

SISTEMAS EN TIEMPO REAL

Técnicas de particionado software

Francisco Andrés Candelas Herías

fcandela@dfists.ua.es

Grupo de Automática, Robótica y Visión Artificial

Dpto. Física, Ingeniería de Sistemas y Teoría de la Señal



Universitat d'Alacant
Universidad de Alicante

Contenido

1. **Introducción.**
2. El particionado de software.
 - 2.1. Identificación de tareas.
 - 2.2. Formalización del problema.
3. Los algoritmos de particionado.
 - 3.1. Tendencias.
 - 3.2. Ejemplo de algoritmo.
4. La división de los módulos software.

1. Introducción

Particionado de software:

- Estructura general de un programa pensada para **optimizar** su implementación.
- **Identificación** de tareas de un proceso de tiempo real que serán objeto de una posterior **planificación**.
- **Agrupamiento** de módulos conceptuales elementales, o **división** de operaciones de alto nivel.

Contenido

1. Introducción.
2. El particionado de software.
 - 2.1. Identificación de tareas.
 - 2.2. Formalización del problema.
3. Los algoritmos de particionado.
 - 3.1. Tendencias.
 - 3.2. Ejemplo de algoritmo.
4. La división de los módulos software.

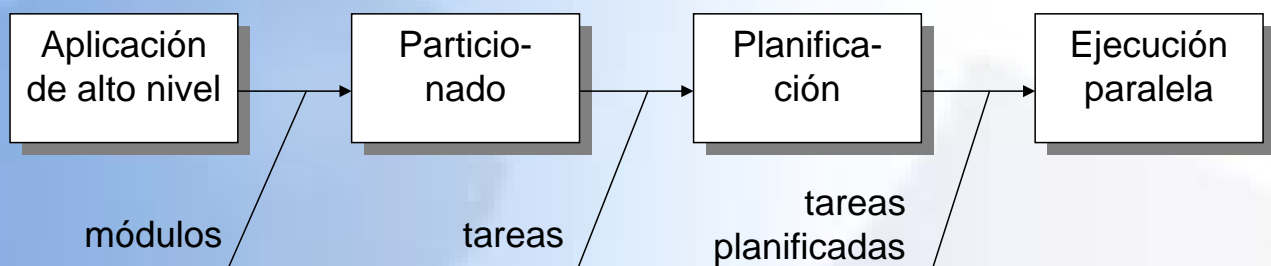
2. Particionado de Software

2.1. La identificación de tareas.

- **Tarea:** entidad que debe procesar una entrada con requerimientos temporales específicos...
 - Relativos a la propia tarea.
 - En relación a otras tareas.
- Puede requerirse **planificación** espacial y temporal para cumplir requisitos en la ejecución.
- Antes de planificar es necesario **identificar** las tareas; etapa de particionamiento.

2. Particionado de Software

2.1. La identificación de tareas.



2. Particionado de Software

2.1. La identificación de tareas.

- Uno de los primeros pasos que debe realizar el diseñador tras la especificación.
- La identificación manual requiere experiencia.
- Si pueden establecerse unos criterios a cumplir:
 - Minimizar tiempo de respuesta del sistema.
 - Utilización efectiva de los recursos.
 - Disminución de costes de comunicación entre tareas.

Contenido

1. Introducción.
2. El particionado de software.
 - 2.1. Identificación de tareas.
 - 2.2. Formalización del problema.
3. Los algoritmos de particionado.
 - 3.1. Tendencias.
 - 3.2. Ejemplo de algoritmo.
4. La división de los módulos software.

2. Particionado de Software

2.2. Formalización del problema.

- El grafo de módulos de entrada:

$$(M, \alpha_m, \varepsilon, \delta)$$

$$M = \{m_1, m_2, \dots\} \neq \emptyset$$

$\alpha_m: M \leftrightarrow M$. No reflexiva, pero puede ser simétrica o antisimétrica

$m_1 \alpha_m m_2 = \text{Cierto} \Leftrightarrow m_1$ se ejecuta inmediatamente antes de m_2

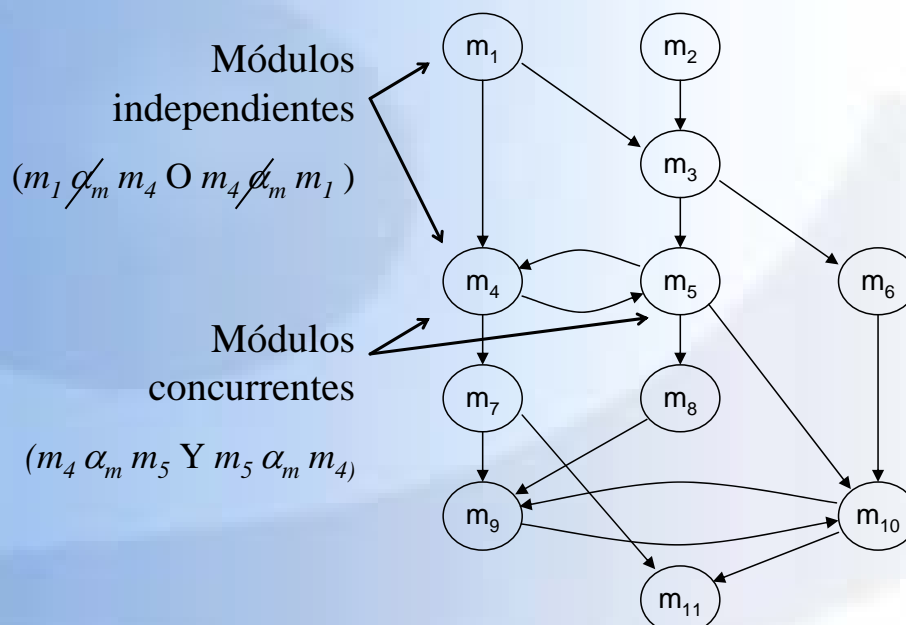
$m_1 \alpha_m m_2 = \text{Falso}$ en otro caso

$\varepsilon: M \rightarrow E$. Donde $e \in E$ es un valor de coste de ejecución

$\delta: M \rightarrow D$. Donde $d \in D$ es un valor de coste de almacenamiento

2. Particionado de Software

2.2. Formalización del problema.



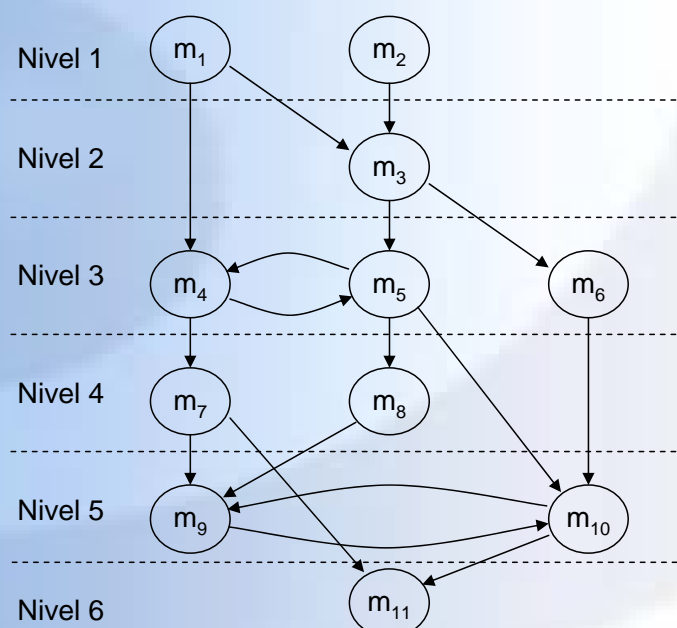
2. Particionado de Software

2.2. Formalización del problema.

- Clasificación de módulos en niveles según α_m :
- Los módulos que no tienen precedentes en α'_m están en el nivel 1.
(α'_m se define a partir de α_m como las relaciones de precedencia de módulos no concurrentes; grafo acíclico)
- Un módulo con exactamente un solo predecesor inmediato en el nivel i está en el nivel $i+k$, para un k minimal de $\{1, 2, \dots\}$.
- Un módulo con varios predecesores inmediatos alguno de los niveles $\{i, i+1, \dots, i+k\}$, está en el nivel $i+k+1$.
- Los módulos concurrentes están en el mismo nivel.

2. Particionado de Software

2.2. Formalización del problema.



2. Particionado de Software

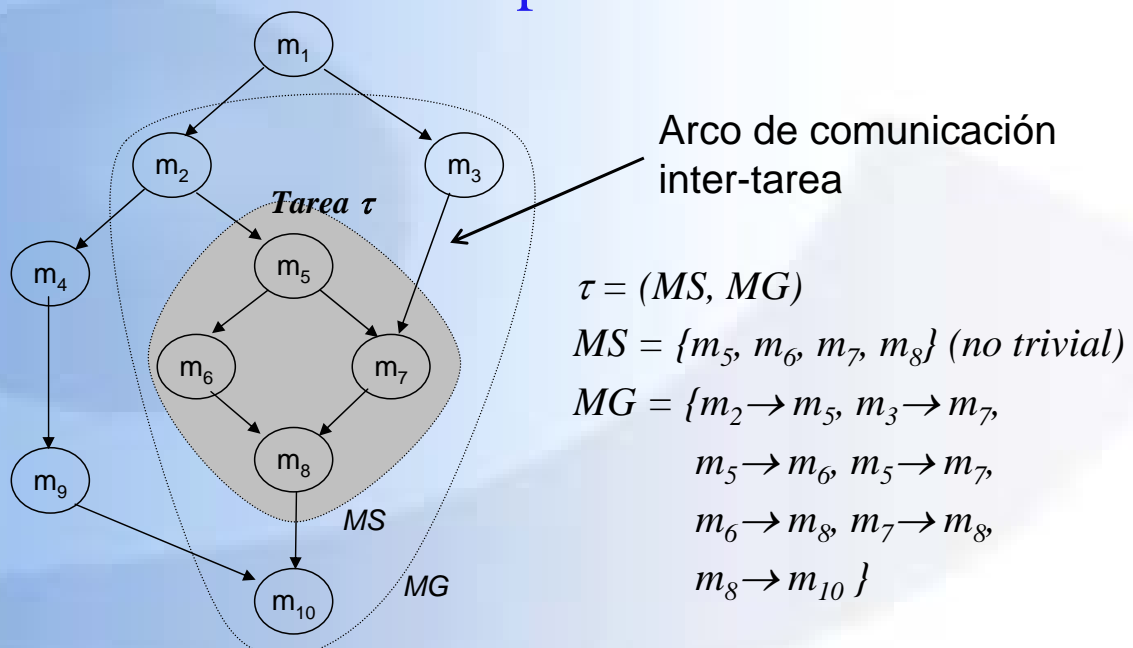
2.2. Formalización del problema.

- Definiendo las tareas:
 - Tarea: agrupación de módulos en conjuntos no vacíos y disjuntos entre si dos a dos.
 - Cada tarea debe estar de acorde con las relaciones de dependencia entre módulos.
 - Conjunto de posibles tareas:

$$T = \{(MS, MG), MS: \text{Particiones}(M), MG: M \leftrightarrow M / MS \neq \emptyset \text{ Y } MG \subseteq \alpha_m \\ \text{Y } (\forall m_1, m_2: M, (m_1, m_2) \in MG \Rightarrow m_1 \in MS \text{ O } m_2 \in MS)\}$$

2. Particionado de Software

2.2. Formalización del problema.



2. Particionado de Software

2.2. Formalización del problema.

Con el particionamiento se pretende obtener un conjunto de tareas P que será una partición del conjunto de posibles tareas T .

$$P \in Particiones(T)$$

2. Particionado de Software

2.2. Formalización del problema.

- Conceptos sobre ejecución de tareas:

- a) Un **módulo es completamente ejecutable** \Leftrightarrow no tiene módulos predecesores O todos sus módulos predecesores ya han sido ejecutados.
- b) Una **tarea es completamente ejecutable** \Leftrightarrow todos sus módulos son completamente ejecutables.
- c) Una **tarea está propiamente activada** si, cuando llega la hora de su activación, tiene acceso a todos los datos externos requeridos por sus módulos del nivel menor.
- d) Una **tarea puede ser propiamente ejecutada** \Leftrightarrow ninguno de sus módulos del nivel menor precede ni sucede a la vez un módulo fuera del conjunto de módulos de la tarea.

2. Particionado de Software

2.2. Formalización del problema.

- Restricciones o condiciones para particionar:

- a) Las debidas a las restricciones de implementación de las propias tareas:

$$P \in \text{Particiones}(T)$$

$$\tau \in P \quad E(\tau) + I(\tau) \leq T_{\max} \quad Y \quad D(\tau) \leq D_{\max}$$

$E(\tau)$, $I(\tau)$: costes directo e indirecto de ejecución de τ

$D(\tau)$: requerimientos de memoria de τ

- b) Cualquiera dos tareas de P no deben compartir un mismo módulo.

$$c) U(MS, \forall (MS, MG) \in P) = U \{m / \exists m' Y (m \alpha_m m' \vee m' \alpha_m m)\}$$

$$d) \forall (MS, MG) \in P \Rightarrow MG \subset \alpha_m \quad Y \quad \alpha_m = U \{MG / \exists MS Y (MS, MG) \in P\}$$

2. Particionado de Software

2.2. Formalización del problema.

- Tipos de ejecución de las tareas:

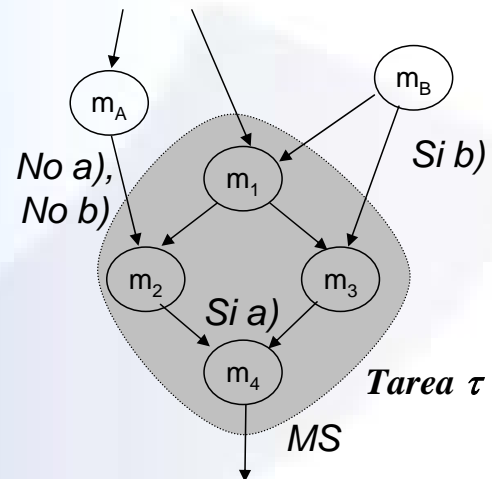
- I. Tareas que devuelven los resultados producidos por sus módulos según van acabando estos.
- II. Tareas que no devuelven sus resultados hasta que la tarea no ha terminado entera.

2. Particionado de Software

2.2. Formalización del problema.

Para que una tarea τ de tipo I propiamente activada sea completamente ejecutable:

- a) Los módulos predecesores inmediatos de cada módulo de τ que no esté en el nivel inferior de τ deben estar en el conjunto de módulos de τ ...
- b) **O** cada uno de los predecesores inmediatos de cada módulo de τ que no pertenecen a su conjunto de módulos también preceden a los módulos en el nivel inferior de τ .



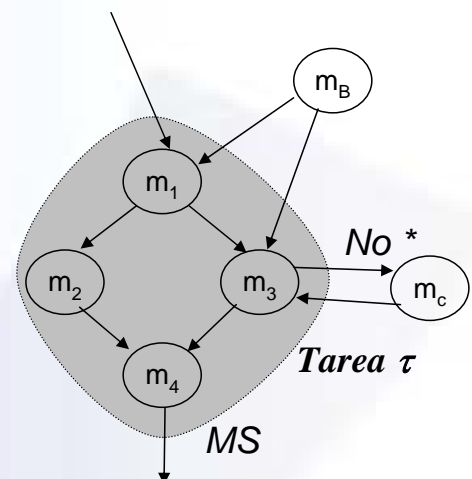
2. Particionado de Software

2.2. Formalización del problema.

Observación:

- * Un tarea no trivial es completamente ejecutable \Leftrightarrow ninguno de sus módulos puede preceder y suceder a otro módulo que no pertenece al conjunto de módulos de la tarea.

Esquema de particionado I.

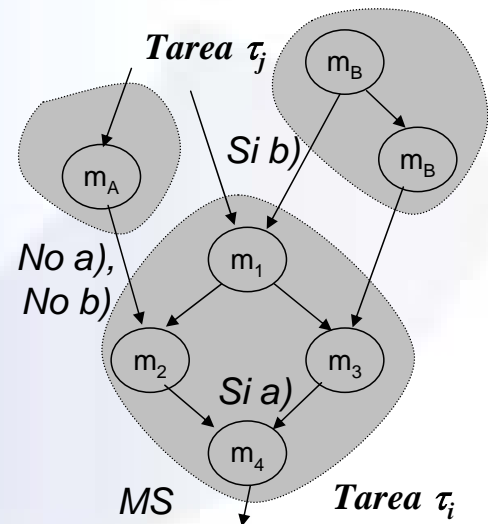


2. Particionado de Software

2.2. Formalización del problema.

Para que una tarea τ_i de tipo II P.A. sea C.E.:

- a) Los módulos predecesores inmediatos de cada módulo de τ_i que no esté en el nivel inferior de τ_i deben estar en el conjunto de módulos de τ_i ...
- b) **O** cada uno de los predecesores inmediatos de cada módulo en τ_i que no pertenezca al conjunto de módulos de τ_i debe pertenecer al conjunto de módulos de otra tarea τ_j , la cual contiene un módulo que precede como mínimo uno de los módulos de menor nivel de τ_i .



2. Particionado de Software

2.2. Formalización del problema.

La observación * anterior sigue siendo aplicable.

Esquema de particionado II.

2. Particionado de Software

2.2. Formalización del problema.

- Esquemas de particionado:
 - I. Las tareas no dependen de que otras tareas acaben para obtener sus datos.
 - II. Ninguna tarea que depende de otra puede comenzar su ejecución hasta que la primera acabe.

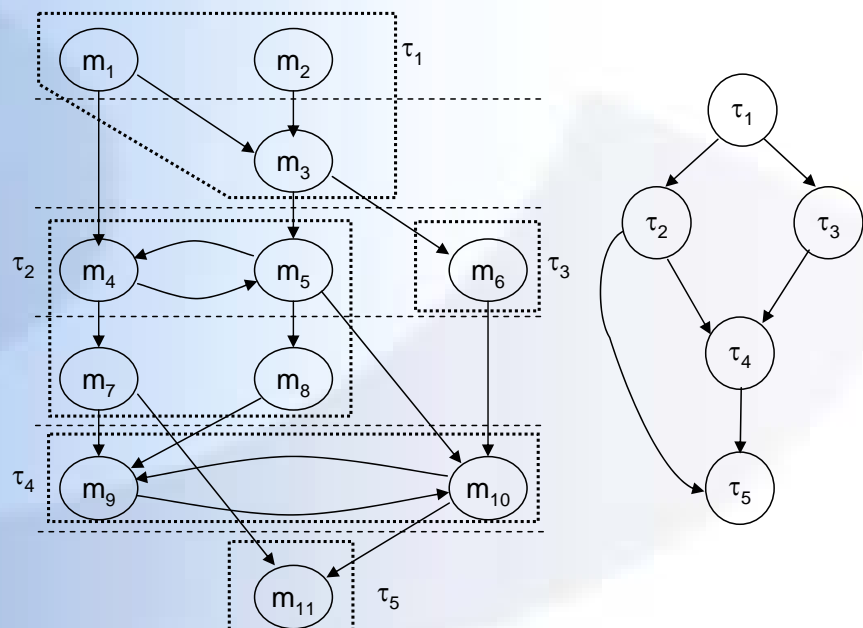
En ambas se debe satisfacer las restricciones planteadas anteriormente.

2. Particionado de Software

2.2. Formalización del problema.

- Ejemplos:

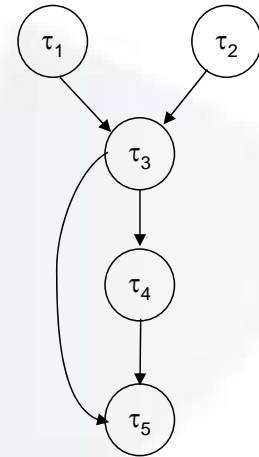
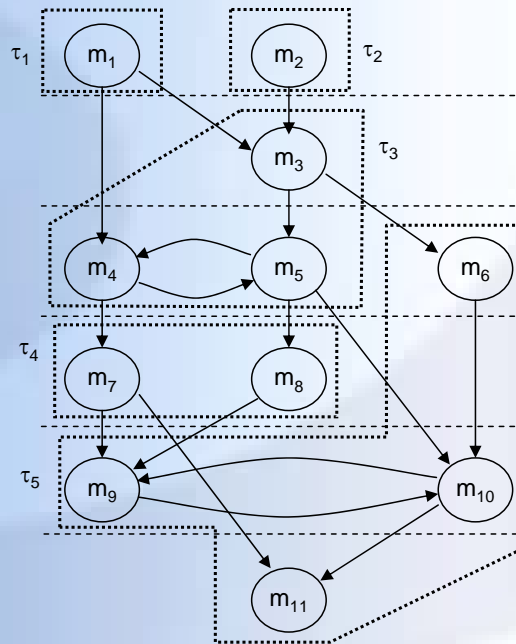
**Cumple
esquemas
I y II**



2. Particionado de Software

2.2. Formalización del problema.

No cumple
esquemas
I ó II



Técnicas de particionado software

25

Contenido

1. Introducción.
2. El particionado de software.
 - 2.1. Identificación de tareas.
 - 2.2. Formalización del problema.
3. Los algoritmos de particionado.
 - 3.1. Tendencias.
 - 3.2. Ejemplo de algoritmo.
4. La división de los módulos software.

Técnicas de particionado software

26

3. Algoritmos de Particionado

- Para aislar al diseñador de detalles de la arquitectura del sistema.
- Menos soluciones para algoritmos de particionado que para planificación.
- En general un algoritmo que de solución óptima al problema de particionado es un problema NP completo.
- Algoritmos basados en heurísticas.
- Difícil llegar a soluciones cercanas a la óptima.

Contenido

1. Introducción.
2. El particionameinto de software.
 - 2.1. Identificación de tareas.
 - 2.2. Formalización del problema.
3. Los algoritmos de particionado.
 - 3.1. Tendencias.
 - 3.2. Ejemplo de algoritmo.
4. La división de los módulos software.

3. Algoritmos de Particionado

3.1.Tendencias.

- Tomar como **entrada un grafo dirigido acíclico** (*Directed Acyclic Graph* o DAG) ponderado.
- Costes operativos de comunicación adicionales entre PEs (*overhead*) **no nulos**.
- Suponer la **disponibilidad de infinitos elementos de proceso** o PEs. Esto no supone limitación.
- **Sucesivas agrupaciones** o particionados de los módulos iniciales o particiones previamente obtenidas hasta llegar un DAG final de tareas.

3. Algoritmos de Particionado

3.1.Tendencias.

- **Heurísticas basadas en la longitud del camino crítico** (*Critical Path Length* o CPL).
- Particionamiento en **tiempo de compilación**.

Otras propuestas....

- Transformación del DAG en un árbol (binario), practicando instancias paralelas con alta relación comunicación/computación.
- Combinación de las técnicas de particionamiento con las de planificación en un mismo algoritmo.

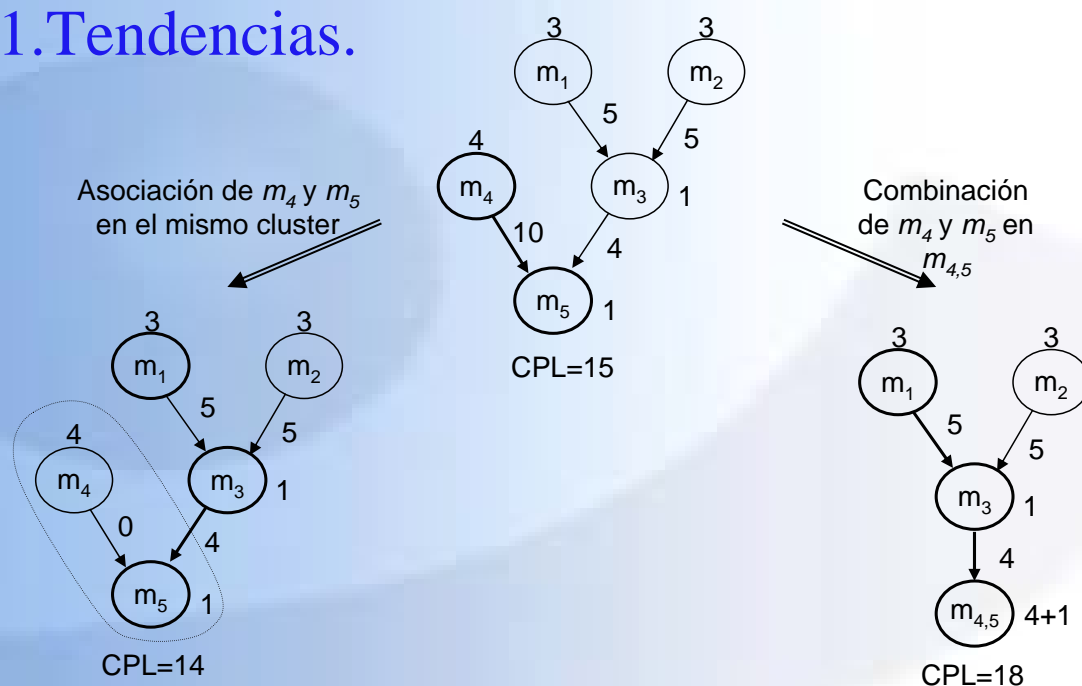
3. Algoritmos de Particionado

3.1.Tendencias.

- Se plantean distintas estrategias de agrupamiento para obtener las tareas, entre las que destacan:
 - **Combinación** (*merging*). Cuando dos módulos se agrupan, esto da lugar a una nueva tarea y algunos arcos pueden ser sustituidos.
 - **Asociación** (*clustering*). Simplemente todos los módulos de un mismo agrupamiento deben ser ejecutados en el mismo PE. No se altera módulos ni arcos.

3. Algoritmos de Particionado

3.1.Tendencias.



Contenido

1. Introducción.
2. El particionado de software.
 - 2.1. Identificación de tareas.
 - 2.2. Formalización del problema.
3. Los algoritmos de particionado.
 - 3.1. Tendencias.
 - 3.2. Ejemplo de algoritmo.
4. La división de los módulos software.

3. Algoritmos de Particionado

3.2. Ejemplo de algoritmo.

- Propuesto por Moez Ayed y Jean-Luc Gaudiot.
- Para un sistema multiprocesador con memoria distribuida.
- En tiempo de compilación.
- Combinación de tareas.
- Coste $O(e \cdot n^3)$, $e = \text{nº arcos}$ y $n = \text{nº de nodos del DAG}$.

3. Algoritmos de Particionado

3.2. Ejemplo de algoritmo.

- Suposiciones:
 - DAG de entrada ponderado.
 - Entradas del DAG disponibles al iniciar ejecución.
 - Tareas no interrumpibles tras comenzar ejecución y esquema II.
 - Sistema destino con enlaces punto a punto entre los procesadores.
 - Infinito numero de procesadores.
 - Costes adicionales de comunicación mínimos no nulos.

3. Algoritmos de Particionado

3.2. Ejemplo de algoritmo.

- Definiciones:
 - Sea G el DAG a particionar, Π es una partición de G :
 $\Pi = \{\tau_1, \tau_2, \dots, \tau_n\}$, $\tau_i \neq \emptyset$ Y $(\tau_i \cap \tau_j) = \emptyset$, $i, j = 1, 2, \dots, n$
 τ_i es una **tarea** que se ejecuta toda en un PE.
 - **Partición trivial**: $\Pi = \{\tau_1, \tau_2, \dots, \tau_n\}$ con $\text{Card}(\tau_i) = 1$
 - Nodo **raíz** de G : el que no tiene predecesores.
 - Nodo **hoja** de G : el que no es predecesor.

3. Algoritmos de Particionado

3.2. Ejemplo de algoritmo.

- **Camino de ejecución** de G : camino que parte de un nodo de raíz y llega a un nodo hoja.
- **Camino crítico** de G : Camino de ejecución de longitud máxima **CPL**.
- Dos nodos de G son **dependientes (independientes)** \Leftrightarrow Hay (no hay) un camino de ejecución entre ellos.
- **ParTime**: Coste temporal de ejecución en paralelo del grafo de tareas obtenido de una partición.

3. Algoritmos de Particionado

3.2. Ejemplo de algoritmo.

- Procedimiento:
 - Comenzar con la *partición trivial*.
 - Secuencia de refinamientos. En cada paso se combinan parejas de tareas según una **Heurística** y se calcula el *ParTime* de la nueva partición.
 - Parar al alcanzar la partición de un solo elemento.
 - Elegir la partición con menor el *ParTime*.

3. Algoritmos de Particionado

3.2. Ejemplo de algoritmo.

- Heurística:

$$ParTime = T_c \text{ (computación)} + T_o \text{ (overhead comunicación)}$$

Más paralelismo $\Rightarrow T_c \downarrow, T_o \uparrow$

Menos paralelismo (combinación módulos) $\Rightarrow T_c \uparrow, T_o \downarrow$

Objetivo: reducir T_o al combinar tareas, aunque disminuya algo el paralelismo.

3. Algoritmos de Particionado

3.2. Ejemplo de algoritmo.

Combinación de tareas independientes $\Rightarrow T_o$ no varía, hay pérdida de paralelismo \Rightarrow No se gana nada.

Combinación de tareas conectadas por un arco $\Rightarrow T_o$ menor, posible pérdida de paralelismo \Rightarrow Se puede ganar algo.

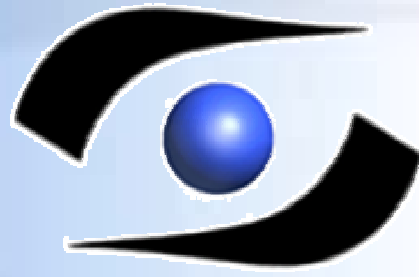
Regla: Combinar solo pares de tareas conectadas por un arco.

Contenido

1. Introducción.
2. El particionado de software.
 - 2.1. Identificación de tareas.
 - 2.2. Formalización del problema.
3. Los algoritmos de particionado.
 - 3.1. Tendencias.
 - 3.2. Ejemplo de algoritmo.
4. La división de los módulos software.

3. La división de módulos

- Etapa previa o que sustituye al particionado.
- Pretende aumentar paralelismo en programas con módulos de muy alto nivel.
- Divide módulos del programa en tareas más operaciones.
- Algunos autores también hablan de *partitioning*, aunque en más correcto *splitting*.
- Para evitar altos costes de comunicación, el resultado se pasa a una etapa de particionado.



Fin

Muchas gracias por vuestra atención