Modelos de programación paralela

Programación Paralela

José María Cecilia

Grado en Ingeniería informática

José M. Cecilia- Tlf: (+34) 968 278 587
Universidad Católica San Antonio de Murcia - Tlf: (+34) 968 00 00 00 info@ucam.edu - www.ucam.edu

Contenido

Introducción

Paralelismo de datos Vs Tareas

Modelos de programación paralela

Paralelización Implícita Vs Explícita

Introducción

Diferencias entre los paradigmas:

- ***Compromiso entre facilidad de programación vs. eficiencia:*** Algunos paradigmas de programación son mas sencillos de programar que otros (mas alto nivel) pero generan código mas alejado al hardware (menos eficiente).
- ***Naturaleza de la aplicación:*** Distintas aplicaciones pueden tener diferentes tipos de paralelismo.
- **Conceptos fundamentales:**
  - Los modelos de programación paralela existen como abstracción de las arquitecturas hardware y memoria.
  - No tienen que estar vinculados a una arquitectura concreta (hay excepciones).

Contenido

Introducción

Paralelismo de datos Vs Tareas

Modelos de programación paralela

Paralelización Implícita Vs Explícita

Paralelismo de datos

- Aplicaciones en las que los datos están sujetos a idéntico procesamiento.
- Apropiado para máquinas SIMD. También en MIMD.
  - Sincronización global después de cada instrucción -> ¡INEFICIENTE!
  - **Solución**: Relajar la ejecución síncrona de las instrucciones.
  - **Modelo SPMD (*Single Program Multiple Data*):** Cada procesador ejecuta el mismo programa asíncronamente
- Lenguajes de paralelismo de datos ofrecen construcciones de alto nivel para compartir información y manejar concurrencia -> Mas fácil de comprender y escribir.

Paralelismo de control o tareas

- Ejecución simultánea de cadenas de instrucciones diferentes (tareas). Las instrucciones se pueden aplicar sobre los mismos datos aunque, normalmente, sobre diferentes datos.
- Adecuados para MIMD ya que requiere múltiples cadenas de instrucciones.
- Se ejecutan tareas distintas.

Contenido

Introducción

Paralelismo de datos Vs Tareas

Modelos de programación paralela

Paralelización Implícita Vs Explícita

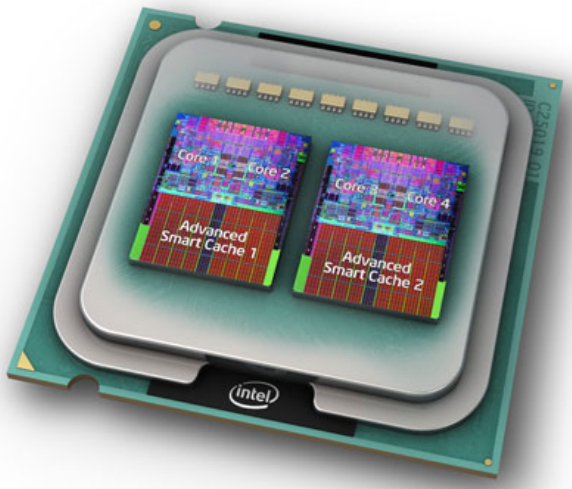
Modelos de programación paralela: Resumen

- Shared Memory (sin threads)
- Threads
- Distributed Memory / Message Passing
- Paralelismo de datos
- Paralelismo Híbrido
- Single Program Multiple Data (SPMD)
- Multiple Program Multiple Data (MPMD)
- Heterogeneous computing

Uso de modelos de programación sobre hardware independiente

- Modelo de memoria compartida en una máquina distribuida (Superdome benarabi).
- Modelo de memoria distribuida en una máquina de memoria compartida.

HP Integrity Superdome SX2000



¿Qué modelo utilizamos?

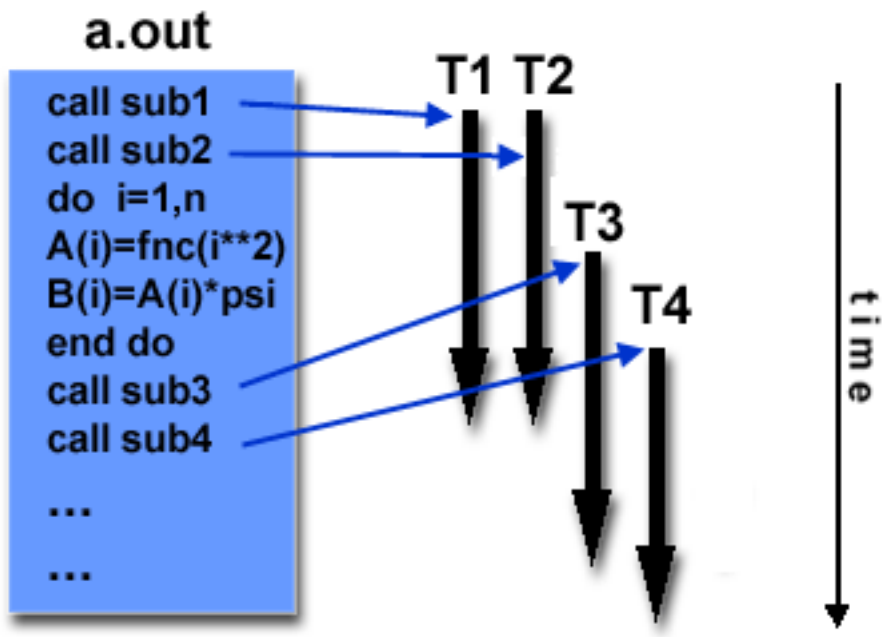
- Combinación entre disponibilidad y elección personal.
  - Rendimiento.
  - Facilidad de uso.
- No existe el mejor modelo pero si hay mejores implementaciones.
- También depende del hardware

Modelo de memoria compartida (sin hilos)

- Las tareas comparten un espacio de memoria que pueden leer/escribir asíncronamente.
- Se deben de utilizar mecanismos como locks o semáforos para controlar el acceso a la memoria compartido.
- Ventaja:
  - No hay necesidad de indicar comunicación entre tareas.
- Desventajas:
  - Mas difícil manejar la localidad de datos.
- Implementaciones: Compiladores establecen variables como globales.

Modelo de Hilos

- Tipo de memoria compartida.
- El programa principal a.out puede desarrollar una tarea Secuencial, y crear hilos Que se ejecuten concurrentemente
- Implementaciones: POSIX Threads y OpenMP



Espacio de direcciones compartido

- **Programa:** Colección de procesos accediendo a una zona central de variables compartidas
- **Exclusión mutua:** Mas de un proceso puede acceder en el mismo instante a una misma zona de memoria
  - Puede producir resultados impredecibles.
  - Necesidad de un mecanismo de protección -> Primitivas de exclusión mutua
- **Sintaxis:** basada en C pero con extensiones (pragmas).
- **Estándares**: OpenMP

Espacio de direcciones compartido

- Primitivas para asignar variables compartidas.
  - 2 tipos de variables: compartidas (shared) y locales (private)
- Primitivas para la exclusión mutua y sincronización
  - **Sección crítica:** Código que sólo puede ser ejecutado por un procesador en un momento determinado.
    - Necesidad de **locks (candados)** para garantizar la exclusión mutua en la ejecución de secciones críticas.
  - **Barreras:** Cuando los procesos necesitan sincronizarse. Cada proceso espera en la barrera a los otros procesos. Después de la sincronización todos los procesos continúan con su ejecución.

Espacio de direcciones compartido

- Primitivas para la creación de procesos
  - **Modelo Fork + join:**
    - Llamada al sistema que crea procesos idénticos al padre (**fork**), comparten las variables compartidas, locks, etc.
    - Cuando los subprocesos terminan se mezclan usando **join**
    - Al principio tenemos el proceso maestro. Si el proceso maestro necesita realizar una tarea en paralelo, crea un número predeterminado de procesos, llamados procesos esclavos. Cuando la tarea se completa, los procesos esclavos terminan y devuelven el control al proceso maestro.

Paso de mensajes

- **Programa:** Colección de procesos con variables privadas cada proceso, y con la posibilidad de enviar y recibir datos mediante paso de mensajes.
- **Arquitectura:** La arquitectura ideal para este paradigma es distribuida pero también se puede utilizar en arquitecturas de memoria compartida.
- **Sintaxis:** La sintaxis del lenguaje de programación suele ser basada en C, pero con extensiones (conjunto de llamadas especiales):
  - paso de mensajes, sincronización de procesos, exclusión mutua.
  - Problema de portabilidad entre arquitecturas.
- **Estándares**: *Message Passing Interface*, MPI

Paso de mensajes

- ***Extensiones básicas:*** Son extensiones en los lenguajes secuenciales para soportar el paso de mensajes.
- ***Dos Primitivas básicas de comunicación:*** SEND y RECEIVE. Ejemplo:

SEND (message, size, target, type, flag)

- message: contiene los datos que se envían
- size: indica el tamaño en bytes
- target: procesador de destino
- type: tipos de mensajes enviados
- flag: si la operación SEND es bloqueante o no.

Paso de mensajes

- La primitiva RECEIVE lee un mensaje del buffer de comunicación en la memoria.

RECEIVE(message, size, source, type, flag)

- message: indica el lugar donde se almacenan los datos

- size: indica el tamaño en bytes

- source: procesador de donde viene el mensaje

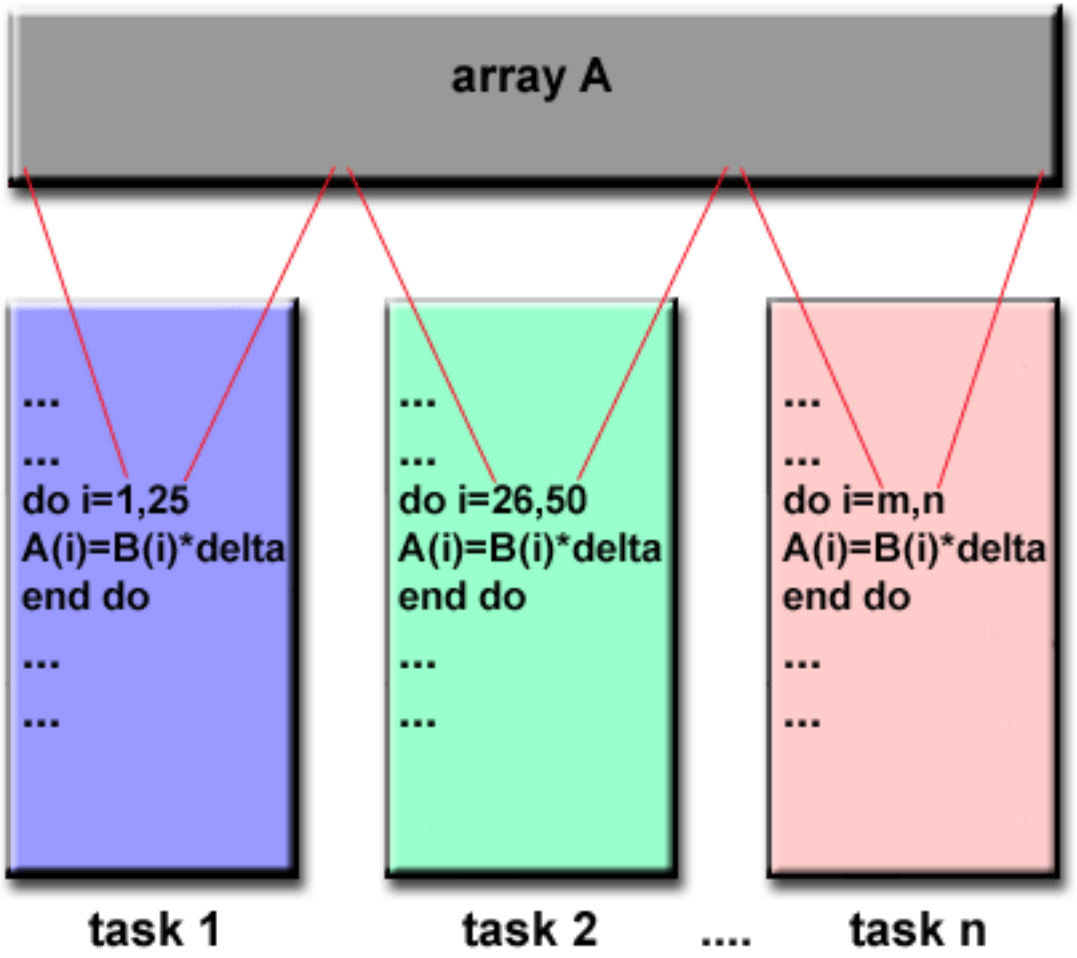
- type: tipos de mensajes enviados. Puede haber más de un mensaje en el buffer de comunicación de los procesadores fuente.

- flag: si la operación de recibir es bloqueante o no.

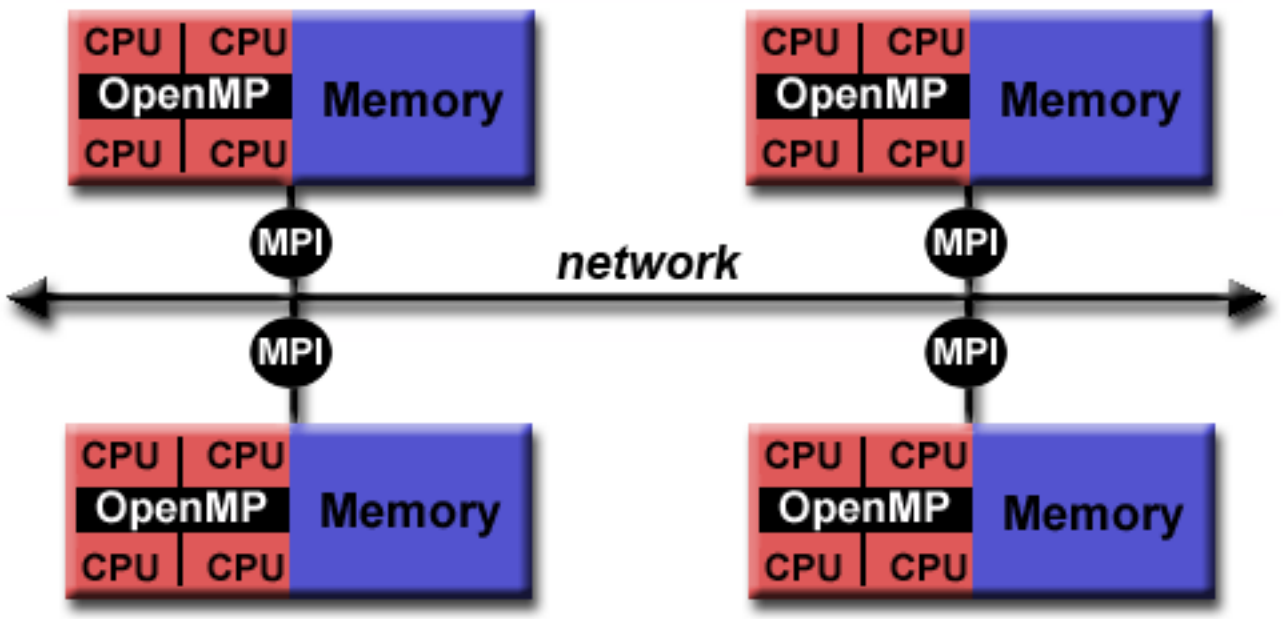
Modelo paralelo de datos

- Partitioned Global Address Space: PGAS
- Siguientes características:
  - Espacio de direcciones tratadas globalmente.
  - Desarrollar operaciones sobre un conjunto de datos (array) diferente.
  - Misma operación sobre distintos datos.
- Implementaciones:
  - Chapel: an open source parallel programming language project being led by Cray. More information: http://chapel.cray.com/

Modelo paralelo de datos

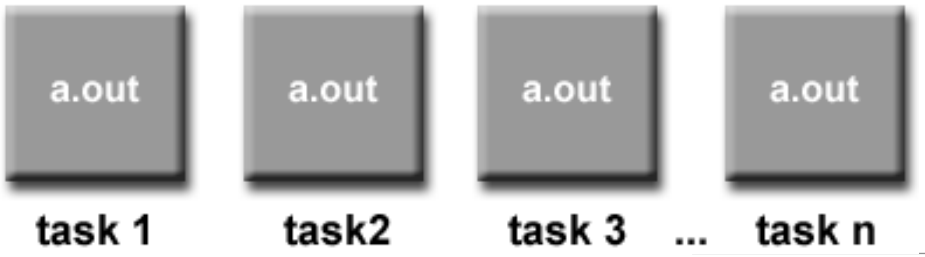


Modelo Híbrido



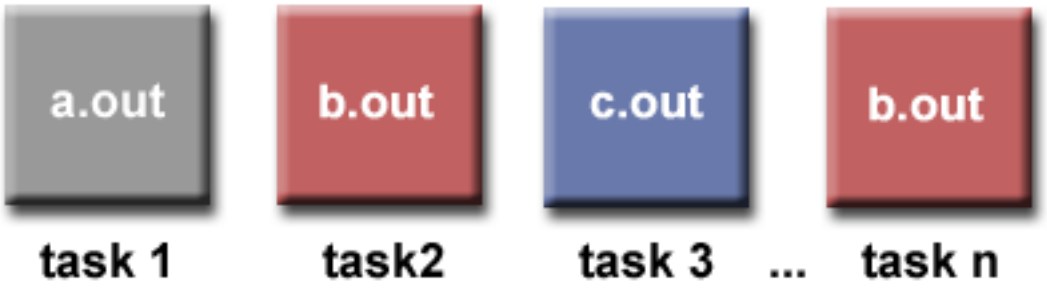
Single Program Multiple Data (SPMD)

- Modelo de alto nivel que se construye sobre los anteriores.
- Todas las tareas ejecutan el mismo programa.
- Las tareas se aplican sobre distintos datos.
- Necesitamos una lógica para que dependiendo la tarea se ejecute unas instrucciones u otras. Condicionales
- Disponible en GPUs



Multiple Program Multiple Data (MPMD)

- Modelo de alto nivel que se construye sobre los anteriores.
- Las tareas pueden ejecutar diferentes programas.
- Las tareas se aplican sobre distintos datos.
- Necesitamos una lógica para que dependiendo la tarea se ejecute unas instrucciones u otras. Condicionales
- Disponible en GPUs bajo streams



Modelos de programación en entornos heterogéneos

- **Computador heterogéneo:** Tenemos varios procesadores dentro de un mismo computador de distinta naturaleza. Secuencial, SIMD, Memoria Compartida, etc…
  - Ejemplo: CPU (multicore) + GPU
- Cada día es mas habitual -> todos los ordenadores tiene procesador y tarjeta gráfica.
- Además la unidad de procesamiento gráfico se está incluyendo dentro del chip. Ejemplo Fusion AMD, Sandy Bridge, etc…
- ¿Cómo programamos en entornos heterogéneos?

Modelos de programación en entornos heterogéneos

- Mezclar paralelismo de tareas y datos.
- Lenguaje mas popular CUDA de NVIDIA.
  - Mezclamos dos paradigmas. Código secuencial para la CPU (C, python, Java, Fortran) + Extensiones para controlar y computar en la GPU.
  - **Problema:** Lenguaje propietario, sólo para GPUs de NVIDIA.
- **Estándar:** OpenCL
  - Diseñado para ejecutar una misma aplicación en cualquier plataforma de cómputo.
  - **Problema:** Demasiado general, consorcio de empresas, etc…

Modelos de programación en entornos heterogéneos

- OpenACC es una colección de directivas de compilación para especificar bucles y regiones de código que enviar a un acelerador. El lenguaje soportado es C, C++ y Fortran.
- Es portable entre SSOO, CPUs y aceleradores.
- Permite crear modelos de alto nivel que combinan códigos host + acelerador
  - No hay inicialización del acelerador.
  - No hay necesidad de transferir información entre host y device
- El compilador y el runtime de OpenACC se encarga de tratar el hardware particular.

Ejemplo OpenACC Vs OpenMP

```
 // Multiplicación de matrices
#pragma acc kernels copyin(a,b) copy(c)
  for (i = 0; i < SIZE; ++i) {
    for (j = 0; j < SIZE; ++j) {
      for (k = 0; k < SIZE; ++k) {
    c[i][j] += a[i][k] * b[k][j];
      }
    }
  }
```

```
 // Multiplicación de matrices
#pragma omp parallel for default(none) shared(a,b,c) private(i,j,k)
  for (i = 0; i < SIZE; ++i) {
    for (j = 0; j < SIZE; ++j) {
      for (k = 0; k < SIZE; ++k) {
    c[i][j] += a[i][k] * b[k][j];
      }
    }
  }
```

Contenido

Introducción

Paralelismo de datos Vs Tareas

Modelos de programación paralela

Paralelización Implícita Vs Explícita

Paralelismo implícito/explícito

- **Explícito:**
  - El algoritmo paralelo debe especificar explícitamente cómo cooperan los procesadores.
  - La tarea del compilador es sencilla. La del programador es bastante difícil.
- **Implícito:**
  - Programación secuencial.
  - Compilador paraleliza el código de forma transparente al programador.
  - El compilador tiene que analizar y comprender las dependencias para asegurar un mapeo eficiente.
  - Desarrollo del compilador muy costoso y generación de código poco eficiente.

Compiladores paralelizantes

- Programa secuencial. El compilador se encarga de paralelizarlo.
- Normalmente paralelización de bucles: dividen el trabajo en los bucles entre los distintos procesadores.
- Si hay (o puede haber) dependencia de datos no paraleliza:

```
para i=1 to n
        a[i] = b[i]+a[i-1]
finpara
```

- Se puede forzar la paralelización con opciones de compilación (--parallel, -o3), o con directivas (OpenMP).
- Generan ficheros con información de bucles paralelizados y no paralelizados, y el motivo

¿Preguntas?