Perfil7
www.wuolah.com/student/Perfil7

414

ApuntesTEMA2AC.pdf

APUNTES con diapositivas y libro

2º Arquitectura de Computadores

Grado en Ingeniería Informática

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación
Universidad de Granada

- ARQUITECTURA DE COMPUTADORES -

Tema 2

Lección 4

PROBLEMAS QUE PLANTEA LA PROGRAMACIÓN PARALELA

La programación secuencial ha dejado paso a la paralela, por lo que para adaptar este cambio, debemos cambiar las herramientas y las propias costumbres del programador. Aún así, la programación paralela, plantea los siguientes problemas:

- División en unidades de cómputo independientes (tareas)
- Agrupación asignación de tareas o carga de trabajo (código, datos) en procesos/threads
- Asignación a procesadores/núcleos
- Sincronización y comunicación

MODOS DE PROGRAMACIÓN MIMD

Por programación paralela, entendemos que podemos trabajar con múltiples instrucciones y múltiples datos, como es el caso de:

SPMD – El mismo programa ejecutado de forma paralela

MPMD – Un programa se fragmenta y cada uno de los fragmentos se ejecuta de forma paralela

HERRAMIENTAS DE PROGRAMACIÓN PARALELA

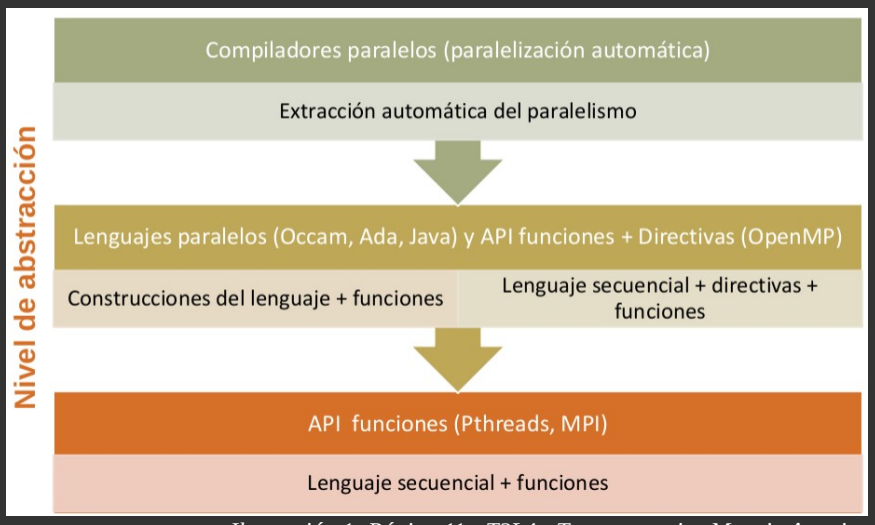


Ilustración 1: Página 11 - T2L4 - Transparencias Mancia Anguita

Dichas herramientas permiten de forma explícita o implícita:

- Localizar paralelismo o descomponer en tareas independientes (descomposition)
- Asignar tareas (código y datos) a procesos/threads
- Crear y terminar procesos/threads (enrolar y desenrolar)
- Comunicar y sincronizar procesos/threads

Además, el programador, o bien la herramienta, o bien el propio SO se encargan de asignar procesos/threads a unidades de procesamiento (mapping)

COMUNICACIONES COLECTIVAS

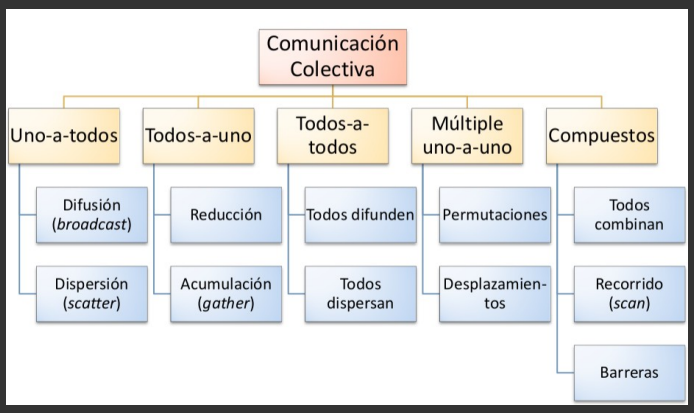


Ilustración 2: Página 15 - T2L4 - Transparencias Mancia Anguita

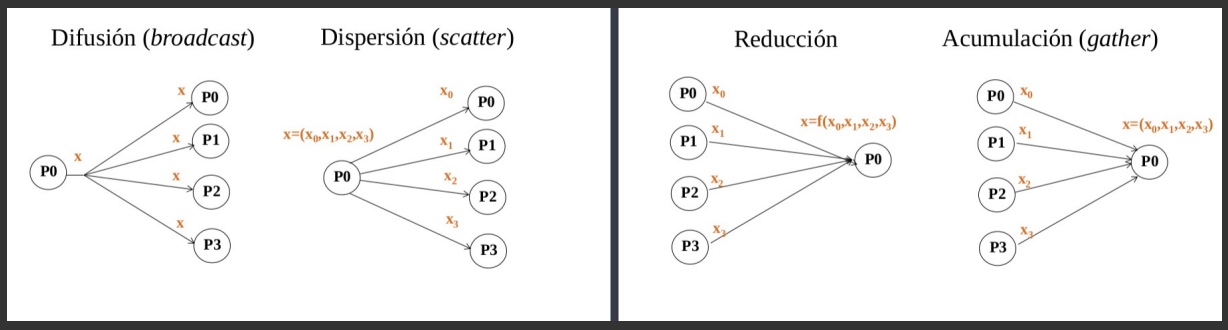


Ilustración 3: Página 16/17 - T2L4 - Transparencias Mancia Anguita

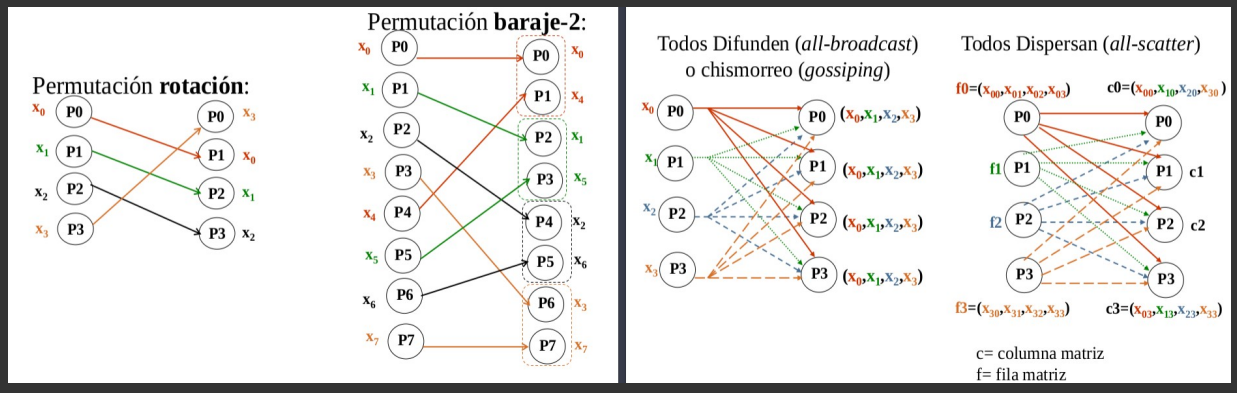


Ilustración 4: Página 18/19 - T2L4 - Transparencias Mancia Anguita

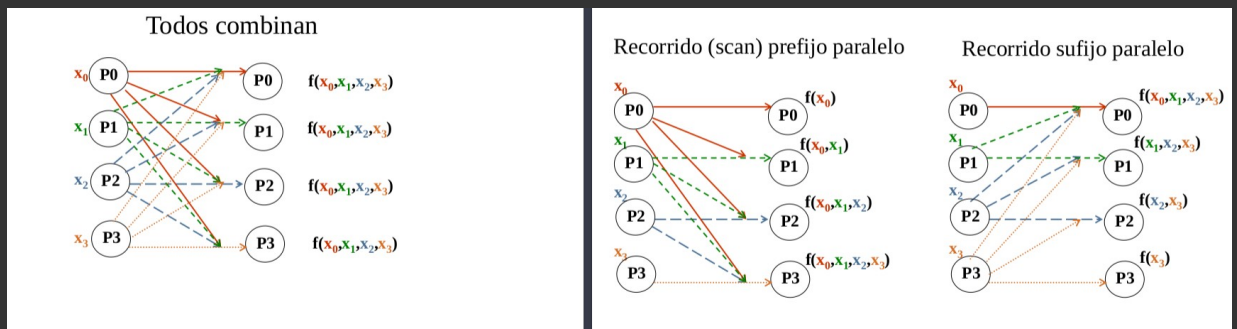


Ilustración 5: Página 20/21 - T2L4 - Transparencias Mancia Anguita

ESTILOS DE PROGRAMACIÓN Y ARQUITECTURAS PARALELAS

Estructura de cómputo	Implementa paralelismo a través de	Herramienta
Multicomputador	→	Paso de mensajes
Multiprocesador	→	Variables compartidas
Procesadores matriciales, GPU	→	Paralelismo de datos

El paso de mensajes se puede observar en lenguajes de programación JAVA o OCCAM y APIs(bibliotecas de funciones) como MPI o PVM

De igual forma, las variables compartidas también se pueden aplicar a lenguajes, APIs(directivas del compilador+funciones. Y bibliotecas de funciones)

Y el paralelismo de datos en lenguajes de programación y APIs(funciones de stream processing) como NVIDIA CUDA

ESTRUCTURAS TÍPICAS DE PROCESOS/TAREAS/THREADS

Hay una serie de estructuras típicas para aprovechar el paralelismo:

-Descomposición de dominio o descomposición de datos → Cada thread procesa su parte y la comparte con los otros threads, de forma que no necesitan ninguna otra entidad que maneje los resultados. Se autogestionan.

-Cliente/Servidor → El servidor responde a las peticiones de los clientes y envía las respuestas (cada cliente es un thread)

-Divide y vencerás o descomposición recursiva → Cada thread se encarga de un flujo de operaciones independiente, de manera que al unir cada uno de los flujos, se pueda obtener el resultado final.

-Segmentación o flujo de datos → Al igual que en descomposición de datos, cada thread hace su trabajo, pero la dirección de flujo es siempre al siguiente thread. De forma que cada nodo de trabajo hace exclusivamente una tarea y la manda al siguiente nodo.

-Master-Slave o granja de tareas → El master envía el trabajo a los slaves y recolecta los resultados de estos

Lección 5

PROCESO DE PARALELIZACIÓN

1. Descomponer en tareas independientes o localizar paralelismo.
2. Asignar tareas a procesos/threads (planificar+mapear)
3. Redactar código paralelo
4. Evaluar prestaciones

DESCOMPONER EN TAREAS INDEPENDIENTES

Para ello debemos hacer un análisis de dependencias entre funciones y dependencias entre bucles

ASIGNAR TAREAS A PROCESOS

Por esta sección entendemos tanto la planificación (scheduling) como el mapeo a los procesadores/cores (mapping).

Planificación → agrupar tareas en threads
Mapeo → asignar dichos threads a cores

Para la granularidad de la carga asignada, debemos considerar el número de elementos de procesamiento disponibles y el tiempo de comunicación frente al tiempo de cálculo

Además también debemos tener en cuenta el equilibrio de la carga (tareas= código+datos), lo que se conoce como load balancing. Con el objetivo de que unos procesos/threads no deben esperar a otros.

Este equilibrio en gran parte depende de la arquitectura. Debemos tener en cuenta si tratamos con una arquitectura homogénea o no homogénea, o uniforme o no uniforme.

Encontramos 2 tipos de asignación:

- Asignación estática: Que establece qué tarea ejecutará cada core
- Asignación dinámica (en tiempo de ejecución) : Distintas tareas se pueden asignar a distintos cores

En ambos casos la asignación explícita es la que ejerce el programador y la implícita la herramienta de programación al general el código ejecutable.

Respecto al mapeo de threads, se suele relegar esta labor al SO (light process). Aunque también lo puede hacer el entorno o sistema en tiempo de ejecución (runtime system de la herramienta de programación). El programador puede influir en esta tarea.

REDACTAR EL CÓDIGO PARALELO

En este apartado es donde se crea el código paralelo. Para ello se debe Crear el código paralelo (enrolar), Localizar el paralelismo, Comunicar/Sincronizar las distintas partes, agrupar y asignar y por último terminar el código paralelo (desenrolar)

Lección 6

EVALUAR PRESTACIONES

Las prestaciones se pueden medir a través de:

- Las medidas usuales para ello son tiempo de respuesta y productividad.

Para el tiempo de respuesta, encontramos dos variantes:

- Real (wall-clock time o elapsed time)
- Usuario, sistema, CPU time = user+sys

- Escalabilidad
- Eficiencia, que no es más que la relación entre prestaciones/prestaciones_máximas
  Rendimiento = prestaciones/num_recursos
  Otras = prestaciones/consumo o área_ocupada

ESCALABILIDAD

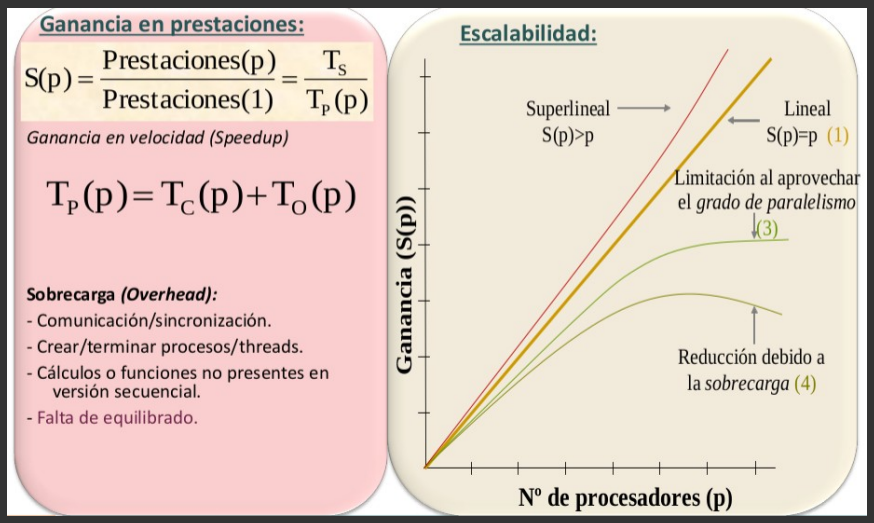
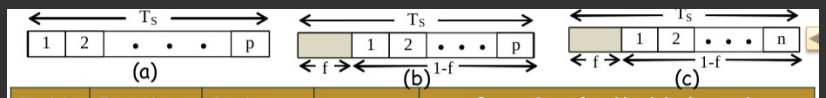


Ilustración 6: Página 68 - T2L4 - Transparencias Mancia Anguita

GANANCIA MÁXIMA



Modelo código	Fracción no paral. en T_S	Grado paralelismo	Overhead	Ganancia en función del número de procesadores p con T_S constante
(a)	0	ilimitado	0	$S(p) = \frac{T_S}{T_P(p)} = p$ Ganancia lineal (1)
(b)	f	ilimitado	0	$S(p) = \frac{1}{f + \frac{(1-f)}{p}} \xrightarrow{p\to\infty} \frac{1}{f}$ (2)
(c)	f	n	0	$S(p) = \frac{1}{f + \frac{(1-f)}{p}} \xrightarrow{p=n} \frac{1}{f + \frac{(1-f)}{n}}$ (3)
(b)	f	ilimitado	Incrementa linealmente con p	$S(p) = \frac{1}{f + \frac{(1-f)}{p} + \frac{T_O(p)}{T_S}} \xrightarrow{p\to\infty} 0$ (4)

Ilustración 7: Página 69 - T2L4 - Transparencias Mancia Anguita

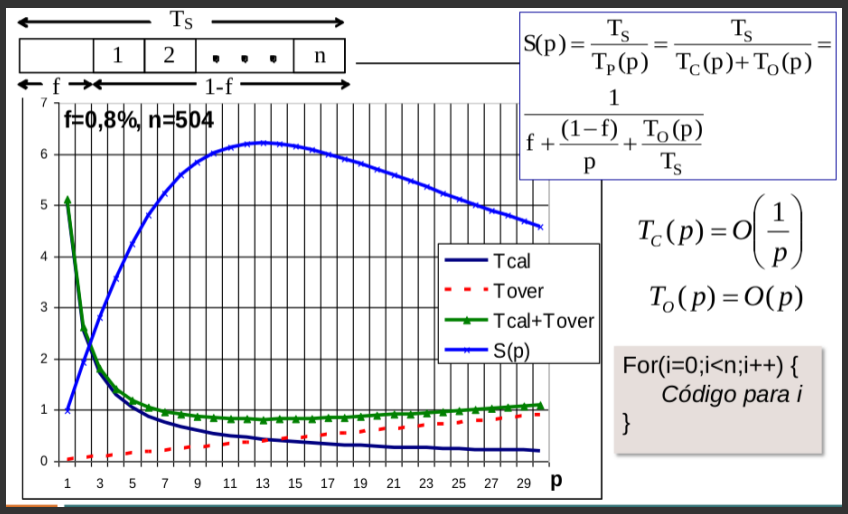


Ilustración 8: Página 70 - T2L4 - Transparencias Mancia Anguita

LEY DE AMDAHL

- Ley de Amdahl: la ganancia en prestaciones utilizando p procesadores está limitada por la fracción de código que no se puede paralelizar (2):

$$S(p)=\frac{T_S}{T_P(p)}\le\frac{T_S}{f\cdot T_S+\frac{(1-f)\cdot T_S}{p}}=\frac{p}{1+f(p-1)}\to\frac{1}{f}(p\to\infty)$$

  - S : Incremento en velocidad que se consigue al aplicar una mejora. (paralelismo)
  - p : Incremento en velocidad máximo que se puede conseguir si se aplica la mejora todo el tiempo. (número de procesadores)
  - f : fracción de tiempo en el que no se puede aplicar la mejora. (fracción de t. no paralelizable)

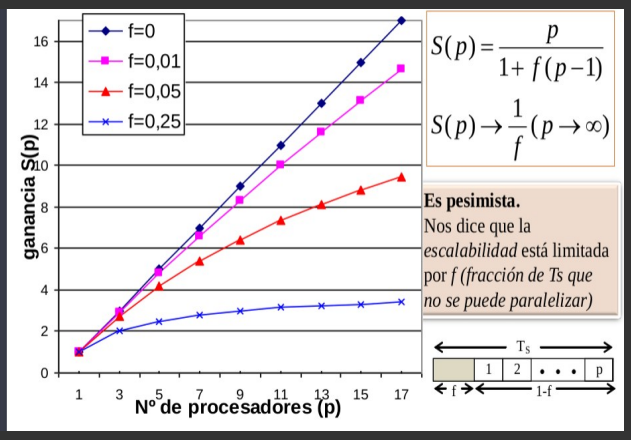


Ilustración 9: Página 72/73 - T2L4 - Transparencias Mancia Anguita

GANANCIA ESCALABLE

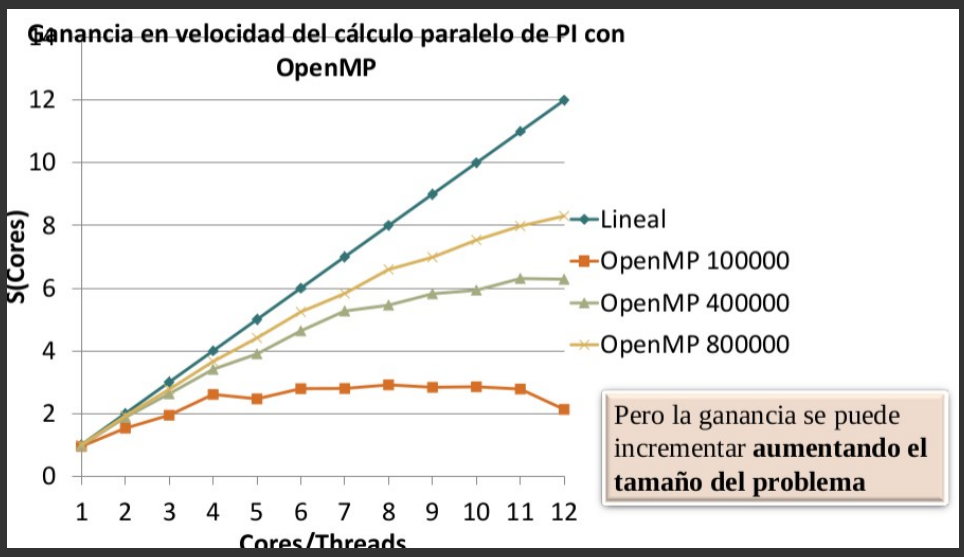


Ilustración 10: Página 74 - T2L4 - Transparencias Mancia Anguita

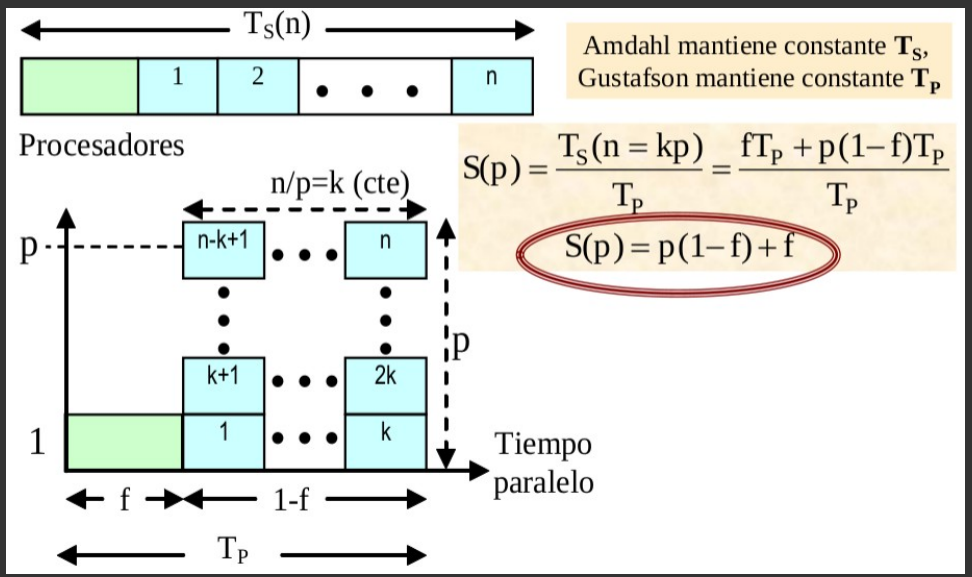


Ilustración 11: Página 72/73 - T2L4 - Transparencias Mancia Anguita