#pragma omp target teams \
num_teams(nteams) thread_limit(mthreads) \
map(to:N,alpha,x[:N],y[:N]) map(from:z[:N])
{
#pragma omp distribute parallel for
for (i = 0; i < N; i++) {
    z[i] = alpha * x[i] + y[i];
  }
}
print_results(N,z);
…
#pragma omp target\ //delimita trabajo para coprocesador
 map(to:N,x[:N],y[:N]) map(tofrom:z[:N])
{…}
```

host
device
host
device

- Más trabajo para la herramienta de programación

Combinar directivas II



daxpyz_off3.c

```
…
init(x,y);
…
#pragma omp target teams distribute parallel for\
num_teams(nteams) thread_limit(mthreads) \
map(to:N,alpha,x[:N],y[:N]) map(from:z[:N])
{
  for (i = 0; i < N; i++) {
    z[j] = alpha * x[j] + y[j];
  }
}
print_results(N,z);
…
#pragma omp target\  //delimita trabajo para coprocesador
 map(to:N,x[:N],y[:N]) map(tofrom:z[:N])
{…}
```

host
device
host
device

- Más trabajo para la herramienta de programación

Contenidos

- OpenMP 5
- Coprocesador en atcgrid
- Arquitectura CPU + coprocesador
- Construcciones/Directivas para ejecutar código en coprocesadores
- Construcciones/Directivas target, teams, distribute, parallel y for
  - Combinar construcciones/directivas
- **Construcción/Directiva target data**
- Construcciones/Directivas target enter data y target exit data
- Cláusulas
- Variables de control y funciones

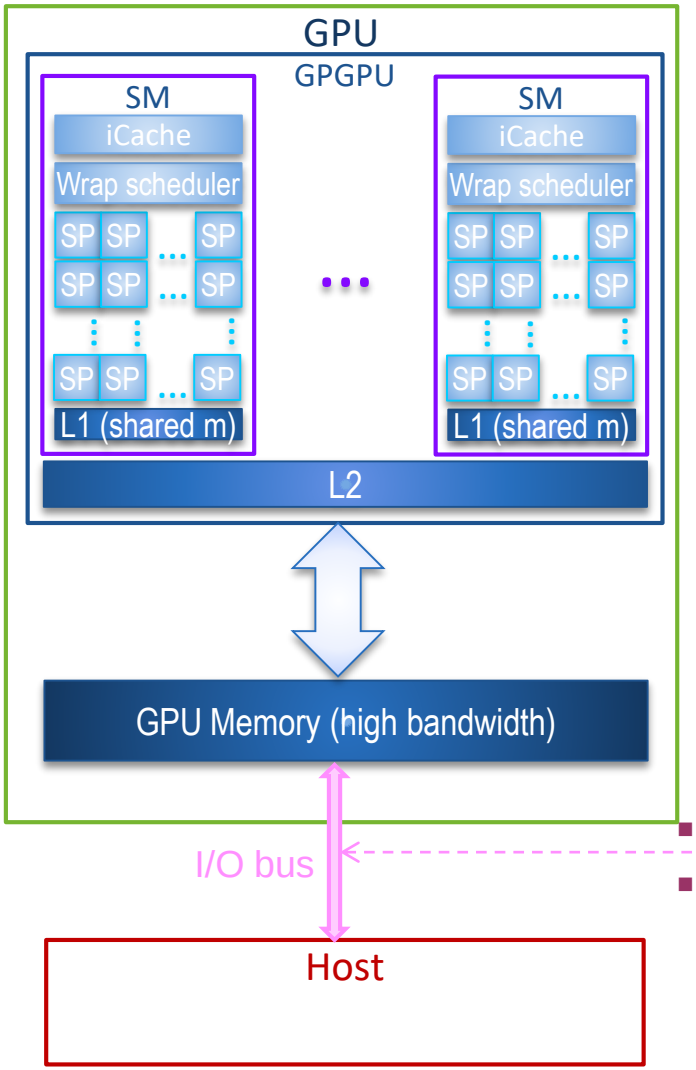
Construcción **target data**

TYPES		Executable				Declarative
Devices		target, **target data**			target enter data/ target exit data, target update	declare target/end declare target, declare target
Teams		teams	distribute			
Threads	Parallel	parallel				
	Work-sharing	single	for			
		sections				
	Synchroni-zation	critical*		atomic	barrier, flush*	

TYPES	With structure block	With an associated loop	With a statement	Stand-alone	With declaration-definition of variables, functions

- **#pragma omp target data**: crea un ámbito de datos de dispositivo

Construcción **target data**: crea un ámbito de datos de dispositivo



daxpyz_off4.c

```
…
init(x,y);
…
#pragma omp target data \
map(to:N,alpha,x[:N],y[:N]) map(from:z[:N])
{
  #pragma omp target teams distribute parallel for\
  num_teams(nteams) thread_limit(mthreads)
    for (i = 0; i < N; i++) {
      z[i] = alpha * x[i] + y[i];
    }

  #pragma omp target
  {…}
}
```

host

device

- Ámbito de datos compartido por varios targets (2 en ej.)
- **Target data** reserva espacio en el dispositivo para las variables en **`map`**, inicializa aquellas con **`to`**, y transfiere al host el contenido de aquellas con **`from`** cuando termine la ejecución de los `targets` que engloba.

Contenidos

- OpenMP 5
- Coprocesador en atcgrid
- Arquitectura CPU + coprocesador
- Construcciones/Directivas para ejecutar código en coprocesadores
- Construcciones/Directivas target, teams, distribute, parallel y for
  - Combinar construcciones/directivas
- Construcción/Directiva target data
- **Construcciones/Directivas target enter data y target exit data**
- Cláusulas
- Variables de control y funciones

Directivas **target enter data, target exit data** y **target update**

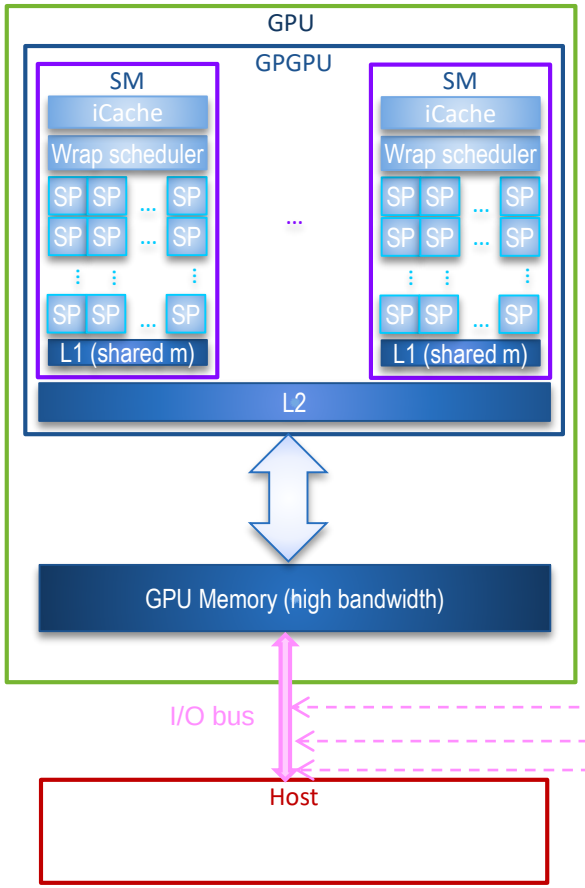
TYPES		Executable				Declarative
Devices		target, target data			**target enter data**/ **target exit data**, **target update**	declare target/end declare target, declare target
Teams		teams	distribute			
Threads	Parallel	parallel				
	Work-sharing	single	for			
		sections				
	Synchroni-zation	critical*		atomic	barrier, flush*	
TYPES		**With structure block**	**With an associated loop**	**With a statement**	**Stand-alone**	**With declaration-definition of variables, functions**

- **#pragma omp target enter data**: mapea variables a un ámbito de datos de coprocesador (reserva espacio variables en el coprocesador)
- **#pragma omp target exit data**: desmapea variables de un ámbito de datos de coprocesador
- **#pragma omp target update**: actualizar variables (host->coprocesador, host<-coprocesador)

Directivas target enter data, target exit data y target update

$z_i = a \times x_i + b \times y_i$

daxpbyz32_off.c




```
//Calcula z = a·x + b · y (x, y, z vectores dinámicos float, a y b escalares)
…
hostlee(N,a,b);
hostreserva(x,y,z); hostinicia(x); p = a;
t1 = omp_get_wtime();
#pragma omp target enter data map(to:N,p,x[:N]) \
map(alloc:z[:N],y[0:N])
t2 = omp_get_wtime();
#pragma omp target teams distribute parallel for
for (int i = 0; i < N; i++) z[i] = p * x[i];
t3 = omp_get_wtime();
hostinicia(y); p = b;
t4 = omp_get_wtime();
#pragma omp target update to(p, y[:N])
t5 = omp_get_wtime();
#pragma omp target teams distribute parallel for
for (int i = 0; i < N; i++) z[i] = z[i] + p * y[i];
t6 = omp_get_wtime();
#pragma omp target exit data map(delete:N,p,x[:N],\
y[0:N]) map(from: z[:N])
t7 = omp_get_wtime();
…
hostprint_result(N,z,t1-t0,t2-t1,t3-t2,t4-t3,t5-t4,t6-
t5,t7-t6,t7-t0);
…
```

Construcción **declare target**

TYPES		Executable				Declarative
Devices		target, target data			target enter data/ target exit data, target update	**declare target**/**end declare target**, **declare target**
Teams		teams	distribute			
Threads	Parallel	parallel				
	Work-sharing	single	for			
		sections				
	Synchroni-zation	critical*		atomic	barrier, flush*	

TYPES	With structure block	With an associated loop	With a statement	Stand-alone	With declaration-definition of variables, functions

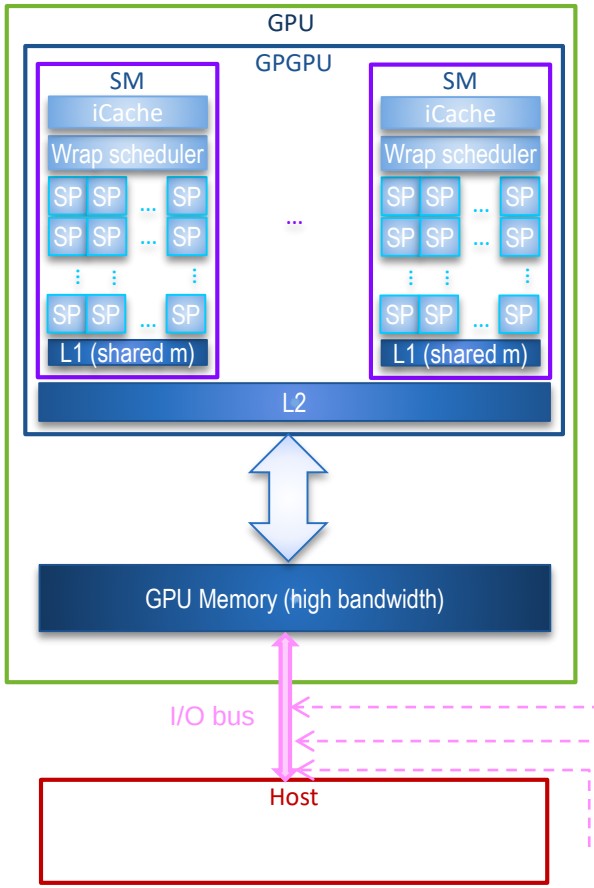
- **#pragma omp declare target**:
  - Solo para variables globales y funciones.
  - Crea versiones en el coprocesador para variables y funciones, y asigna almacenamiento en el dispositivo.

#pragma omp declare target
declarations-definition-seq
#pragma omp end declare target
#pragma omp declare target (*extended-list***)**
#pragma omp declare target *clause[[,]clause ...]*

Directiva **declare target**

$z_i = a \times x_i + b \times y_i$

daxpbyz32_off.c




```
//Calcula z = a·x + b · y (x, y, z vectores globales float, a y b escalares float)
…
#define MAX 50331648 //100663296
#pragma omp declare target
 float x[MAX], y[MAX], z[MAX], p;  int N;
#pragma omp end declare target
int main(int argc, char **argv) {
…
hostlee(N,a,b);
hostinicia(x); p = a;
t1 = omp_get_wtime();
#pragma omp target update to(N, p, x[:N])
t2 = omp_get_wtime();
#pragma omp target teams distribute parallel for
for (int i = 0; i < N; i++) z[i] = p * x[i];
t3 = omp_get_wtime();
hostinicia(y); p = b;
t4 = omp_get_wtime();
#pragma omp target update to(p, y[:N])
t5 = omp_get_wtime();
#pragma omp target teams distribute parallel for
for (int i = 0; i < N; i++) z[i] = z[i] + p * y[i];
t6 = omp_get_wtime();
#pragma omp target update from(z[:N])
t7 = omp_get_wtime();
…
hostprint_result(N,z,t1-t0,t2-t1,t3-t2,t4-t3,t5-t4,t6-t5,t7-
t6,t7-t0);
…
```

Contenidos

- OpenMP 5
- Coprocesador en atcgrid
- Arquitectura CPU + coprocesador
- Construcciones/Directivas para ejecutar código en coprocesadores
- Construcciones/Directivas target, teams, distribute, parallel y for
  - Combinar construcciones/directivas
- Construcción/Directiva target data
- Construcciones/Directivas target enter data y target exit data
- **Cláusulas**
- Variables de control y funciones

Cláusulas que se van a utilizar en el bloque práctico

Tipo	Clause	target	teams	distribute	parallel	for	target data	target enter data	target exit data	target update	declare target (exteded-list)
Dispositivo	**device**(integer-expression) (1)	X					X	X	X		
Control nº threads	**thread_limit**(integer-expression)		X								
	num_teams(upper-bound)		X								
Control ámbito	**reduction** (reduction-identifier: list)		X		X	X					
Compartición datos dispositivo	**map**([map-type:] list item)	X					X	X	X		

- (1): se puede usar una vez
- ***list***: lista de variables separadas por comas
- ***list item*****:** pueden ser variables o secciones de vectores o matrices
- ***map-type***: puede ser `alloc`, `to`, `from`, `tofrom`, `release`, `delete`
  - Por defecto, en OpenMP 4.0 se usa tofrom, a partir de OpenMP 4.5 firstprivate.

Contenidos

- OpenMP 5
- Coprocesador en atcgrid
- Arquitectura CPU + coprocesador
- Construcciones/Directivas para ejecutar código en coprocesadores
- Construcciones/Directivas target, teams, distribute, parallel y for
  - Combinar construcciones/directivas
- Construcción/Directiva target data
- Construcciones/Directivas target enter data y target exit data
- Cláusulas
- **Variables de control y funciones**

Variables de entorno relacionadas con coprocesadores

Variable de control	Ámbito	Valor (valor inicial)	¿Qué controla?	Consultar /Modificar
default-device-var	entorno de datos	Es un entero no negativo (depende de la implementación)	Nº de dispositivo a utilizar por defecto en las construcciones (directivas) de dispositivo	sí(f) /sí(ve,f)
target-offload-var	global	mandatory | disabled | default (default)	Controla la descarga de código a dispositivos externos	no /sí(ve)
team-size-var	team	(one)	Tamaño del equipo actual. Una copia por entorno de datos	sí(f) /no
nteams-var	dispositivo	(zero)	Nº de equipos que solicitará una región teams	sí(f) /sí(ve,f)
teams-thread-limit-var	dispositivo	(zero)	Nº máximo de threads en un team	sí(f) /sí(ve,f)

v5.1 no disponible en nvc

Variables de entorno relacionadas con descarga de código a dispositivos externos

Variable de control	Variable de entorno para inicializar valor	Función para modificar	Función para leer
default-device-var	OMP_DEFAULT_DEVICE	omp_set_default_device()	omp_get_default_device()
target-offload-var	OMP_TARGET_OFFLOAD		
team-size-var			omp_get_num_threads()
nteams-var	OMP_NUM_TEAMS	omp_set_num_teams()	omp_get_max_teams()
teams-thread-limit-var	OMP_TEAMS_THREAD_LIMIT	omp_set_teams_thread_limit()	omp_get_teams_thread_limit()

v5.1 no disponible en nvc

Otras rutinas del entorno de ejecución

- omp_get_team_num()
  - Devuelve al *thread* el identificador del *team* al que pertenece
- omp_get_num_teams()
  - Devuelve el total de *teams* en un región
- omp_get_num_devices()
  - Devuelve el número de coprocesadores disponibles en el nodo

Funciones relacionadas con ejecución en coprocesadores

```
#include <stdlib.h>
#include <stdio.h>
#include <omp.h>
#define MAX 33554432
int inf[MAX][4];

int main(int argc, char **argv) {
  if (argc < 4) { printf("Faltan nº de iteraciones bucle, nº de teams o nº de threads/teams");
                  exit(-1);}
  int N = atoi(argv[1]);  int nteams = atoi(argv[2]);  int mthreads = atoi(argv[3]);
  printf("Hay %d dispositivo(s) además del host \n ", omp_get_num_devices());
  printf("Nº de núcleos lógicos en CPU=%d\n", omp_get_num_procs());
  #pragma omp target teams distribute parallel for map(from:inf[:N][4]) map(to:N) \
   num_teams(nteams)  thread_limit(mthreads)
    for (i = 0; i < N; i++) {
      inf[i][0] = omp_get_thread_num();
      inf[i][1] = omp_get_num_threads();
      inf[i][2] = omp_get_team_num();
      inf[i][3] = omp_get_num_teams();
    }
  for (int j=0; j < N; j++)
   printf("Iteracción %d, en thread %d/%d del team %d/%d\n",j,inf[j][0],inf[j][1],inf[j][2],in
f[j][3]);
  return 0;
}
```