Tema 1: Evolución de la arquitectura de los sistemas paralelos y de sus modelos de programación

Modelos de Programación

Asignatura :Arquitectura de Sistemas Paralelos

Profesor: Francisco Javier Gómez Arribas
Departamento de Tecnología Electrónica y de las Comunicaciones

UNIVERSIDAD AUTONOMA DE MADRID

Escuela Politécnica Superior

Contenidos

- Modelos de Programación Paralela
  - Características distintivas de cada modelo
  - Unidades de Ejecución: Threads o Procesos
- Programación basada en paso de mensajes (MPI).
- Programación en sistemas de memoria compartida.
  - Modelo de programación (OpenMP).
  - Modelo de Hilos (Posix Threads)
- Modelo de programación CUDA.

Modelos de Programación

Arquitectura paralela:

- Arquitectura de computadores: Conjunto de instrucciones + organización
- Arquitectura de comunicaciones: Conjunto de instrucciones de comunicación y sincronización + organización

Independencia:

Modelo de Programación
≠
Arquitectura

Tipos de Paralelismo

- ¿Qué es el paralelismo y dónde está?
  - PARALELISMO: Posibilidad de ejecutar varias acciones simúltaneamente con el objetivo de incrementar el trabajo realizado y disminuir el tiempo de ejecución.
    - A nivel de programas
    - A nivel de subrutinas

    → PARALELISMO GRANO GRUESO

    - A nivel de bucles
    - A nivel de sentencias

    → PARALELISMO GRANO FINO

- Qué es un programa paralelo?

Son programas que usan más de un elemento de proceso.

Tipos de paralelismo a explotar

Paralelismo de datos

```
do i = 1, 3000
  A(i) = func(i)
enddo
```

P0

```
do i = 1, 1000
A(i) = func(i)
enddo
```

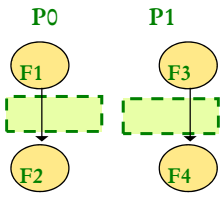
P1

```
do i = 1001, 2000
 A(i) = func(i)
enddo
```

P2

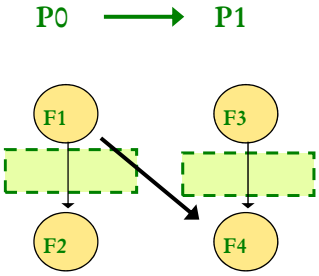
```
do i = 2001, 3000
A(i) = func(i)
 enddo
```

Paralelismo de tareas o de función



Paralelismo con dependencias

¿ Cómo tratar la dependencias entre tareas?



Sincronización

- global (barreras)
- punto a punto (eventos)

Paralelismo

¿Qué paralelismo utilizar? Elegir arquitectura y su programación

- Programación en Memoria Compartida (OpenMP, pTthreads)
  - Explota el paralelismo de grano fino
  - Fácil programación. Paralelismo extraído por el compilador
  - Poca escalabilidad (O(16) procesadores)
  - Requieren un hardware más costoso si el número de procesadores es grande.

- Programación en Memoria Distribuida (Paso de Mensajes)
  - Explota el paralelismo grano grueso
  - Programación compleja
  - Usualmente mayor aprovechamiento de la localidad
  - Alta escalabilidad
  - Alta portabilidad => Se puede ejecutar el código en plataformas hardware sencillas

Modelos de Programación

Memoria Compartida (MC)

- Varios procesos secuenciales ejecutándose en un espacio de direcciones común
- Comunicación implícita en acceso (lectura o escritura) a memoria compartida (MC)
- Sincronización ≡ escritura en MC
- Paso de ejecución de procesos (pesados) a → hebras o *threads* (ligeros)

Paralelismo de Datos (PD)

- Computación científica masivamente paralela SIMD
- Compiladores paralelos → código vectorial
- En MIMD → mediante el modelo SPMD (*Single Program Multiple Data*)
- Las operaciones son ejecutadas en paralelo sobre colecciones de datos estructurados
- Lenguajes: F90+ librerías , HPF

Modelos de Programación

Paso de Mensajes (PM)

- Procesos independientes → interacción sólo mediante envío/recepción de mensajes
- Proceso → sólo memoria local o de acceso privado
- Librerías: PVM (Parallel Virtual Machine), MPI (Message Passing Interface)
- Problemas de encapsulado

Orientado a Objetos (OO)

- Lenguajes Orientados a Objetos (OO): Java, Smalltalk, C++, etc.
- Lenguajes OO con extensiones para paso de mensajes
- Lenguajes OO con modelo de MC mediante hebras (threads) y paso de mensajes: Java + Corba

Entidades de Ejecución

Procesos (proceos pesado)

- El programa se divide en procesos que se ejecutan concurrentemente.
- Cada proceso tiene su código y sus datos
- En cada proceso sólo hay un contexto de ejecución
- Cambio de contexto en la CPU es costoso.
- Cada proceso puede ser ejecutado por un procesador distinto

Threads (procesos ligeros = hebras) flujo de ejecución concurrente dentro de un proceso pesado

- Unidad básica de ejecución de CPU con: Contador de programa, registros, variables locales y pila
- Pueden existir varios hilos de ejecución en el mismo proceso compartiendo código y datos (contexto)
- Creación y destrucción rápida
- Estándares: Pthreads (Posix threads) y Java.

Estrategia de Paralelismo

- ¿Qué estrategia de programación seguir?
  - MPI (Message Passing Interface)
    - Algo más complejo de programar
    - Máxima flexibilidad
  - OpenMP
    - Fácil programación a nivel de directivas.
  - POSIX Threads (Pthreads) o Windows Threads
    - El programador controla todo: creación, acceso a recursos compartidos,….
  - Compiladores paralelizadores: HPF (High Performance Fortran)
    - Simple programación
    - Faltan directivas de distribución de datos realmente eficaces

Estrategia de Paralelismo

- Problemas de la Paralelización
  - Balanceo de Carga
    - Distribución de datos adecuada para TODAS las fases de la ejecución
  - Minimización de las comunicaciones
    - Mínimo número de mensajes
    - Mínima longitud de los mensajes
    - Optima distribución de las comunicaciones sobre la topología de la red de interconexión

Los Algoritmos Paralelos

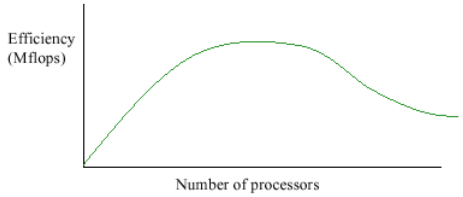
¡El problema debe ser paralelo!

- Ciertos algoritmos serie no se paralelizan bien. El desarrollo de nuevos algoritmos paralelos para reemplazar un algoritmo serie puede ser una de las tareas más difíciles en la computación paralela.
- No hay que olvidar que el algoritmo paralelo puede suponer más trabajo o más operaciones.

Eficiencia de la Computación Paralela

Eficiencia y limitaciones

- Dependiendo del problema, incrementar el número de elementos de proceso no mejora el rendimiento.



- La eficiencia del sistema se mejora si:
  - Se balancea la carga entre elementos de proceso.
  - Se minimiza la comunicación entre ellos.

Paradigma de paso de mensajes

Paso de mensajes:

- Principio general, aplicables a casi todos los tipos de arquitecturas paralelas (Sistemas con memoria distribuida / compartida)
- Es el paradigma más extendido hoy día en programación de aplicaciones paralelas.

El modelo de paso de mensajes es válido en:

- Sistemas de memoria distribuida
  - Cluster de Workstation(COW) Arquitectura homogenea con uso dedicado e inteconectado con red de alta velocidad (Infinibad)
  - Red de Estaciones de trabajo (NOW) Arquitectura heterogenea , sistemas no dedicado e interconectado con red estándar (Ethernet)
- También en sistemas de memoria compartida.

Principios del paso de mensajes

Programación de una aplicación paralela con P procesos con diferentes espacios de memoria.

La distribución de tareas se decide por el programador y un modelo muy utilizado es el paradigma **maestro-esclavo**.

- Un proceso (master) es el encargado de distibuir datos y recolectar resultados.
- El resto de procesos ejecutan la misma tarea sobre su porción de datos (paralelismo de datos - SPMD).

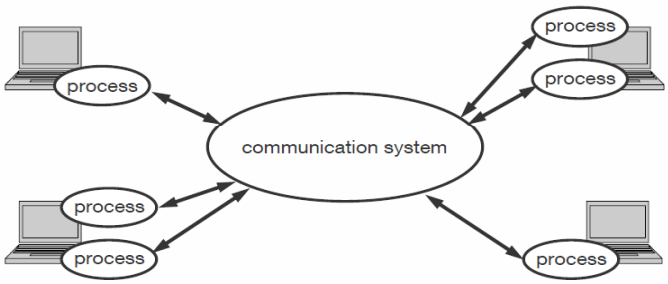
La comunicación tiene lugar por intercambio de mensajes.

- El intercambio de mensajes se realiza vía funciones de librería
  - Diseñado sin dependencias de Hardware o Lenguajes de programación.
- El intercambio de mensajes es cooperativo:
  - los datos deben ser tanto enviados como recibidos explícitamente.

Comunicación por paso de mensajes

El paradigma contempla un conjunto de procesos que interactúan por paso de mensajes.

- Cada proceso puede ejecutar código distinto y sobre diferentes datos (MIMD)
- Las funciones de librería son la única interfaz de comunicación.



Características del paso de mensajes

Escrito en un lenguaje convencional (C, FORTRAN,...)

Se ejecutan múltiples procesos en paralelo (idealmente uno en cada procesador del sistema).

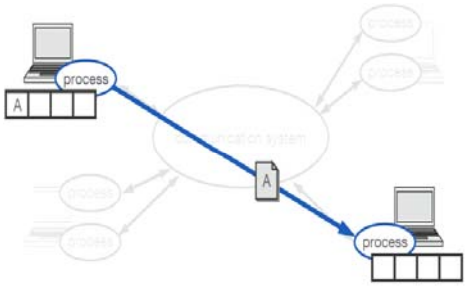
Todas la variables son privadas para cada proceso.

Los procesos se comunican vía llamadas a subrutinas

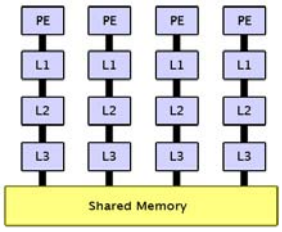
El Interfaz es genérico:

Las comunicaciones se ocultan al programador.

Los mensajes se intercambian explícitamente vía send() y receive()

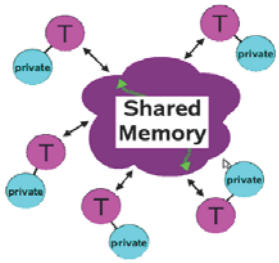


Sistemas de Memoria compartida



- Los procesadores ejecutan tareas que comparten un espacio de direcciones común, que se denomina memoria compartida.
- El acceso a la memoria se hace por buses y la jerarquía de memoria debe disponer de los mecanismos suficientes que garanticen la coherencia de contenidos.
- La gestión de la memoria compartida es invisible al programador.

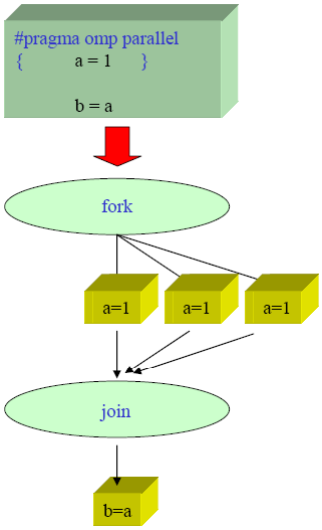
Paradigma de Memoria compartida: OpenMP



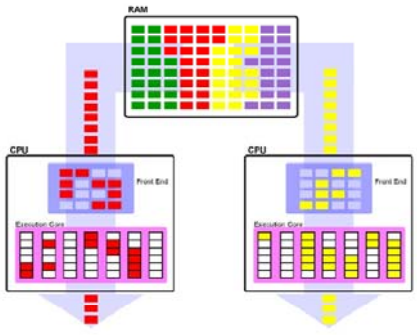
- El acceso a la memoria es compartido por todos los elementos de proceso.
- Cada unidad de ejecución (thread) puede tener datos compartidos y/o privados
  - **Los datos compartidos son comunes a todos los threads**
  - **Los datos privados solo los procesa el thread propietario**
- La transferencia de datos es transparente al programador
- Se producen sincronizaciones (implícitas)

Modelo de Ejecución OpenMP

- OpenMP se basa en el modelo fork/join
  - Un programa OpenMP empieza con un solo thread(Master Thread).
  - Se lanzan varios threads (Team) en un región de código paralelo.
  - Al salir de la región paralela los threads lanzados retornan.
- Un "Team" tiene una cantidad fija de threads ejecutando código replicado en la región paralela.
  - Construcciones para compartir trabajo especifican que threads ejecutan cada parte de trabajo
  - Una barrera de sincronización para todos los threads del team finaliza la región paralela.
- El código después de una región paralela se ejecuta secuencialmente por el Master thread.



Memoria compartida: Hilos a nivel S.O.



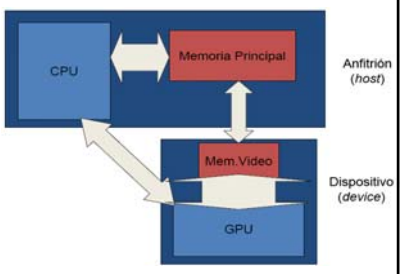
En sistemas de memoria compartida

- Un hilo es un contexto de ejecución independiente.
- Se pueden lanzar varios hilos dentro de un proceso del S.O.
- Si los hilos se ejecutan en diferentes procesadores se consigue paralelismo.
- Si los hilos se ejecutan en un procesador se planifican Round-robin.
- La creación de hilos y las sincronizaciones son explícitas.
- POSIX Threads, Windows Threads, Java Threads,

Paradigma de programación CUDA

La GPU es un dispositivo de computación (compute device) que:

- Es un coprocessor para la CPU (host)
- Tiene su propia DRAM (device memory)
- Ejecuta muchos threads en paralelo



Fragmentos de una aplicación con PARALELISMO DE DATOS se ejecutan en la GPU (device) como kernels que corren en paralelo en muchos threads

Diferencias entre threas de GPU y de CPU

- GPU threads son ligeros (extremely lightweight)
  - Muy poco overhead de creación
- GPU necesita 1000s de threads para ser realmente eficiente
  - Multi-core CPUsolo necesita unas pocos.

Modelo de ejecución CUDA

Warp = Threads agrupados en una instrucción SIMD

- CUDA utiliza SIMT (Single Instruction Multiple Thread)
- Warps son grupos de 32 threads. Cada warp recibe una instrucción y hace "broadcasts" de ella a todos sus threads.
- CUDA proporciona "zero-overhead" en planificación de warps y threads. En media, el overhead de creación de threads es de 1 ciclo de reloj.

Warp: In the textile industry, the term "warp" refers to "the threads stretched lengthwise in a loom to be crossed by the weft".