Metodología de la Programación Paralela

Facultad Informática, Universidad de Murcia

Introducción a la Computación Paralela

Bibliografía básica

- Del curso, capítulos 1 a 6
- De esta sesión, capítulo 2

Contenido

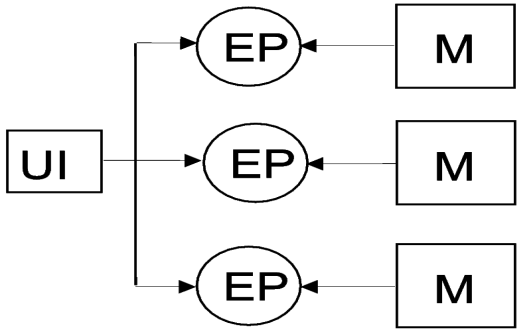
Clasiflicación de Flynn

SD=*Single Data*

MD=*Multiple Data*

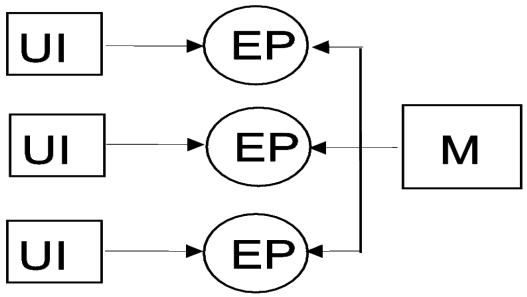
SI=*Single Instruction*

SISD

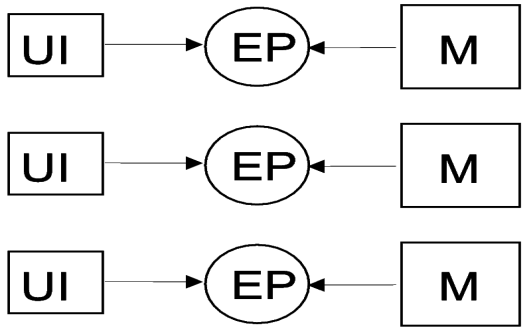


SIMD

MI=*Multiple Instruction*



MISD



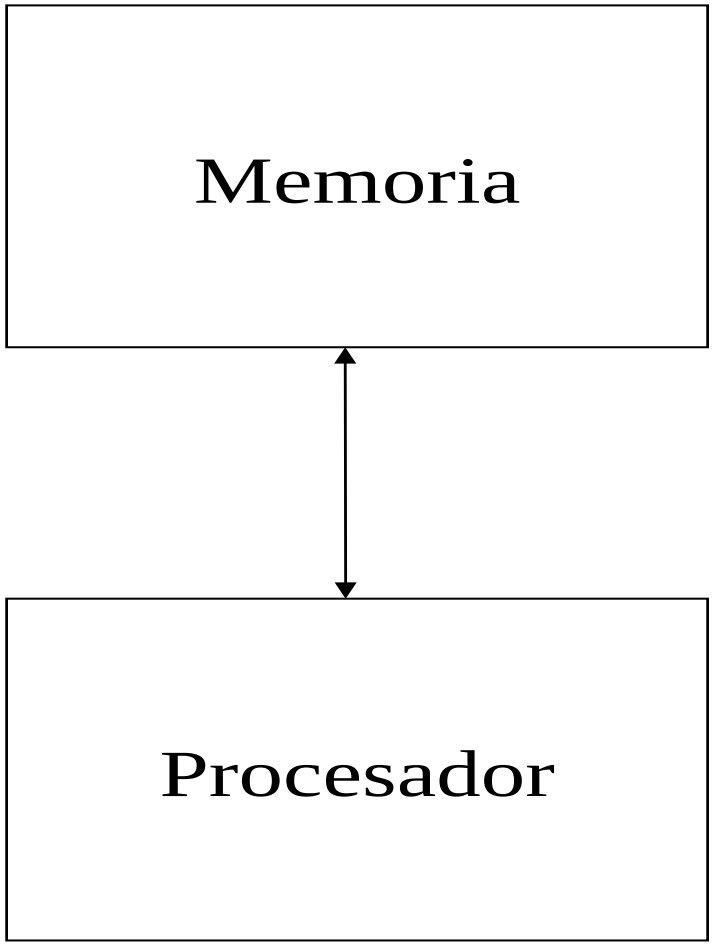
MIMD

EP=Elemento de proceso; M=Memoria; UI=Unidad de instrucción

Secuencial - SISD

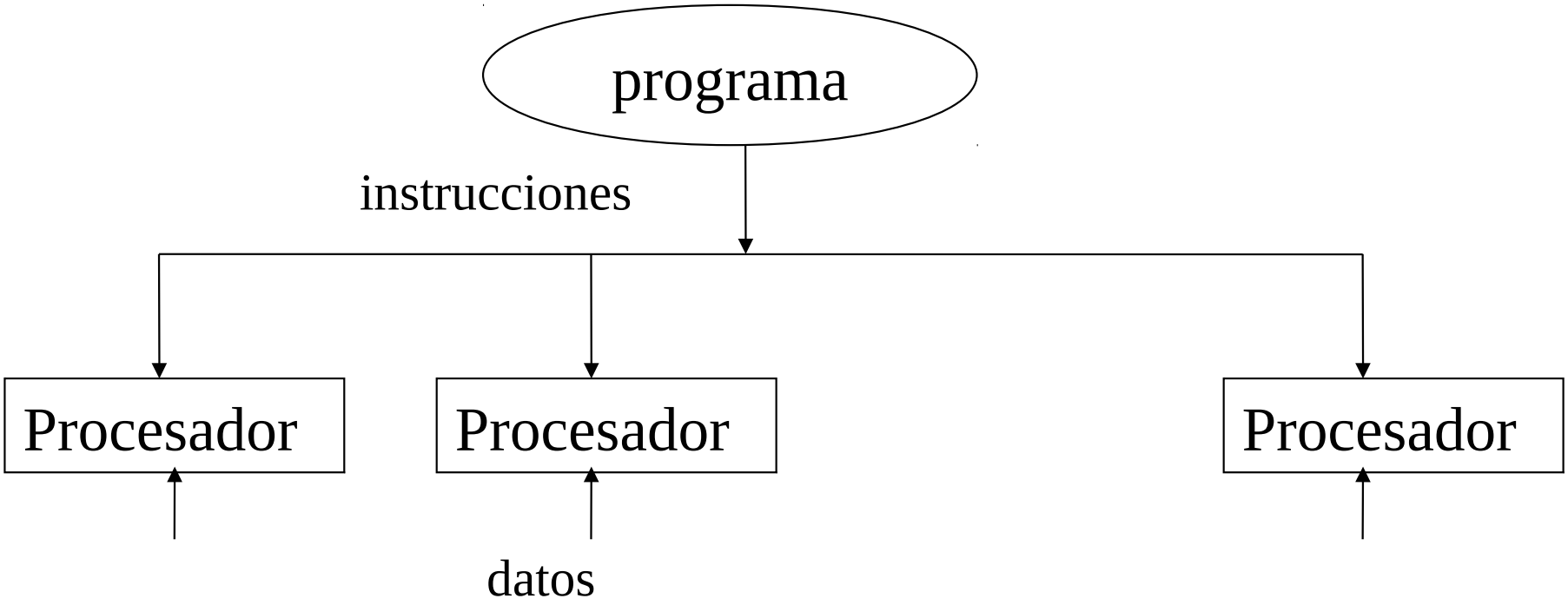
Modelo Von Neuman

- Instrucciones
  de memoria a procesador
- Datos
  entre memoria y procesador



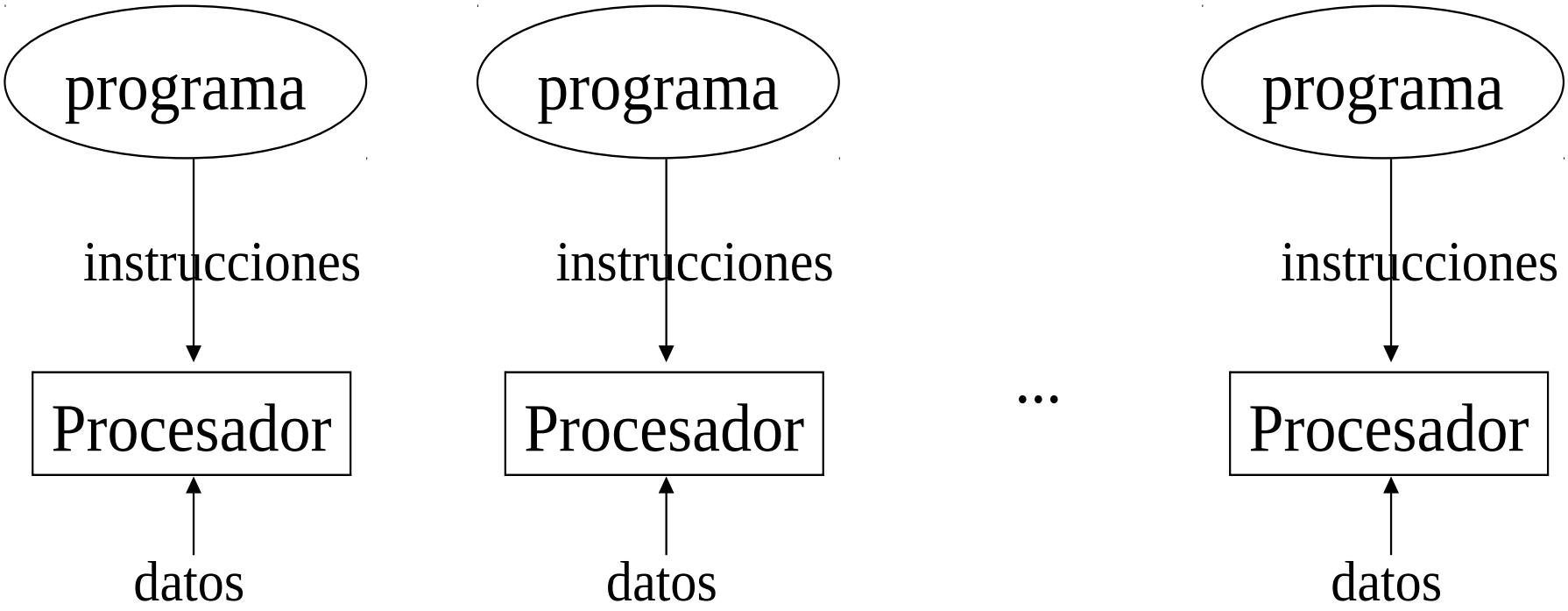
SIMD

Una única unidad de control.
La misma instrucción se ejecuta síncronamente por todas las unidades de procesamiento.
Normalmente se consideran en este paradigma las GPU.



MIMD

Cada procesador ejecuta un programa diferente independientemente de los otros procesadores.
Es el **modelo** que usamos **en esta asignatura**



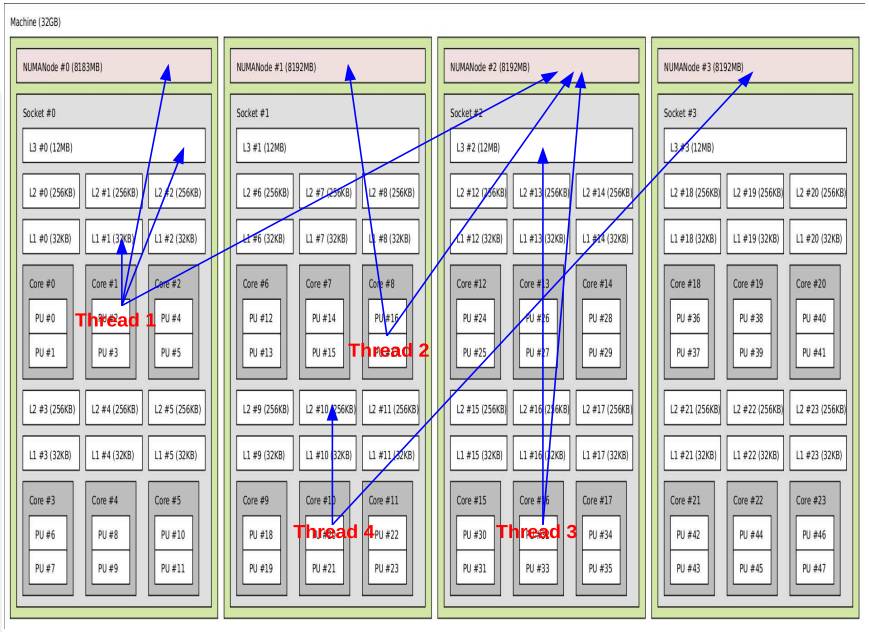
Consideramos tres paradigmas básicos que se pueden considerar los **estándares actuales**:

- Programación con Memoria Compartida.
- Programación con Paso de Mensajes.
- Simple Instrucción Múltiple Dato (SIMD).

MC- Sistemas

- Los sistemas donde se realiza permiten ver la memoria compartida por los distintos elementos de computación: los distintos procesos o hilos, independientemente del procesador o núcleo donde estén, tienen acceso directo a todas las posiciones de memoria.
- Típicamente son sistemas multicore, con memoria dividida en bloques y organizada jerárquicamente, pero en el programa se considera la memoria común, aunque habrá distinto coste de acceso a los datos dependiendo de dónde se encuentren.

Imagen de *saturno*, con `hwloc`

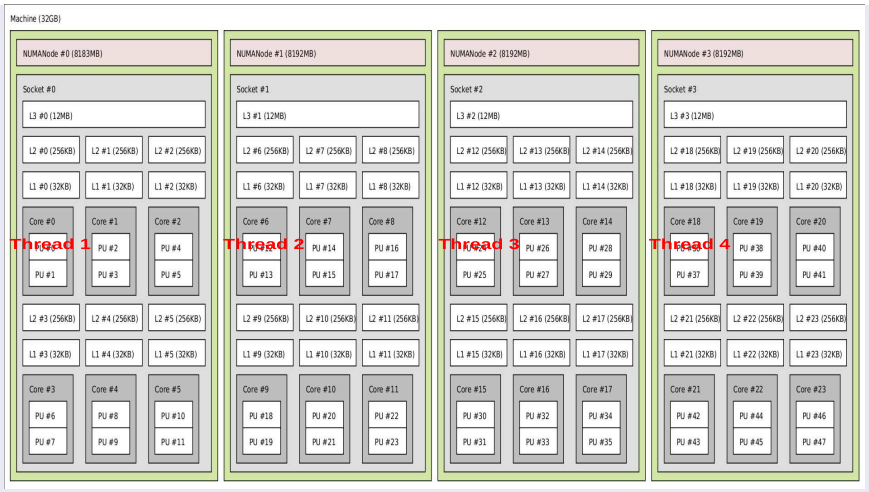


Problemas de:
coherencia de datos (resuelto por el sistema operativo)

y contención (empeoran el tiempo de ejecución).

MC- Hilos

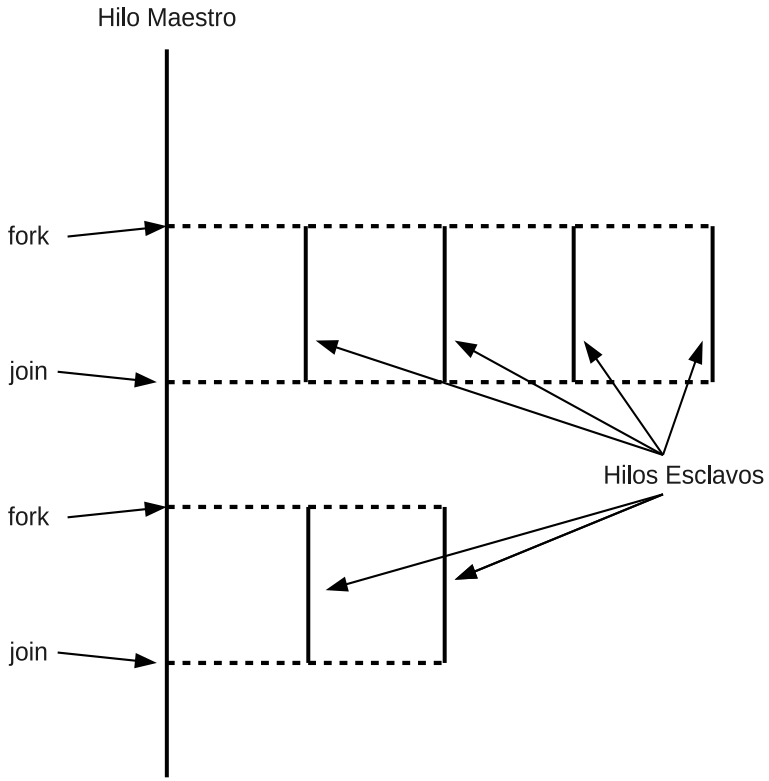
- Los elementos lógicos de computación son los hilos (*threads*).
- Se asocian a elementos físicos de computación, que son los núcleos (*cores*).
- La asociación hilos-cores se puede realizar de distintas formas (afinidad).



MC- Modelo programa

Modelo fork-join:

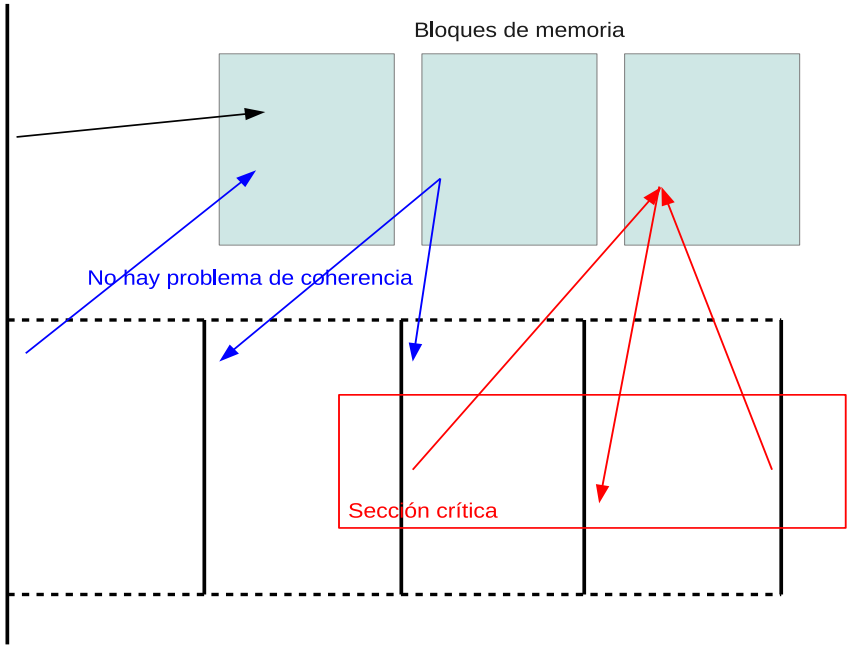
- un hilo genera nuevos hilos (fork)
- y los espera para sincronizarse (join)



MC- Regiones críticas

Cuando los hilos acceden a zonas comunes de datos para variarlos es necesaria sincronización. Los entornos de programación en memoria compartida proporcionan herramientas:

- Sección crítica: sólo un hilo puede estar ejecutando esa parte del código en un momento dado.
- Llaves, semáforos: se puede acceder a esa zona de código cuando se cumple una condición.



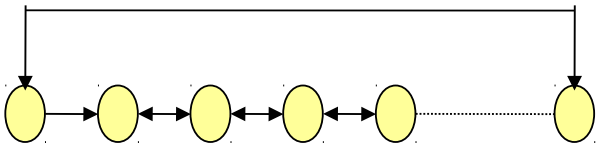
Entornos de programación en Memoria Compartida

- C, C++: tienen llamadas a rutinas del sistema, fork, join y otras.
- Java: tiene bibliotecas de concurrencia.
- Pthreads: interface de programación (API) para trabajo con hilos.
- OpenMP: especificación de API para programación paralela en Memoria Compartida. Se puede considerar el estándar para computación en MC.
  Se encuentra en implementaciones de lenguajes, como `gcc`.

PM- Sistemas

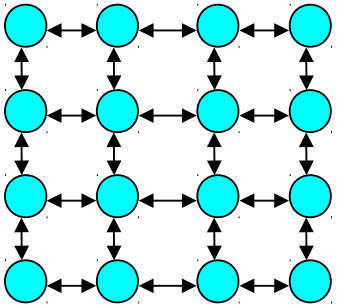
- En los sistemas donde se realiza esta programación no se puede acceder a todas las posiciones de memoria desde todos los elementos de proceso.
- Cada nodo o procesador tiene asociados unos bloques de memoria, a los que puede acceder directamente.
- Un proceso asignado a un procesador, para poder acceder a datos en bloques de memoria no accesibles desde ese procesador, tiene que comunicarse con procesos en procesadores a los que está asociada esa memoria.
- Son redes (*clusters*) de ordenadores, formados por nodos multicore conectados en red, sistemas distribuidos..., pero también se puede usar programación por paso de mensajes en sistemas de memoria compartida.

PM- Topologías de red



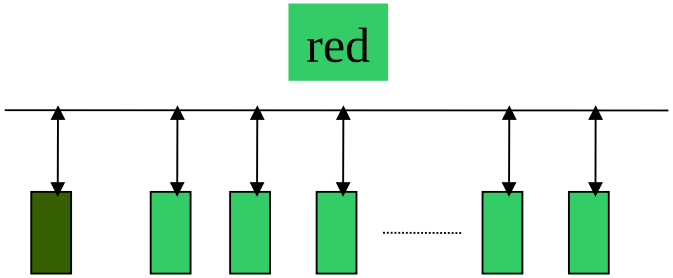
anillo

Diámetro: p/2



Malla

Diámetro: √p



Hipercubo

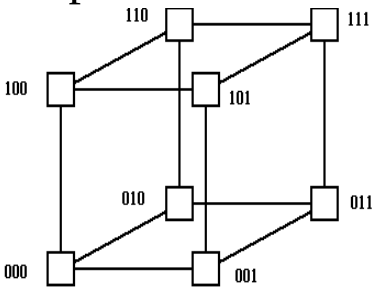


Figure 1.10 Three-dimensional hypercube

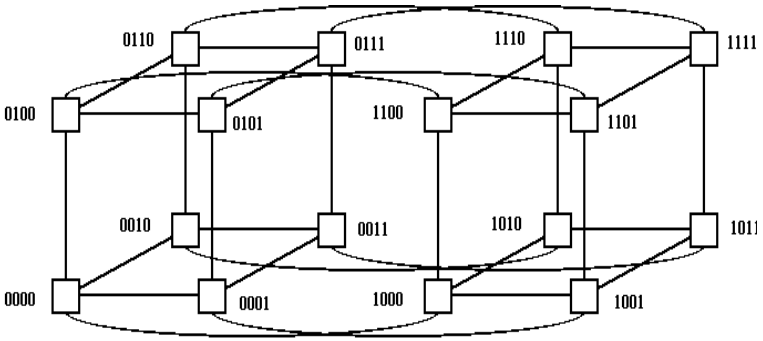


Figure 1.11 Four-dimensional hypercube

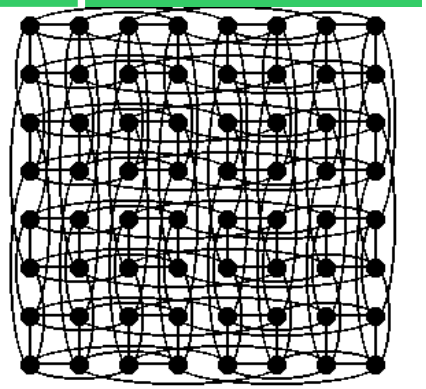
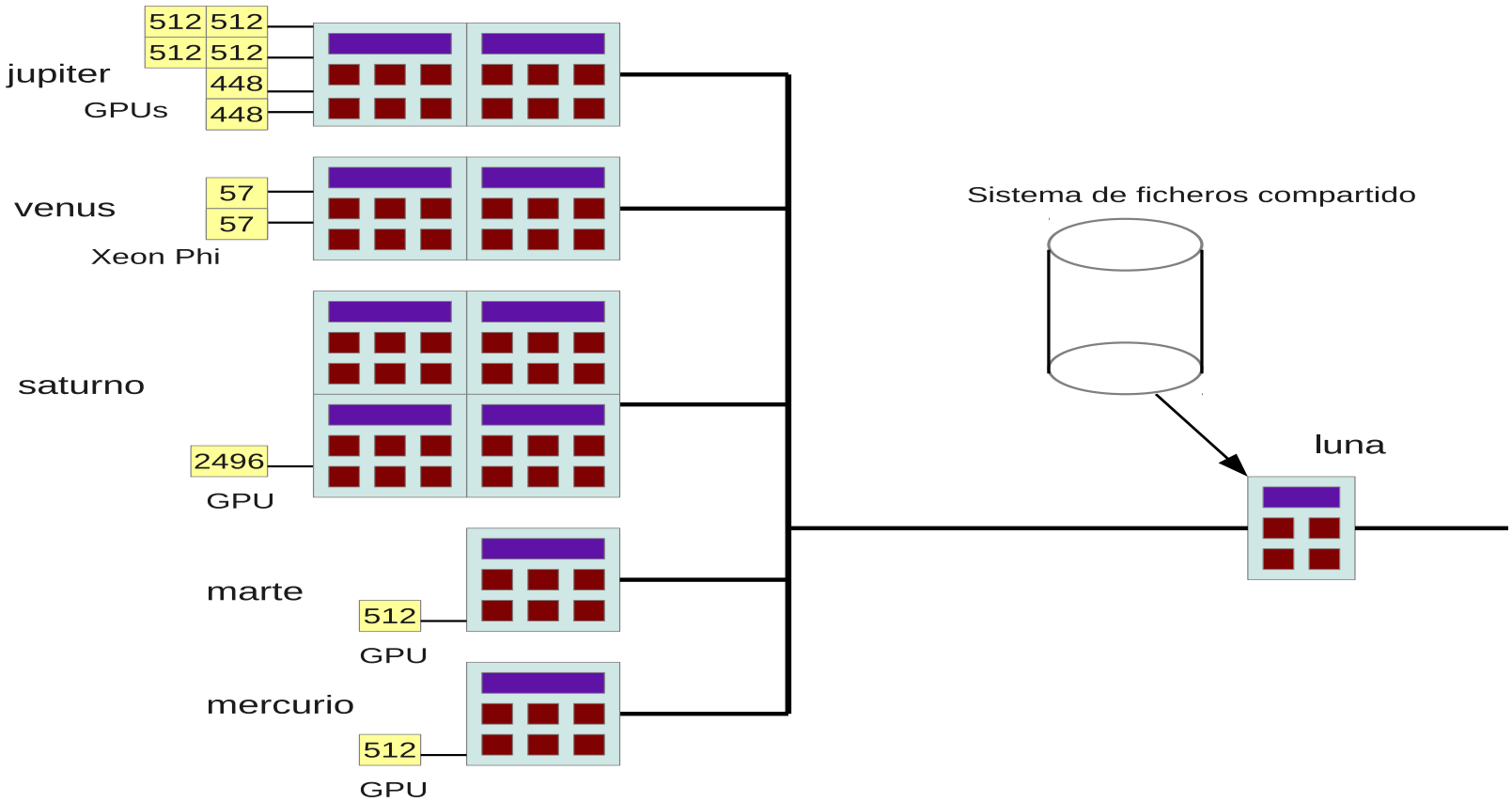


Figure 1.12 Six-dimensional hypercube laid out in one plane

Para programación puede ser más importante la **topología lógica de procesos** que la física de procesadores.

PM- Cluster del laboratorio de CCPP



Consultar información en `luna.inf.um.es/grupo_investigacion` (no actualizado)

PM- Procesos

Programa:

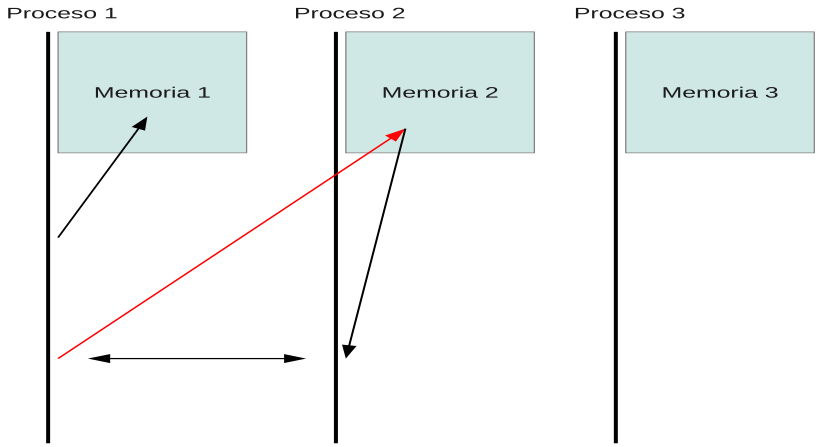
- Puede haber un único programa y ponerse en marcha varios procesos con el mismo código. Modelo Simple Programa Múltiple Dato (**SPMD**).
- Aunque sea el mismo programa los códigos que se ejecutan pueden ser distintos si se compila para arquitecturas distintas.
- Puede haber varios programas y generarse procesos con códigos distintos.

Generación de procesos:

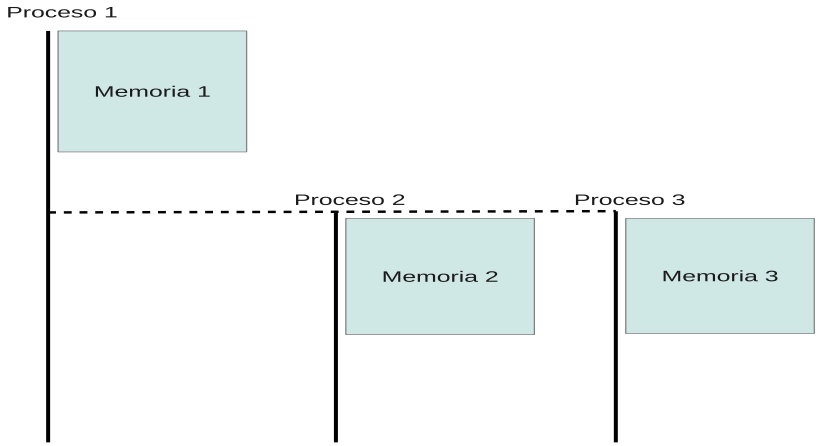
- Generación estática: todos los procesos se ponen en marcha al mismo tiempo.
- Generación dinámica: unos procesos ponen en marcha otros durante la ejecución.

PM- Memoria

- Generación estática



- Generación dinámica



PM- Mensajes

Los procesos se comunican con mensajes, que pueden ser:

- Según el número de procesos:
  - Punto a punto: un proceso envía y otro recibe.
  - Globales: intervienen varios procesos, posiblemente uno enviando o recibiendo datos de todos los demás.
- Según la sincronización:
  - Síncronos: los procesos que intervienen se bloquean hasta que se realiza la comunicación.
  - Asíncronos: los procesos no se bloquean. El que envía manda los datos y sigue trabajando, el que recibe, si no están disponibles los datos continúa con su trabajo.

Entornos de programación por Paso de Mensajes

- Java: tiene bibliotecas de paso de mensajes.
- MPI (Message Passing Interface): especificación de API para programación con Paso de Mensajes. Se puede considerar el estándar para computación en sistemas distribuidos.
  API para varios lenguajes: C/C++, Fortran...
  Varias implementaciones gratuitas: MPICH, LAMMPI, **OpenMPI**...

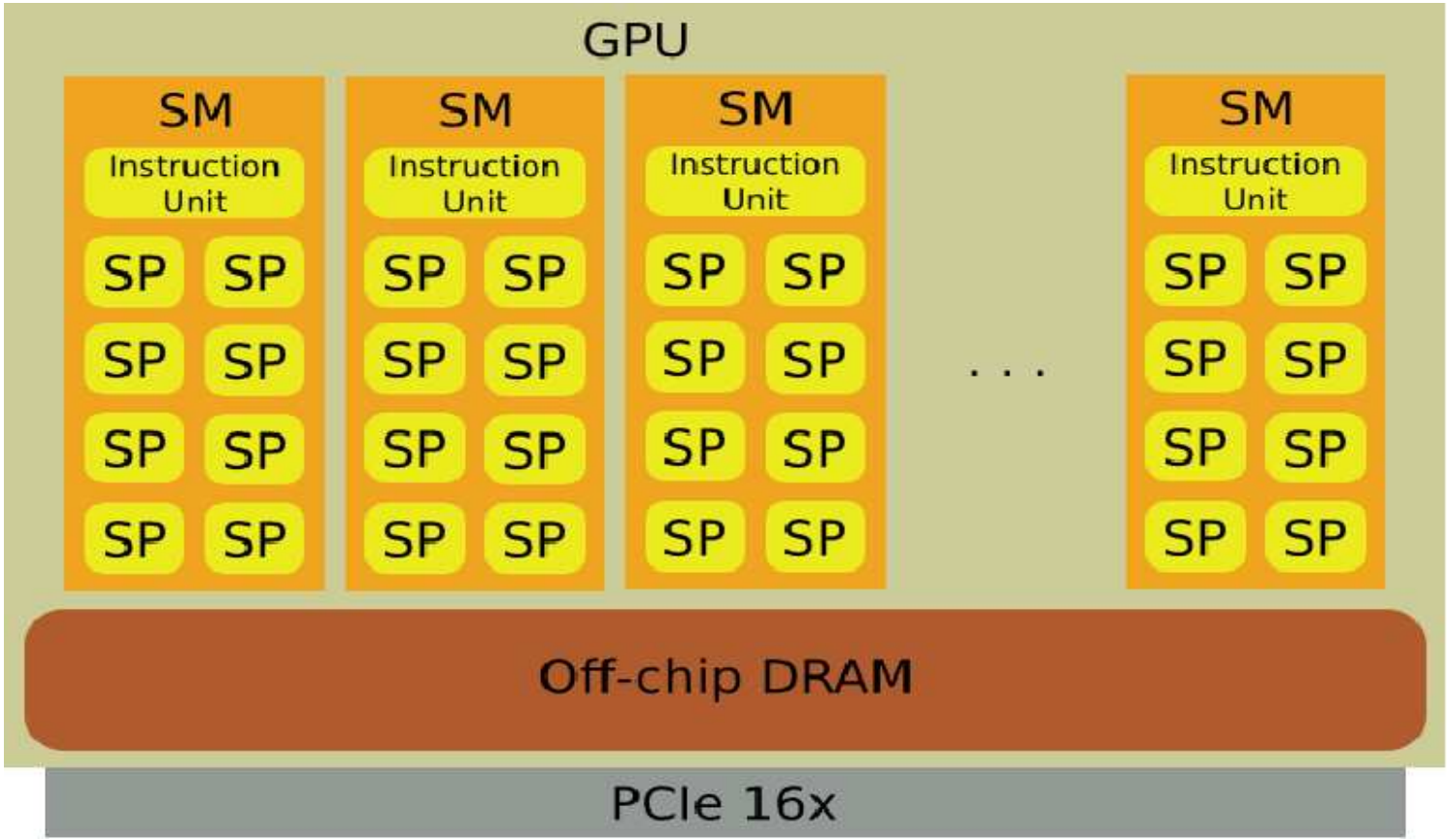
SIMD- Sistemas

- Modelo SIMD: muchas unidades de proceso, cada una realizando la misma operación (SI) y cada una sobre sus datos (MD).
- En la actualidad podemos considerar sistemas SIMD coprocesadores como:
  - tarjetas gráficas (GPU), hasta $\approx$ 2500 cores. En los sistemas para gráficos, se pueden programar para propósito general.
  - Intel Xeon Phi, entre 57 y 61 cores, cada uno hasta 4 threads por hardware.

  pero también hacen computación de forma asíncrona, normalmente trabajo en CPU y se manda parte del procesamiento al coprocesador.

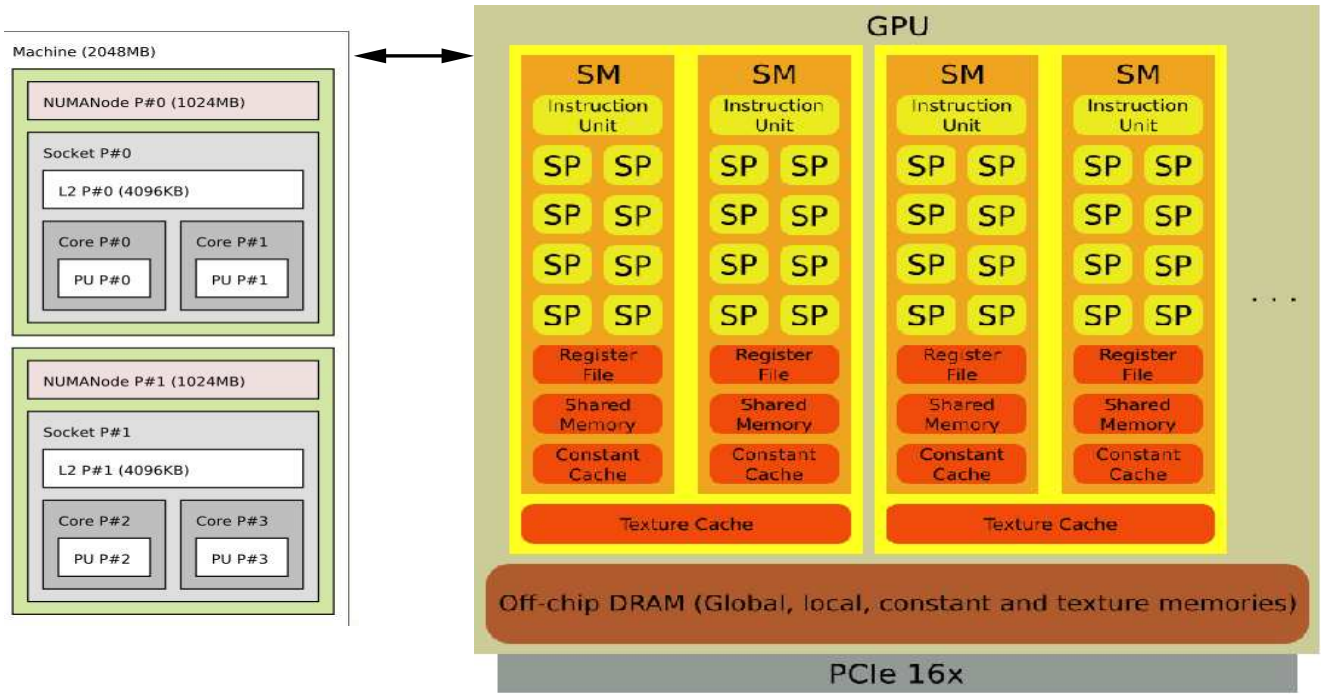
GPU- Estructura

Constan de varios Streaming Multiprocessors (SMs), cada uno con varios Streaming Processors (SPs):



GPU- Memoria

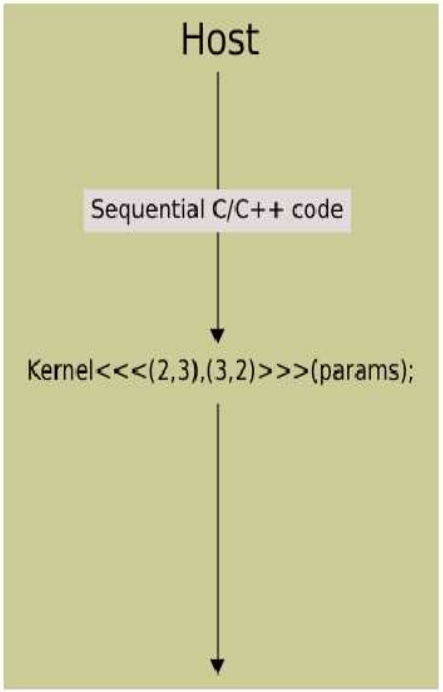
- Memorias independientes en CPU y GPU.
- Con jerarquía de memoria en cada una.
- Más compleja la de GPU: optimizar su uso para tener buenas prestaciones.
- Y necesario minimizar copias entre memorias de CPU y GPU, o solapar las copias con computación.

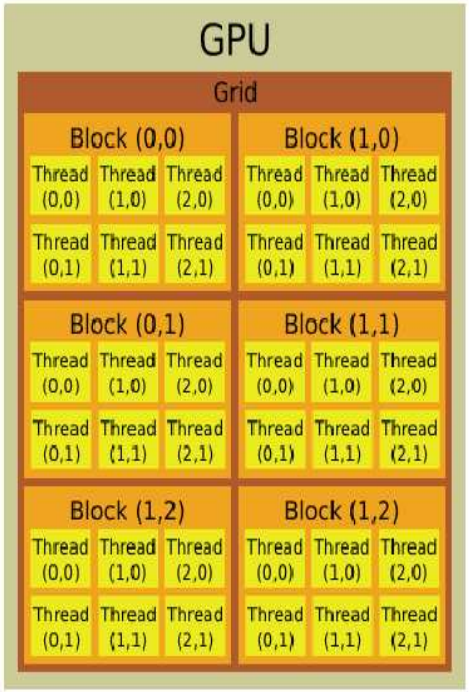


GPU- Modelo programación

- Los procesadores (SPs) de un SM ejecutan hilos independientes, pero en cada instante ejecutan la instrucción leída por la Instruction Unit (IU).
- En cada SM los hilos los gestiona el hardware: bajo coste.
- kernel es la parte de código en la CPU que lanza ejecución a GPU:

- Descompone un problema en subproblemas y lo mapea sobre un grid, que es un vector 1D o 2D de bloques de hilos.
- Cada bloque es un vector 1D, 2D o 3D de hilos.
- Los hilos usan su identificador de thread dentro de un bloque y de bloque dentro del grid para determinar el trabajo que tienen que hacer.





Entornos de programación de sistemas SIMD

- Programación de GPU es programación específica:
  - CUDA para tarjetas NVIDIA
  - OpenCL es estándar para tarjetas de diferentes fabricantes.
- En Intel Xeon Phi se puede usar OpenMP y MPI,
  con compilación diferenciada para CPU y para el Xeon Phi,
  pero
  pudiendo trabajar de forma conjunta.

Vemos ejemplos básicos de programación paralela en:

- Java
- C/C++ con fork-join
- Pthreads

quizás se trabaje con alguno de ellos u otros similares en la sesión práctica no evaluable del 27 de septiembre.
Otros entornos se verán en las sesiones siguientes:

- OpenMP
- MPI
- CUDA

El programa `ejemplojava.java` se compila con:
`javac ejemplojava.java`
se ejecuta `java ejemplojava X`, con `X` el número de datos a ordenar.
Revisar estas clases y métodos de Java para la sesión de prácticas:

- Se crea una clase (threadordenar) que implementa **Runnable**.
- el método `run` llama a ordenar la segunda mitad del array.
- Los datos y su tamaño se declaran globales para que puedan acceder los dos hilos.
- Se declara un hilo de esa clase: `Thread t = new Thread(new threadordenar());`
- y se inicia su ejecución con `t.start()`.
- El hilo maestro espera mientras el hilo esclavo está activo: `t.isAlive()`.

El programa `ejemplofork.cpp` se compila con:

```
gcc -O3 ejemplofork.cpp -o ejecutable
```

se ejecuta `ejecutable` y se introduce el número de datos.

Revisarlo para la sesión de prácticas:

- Un proceso puede crear un proceso hijo con la función **fork**.
- fork devuelve al proceso que lo llama el identificador del proceso hijo, y al hijo el valor cero.
- El proceso hijo tiene una copia de las variables del proceso padre.
- y ejecuta el mismo código del padre a partir de la zona en que se ha creado.
- El padre puede usar la función **wait** para esperar que el hijo acabe.
- Para que padre e hijo trabajen con datos compartidos hay que utilizar funciones de compartición de memoria, como por ejemplo la **mmap**.

El programa `ejemplopthreads.cpp` se compila con:
`gcc -O3 ejemplothreads.cpp -o ejecutable -lpthread`
se ejecuta `ejecutable` y se introduce el número de datos.
Revisar las siguientes funciones y tipos para la sesión de prácticas:

- Se usa la librería `pthreads.h`
- Se declara un array de hilos:
  `pthread_t threads[NUM_THREADS]`
- Los hilos se ponen en marcha con
  `pthread_create`
- que recibe la dirección del hilo (`&threads[i]`), la dirección de memoria de la función que ejecuta el hilo, y la dirección de la estructura que contiene los parámetros que se pasan a la función (`(void *) &thread_data_array[i]`)
- Se espera a que acaben los hilos con
  `pthread_join(threads[i], &status)`

Sesiones de la semana próxima

- De teoría:
  Programación de memoria compartida, OpenMP.
  Consultar la parte correspondiente del capítulo 3 del libro de IPP.
- Prácticas:
  Sesión no evaluable sobre entornos de programación paralela.