Tema 4.1 Programación mediante directivas OpenMP

Computación de Altas Prestaciones

Carlos García Sánchez

5 de octubre de 2022

- 'Using OpenMP : portable shared memory parallel programming", Barbara Chapman, et all. 2008
- "OpenMP 5.0", `https://www.openmp.org/wp-content/uploads/OpenMP-API-Specification-5.0.pdf`

Outline

1 Introducción

2 Arquitecturas

3 Modelos de programación paralela

4 API

¿Qué es OpenMP?

OpenMP es

- Una API para expresar explícitamente el paralelismo
- Compuesto por tres componentes:
  - Directivas del compilador
  - Rutinas de biblioteca de tiempo de ejecución
  - Variables de entorno

OpenMP no es un lenguaje

- Para sistemas paralelos de memoria distribuida
- Para hacer un uso más eficiente de la memoria compartida
- Requiere verificar dependencias de datos, conflictos de datos, condiciones de carrera...

Objetivos

Estandarización

- Proporciona un estándar entre una variedad de arquitecturas / plataformas de memoria compartida
- Definido y respaldado conjuntamente por un la mayoría de los proveedores hardware y software

Definición sencilla

- Conjunto simple y limitado de directivas
- Se puede expresar un paralelismo elevado con solo 3 o 4 directivas
  - Ahora obj. menos significativo (versiones más complejas)

Objetivos

Facilidad de uso

- Permite paralelizar incrementalmente un programa secuencial
- Permite implementar paralelismo de grano grueso y grano fino

Portabilidad

- La API está disponible para C/C ++ y Fortran
- La mayoría de las plataformas soportadas (Unix/Linux y Windows)

Historia

- A principios de los 90s, los proveedores de sistemas de memoria compartida suministraron sus propias extensiones de programación basadas en directivas:
  - El usuario especificaba con directivas qué bucles se deben paralelizar
  - El compilador del fabricante sería responsable de paralelizar automáticamente bucles en sistemas SMP
  - Implementaciones con funcionalmente similares, pero divergentes
- En 1997 aparece estándar OpenMP[1]

[1] Especificaciones en `https://www.openmp.org/`

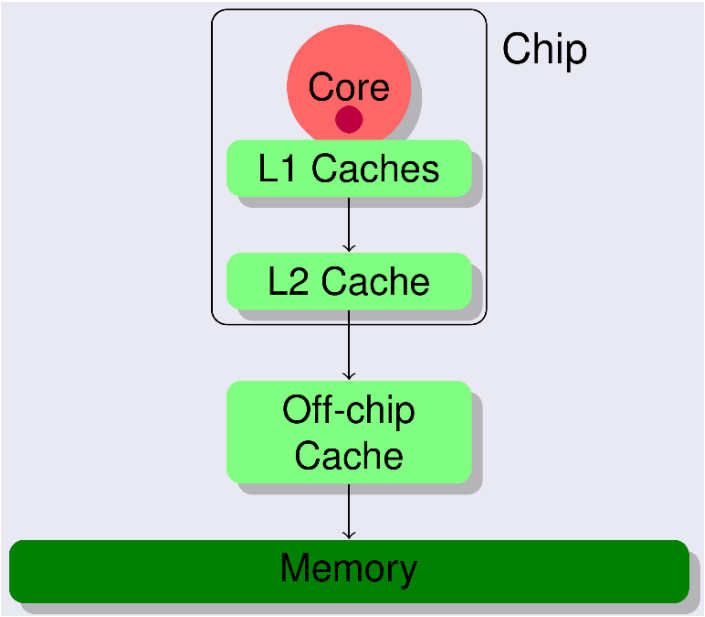
Historia

- OpenMP evoluciona periódicamente y es considerado estándar de programación paralela para sistemas memoria compartida
  - Nuevas construcciones y características añadidas en sucesivas versiones
- Inicialmente, aparece una API para C y Fortran separada
  - Desde 2005, cada versión está unificada
- Mantenida por *Architecture Review Board (ARB)*, un consorcio de la industria y la academia

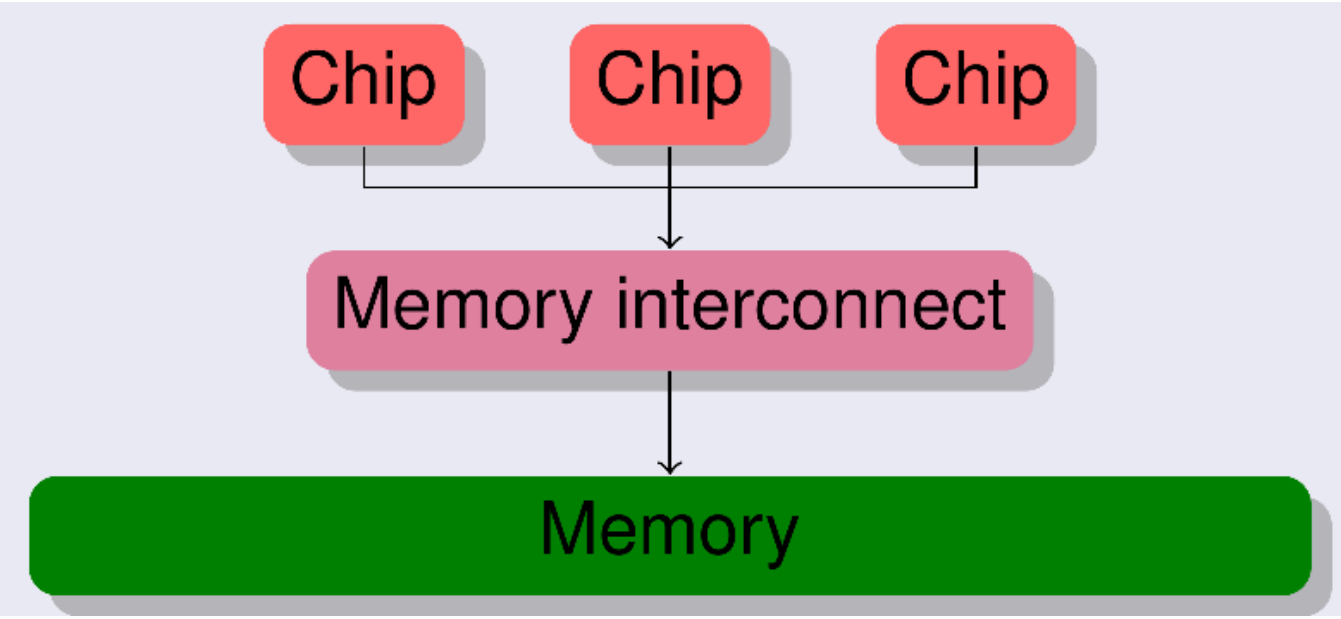
Historia

Fecha	Versión
Oct 1997	Fortran 1.0
Oct 1998	C/C++ 1.0
Nov 1999	Fortran 1.1
Nov 2000	Fortran 2.0
Mar 2002	C/C++ 2.0
May 2005	OpenMP 2.5
May 2008	OpenMP 3.0
Jul 2011	OpenMP 3.1
Jul 2013	OpenMP 4.0
Nov 2015	OpenMP 4.5
Nov 2018	OpenMP 5.0

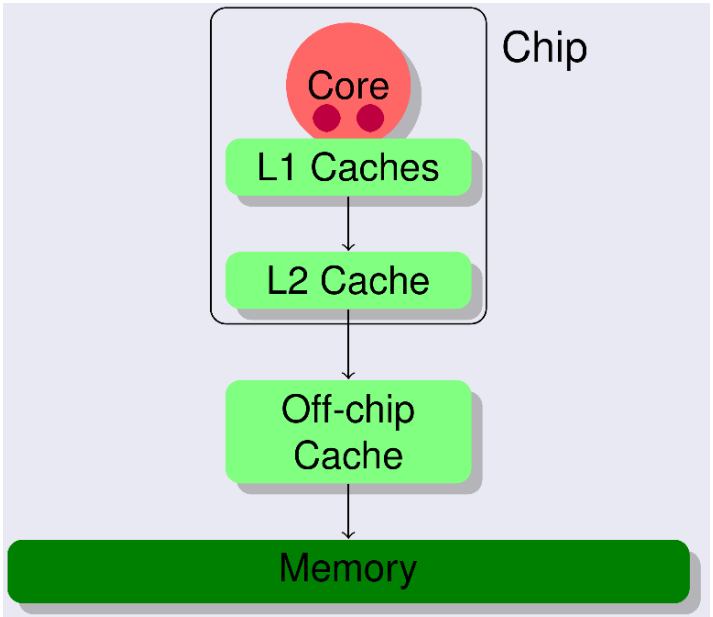
Procesadores mono-core



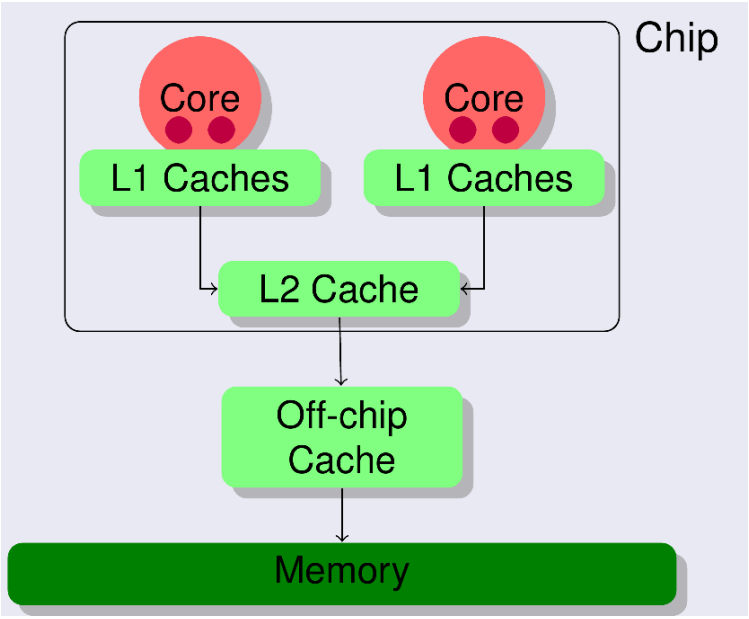
Procesadores SMP



Procesadores SMT



Procesadores multi-core



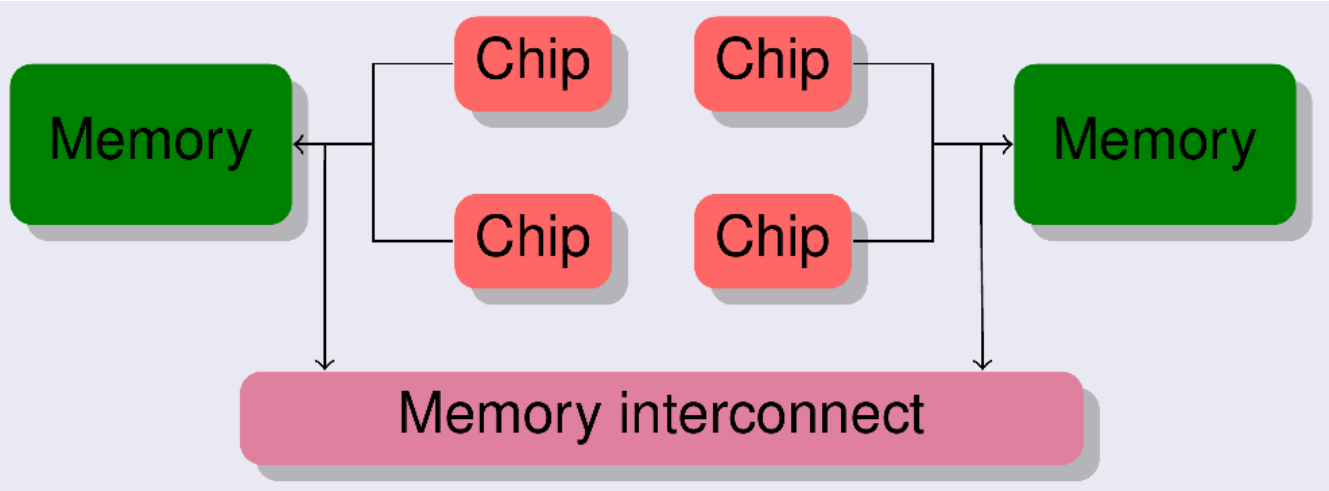
Memoria

Coherencia cache

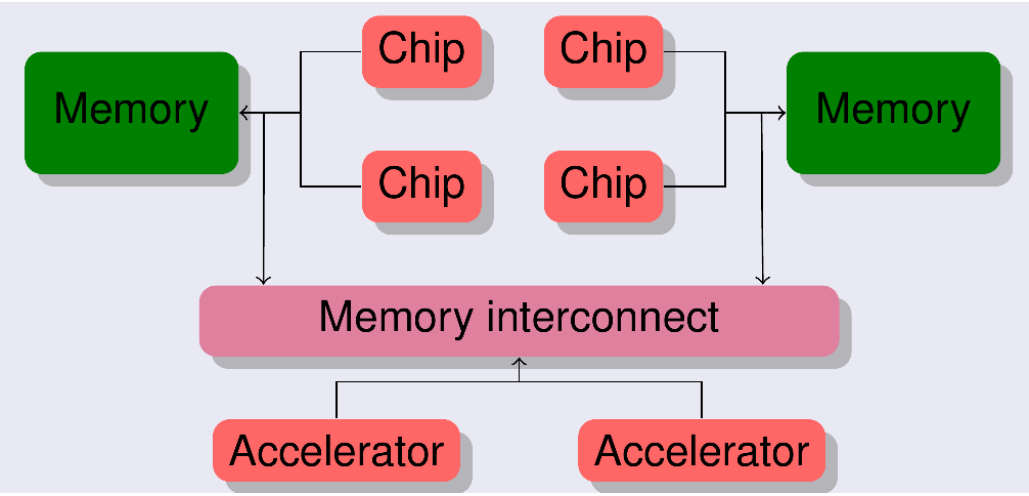
- El hw normalmente permite disponer de la misma visión de la *jerarquía cache*
- Para permitir coherencia cache: **cada línea de caches tiene su tags**
  - Escrituras invalidan la línea de cache (posiciones contiguas de mem)

NUMA

- *Non-Uniform Memory Access*
- La migración y la localidad de datos tiene repercusiones importantes en el rendimiento

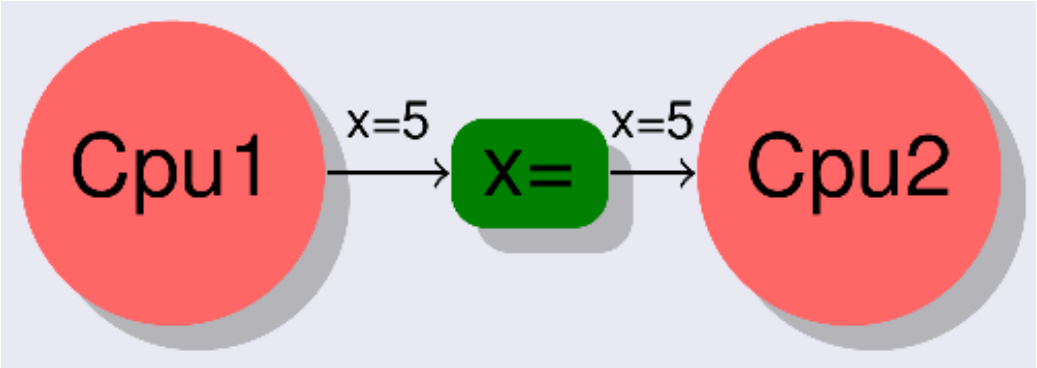


Arquitecturas heterogéneas



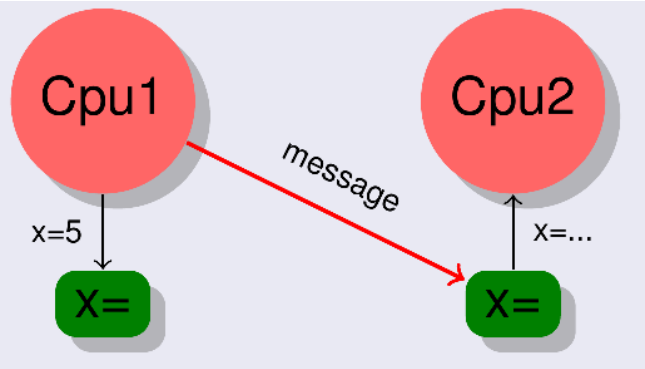
Modelos de memoria compartida

- Memoria es compartida por todos los procesadores
- Existen mecanismos de comunicación y sincronización a través de la memoria compartida



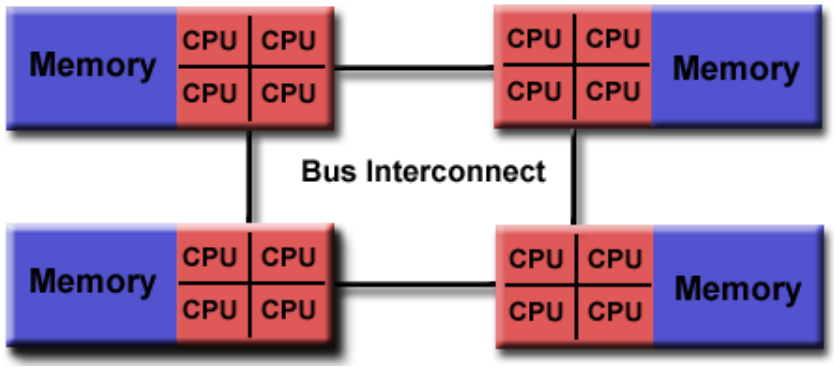
Modelos de memoria distribuida

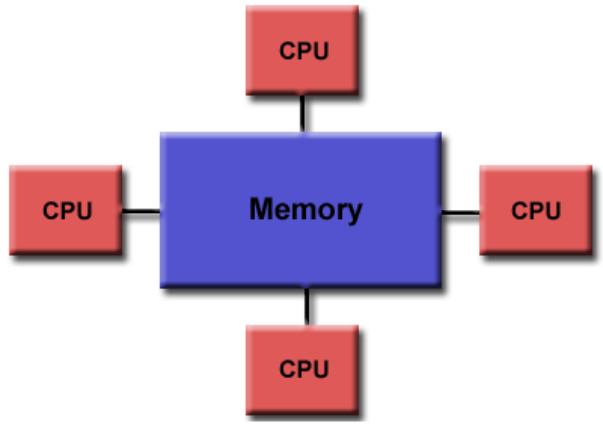
- Cada proceso tiene su propio espacio de direcciones de memoria
  - Ese espacio de memoria no es accesible por otros procesos
- La comunicación y sincronización se lleva a cabo explicitamente mediante mensajes



Uso en HPC

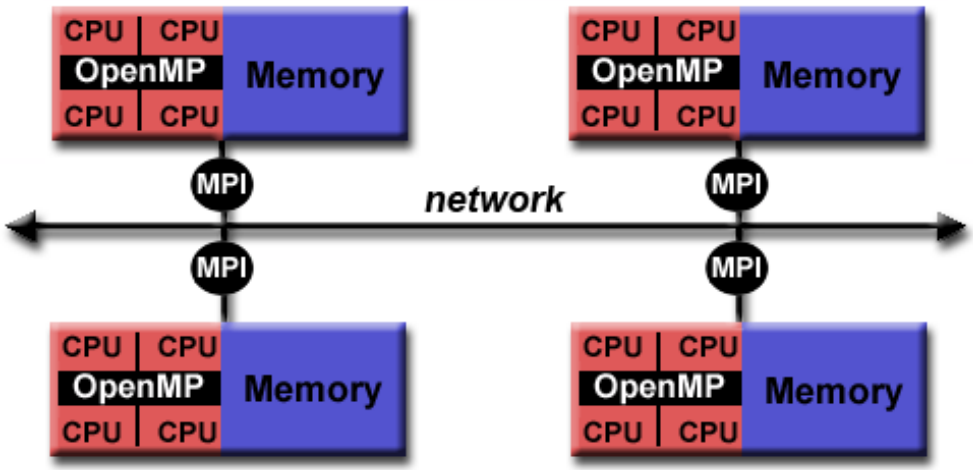
- Memoria Compartida
  - Diseñado para multiprocesadores de memoria compartida tipo UMA/NUMA
    - Ha ido evolucionando a sistemas multiprocesador-multicore





Uso en HPC

- Por defecto el paralelismo expresado funciona para **un nodo**
- En el caso de HPC, se puede combinar con sistemas para memoria distribuida
  - OpenMP es usado para trabajo intensivo dentro del nodo
  - MPI para comunicar y compartir datos entre nodos



Paralelismo con hilos

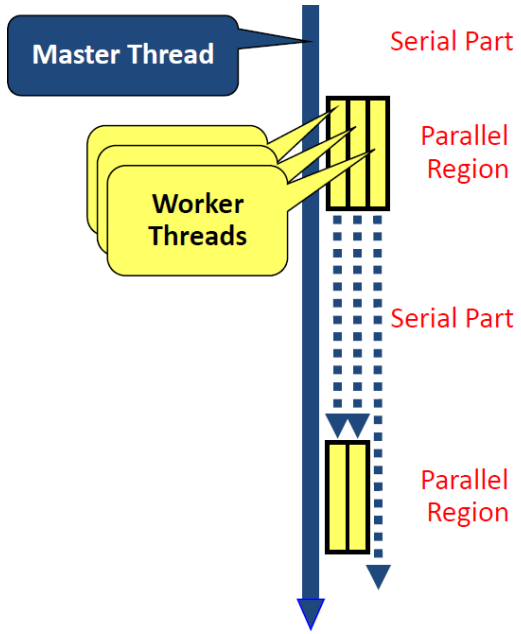
- OpenMP explota el **paralelismo mediante hilos/threads**
- Un hilo es entidad más pequeña de procesamiento
  - Más liviano que un proceso
- Habitualmente, un número de hilos se pueden mapear sobre una máquina multiprocesador/core

Paralelismo explícito

- OpenMP es un paradigma de programación explícito (no automático)
- Expresado mediante **directivas**
- Utiliza modelo **fork-join**

Modelo de ejecución

- Los programan comienzan con un hilo: **hilo maestro**
- *Regiones paralelas* por hilos *worker* o "equipo de hilos"
- Entre las regiones paralelas *worker* están dormidos
  - Modelo **fork-join**

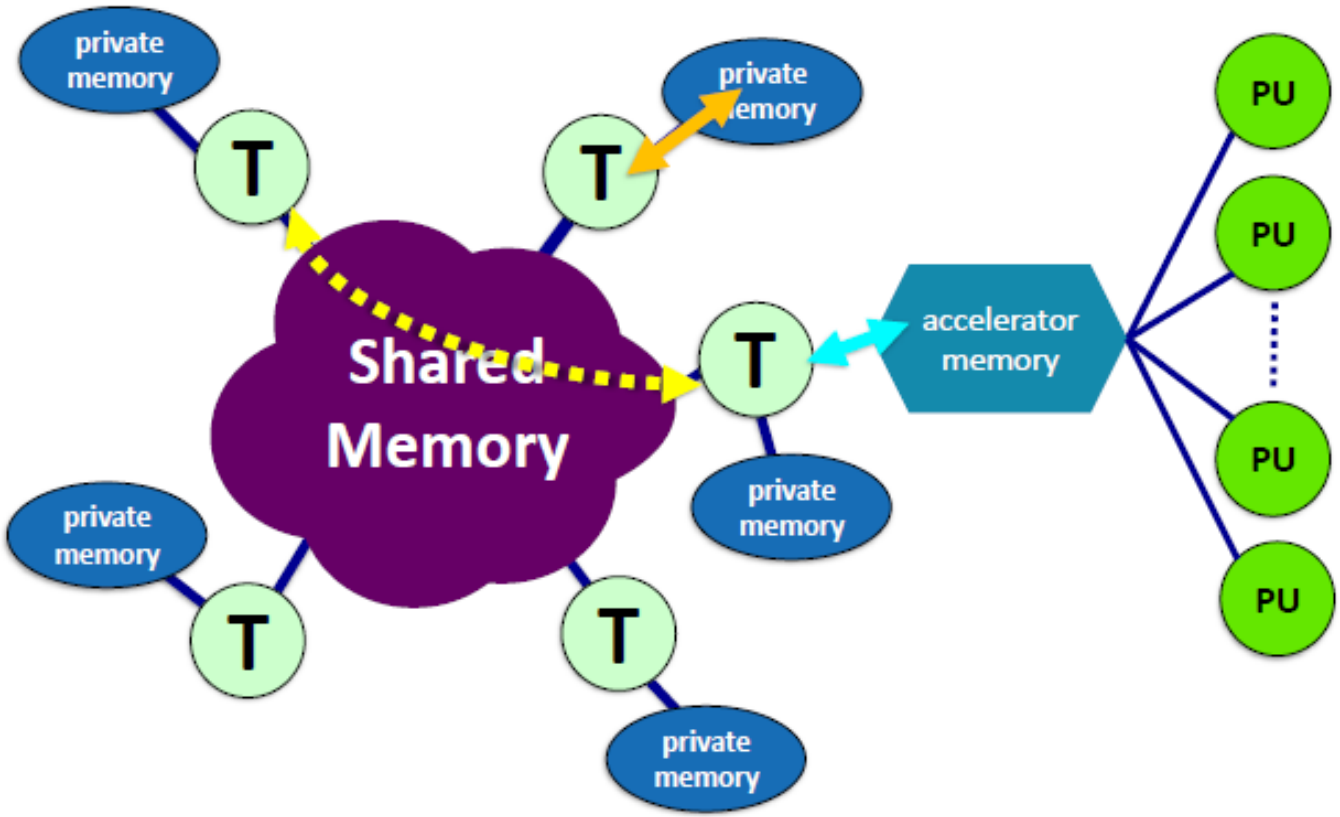


Modelo de memoria

- OpenMP es utilizado para memoria compartida, por defecto en un región paralela los datos son **compartidos**
- OpenMP define el modelo de memoria como **relajado**
  - Hilos pueden ver diferentes valores de la misma variable
  - La consistencia de memoria se garantiza en puntos específicos
  - Cuando una el valor de una variable es **crítica**, es responsabilidad del programador asegurarse que hace un *flush* al resto de hilos
- Las variables se pueden compartir, o ser **privadas**

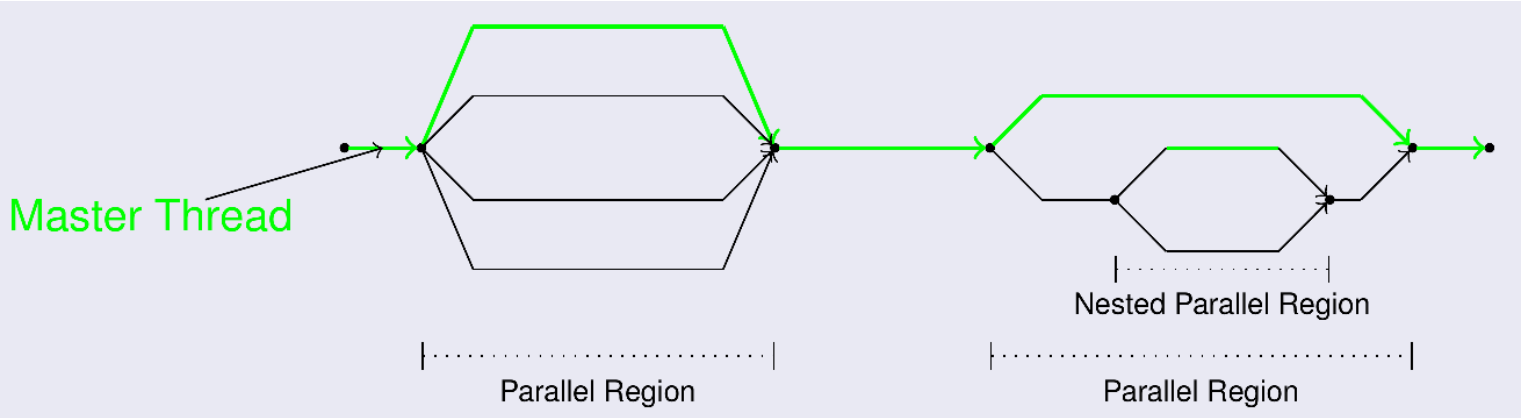
Modelo de memoria

- Los datos transferidos por la memoria compartida son transparentes a la aplicación



Paralelismo anidado

- La API permite crear regiones paralelas dentro otras regiones paralelas
- Antiguamente las implementaciones no lo soportaban aunque en la actualidad la mayoría si



Entrada/salida

- OpenMP no especifica nada respecto a E/S
  - Varios hilos pueden escribir/leer a la vez sobre el mismo fichero
  - Necesario sincronización

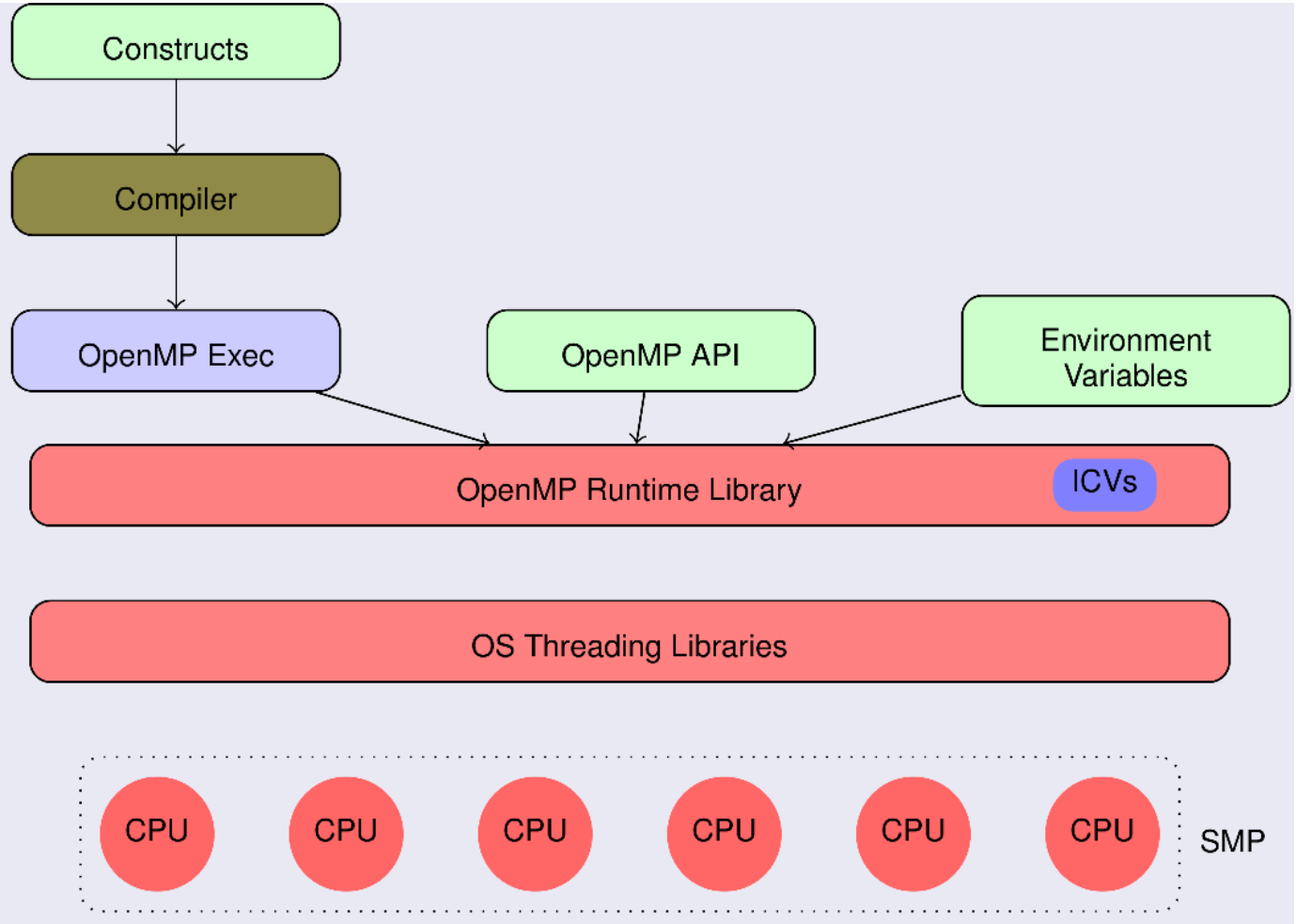
Resumen: ventajas de OpenMP

- Implementación y estándard maduro
- Buena escalabilidad (buen rendimiento)
- Es portable en múltiples arquitecturas
- Permite paralelización incremental
- Leves variaciones de la versión secuencial
- Suporta ambos modelos de paralelismo: funcional y de datos
- Comunicación implícita

Resumen: desventajas de OpenMP

- **Comunicación implícita**
- Modelo de memoria único
  - Múltiples modelos de memoria (jerarquías, GPUs...)
- Paralelización incremental crea falta sensación de *gloria/infierno*
- No funciona en *clusters* (memoria distribuida)

Componentes



Sintaxis

Fortran

- A través de comentarios con formato específico
  - `!$OMP` o `C$OMP` o `*$OMP`

C/C++

- Con una directiva de compilación
  - `#pragma omp construct [clauses]`
- La sintaxis de OpenMP se ignora si el compilador no reconoce OpenMP

Cabeceras

C/C++

- `#include <omp.h>` contiene los prototipos y tipos de datos definidos
- La macro `_OPENMP` está habilitada por el compilador de OpenMP

Bloque estructurado

Definición

- La mayoría de las directivas se aplican en un **bloque estructurado**
  - Cada bloque puede tener una o varias declaraciones
  - Todos tiene un punto de entrada y otro de salida
    - **saltos a un bloque o desde un bloque al exterior no permitidos**
  - Se permite la finalización de un programa

Construcción paralela

Especificando el número de hilos

- Se controla con una variable de control interna (ICV) llamada *nthreads-var*
- En una región paralela se crean *nthreads-var* hilos
  - Se permite anidamiento una construcción paralela dentro de otra: *nested parallelism*
- La variable *nthreads-var* puede modificarse
  - Con la llamada `omp_set_num_threads`
  - Variable entorno `OMP_NUM_THREADS`
- La clausula *num_threads* inhibe la variable *nthreads-var* en la región correspondiente

Construcción paralela

Evitando la región paralela

- A veces, se quiere ejecutar la región paralela bajo ciertas circunstancias
  - ej: poco datos de entradas, etc
- La clausula *if* permite evaluar una condición (*true/false*)y crear región paralela o que se ejecute secuencialmente
  - Nota: existe overhead porque se crea región paralela con un solo hilo

Construcción paralela

first_OMP.c

```
void main ( ) {
#pragma omp parallel
... /* An unkown number of threads here. Use OMP_NUM_THREADS */

omp_set_num_threads(2) ;
#pragma omp parallel
... /* A team of two threads here */

#pragma omp parallel num_threads(random()%4+1) if(0)
... /* A team of 1 thread here */
}
```

Llamadas a la API de OpenMP

Rutinas de interés

- `int omp_get_num_threads()` Devuelve el **número de hilos**
- `int omp_get_thread_num()` Devuelve el **id** del hilo
- `int omp_get_num_procs()` Devuelve el número de procesadores
- `int omp_get_max_threads()` Devuelve el **número máximo de hilos**
- `double omp_get_time()` Devuelve el número se segundos (reloj sistema)

Directiva `parallel`

- `#pragma omp parallel [clauses]`

Clausulas

- *num_threads*
- *if(expression)*
- *shared (var-list)*
- *firstprivate (var-list)*
- *default(none|shared|private|firstprivate)*
- *reduction(var-list)*

Ejemplo

example_OMP.c

```
#include <omp.h>
void main () {

   int var1, var2, var3;

   // Serial code
   ...

   // Beginning of parallel region. Fork a team of threads.
   // Specify variable scoping

   #pragma omp parallel private(var1, var2) shared(var3)
   {
      // Parallel region executed by all threads
      ...

      //Other OpenMP directives
      ...
   }

   // Resume serial code
   ...
}
```