Práctica 4.

Explotar el potencial de las arquitecturas modernas.

Arquitectura de Ordenadores

VÍCTOR MORENO

ESCUELA POLITÉCNICA SUPERIOR. UAM.

Contenidos

- **Arquitectura de computadores en 2013**
  - Multicomputador, Multiprocesador, Multicore
  - Hyperthreading

- **Programación paralela en sistemas *multicore***
  - POSIX threads: pthreads
  - Afinidad de un proceso
  - OpenMP

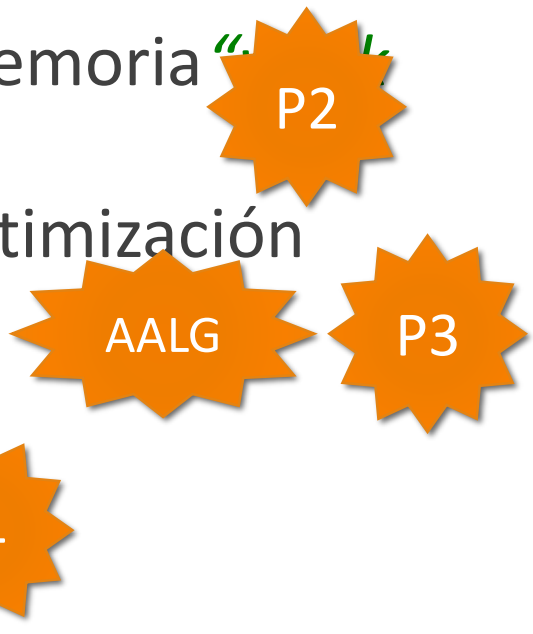
- **Práctica 4**
  - Entorno
  - Ejercicios

Motivación: HPC

- **¿Se necesita la programación paralela y la computación de altas prestaciones (HPC)?**
  - "grand challenges"
    - Simulaciones climáticas y/o geofísica (tsunami, tornados)
    - Simulaciones estructurales, flujos,…(crash test, CFD)
    - Sistemas avanzados de diseño, realidad Virtual (CAD, efectos)
    - Análisis de datos (Large Hadron Collider CERN, Carnivore,..)
    - Aplicaciones militares, médicas (crypto analysis,…)

Motivación: HPC

- **Incrementar de prestaciones con:**
  - Hardware mas rápido, más memoria *"[illegible] harder"*
  - Algoritmos más eficientes, optimización *"work smarter"*
  - Con arquitecturas paralelas *"get some help"*

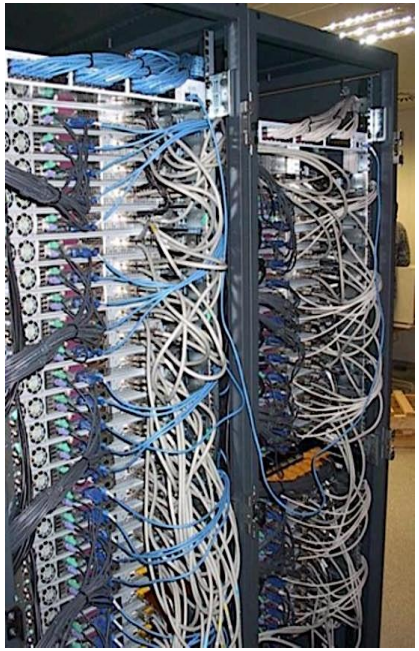


Computación paralela

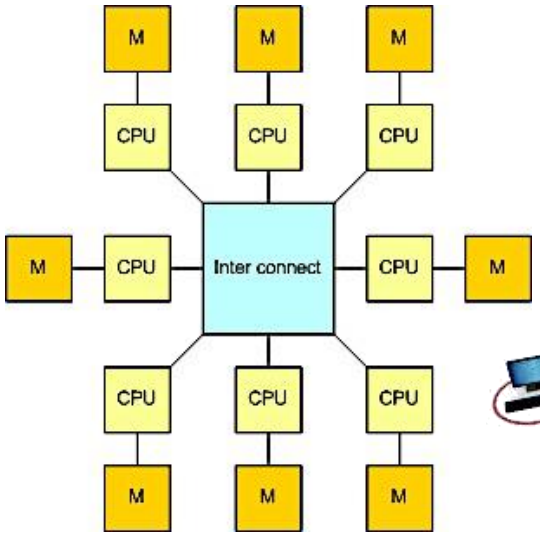
- **Hoy en día las máquinas con las que trabajamos disponen de muchas unidades de proceso interconectadas**
- **Diversos niveles de interconexión**
  - Multicomputador
  - Multiprocesador
  - Multicore
- **Mediante computación/programación paralela podemos sacar partido de estos sistemas**

Multicomputador

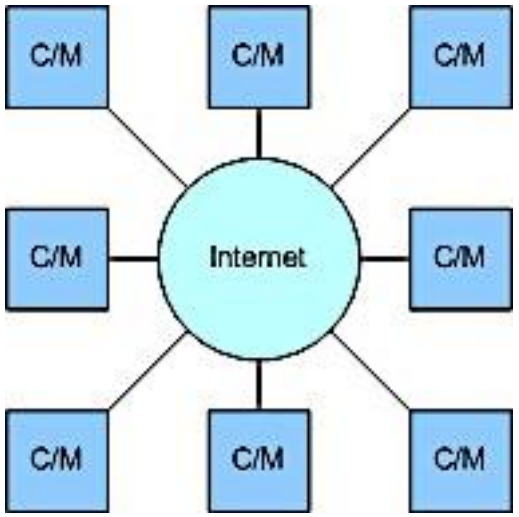
- **Diversos ordenadores conectados entre sí**
- **Comunicación basada en paso de mensajes**



- **Clúster**



- **Grid**

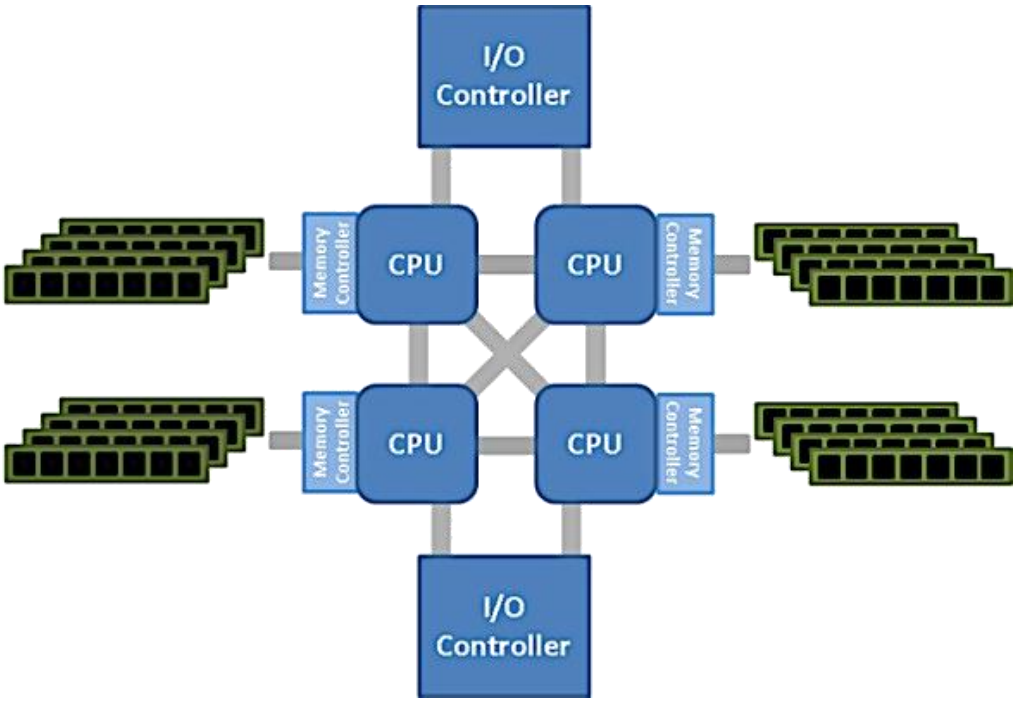


Multiprocesador

- **Ordenador con más de una unidad física de proceso (procesador/nodo) conectadas a una memoria compartida**
- **Dos tipos**
  - **UMA**

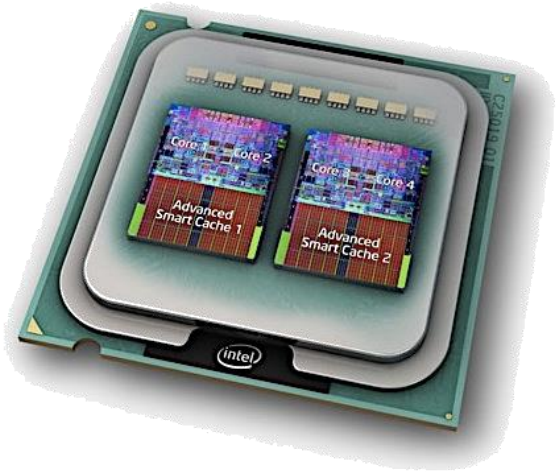
    Uniform Memory Access
  - **NUMA**

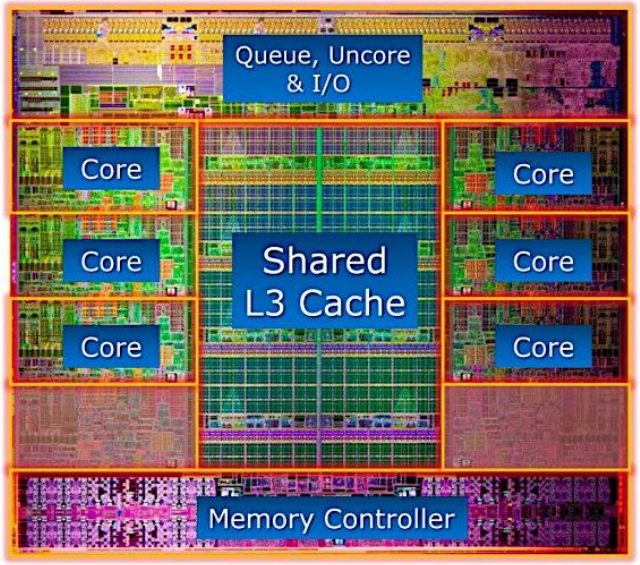
    Non-Uniform Memory Acces
- **Paralelismo**
  - **Procesos**
  - **Hilos (*Threads*)**



Multicore

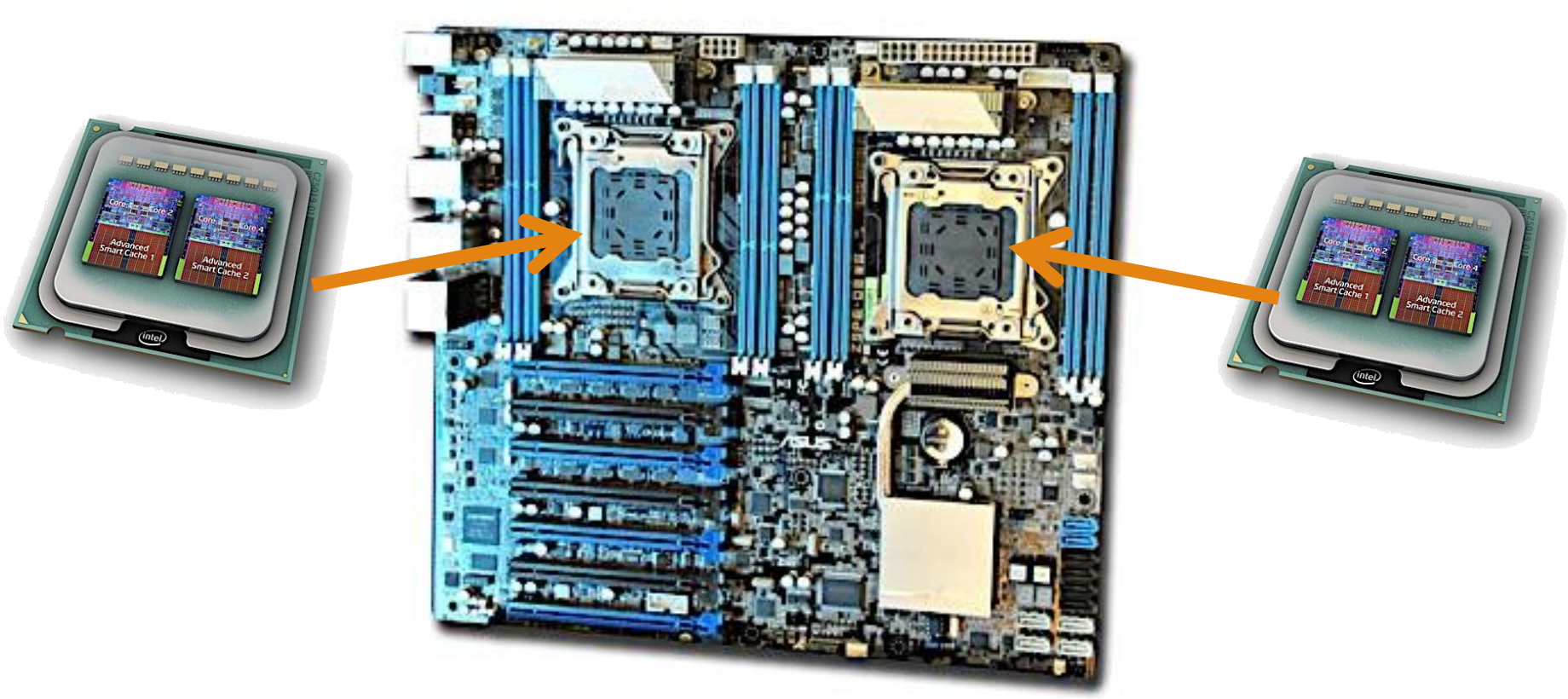
- **Un procesador *multicore* es aquel que combina en un mismo encapsulado dos o más microprocesadores independientes**
- **A cada uno de estos microprocesadores se les demonina *core* o núcleo**





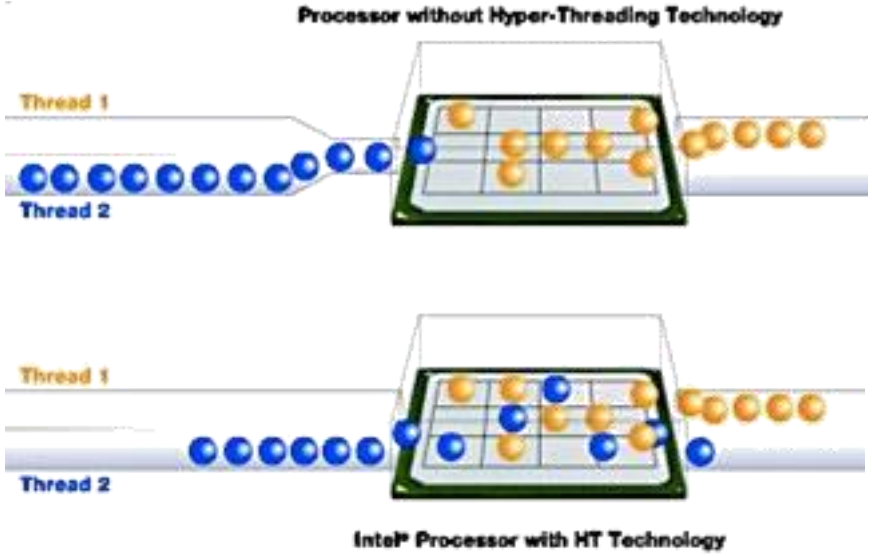
Ordenadores comunes hoy en día

- **Multiprocesador-multicore (MP-MC)**



Hyperthreading

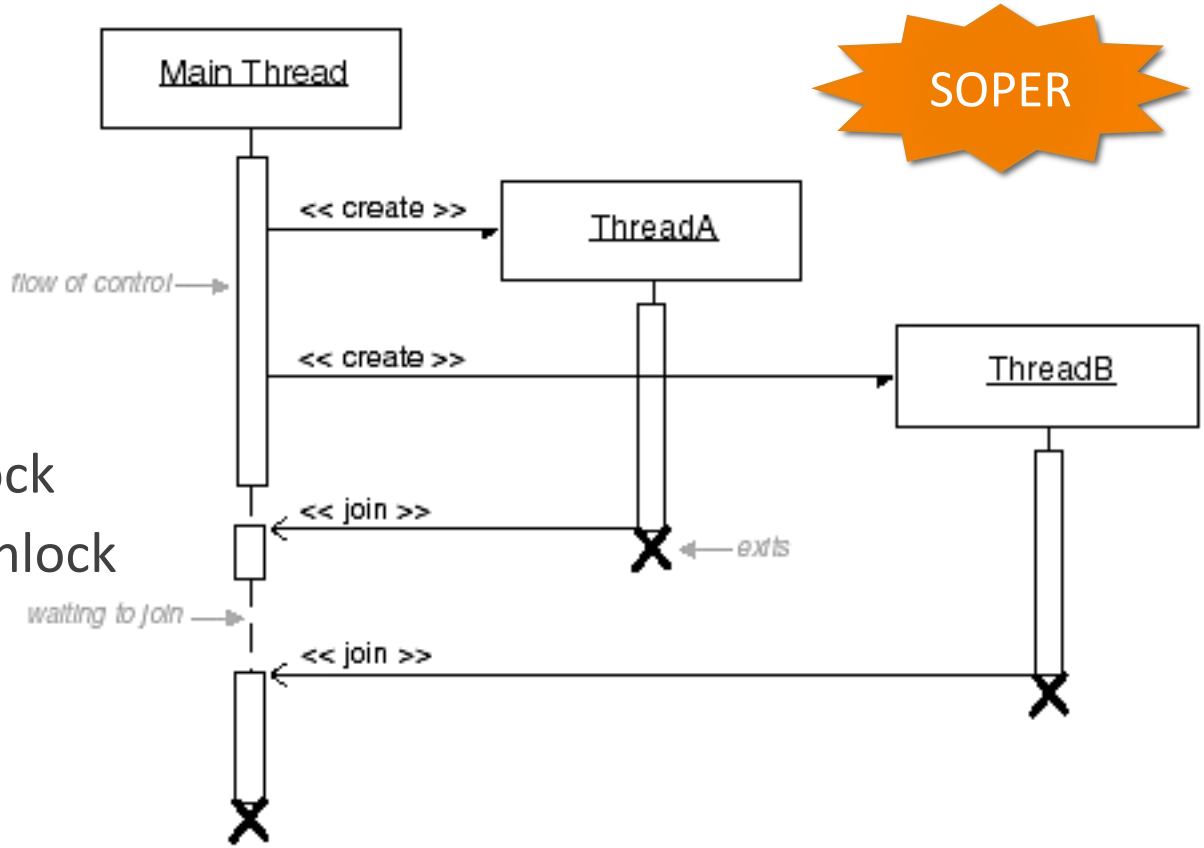
- **Cada procesador maneja dos *threads***
  - Cuando el que está ejecución se bloquea, entra el otro
  - Estructura HW para hacer un cambio de contexto del procesador (registros, …)
- **Número de procesadores x 2**
  - ¡No es real!
  - Útil en sistemas de sobremesa
    - Muchos bloqueos
  - No siempre útil en HPC
    - Nº bloqueos mínimo



Programar en sistemas MP-MC

- **Programación basada en hilos comunicados por memoria compartida**

- **POSIX threads**
  - pthread_create
  - pthread_join
  - pthread_mutex_lock
  - pthread_mutex_unlock
  - …



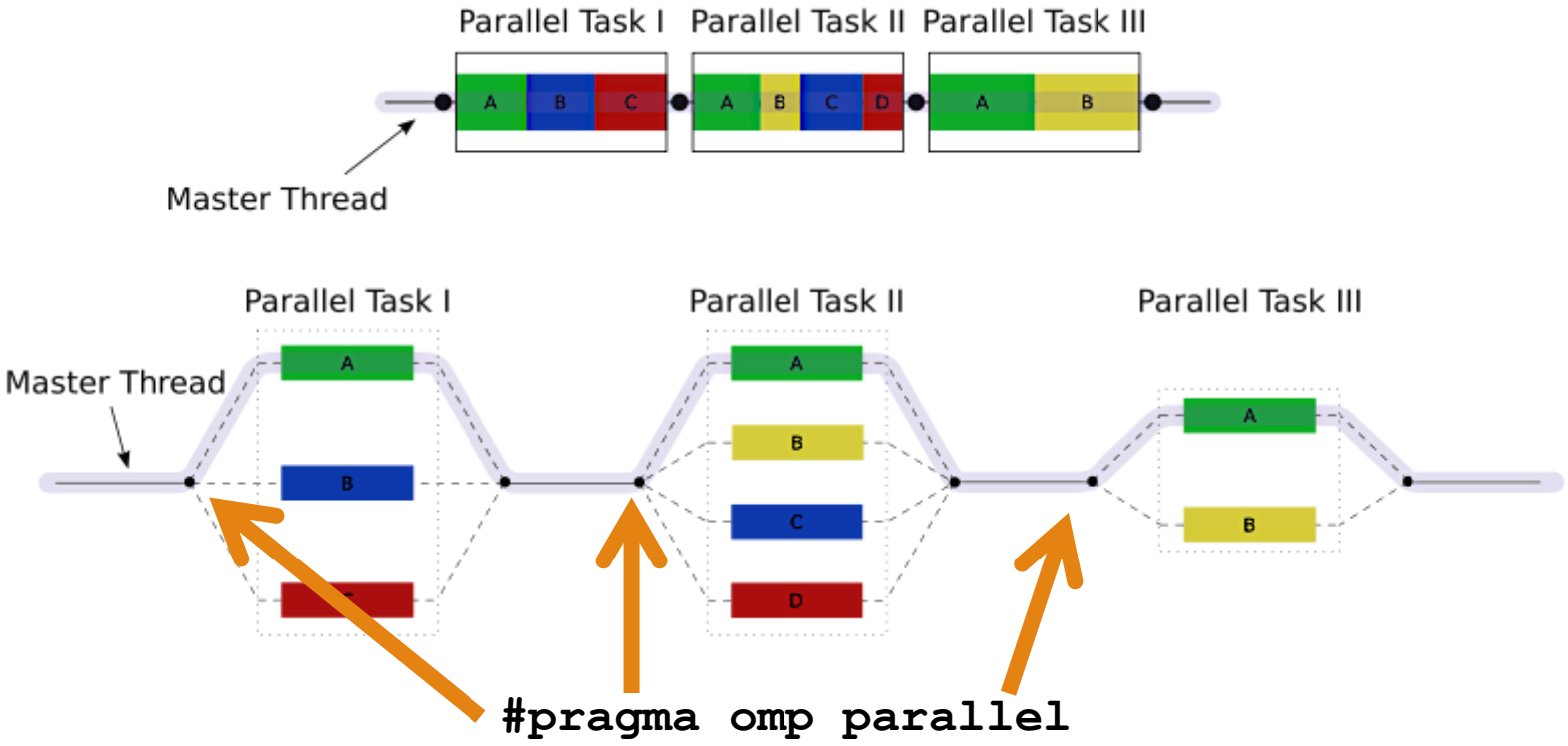
SOPER

Programar hilos

- **Asignar afinidad (*affinity*) a un proceso/hilo implica que siempre se ejecutará en el mismo *core***
  - pthread_setaffinity_np
- **Imprescible evitar que los *threads* se molesten entre ellos para obtener el máximo rendimiento**
  - no existen garantías si no asigna de forma explícita
- **Complicaciones añadidas:** reparto de trabajo, variables compartidas, …

OpenMP

- **Ofrece un API basado en *pragmas* para simplificar el manejo de hilos en sistemas MP-MC**



OpenMP: programa de ejemplo

```
1  #include <omp.h>
2  #include <stdio.h>
3
4  int main(int argc, char *argv[])
5  {
6      int tid,nthr,nproc;
7      int arg;
8
9      nproc = omp_get_num_procs();
10     printf("Hay %d cores disponibles\n", nproc);
11
12     arg = atoi( argv[1] );
13     omp_set_num_threads(arg);
14     nthr = omp_get_max_threads();
15     printf("Me han pedido que lance %d hilos\n", nthr);
16
17     #pragma omp parallel private(tid)
18     {
           tid = omp_get_thread_num();
           nthr = omp_get_num_threads();
21         printf("Hola, soy el hilo %d de %d\n", tid, nthr);
22     }
23 }
```

región paralela

Qué resuelve OpenMP

- **Creación de los equipos de hilos**
- **Sincronización al terminar la región paralela**
- **Librería de funciones para acceder a/modificar datos relacionados con el equipo de trabajo**
  - ¿Quién soy?: omp_get_thread_num()
  - ¿Con quién voy?: omp_get_num_threads()
  - Cambiar tamaño del equipo: omp_set_num_threads()
  - …
- **Generación de copias locales de las variables definidas como privadas**
- **Asignación de afinidad a cada *thread* del equipo**

Consideraciones al programar OpenMP

- **Definir correctamente la accesibilidad de las variables utilizadas**
  - public/private
  - Ejercicio 1
- **Gestionar la recogida de resultados del equipo de hilos**
  - Reduction
  - Ejercicio 2
- **Colocación más adecuada del pragma `parallel`**
  - Ejercicio 3

Cláusula `parallel`

- **Su aparición en el código significa la delimitación de una región paralela**
- **Al inicio de la región paralela, se lanzan tantos hilos como se haya indicado**
- **Al final de la región paralela, se realiza una sincronización de todos los hilos**
  - Hasta que todos hayan acabado no se continúa con el programa
- **Este lanzamiento/sincronización de hilos implica un coste en términos de tiempo de ejecución que no se puede obviar**

Cláusula `for`

- **Puede usarse dentro de una región paralela ó combinado con `parallel` antes de un bucle (que pasará a ser la región paralela)**
- **Reparte de forma automática el trabajo entre los *threads* del equipo**
  - Declara automáticamente la variable índice como privada

```
...
for(i=0;i<100;i++)
{
   ...
}
...
```

Cláusula `for`

```
#pragma omp parallel
{
   ...
   #pragma omp for
   for(i=0;i<100;i++)
   {
      ...
   }
   ...
}
```

```
...
for(i=0;i<25;i++)
{
   ...
}
...
```

Thread 0

```
...
for(i=50;i<75;i++)
{
   ...
}
...
```

Thread 1

```
...
for(i=25;i<50;i++)
{
   ...
}
...
```

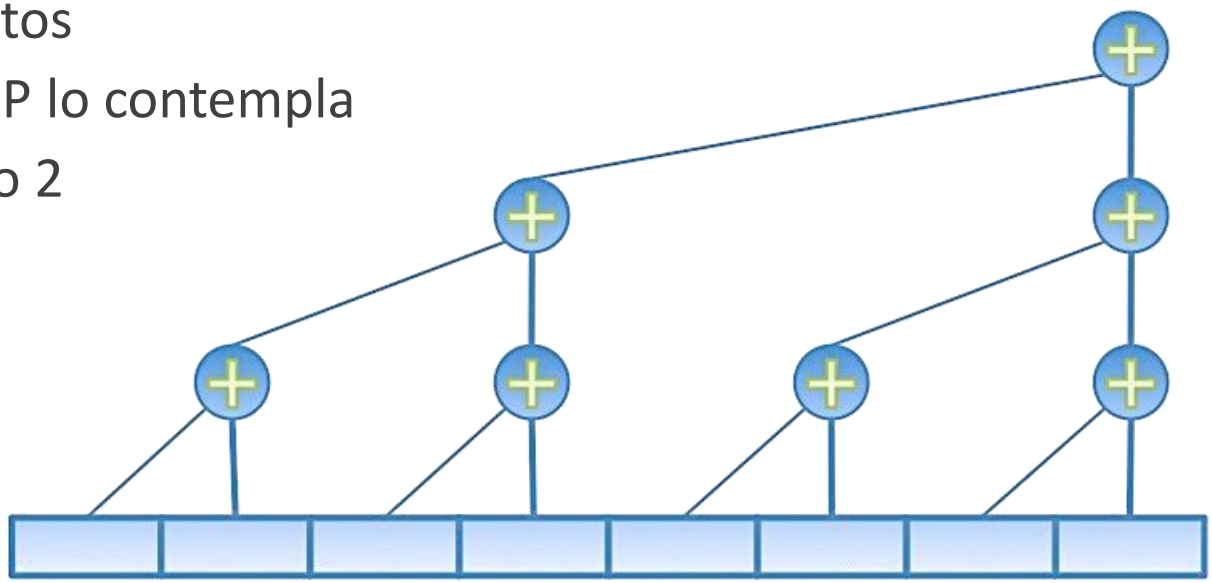
Thread 2

```
...
for(i=75;i<100;i++)
{
   ...
}
...
```

Thread 3

Reducción

- **Problema típico en programación paralela**
- **Cada hilo genera un resultado parcial, que se ha re recoger y computar para obtener un resultado final**
  - Ej: suma de N elementos con T hilos, cada hilo ha sumado N/T elementos
  - OpenMP lo contempla
  - Ejercicio 2



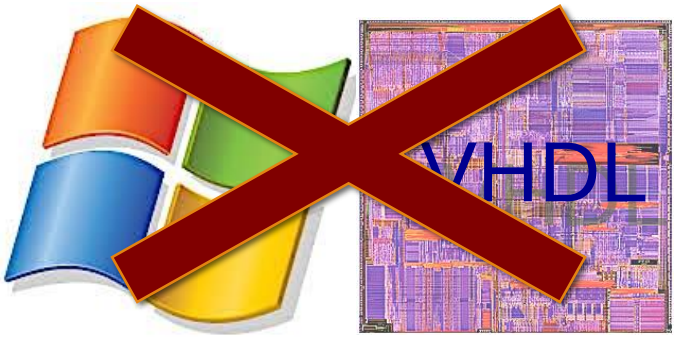
Compilar programas OpenMP

- **Muchos compiladores soportan el uso de OpenMP**
- **Vamos a utilizar el soporte de gcc para openmp**
  - Flags al compilar:
    - -lgomp : librería con la funciones
    - -fopenmp : soporte de los pragmas

- **Está incluido en el fichero `Makefile` de partida**

Práctica 4: entorno de de trabajo

- A lo largo de las prácticas 3 y 4 vamos a trabajar utilizando *software* para poner en práctica los conceptos *hardware* de teoría



Práctica 4: material de partida

- **arqo4.c/arqo4.h:** librería de manejo de matrices/vectores de números en coma flotante
- **omp1.c/omp2.c**: ejemplos de programación utilizando OpenMP
- **pescalar_serie.c**: versión serie de un programa que realiza el producto escalar de dos vectores
- **pescalar_par1.c/pescalar_par2.c**: versiones paralelas del producto escalar
- **Makefile**: fichero utilizado para la compilación de los ejemplos aportados.

Ejercicio 0

- **Identificar de qué tipo es el PC del laboratorio**
  - ¿Multiprocesador?
    - #procesadores
  - ¿Multicore?
    - #cores
  - ¿Hyperthreading?

```
cat /proc/cpuinfo
```

Ejercicio 1

- **Ejecutar y comprender ejemplos básicos de OpenMP**
  - Lanzamiento de un equipo de hilos
  - Utilización de las funciones básicas de OpenMP
  - Declaración de variables privadas/públicas

- **Contestar a las preguntas de la memoria**

Ejercicio 2

- **Producto escalar de dos vectores**
  - Dos versiones paralelas
  - ¿Funcionan correctamente?
  - Análisis del rendimiento al variar parámetros
    - Tamaño del vector
    - Número de hilos
  - Contestar a las preguntas de la memoria

Ejercicio 3

- **Multiplicación de matrices**
  - ¿Qué bucle es mejor paralelizar?
  - Análisis de rendimiento
  - Contestar preguntas de la memoria

Práctica 4: entrega

IMPORTANTE:

- En dos semanas (del 5 al 9 de Diciembre) no hay clase. **SOLO tutorías el miércoles 7.**

- **Entregas:**
  - **Grupos Miércoles –** hasta el martes 13 de Diciembre de 2016 a las 23:59.
  - **Grupos Jueves –** hasta el miércoles 14 de Diciembre de 2016 a las 23:59.

Prácticas 3 y 4: examen

- **Presencia OBLIGATORIA!**
  - **Miércoles 14 de diciembre**
  - **Jueves 15 de diciembre**

Encuestas en SIGMA

- Profesor de prácticas
- Profesor de teoría
- Asignatura

- SIGMA
  - Encuestas Web SIGMA
    - Rellenar encuestas