Programación paralela

Programación paralela.

Nociones de programación paralela.

Creación de un programa paralelo.

- Descomposición.
- Asignación.
- Orquestación.
- Mapeado.

Paralelización de un programa.

¿Para quién es importante?

Diseñadores de Algoritmos

- Diseñar algoritmos que corran bien en sistemas reales.

Programadores

- Comprender dónde radican las claves del rendimiento para obtener el mejor rendimiento posible en un sistema dado.

Arquitectos

- Comprender las cargas, las interacciones, y la importancia de los grados de libertad.
- Importante para el diseño y la evaluación.

Importancia de la programación

Diseñamos máquinas para que ejecuten *programas*. Así pues, éstos:

- ayudan a tomar decisiones en el diseño hardware;
- ayudan a evaluar los cambios en los sistemas.

Son la clave que dirige los avances en la arquitectura uniprocesador.

- Caches y diseño del conjunto de instrucciones.

Mayor importancia en multiprocesadores.

- Nuevos grados de libertad.
- Mayor penalización si hay desacoplo entre programas y arquitectura.
- Más espacio para optimaciones en software.

Importancia de la programación

Aun desde el punto de vista arquitectónico, necesitamos abrir la “caja negra” del software.

Aunque un problema disponga de un buen algoritmo secuencial, paralelizarlo no es trivial.

Hay que ser capaces de

- extraer el paralelismo y
- combinarlo con las prestaciones ofrecidas por la arquitectura.

Ejemplos de problemas paralelizables

Simulación de corrientes oceánicas.

- Estructura regular de comunicación, computación científica.

Simulación de la evolución de galaxias.

- Estructura irregular de comunicación, computación científica.

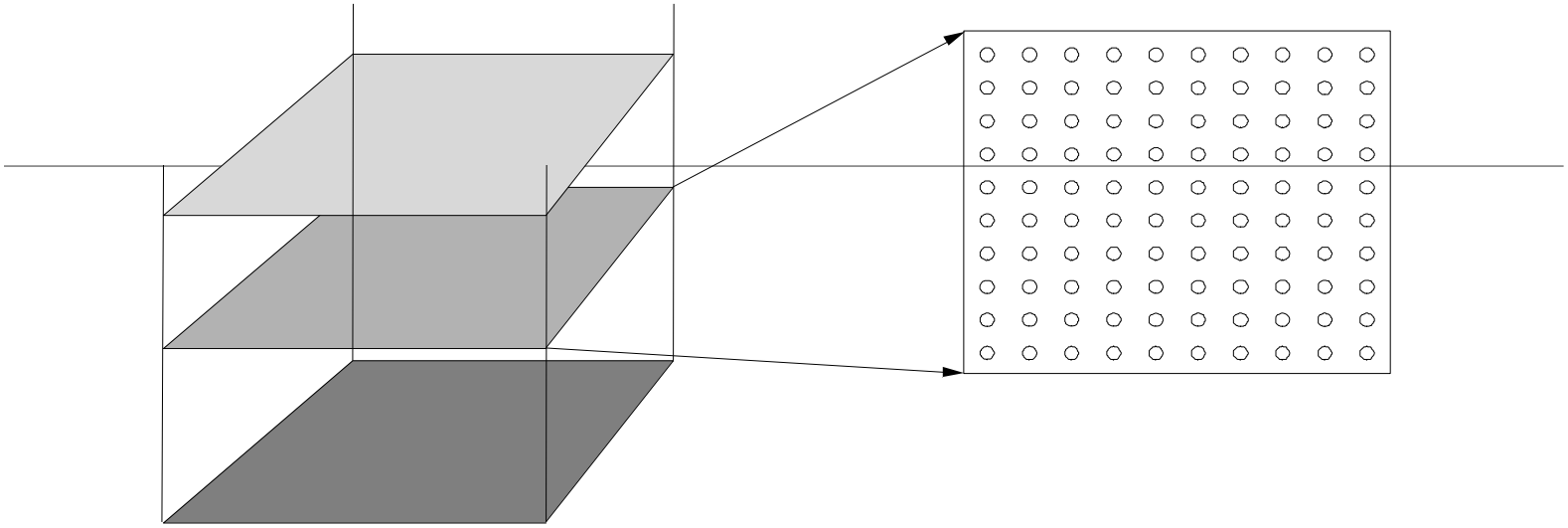
Reproducción de escenas por trazado de rayos.

- Estructura irregular de comunicación, computación gráfica.

Data mining.

- Estructura irregular, procesado de información.

Simulación de corrientes oceánicas



Modelado como parrillas de dos dimensiones.

Separación en espacio y tiempo.

- **Finura en la resolución espacial y temporal => gran precisión.**

Muchas operaciones diferentes por unidad de tiempo.

- **Planteamiento y solución de ecuaciones.**

Concurrencia de computación a través y entre la parrilla.

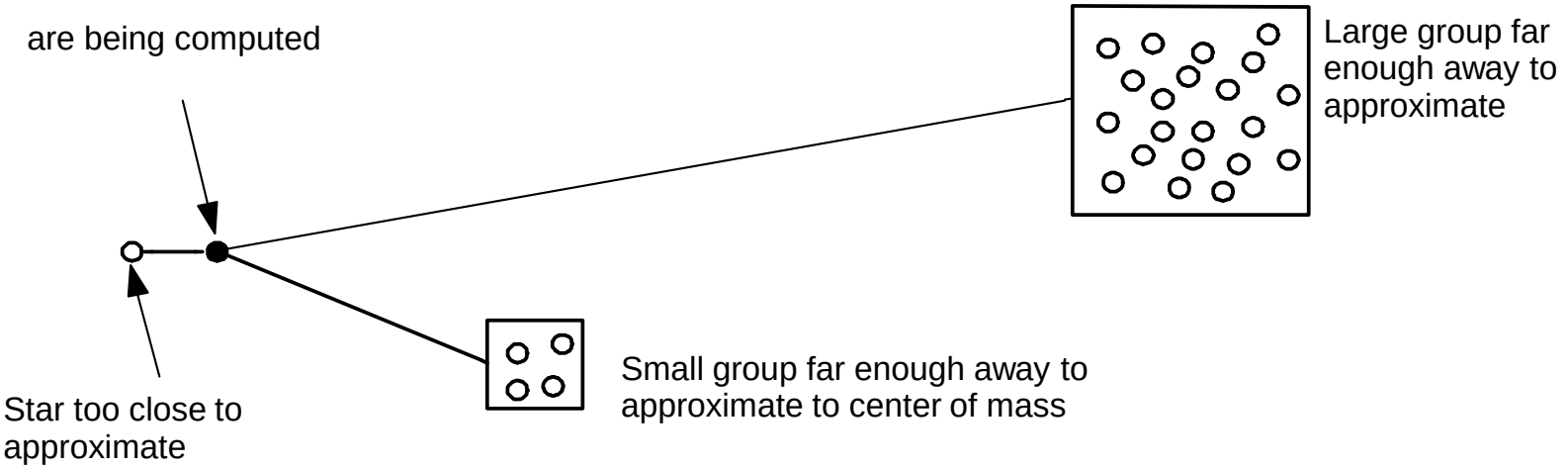
Simulación de la evolución de galaxias

Simula la interacción de multitud de estrellas en el tiempo.

La computación de las fuerzas es cara.

Aproximación bruta: la computación entre pares de estrellas tarda $O(n^2)$.

Los métodos jerárquicos tardan $O(n \cdot \log n)$. Se utiliza el hecho de que la fuerza decae con el cuadrado de la distancia.



Reproducción de escenas por trazado de rayos.

Dispara rayos dentro de la escena a través de pixels en el plano de la imagen.

Seguimiento de sus caminos:

- Rebotan al chocar con los objetos.
- Generan nuevos rayos: árbol de rayos por rayo entrante.

El resultado es el color y la opacidad de los pixels.

Paralelismo entre rayos.

Minería de datos *(Data mining)*

Extracción de información útil (“conocimiento”) a partir de grandes volúmenes de datos.

Un problema particular es el hallazgo de relaciones entre conjuntos en principio no correlacionados de datos.

Existe paralelismo a la hora de examinar conjuntos de datos relacionados de tamaño $k - 1$ para tratar de obtener un conjunto de datos relacionados de tamaño k.

Creación de un programa en paralelo

Asumimos: se nos da un algoritmo secuencial.

- Si éste no se presta a la paralelización, se hace necesario un algoritmo completamente diferente.

Proceso:

- Identificar el trabajo que se puede hacer en paralelo.
- Dividir el trabajo y/o los datos entre procesos.
  - Nota: El trabajo incluye la computación, el acceso a datos y la entrada/salida.
- Gestionar los accesos a datos, las comunicaciones y la sincronización.

Objetivo:

- Obtener alta productividad con costo bajo en programación y recursos.

Creación de un programa en paralelo

Objetivo principal: incrementar el *Speedup.*

$$Speedup\ (p) = \frac{Performance(p)}{Performance(1)}$$

Para un problema fijo:

$$Speedup\ (p) = \frac{Time(1)}{Time(p)}$$

Creación de un programa en paralelo

Tarea:

- Orden de trabajo que no puede descomponerse en subórdenes ejecutables en paralelo.
- Se ejecuta secuencialmente; la concurrencia solo se da entre tareas.

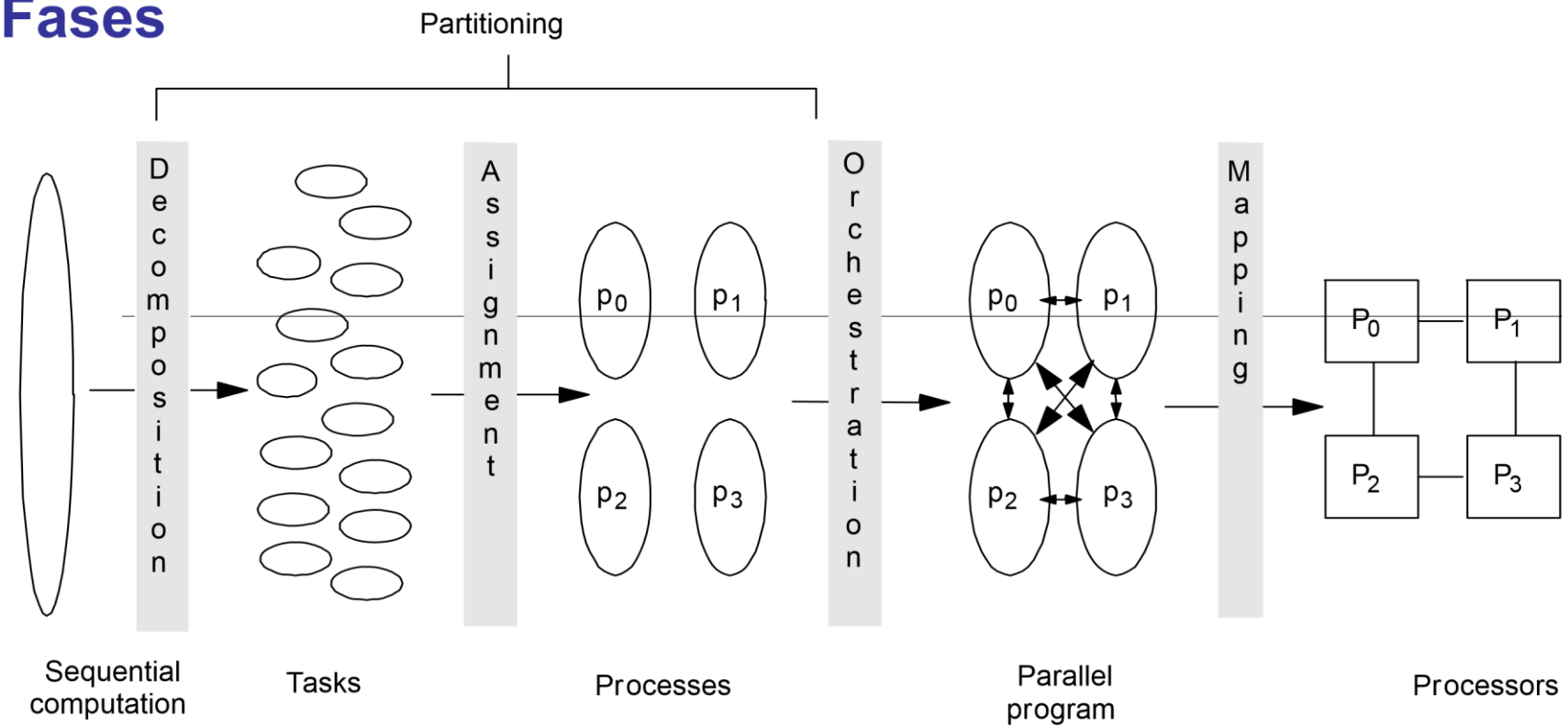
Proceso (thread o hebra):

- Entidad abstracta que ejecuta las tareas que le son asignadas.
- Un programa paralelo está constituido por “muchos procesos cooperando”.
- Hay una comunicación y sincronización en los procesos para la ejecución de las tareas.

Procesador:

- Elemento físico en el que se ejecutan los procesos.
- Los procesos constituyen la máquina virtual para el programador.
  - Primero se escribe el programa en términos de procesos y después de mapea a los procesadores.

Fases



4 pasos: Descomposición, Asignación, Orquestación, Mapeado.

- *Hecho por el programador o por el sistema (compilador, runtime, ...)*
- *Las acciones son las mismas; si las hace el programador, se realizan de forma explícita.*

Descomposición

División del programa en tareas que serán distribuidas entre los procesos.

Las tareas pueden estar dispuestas para ejecución dinámicamente.

- El número de tareas disponibles puede variar con el tiempo.
- Identificar la concurrencia y decidir a qué nivel se va a explotar.

Objetivo: Suficientes tareas para mantener ocupados los procesos pero no demasiadas.

- El número de tareas disponibles en un momento define el límite del *speedup* que podemos conseguir.

Descomposición

Limites de Concurrencia: Ley de Amdahl

Es la limitación fundamental del *speedup.*

Si la fracción *s* del programa es inherentemente serie, *speedup* ≤ 1/*s.*

Ejemplo: Supongamos un cálculo en dos fases con *p* procesadores.

- Primera fase: Operación independiente sobre cada punto de una rejilla de $n \times n$ elementos.
- Segunda fase: Suma global de los n^2 puntos.

Tiempo para la primera fase = n^2/p.

Tiempo para la segunda fase = n^2.

Descomposición

Limites de Concurrencia: Ley de Amdahl

$$Speedup \leq \frac{2n^2}{\frac{n^2}{p} + n^2} = \frac{2p}{p+1}$$

Tips: dividir la segunda fase en dos:

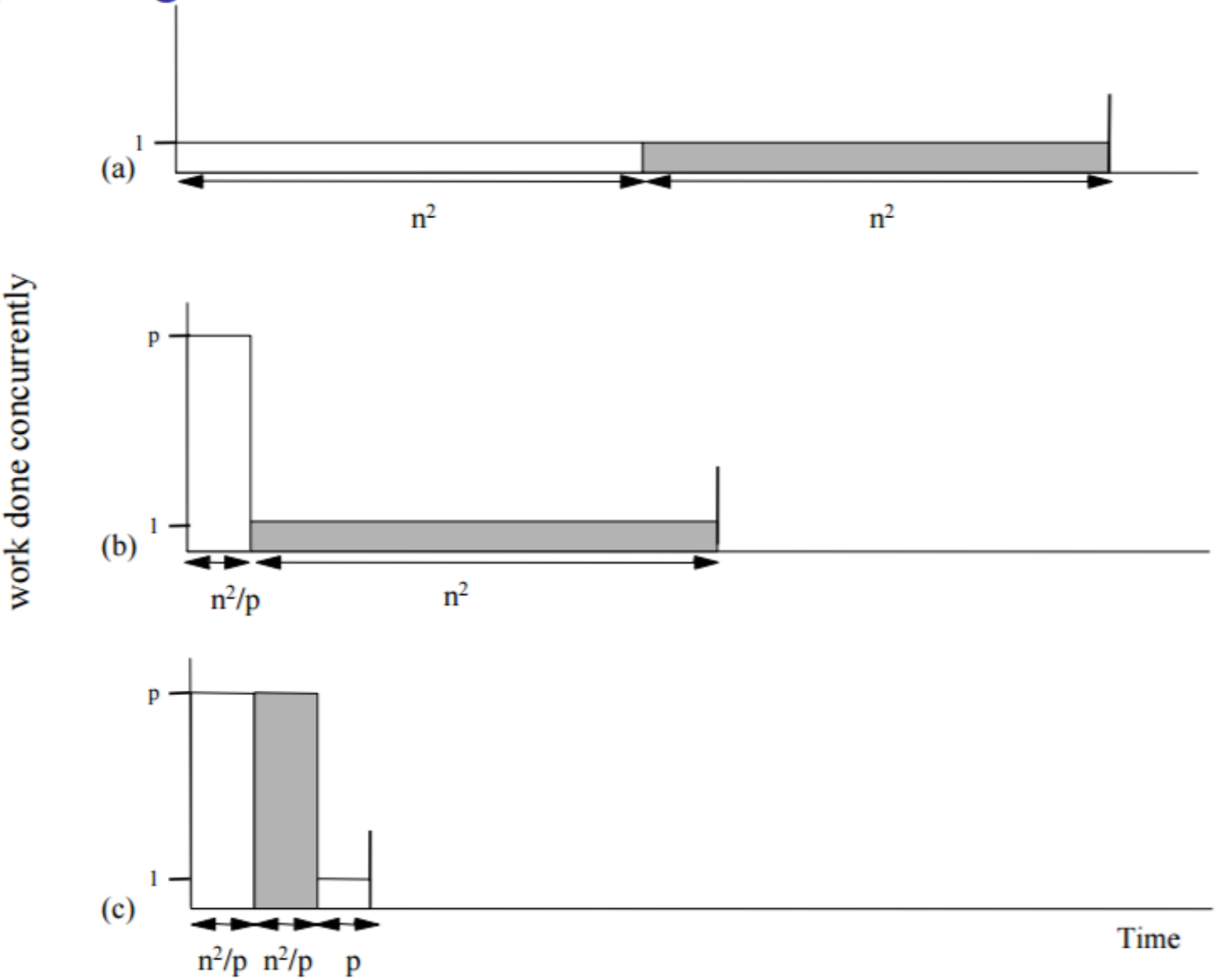
acumular en sumas locales durante el primer barrido,

sumar los valores locales para alcanzar la suma total.

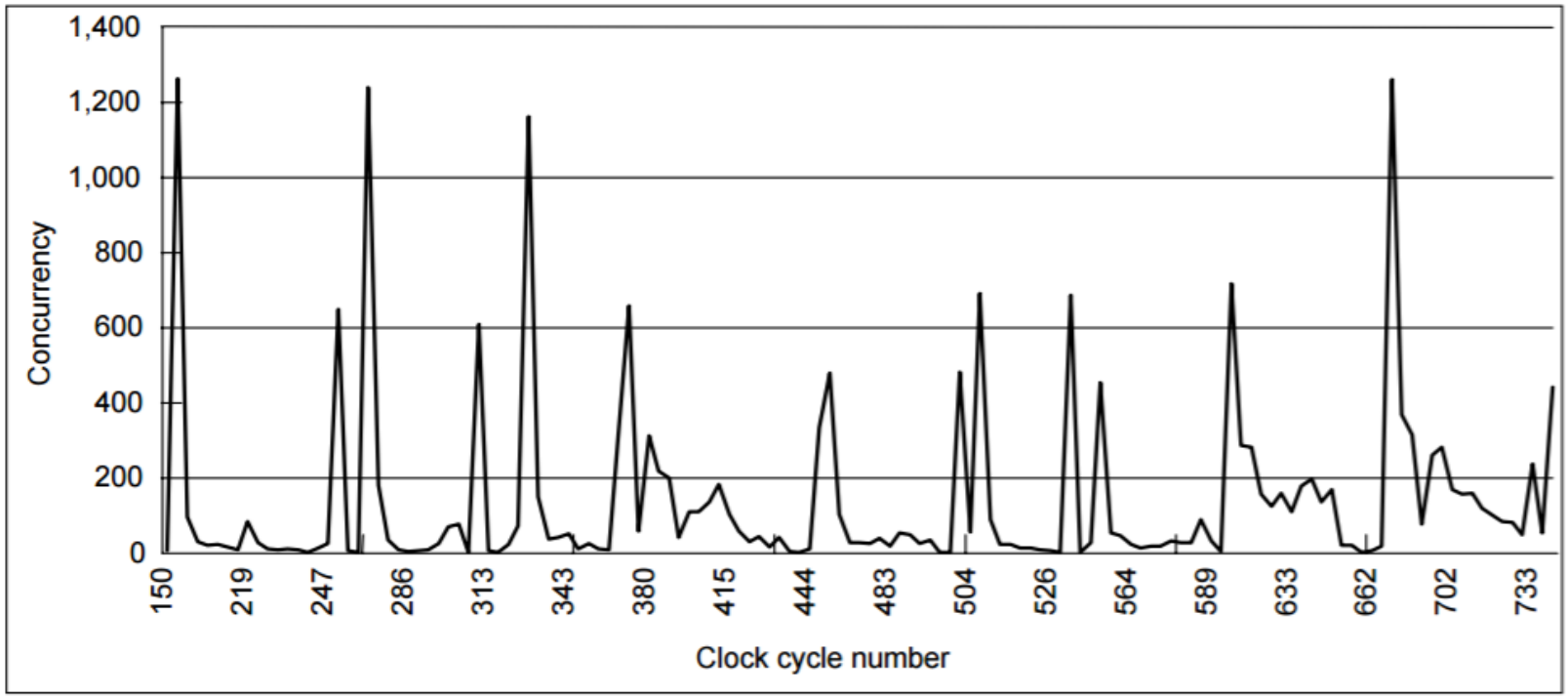
Tiempo $n^2/p + n^2/p + p$, y mejora en *speedup*: $\frac{p\ 2n^2}{2n^2 + p^2}$

si n es muy grande, el speedup es lineal en p.

Descripción gráfica



Perfiles de concurrencia



Simulación lógica de un circuito MIPS R6000. El eje *y* muestra el número de elementos lógicos evaluables para cada ciclo de reloj simulado (eje *x*).

Descomposición

Perfiles de concurrencia

- Área bajo la curva: trabajo total o tiempo para un procesador.
- Extensión horizontal: límite inferior de tiempo (infinitos procesadores).

El *speedup* es: $$\frac{\sum_{k=1}^{\infty} f_k k}{\sum_{k=1}^{\infty} f_k \left\lceil \frac{k}{p} \right\rceil}$$

La ley de Amdahl es aplicable a cualquier tipo de sobrecarga y no sólo al límite de concurrencia.

Asignación

Mecanismo específico para dividir el trabajo entre procesos.

- Junto con la descomposición recibe el nombre de partición.
- Objetivos:
  - equilibrar la carga;
  - minimizar comunicación y coste de la gestión de la asignación.

Aproximarse a las estructuras suele ser buena idea.

- Inspección del código (paralelizar bucles).
- Existen procesos heurísticos ya establecidos.
- Asignación *estática* frente a asignación *dinámica*

La partición es la primera preocupación del programador.

- *Generalmente* independiente de la arquitectura o el modelo de programación.
- Pero el costo y complejidad de las primitivas usadas puede afectar a la decisión.

Orquestación

Para ejecutar las tareas asignadas, los procesos necesitan mecanismos para:

- referirse y acceder a los datos;
- intercambiar datos con otros procesos (comunicación); sincronizar las actividades.

Las decisiones tomadas en la orquestación son muy dependientes

- del modelo de programación;
- de la eficiencia con que las primitivas del modelo están implementadas.

Orquestación

En la orquestación se incluyen, entre otras cosas:

- cómo organizar las estructuras de datos;
- cómo planificar temporalmente las tareas asignadas a un proceso para explotar la localidad de datos;
- cómo organizar la comunicación entre procesos resultante de la asignación.

Orquestación

Objetivos

- Reducir el coste de comunicación y sincronización.
- Preservar la referencias a datos locales (organización de las estructuras de datos).
- Programar las tareas para satisfacer rápidamente las dependencias.
- Reducir la sobrecarga que introduce el paralelismo.

Cercanos a la Arquitectura (y al modelo de programación)

- La elección depende mucho de la comunicación, eficiencia y las primitivas.
- La arquitectura debe proporcionar las primitivas adecuadas de manera eficiente.

Mapeado

Después de la orquestación ya tenemos un programa paralelo.

Dos aspectos del mapeado:

- Qué procesos se ejecutarán en el mismo procesador.
- Qué procesos se ejecutan en un procesador determinado.
  - El mapeado se realiza en una topología determinada.

Mapeado

Un extremo: *space-sharing:*

- La máquina se divide en subconjuntos con sólo una aplicación en un momento dado en un subconjunto.
- Los procesos pueden ser asignados a un procesador o al sistema operativo.

Otro extremo: el sistema operativo maneja todos los recursos.

El mundo real es una mezcla de los dos.

- El usuario manifiesta deseos en algún aspecto pero el sistema puede ignorarlos.