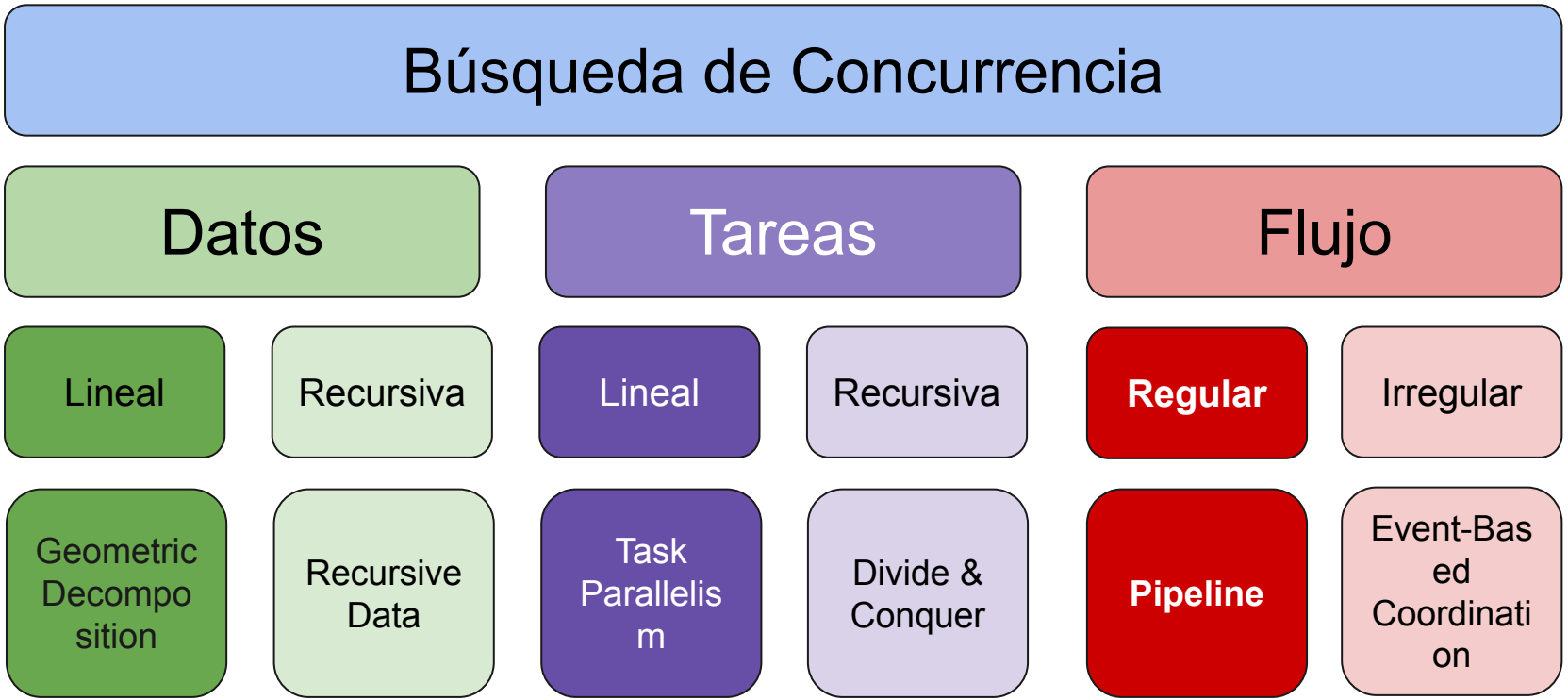
Tema 2
Modelos de programación paralela adaptados a la arquitectura

Nicolás Calvo Cruz
Dpto. de Arquitectura y Tecnología de los Computadores
@ncalvocruz
ncalvocruz@ugr.es

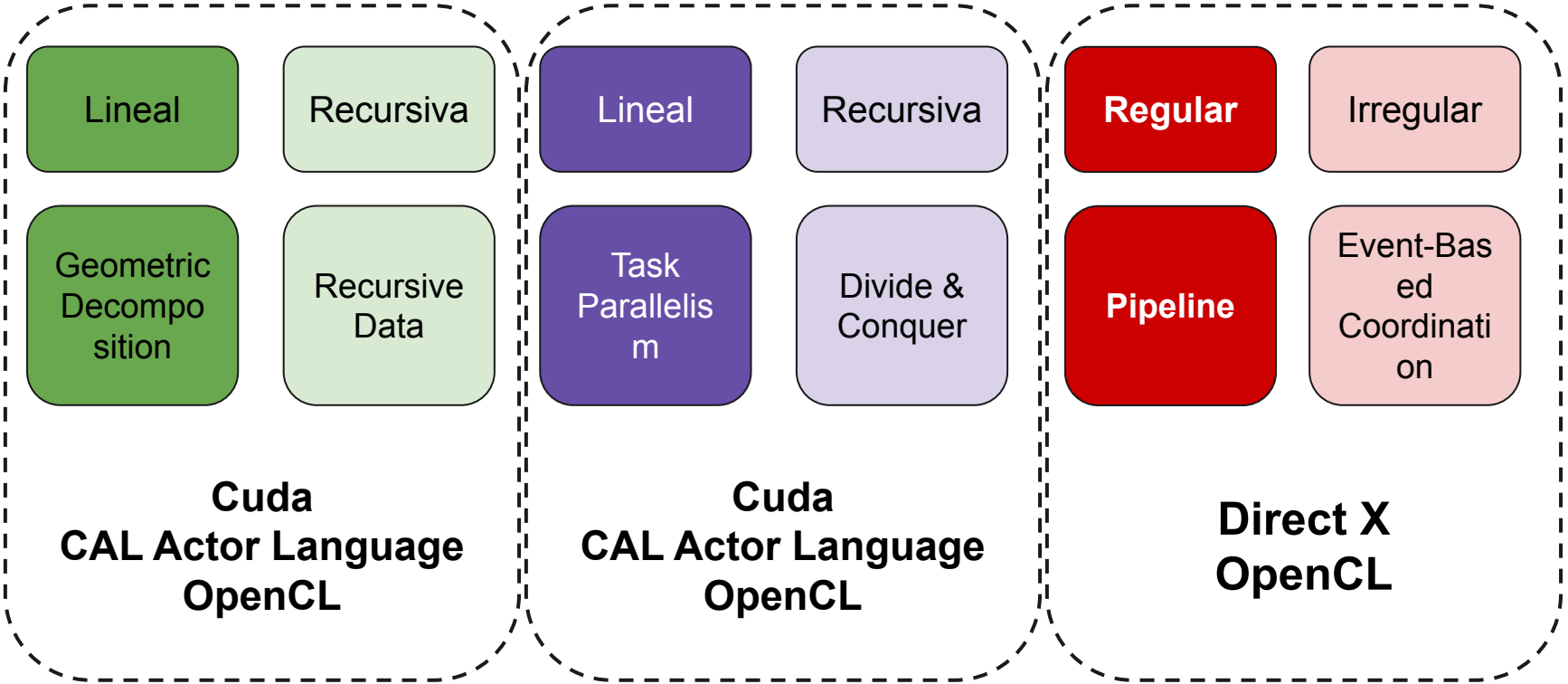
Índice

3. Patrones de los principales algoritmos paralelos

3. Patrones de los principales algoritmos paralelos

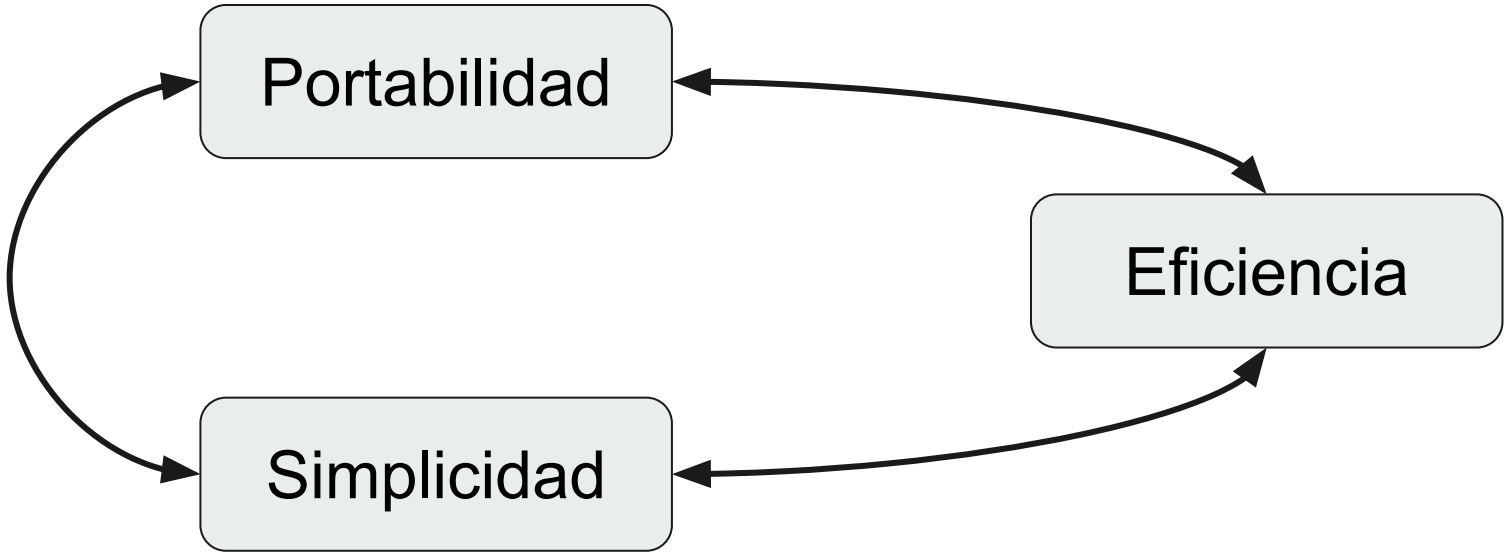


3. Patrones de los principales algoritmos paralelos



3. Patrones de los principales algoritmos paralelos

¿Cómo elegimos la mejor estructura?



3. Patrones de los principales algoritmos paralelos

¿Cómo elegimos la mejor estructura?

Orden de magnitud de UE necesarias

¿Seguimos siendo flexibles?

¿Qué lenguajes tenemos disponibles?

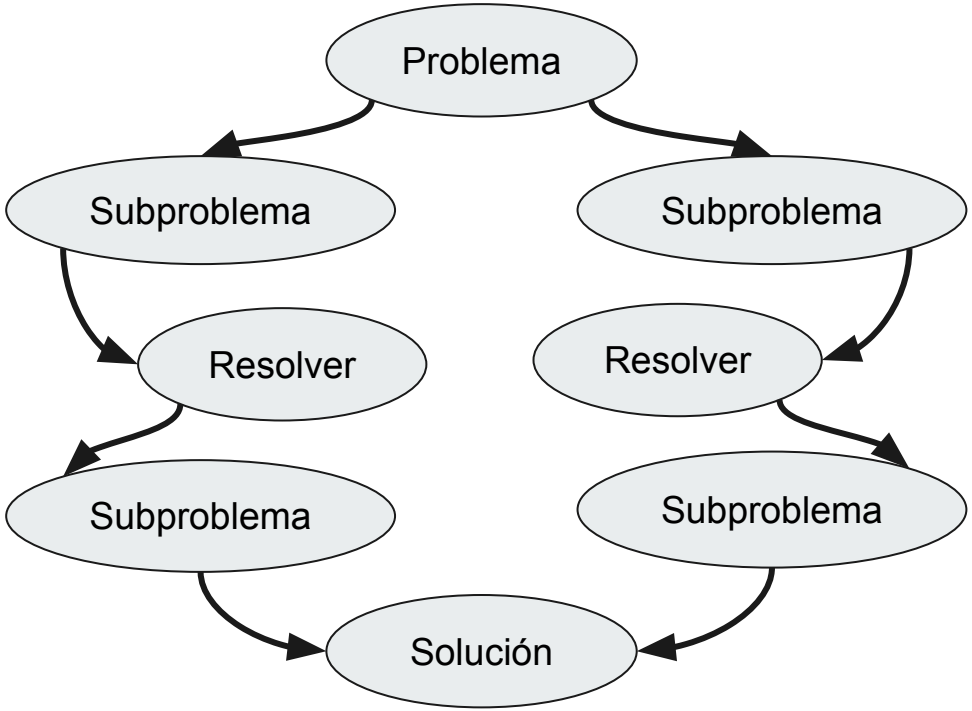
¿Cuánto cuesta la comunicación?

3. Patrones de los principales algoritmos paralelos
3.1 Descomposición por tareas

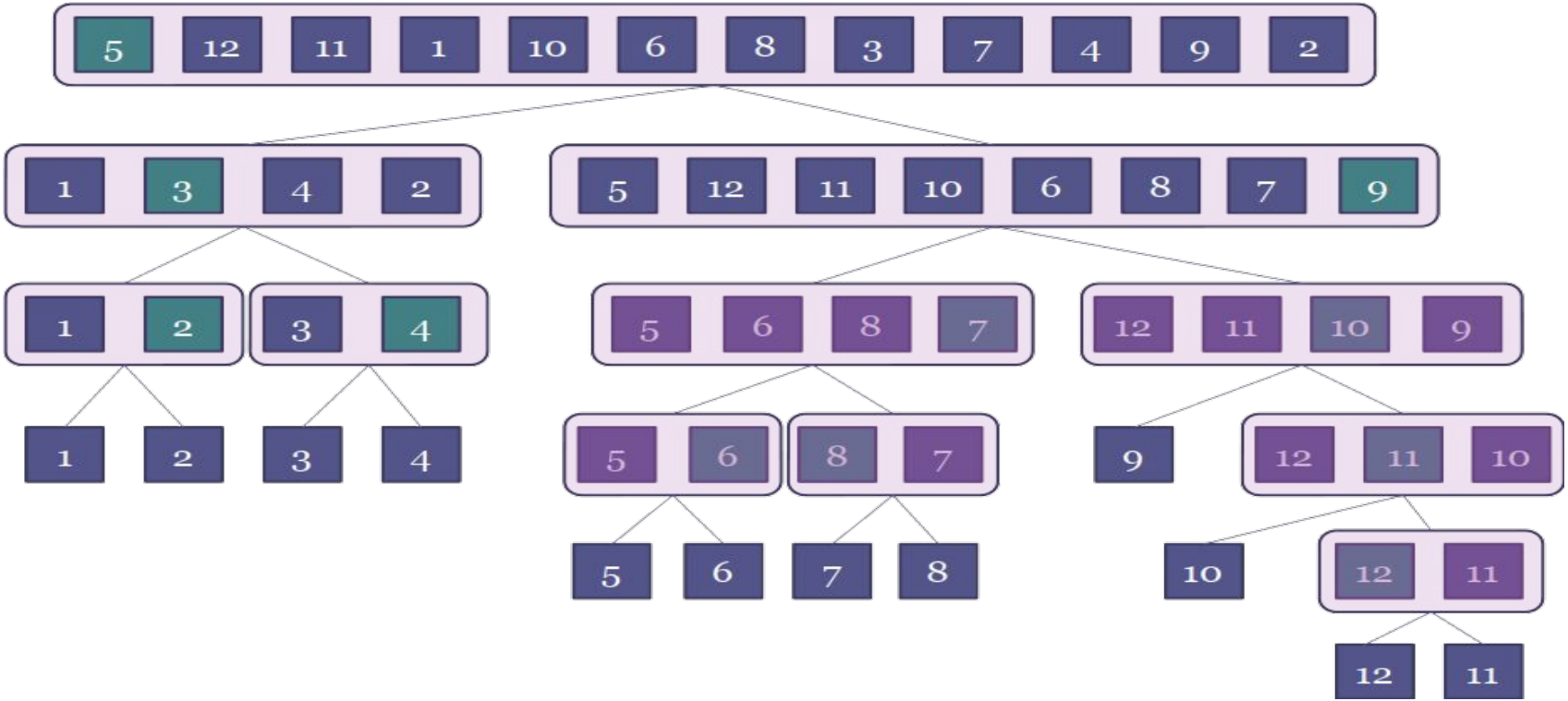
- Las tareas pueden ser:
  - Independientes (Ejemplo de procesado de imágenes)
  - Dependientes
    - Porque acceden a una estructura común
    - Porque tienen que coordinarse por mensajes
- Si las tareas se pueden organizar de forma recursiva: **Divide & Conquer**
- Si las tareas no son recursivas: **paralelismo de tareas general** (hebras o procesos) (*Task Parallelism*)

3. Patrones de los principales algoritmos paralelos
3.1 Descomposición por tareas

- Divide & Conquer
  - Las tareas no tienen porqué ser de igual tamaño
  - Normalmente necesita balanceo de carga



3. Patrones de los principales algoritmos paralelos
3.1 Descomposición por tareas: Quicksort



3. Patrones de los principales algoritmos paralelos
3.2 Descomposición por datos

- Si la estructura se divide siguiendo dimensiones:
  - Algoritmos con una **descomposición geométrica** (Vectores, matrices) (Ej: Multiplicación de matrices)

- Si se divide de forma recursiva:
  - Algoritmos con un esquema recursivo (**Recursive Data**) (Para árboles, grafos, listas,...) (Ej: Encontrar la raíz)

3. Patrones de los principales algoritmos paralelos
3.3 Descomposición por flujo de datos

- **Adecuada** cuando el **flujo de los datos es el que ordena** los grupos de **tareas** definiendo un orden entre ellas
- Si el flujo es regular y estático y suele ser de una sola dirección-> **Pipeline**. Ejemplo compresión JPEG
- Si no: -> **coordinación basada en eventos**. Ejemplo, gestión de las peticiones del servidor web.
    - Monitorización de las peticiones
    - Clasificación
    - Envío para su tratamiento

3. Patrones de los principales algoritmos paralelos
3.4Propuesta de análisis

N-Body problem

http://en.wikipedia.org/wiki/N-body_problem

(Video: https://www.youtube.com/watch?v=Cn2Z8bZDuyw)

```
Array or Real :: atoms(3,N)
Array of Real :: velocities (3,N)
Array or Real :: forces (3,N)
Array or List :: neighbors(N)
Loop over time steps
      vibrational_forces(N, atoms, forces)
      rotational_forces(N, atoms, forces)
      neighbor_list(N, atoms, neighbors)
      non_bonded_forces(N, atoms, neighbors, forces)
      update_atoms_pos_and_vel(N, atoms, velocities,
forces)
      physical_properties(….)
endloop
```

Índice

4. Estructuras de algoritmos más comunes

SPMD: Single Program, Múltiple Data

Todas las hebras/procesos se derivan del mismo código en el que, en última instancia, se acaba operando sobre distintos datos.

Master/Worker: Un proceso organiza el reparto y la recolección de tareas y todos los demás suelen hacer la misma tarea o especializarse en algún tipo de ellas, y se crean y se destruyen de forma dinámica en la mayoría de los casos.

Loop Parallelism: Consiste en la división de iteraciones de bucles en hebras diferentes que normalmente son independientes o se pueden transformar de alguna forma para que lo sean.

Fork/Join: Cada unidad de ejecución se divide tantas veces como haga falta en hijos que luego se reúnen de nuevo cuando el trabajo está realizado.

4. Estructuras de algoritmos más comunes

	Task Parallelism	Divide & Conquer	Descomp. Geométrica	Recursive Data	Pipeline	Event-Based Coordination
SPMD	****	***	****	**	***	**
Loop Parallelism	****	**	***			
Master/Worker	****	**	*	*	*	*
Fork/Join	**	****	**		****	****

4. Estructuras de algoritmos más comunes

	OpenMP	MPI	Java	CUDA (OpenCL)
SPMD	***	****	**	***
Loop Parallelism	****	*	***	*
Master/Worker	**	***	***	*
Fork/Join	***		****	**

4. Estructuras de algoritmos más comunes
1. SPMD

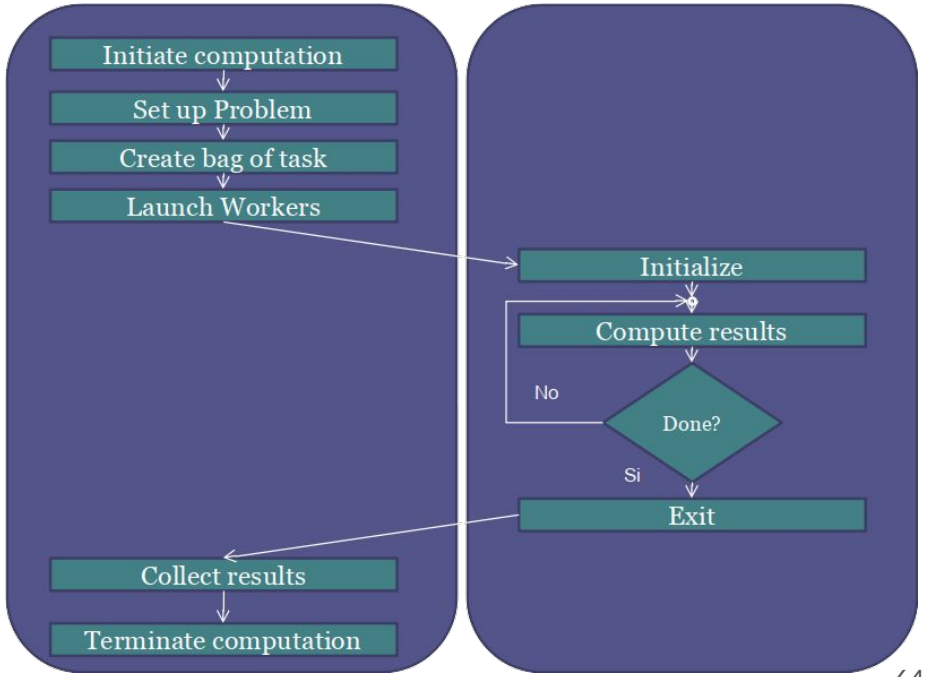
Pasos:

- Inicializaciones
- Obtener identificaciones únicas
- Reparto de trabajo
- Ejecución del trabajo
- Recolección de resultados
- Finalizaciones

4. Estructuras de algoritmos más comunes
2. Maestro - Trabajador

Características:

- Tareas de tamaños diferentes o no predecibles o con dependencias y sin comunicación necesaria entre tareas
- Se necesita balanceo de carga y tiene que estar adaptado a la máquina
- El grueso del trabajo no es un solo bucle
- Las estructuras de computación también pueden ser heterogéneas

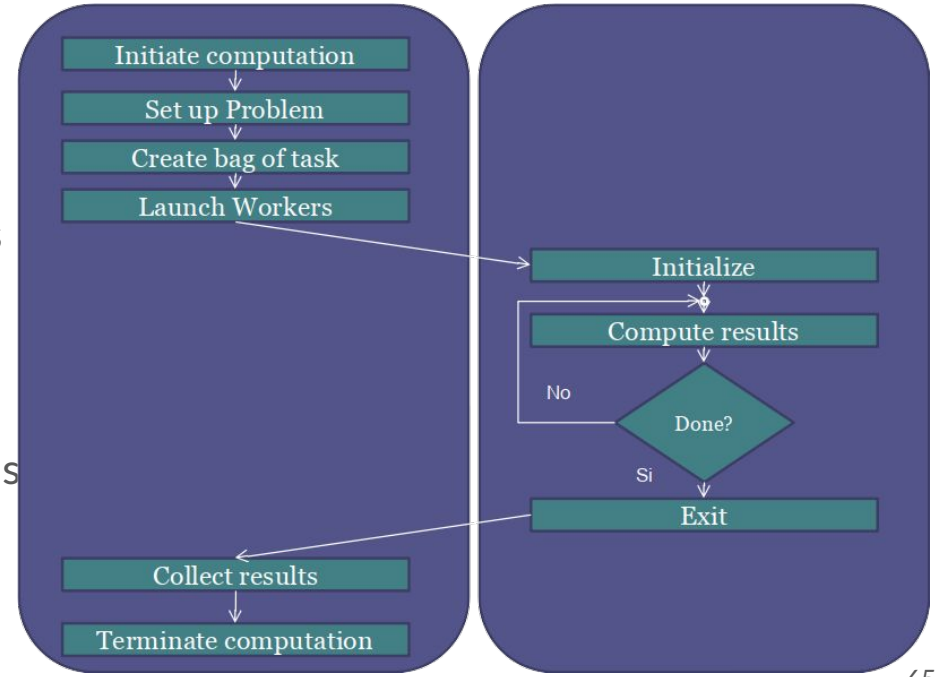


4. Estructuras de algoritmos más comunes
2. Maestro - Trabajador

Problemas:

- Comprobación de finalización.
- Asegurar la robustez incluso si los workers caen
- Implementación de cola de tareas pendientes
- Se puede necesitar jerarquización con colas de tareas también en workers.

Variaciones: El Maestro también es worker



4. Estructuras de algoritmos más comunes
3. Loop Parallelism

Pasos:

- Seleccionar bucles candidatos (bottlenecks con profilers)
- Eliminación de dependencias entre iteraciones sin afectar al resultado final
- Paralelización de bucle
- Optimización de la planificación de iteraciones entre unidades de ejecución

Problemas:

- El principal problema es el acceso simultáneo a posiciones de memoria contiguas de varias hebras
- Sincronización de hebras cuando sea necesaria por operaciones críticas
- Gestión de datos compartidos que podrían no serlo para simplificar

4. Estructuras de algoritmos más comunes
3. Loop Parallelism

Trucos:

- Unir bucles anidados
- Unir bucles adyacentes
- Invertir el orden de los bucles si es posible
- Desenrollar bucles para generar menos hebras y más cantidad de trabajo por hebra

4. Estructuras de algoritmos más comunes
4. Fork/Join

- Se suele usar solo con Java o usando el estándar POSIX, pero es más difícil de controlar.
- Las tareas se relacionan mediante jerarquía.
- El mapeo de trabajo se realiza de dos formas
  - Forma directa : 1 tarea => 1 hebra o 1 proceso
  - Forma indirecta: se asignan tareas a procesadores lo que ya implica un reparto de las hebras o de los procesos, pero los procesos y las hebras no se crean y destruyen durante toda la ejecución.

4. Estructuras de algoritmos más comunes
4. Fork/Join

- Ejemplos:
  - Implementar un patrón de maestro-trabajador con Fork/Join
  - Implementar un patrón de Loop-parallel con Fork/Join

Índice

Estructuras de datos en algoritmos paralelos

Tipos de implementaciones

Las estructuras de datos que son compartidas entre varias unidades de ejecución, hebras o procesos, pueden ser de tres tipos:

- **Shared Data**: hay que mantener coherencia y eso afecta al rendimiento, además de secuencializar los accesos
- **Shared Queue**: Se implementa con colas de tipos abstractos que mantienen corrección en los datos incluso con varias hebras o procesos. Normalmente mediante cerrojos y/o barreras
- **Distributed Array**: vector que se divide entre las tareas o procesos según los datos que necesite cada uno y que no se accede a posiciones por más de una tarea o proceso.

Estructuras de datos en algoritmos paralelos

Puntos clave

Los puntos a tener en cuenta son:

- Si la estructura es compartida o no es necesario, porque nos ahorramos la sincronización
- Si cada tarea escribe en una parte y las demás tareas solo leen, tampoco es muy problemático.

Como reglas generales:

- El resultado debe ser el mismo ante accesos múltiples o no.
- Los accesos a estructuras deben ser mínimos para evitar secuencializar las lecturas.
- Se podría limitar el número de accesos, por ejemplo en las sesiones de una BBDD.

Estructuras de datos en algoritmos paralelos

Pasos a seguir

- Asegurate de que la estructura es necesaria
- Define un tipo de dato abstracto (ADT) donde tengas bien definidas las operaciones básicas que se pueden realizar sobre dicho tipo.
- Evalúa la posibilidad de implementar un protocolo de acceso a la estructura que podrá ser:
  - Acceso One-At-A-Time:
    - Bloquear con mecanismos de cerrojos y hebras.
    - Pedir acceso al proceso que tiene el acceso mediante un mensaje.
- Establecer procesos o hebras que solo hagan una tarea, o leer del ADT o escribir en ella, facilitando la programación.
- Reducir las secciones críticas al mínimo.

Gracias.