DESARROLLO DE SOFTWARE EN ARQUITECTURAS PARALELAS

1. Motivación y aspectos de la programación paralela.
2. Tipos de sistemas paralelos. Paradigmas de programación paralela.
3. Conceptos básicos y medidas de paralelismo.
4. Diseño de programas paralelos.
   - Paralelización manual o automática.
   - Estudio del problema y del programa.
   - Descomposición del problema.
   - Comunicaciones.
   - Equilibrio de la carga.
   - I/O.
5. La interface de paso de mensaje: el estándar MPI.
6. Paralelización de algoritmos: ejemplos y aplicaciones.

Paralelización manual o automática

- El diseño y desarrollo de programas paralelos es un proceso sustancialmente manual.
- Existen herramientas para asistir en la conversión de programas paralelos: compiladores paralelos y preprocesadores.
- El **compilador paralelo** normalmente trabaja de dos formas:
  - Totalmente automático:
    - El compilador analiza el código e identifica las posibilidades de paralelización.
    - Los principales puntos de paralelización son los bucles.
  - Dirigido por el programador:
    - Se usan directivas de compilación que indican cómo paralelizar.
    - Suelen estar en combinación con el paralelismo automático.

Paralelización manual o automática

- Aunque la paralización automática puede ser la respuesta inicial, puede tener inconvenientes:
  - Pueden producirse resultados erróneos.
  - El rendimiento puede decaer.
  - Mucho menos flexible que la paralelización manual.
  - Limitado a un subconjunto (muchas veces solo bucles) de código.
  - Puede no paralelizar nada si el análisis automático detecta inhibidores del paralelismo o código muy complejo.

- Nos dedicaremos a partir de ahora a la paralelización manual.

Estudio del problema y del programa

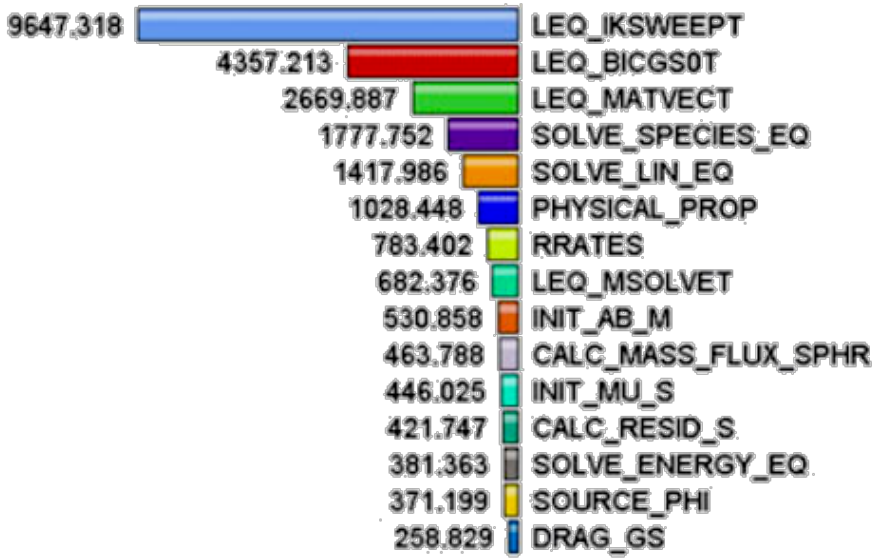
1. El primer paso en el desarrollo de software paralelo es **entender el problema** en cuestión. Y si partimos de un código secuencial, entenderlo también.
2. Determinar si nuestro problema **puede paralelizarse** identificando, en su caso, las **partes independientes**:
   - En el producto escalar de dos vectores cada multiplicación componente a componente es independiente del resto.
   - Por el contrario, en el cálculo de la sucesión de Fibonacci (0,1,1,2,3,5,8,13,21,...) a partir de la relación

     $$F(n) = F(n-1) + F(n-2),$$

     no existen partes independientes.

Estudio del problema y del programa

3. Identificar los puntos calientes (***hotspots***):
   - Conocer dónde se realiza la **mayoría del cálculo**. La mayoría de los programas científicos y técnicos lo hacen en unos pocos puntos.
   - Existen **herramientas de análisis de rendimiento** que pueden ayudar a identificar estos puntos.
   - Nos centraremos en estos puntos, ignorando aquellas partes de código con muy poco uso de CPU.



Estudio del problema y del programa

4. Identificar cuellos de botella (***bottlenecks***):
   - Localizar áreas desproporcionadamente **lentas** o áreas que causan un retraso en el trabajo paralelo. Por ejemplo, **I/O** ocasiona normalmente una lentitud de nuestro programa.
   - Si fuera preciso, sería necesario **reestructurar** nuestro programa o **buscar otro algoritmo** que reduzca estos cuellos de botella.

Estudio del problema y del programa

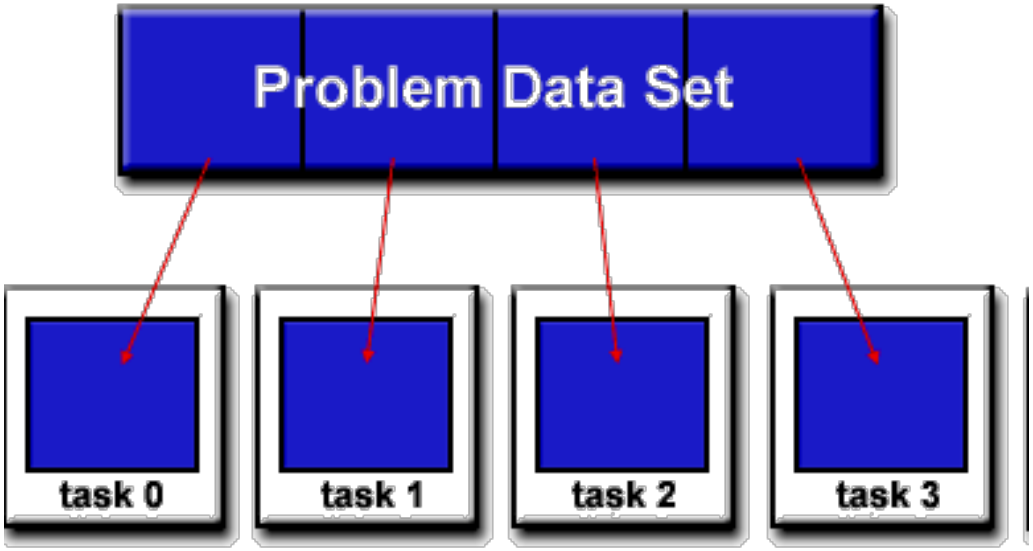
5. Identificar **inhibidores del paralelismo**. Por ejemplo, la dependencia de datos es un gran inhibidor del paralelismo (Fibonacci).
6. Investigar **otros posibles algoritmos**. Esta puede ser una de las consideraciones más importantes en el diseño de una aplicación en paralelo.
7. Utilizar las ventajas que nos ofrece el **software ya desarrollado y altamente optimizado** (librerías matemáticas, software paralelo,...).

Descomposición del problema

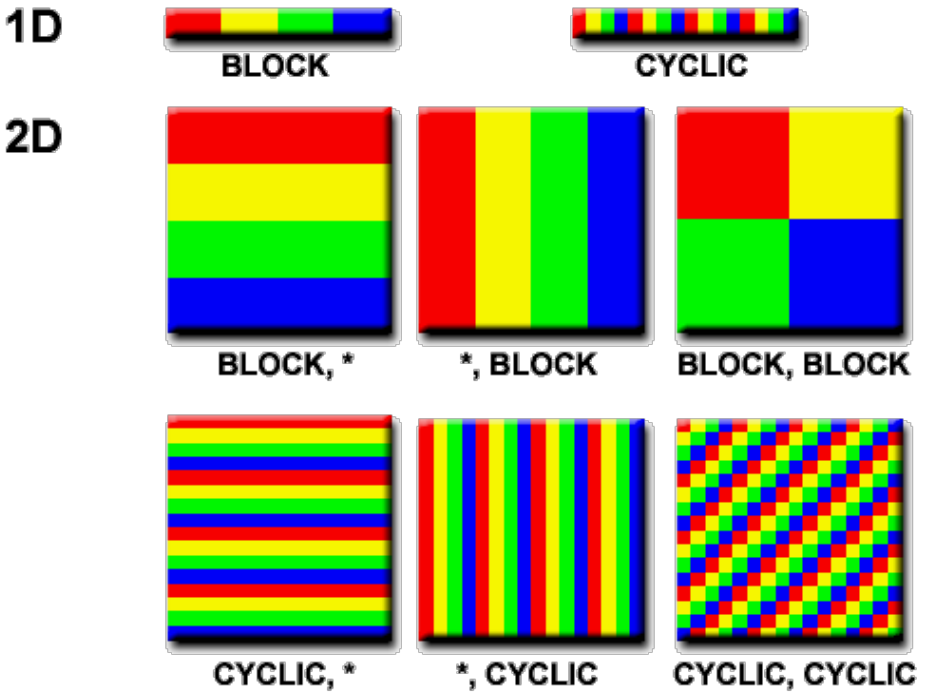
- Uno de los primeros pasos en el diseño de un programa paralelo radica en cómo “romper” nuestro problema en “trozos” de trabajo que puedan ser asignados a distintas tareas. Estamos hablando de la **descomposición** o **particionado** del problema.
- Básicamente existen dos formas de descomponer el trabajo computacional entre distintas tareas paralelas:
  - **Descomposición de dominios**.
  - **Descomposición funcional**.

Descomposición del problema
(Descomp. de dominios)

- En la **descomposición de dominios** los **datos** asociados con el problema se descomponen.
- **Cada tarea** paralela trabaja con **una porción de los datos**.

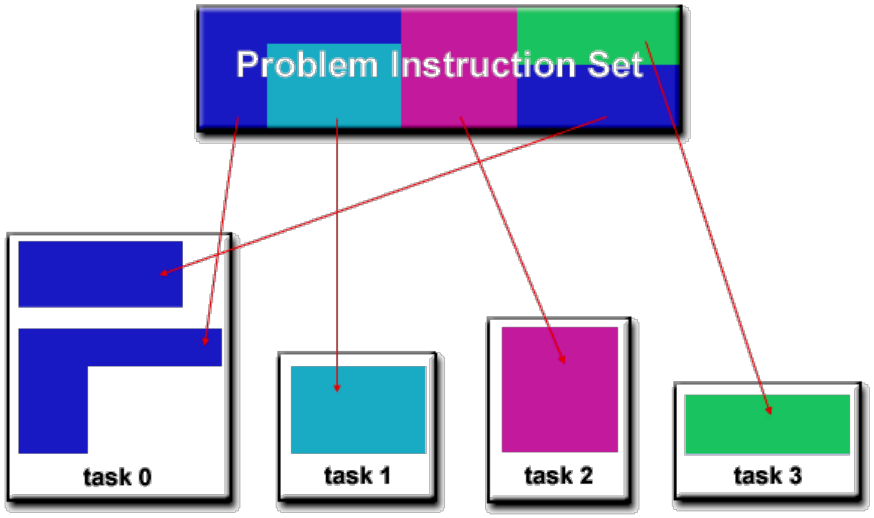


- Diferentes formas de particionar los datos:

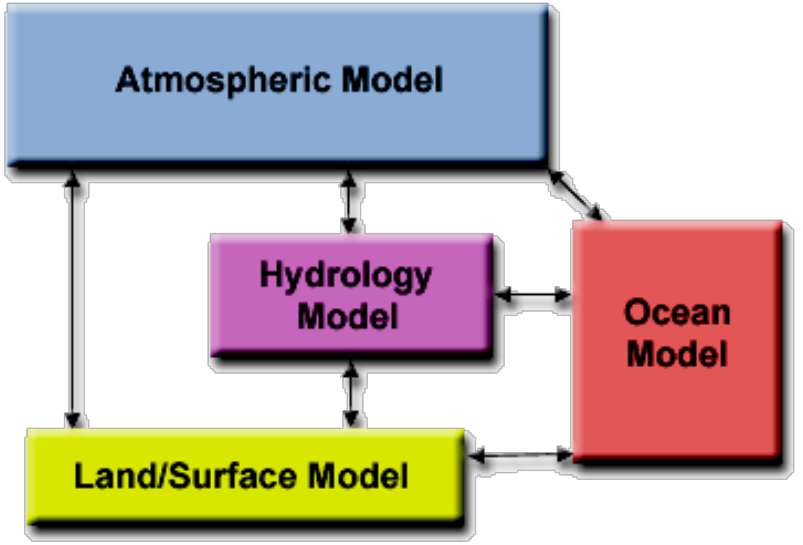


Descomposición del problema
(Descomp. funcional)

- En la **descomposición funcional** nos **centramos en el cálculo** a realizar en lugar de los datos.
- El problema se descompone en función del **trabajo a realizar**.
- Cada tarea realiza una **porción de trabajo** total.



- Un ejemplo: el **modelado del clima**. Cada componente del modelo puede computarse en una tarea separada.



Comunicaciones

- La necesidad de comunicaciones entre tareas depende del problema que estemos tratando:
  - **Pocas o nulas comunicaciones**. Algunos problemas se pueden descomponer y ejecutar en paralelo sin la necesidad de comunicaciones. Por ejemplo, invertir los tonos de grises en una imagen.
  - **Necesitamos comunicaciones**. La mayoría de las aplicaciones paralelas necesitarán comunicar datos y cálculos intermedios entre tareas. Por ejemplo, la aplicación de un filtro de Laplace en una imagen.

Comunicaciones (Factores a considerar)

Existen diversos **factores** que influyen en el diseño de las **comunicaciones** entre tareas:

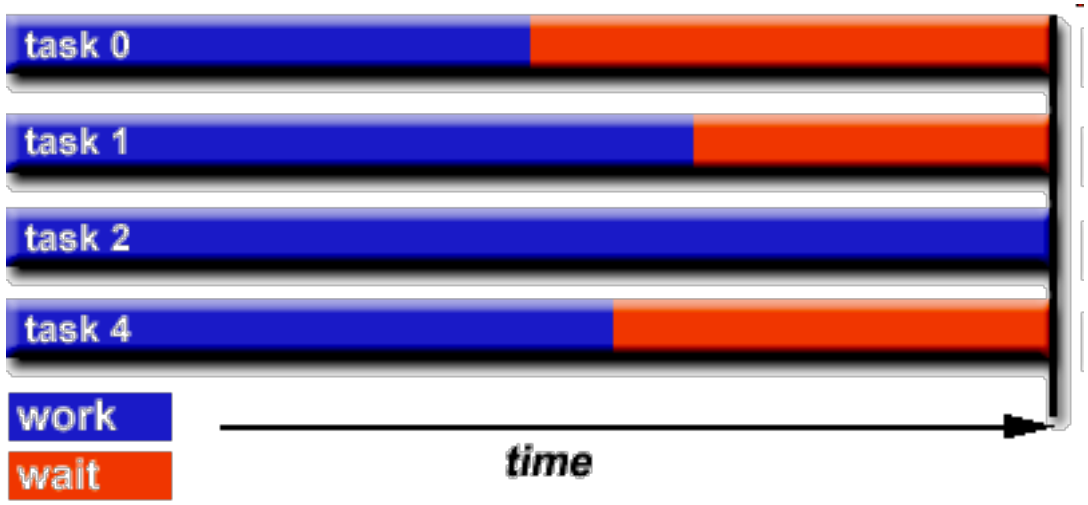
1. **El coste de las comunicaciones:**
   - Se usan recursos para empaquetar datos y transmitirlos.
   - Frecuentemente implican sincronización entre tareas lo cual implica tiempos de espera.
   - El tráfico en la red puede ocasionar problemas de rendimiento.
2. **Latencia vs. ancho de banda:**
   - A menudo es más eficiente empaquetar mensajes pequeños en uno mayor de manera que se minimice la latencia y se aumente el ancho de banda.

Comunicaciones (Factores a considerar)

3. **Comunicación síncrona o asíncrona:**
   - Las comunicaciones síncronas requieren de cierto tipo de protocolo de enlace entre las tareas que comparten los datos.
   - En las comunicaciones asíncronas se permite que las tareas transfieran sus datos de forma independiente.
4. **Ámbito de las comunicaciones:**
   - El conocimiento de qué tareas comunican con otras es crítico en el diseño de nuestra aplicación.
   - **Comunicaciones punto a punto**: involucra a dos tareas, una envía los datos y la otra los recibe.
   - **Comunicaciones colectivas**: más de dos tareas comparten datos de distintas formas.

Equilibrio de la carga

- El **equilibrio de la carga** consiste en distribuir aproximadamente la misma cantidad de trabajo entre todas las tareas de manera que todas permanezcan ocupadas todo el tiempo.
- Puede considerarse como una minimización de los tiempos de inactividad de las tareas.



Equilibrio de la carga

- El **equilibrio de la carga** puede conseguirse de varias formas:
  - **Particionando de forma equilibrada** el trabajo que cada tarea realiza. Por ejemplo:
    - En operaciones matriciales, dividiríamos de forma equitativa los elementos.
    - En bucles con idéntico trabajo en cada iteración, dividiríamos las iteraciones entre las tareas.
  - **Usando una asignación dinámica**:
    - En ocasiones, aunque los datos se distribuyan uniformemente entre las tareas, se ocasiona un desequilibrio de la carga, que puede ser, incluso, impredecible.
    - Es necesario trabajar con un **pool de procesos**:
      - Se generan muchos más trabajos que tareas.
      - Conforme cada tarea finaliza su trabajo, se le asigna uno nuevo.