Síntesis arquitectónica y de alto nivel

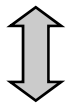
Módulo 2: Planificación temporal de operaciones (scheduling)

Módulo 1. Planificación Temporal de Operaciones

- Contenido
  1. **Introducción.**
  2. **Planificaciones ASAP y ALAP**
  3. **Movilidad de operaciones**
  4. **Métodos básicos: Planificación con límite de tiempo**
     - **4.1. Programación lineal entera 0-1**
     - **4.2. Planificación basada en fuerzas.**
  5. **Métodos básicos: Planificación con límite de área**
     - **5.1. Planificación por listas.**
  6. **Técnicas avanzadas**
     - **6.1. Encadenamiento, multiciclo y segmentación**
     - **6.2. Ejemplo: Slicer**
     - **6.3. Tratamiento de condicionales**
     - **6.4. Tratamiento de bucles: desarrollo y solapamiento de iteraciones.**

Introducción

- El problema de la planificación en SAN
  - Formulación más sencilla: Dado un GFDC partirlo en un conjunto de subgrafos de tal forma que:
    - Cada subgrafo se ejecute en un solo paso de control (*cstep*)
    - Los operandos de cada operación de un *cstep* dado hayan sido calculados en algún *cstep* anterior
  - Cada *cstep* se corresponde a un estado de una FSM => hacer la planificación define el comportamiento del controlador.
  - Ligadura básica: cada una de las operaciones asignadas a un mismo *cstep* han de ser ejecutadas por una UF diferente



Interrelación con la asignación de HW

Introducción

- Variantes del problema
  - Planificación con límite de tiempo

    Objetivo: minimizar el HW necesario para implementar el comportamiento, de forma que el tiempo de operación no sobrepase el límite dado (*csteps* o tiempo físico)

  - Planificación con límite de área

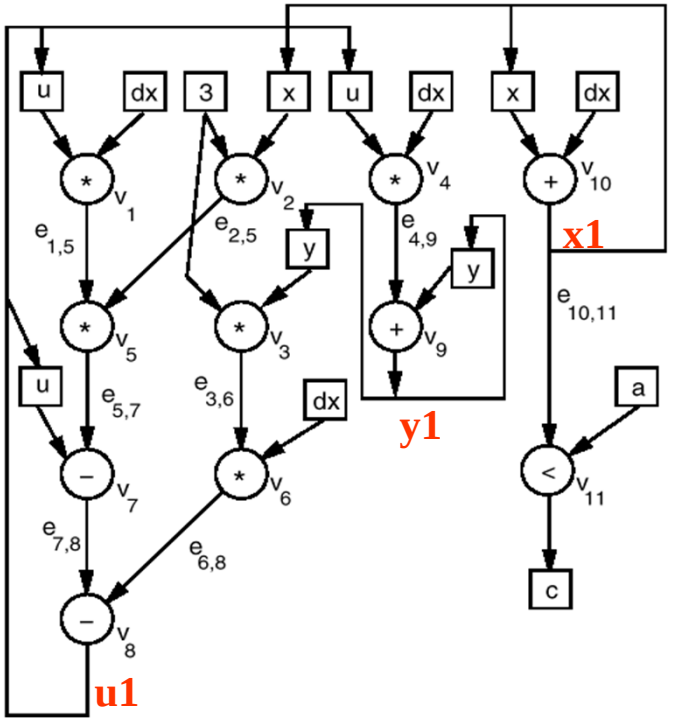
    Objetivo: minimizar el tiempo de operación, de forma que los recursos HW necesarios no sobrepasen el coste especificado (módulos, puertas lógicas, área)

Planificaciones ASAP y ALAP

- Suposiciones simplificadoras:
  - Grafos sin lazos ni condicionales
  - Cada operación se ejecuta en un solo paso de control
  - Cada tipo de operación puede ser ejecutada por una sola UF
- Definiciones
  - GFDC: G(V,E)
    (V = nodos, E = arcos)
  - Operaciones: o_i representadas por los nodos $v_i \in V$
  - Transferencias: $e_{i\,j}$ representa el arco v_i --> v_j
  - *Pred* v_i : {Predecesores inmediatos de v_i }
  - *Succ* v_i : {Sucesores inmediatos de v_i }

Ejemplo de trabajo: Texto fuente y GFDC

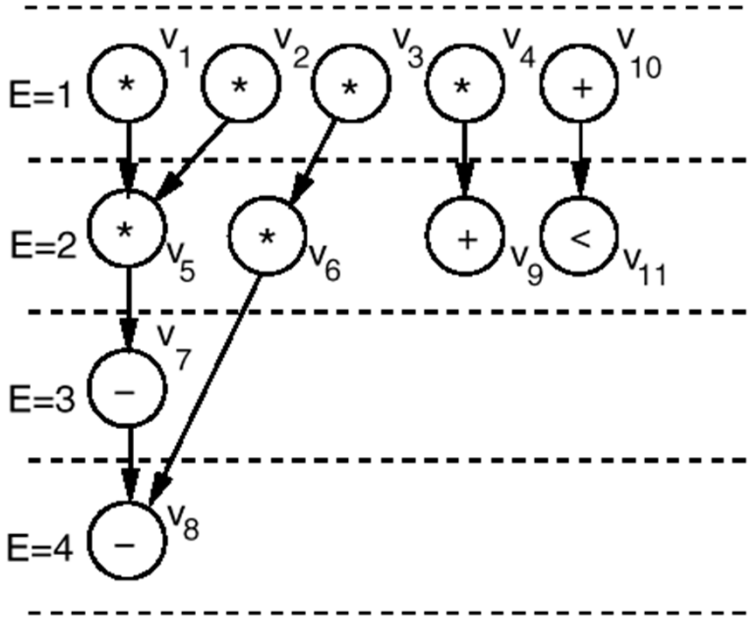
```
while (x<a) do
        x1 := x + dx;
        u1 := u - (3*x*u*dx) - (3*y*dx);
        y1 := y + (u*dx);
        x := x1; u := u1; y := y1;
endwhile
```



Planificación ASAP (As Soon As Possible)

- Idea básica: hacer las operaciones tan pronto como sea posible
- Estrategia: Asignar cada operación al primer *cstep* después de que todos sus predecesores inmediatos hayan sido planificados.
- La planificación ASAP determina el número mínimo de *csteps* necesarios para ejecutar el comportamiento
  - Caso del ejemplo:
    - 4 csteps
    - Módulos necesarios:
      4 multiplicadores
      1 sumador / restador (o 1 sumador y 1 restador)
      1 comparador

Aplicación al ejemplo de trabajo:



Planificación ASAP

- Notación:
  - E_i : número del *cstep* al que se asigna la operación o_i
  - ALL_NODES_SCHED(*Pred* v_i , E): TRUE si todos los nodos de *Pred* v_i han sido asignados
  - MAX(*Pred* v_i , E): valor máximo del conjunto de *csteps* al que han sido asignados los nodos de *Pred* v_i

Algoritmo

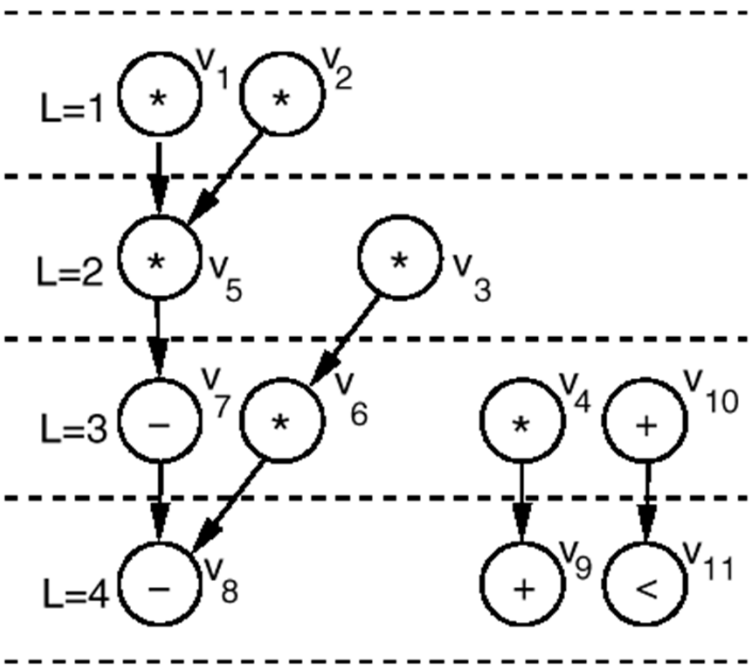
```
for each node vi ∈ V do
    if Pred vi = ∅ then
        Ei = 1;
        V = V - {vi };
    else
        Ei = 0;
    endif
endfor

while V ≠ ∅ do
    for each node vi ∈ V do
        if ALL_NODES_SCHED
            (Pred vi, E) then
        Ei = MAX(Pred vi, E) + 1;
        V = V - {vi };
        endif
    enfor
endwhile
```

Planificación ALAP (As Late As Possible)

- Idea básica: no hacer las operaciones antes de que sea necesario
- Estrategia: dado un tiempo límite T, asignar cada operación al último *cstep* antes de que sus resultados deban ser consumidos por otra operación. (T puede ser el mismo de la planificación ASAP)
  - Módulos necesarios en ejemplo:
    - 2 multiplicadores
    - 1 sumador
    - 1 restador
    - 1 comparador

Aplicación al ejemplo de trabajo (con T=4)



Planificación ALAP

- Notación:
  - L_i : número del *cstep* al que se asigna la operación o_i
  - ALL_NODES_SCHED (*Succ* v_i , E): TRUE si todos los nodos de *Succ* v_i han sido asignados
  - MIN(*Succ* v_i , E): valor mínimo del conjunto de *csteps* al que han sido asignados los nodos de *Succ* v_i

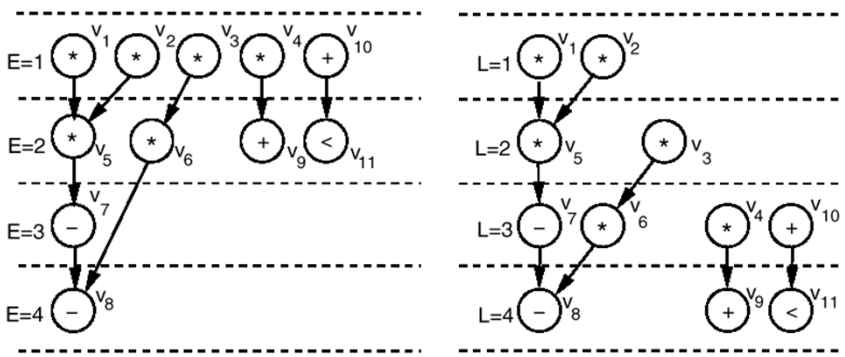
Algoritmo

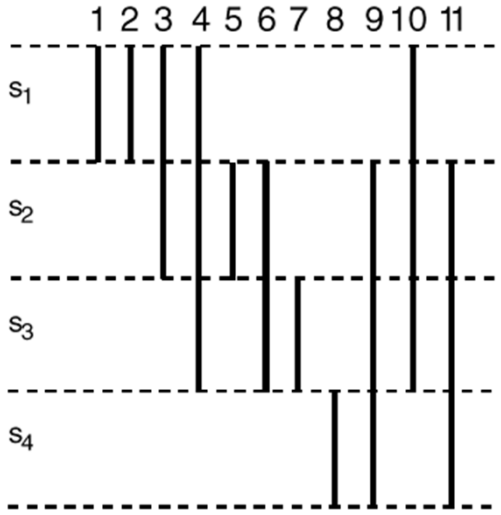
```
for each node vi ∈ V do
    if Succ vi = ∅ then
        Li = T;
        V = V - {vi };
    else
        Li = 0;
    endif
endfor
while V ≠ ∅ do
    for each node vi ∈ V do
        if ALL_NODES_SCHED
        (Succ vi, E) then
            Li = MIN(Succ vi, E) - 1;
            V = V - {vi };
        endif
    enfor
endwhile
```

Planificaciones ASAP y ALAP: Movilidad

- Movilidad de operaciones
  - Una vez determinadas
    - la planificación ASAP de una operación o_i: E_i
    - la planificación ALAP de o_i: L_i
  - la diferencia $L_i - E_i + 1$ (≥ 1) se denomina movilidad de v_i
- Rango de movilidad de o_i
  - Conjunto de *csteps* comprendidos entre E_i y L_i
  - En toda planificación válida, o_i debe estar comprendida entre E_i y L_i
  - Si $E_i = L_i$, entonces o_i pertenece al camino crítico

- Aplicación al ejemplo de trabajo





Métodos básicos: planificación con límite de tiempo

- Programación lineal entera 0-1
  - Crea una planificación óptima respecto al uso de recursos
  - Planteamiento matemático elegante
  - Enorme complejidad computacional para problemas de tamaño mediano
  - La solución se puede obtener mediante paquetes matemáticos estándar
- Definiciones
  - N_k: nº de UFs del tipo k empleadas en el diseño
  - C_k: coste de una UF del tipo k
  - Variables:
    - $x_{ij} = 1$ si la operación o_i se asigna al *cstep* s_j
    - $x_{ij} = 0$ en caso contrario

Métodos básicos: planificación con límite de tiempo

- Formulación del problema

Minimizar $$\sum_{k \in \{\text{tipos de FUs usadas}\}} (C_k * N_k)$$

con las siguientes restricciones:

1. Toda operación, o_i, se planificará en un y sólo un *cstep*, s_j, comprendido entre sus planificaciones ASAP y ALAP

$$\forall i = 1..n, \quad \sum_{E_i \le j \le L_i} x_{ij} = 1$$

2. Ningún *cstep* requerirá más de N_k UFs del tipo k

$$\forall \text{cstep } s_j, \forall \text{tipo de UF } k, \quad \sum_{\forall i | i \text{ se implementa con UF tipo } k} x_{ij} \le N_k$$

3. ∀ operación, sus predecesores deben planificarse en un *cstep* previo

$$\forall i, j \mid o_i \in \text{Pred } o_j, \left[\sum_{E_i \le k \le L_i} (k * x_{ik}) - \sum_{E_j \le l \le L_j} (l * x_{jl}) \right] \le -1$$

Métodos básicos: planificación con límite de tiempo

- Programación lineal entera 0-1
  - Aplicación al ejemplo de trabajo. **Minimizar**:

$$C_m * N_m + C_s * N_s + C_r * N_r + C_c * N_c$$

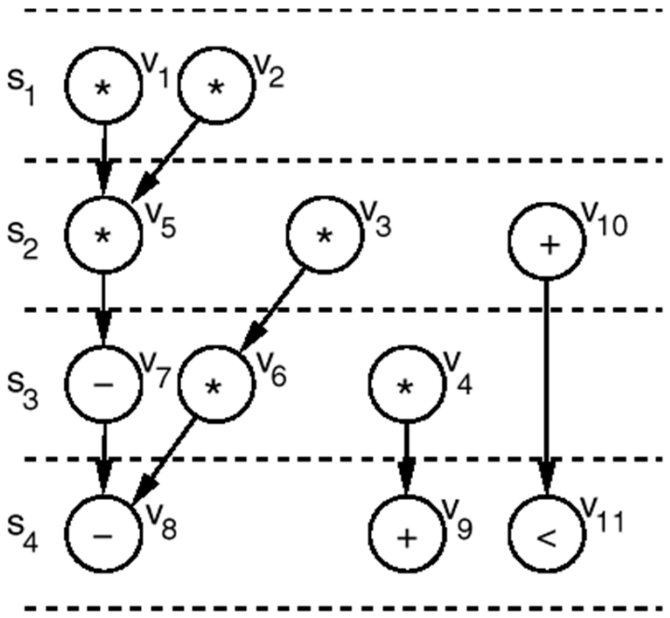
con las **ligaduras**:

$$x_{1,1} = 1$$
$$x_{2,1} = 1$$
$$x_{3,1} + x_{3,2} = 1$$
$$x_{4,1} + x_{4,2} + x_{4,3} = 1$$
$$x_{5,2} = 1$$
$$x_{6,2} + x_{6,3} = 1$$
$$x_{7,3} = 1$$
$$x_{8,4} = 1$$
$$x_{9,2} + x_{9,3} + x_{9,4} = 1$$
$$x_{10,1} + x_{10,2} + x_{10,3} = 1$$
$$x_{11,2} + x_{11,3} + x_{11,4} = 1$$

$$x_{1,1} + x_{2,1} + x_{3,1} + x_{4,1} \leq N_m$$
$$x_{3,2} + x_{4,2} + x_{5,2} + x_{6,2} \leq N_m$$
$$x_{4,3} + x_{6,3} \leq N_m$$
$$x_{7,3} \leq N_s$$
$$x_{8,4} \leq N_s$$
$$x_{10,1} \leq N_a$$
$$x_{9,2} + x_{10,2} \leq N_a$$
$$x_{9,3} + x_{10,3} \leq N_a$$
$$x_{9,4} \leq N_a$$
$$x_{11,2} \leq N_c$$
$$x_{11,3} \leq N_c$$
$$x_{11,4} \leq N_c$$
$$1x_{3,1} + 2x_{3,2} - 2x_{6,2} - 3x_{6,3} \leq -1$$
$$1x_{4,1} + 2x_{4,2} + 3x_{4,3} - 2x_{9,2} - 3x_{9,3} - 4x_{9,4} \leq -1$$
$$1x_{10,1} + 2x_{10,2} + 3x_{10,3} - 2x_{11,2} - 3x_{11,3} - 4x_{11,4} \leq -1$$

Métodos básicos: planificación con límite de tiempo

- Programación lineal entera 0-1
  - Aplicación al ejemplo de trabajo (cont.):
    - Solución: $N_m = 2$, $N_s = N_r = N_c = 1$
    - $x_{1,1} = x_{2,1} = x_{3,2} = x_{4,3} = x_{5,2} = x_{6,3} = x_{7,3} = x_{8,4} = x_{9,4} = x_{10,2} = x_{11,4} = 1$
    - Restantes $x_{i,j} = 0$
  - Planificación resultante:



Métodos básicos: planificación con límite de tiempo

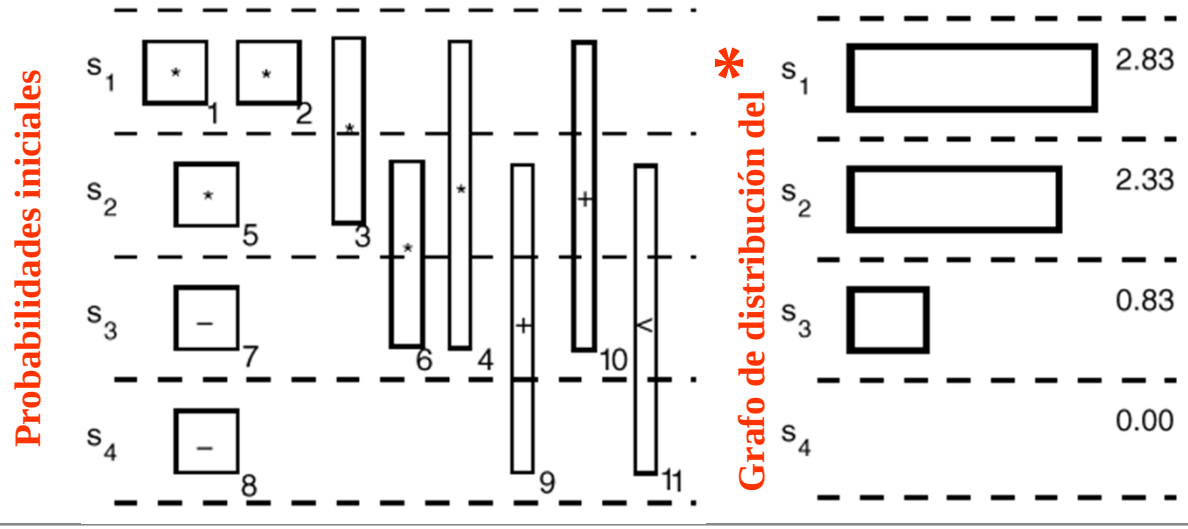
- Planificación basada en fuerzas (modelo físico inspirado en ley de Hooke)
  - Grafo de distribución: Existe uno para cada tipo de operación
    - Para cada tipo de UF, estima el nº de instancias de ese tipo que serán necesarias para implementar el comportamiento
    - Se asume que existe idéntica probabilidad de asignar cada operación a cada uno de los *csteps* de su rango de movilidad
  - Probabilidad de asignar la operación o_i al *cstep* s_j

    $p_j(o_i) = 1 / (L_i - E_i + 1)$, si $E_i \leq j \leq L_i$

    $p_j(o_i) = 0$, en caso contrario

    $$EOC_{jk} = C_k * \sum_{\forall o_i \text{ del tipo } k} p_j(o_i)$$

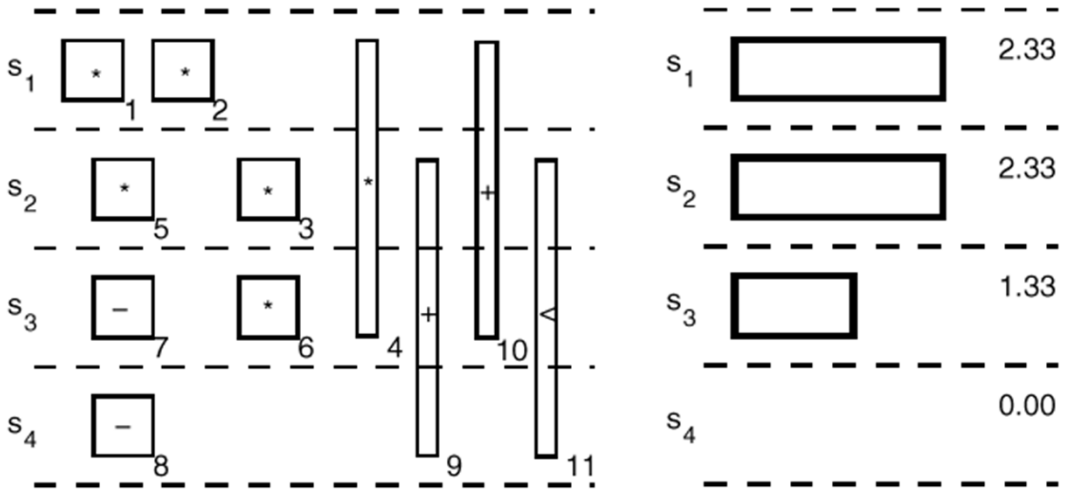
  - Coste estimado de las operaciones de tipo *k* en el *cstep* s_j



Métodos básicos: planificación con límite de tiempo

- Planificación basada en fuerzas
  - Base intuitiva: tiende a planificar las operaciones de forma que el uso de las UFs a lo largo de la planificación sea lo más uniforme posible.
    - Evitar el uso de muchas UFs en un *cstep* para mantenerlas inactivas en los restantes.
    - Equilibrar los Grafos de Distribución.
  - Planificación de una operación ⇒ Ajustar el GD

    Ejemplo: Efecto de asignar la multiplicación o_3 al *cstep* s_2



Métodos básicos: planificación con límite de tiempo

- Coste estimado de una planificación, S: Suma de los máximos EOC de cada tipo de UF

$$COST(S) = \sum_{\forall k \in \{\text{tipos de UF}\}} \left(\max_{\forall j | s_j \in \{csteps\}} EOC_{jk} \right)$$

Algoritmo

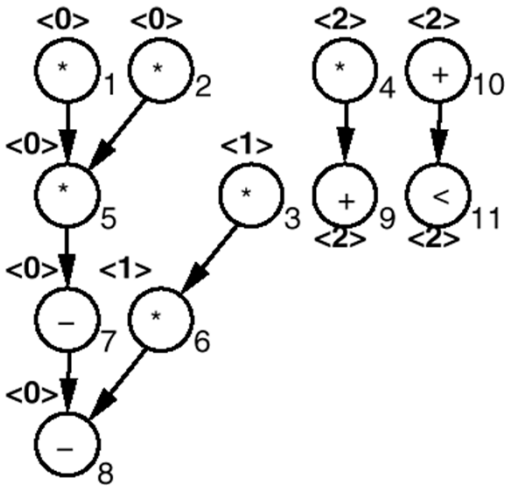
Call ASAP(V);
Call ALAP(V);
while there exists o_i such that $E_i \neq L_i$ **do**
 MaxGain = -∞;
/* Try scheduling all unscheduled operations to every */
/* state in its range */
for each o_i, $E_i \neq L_i$, **do**
 for each j, $E_i \leq j \leq L_i$ **do**
  S_{work} = SCHEDULE_OP($S_{current}$, o_i , s_j);
  ADJUST_DISTRIBUTION(S_{work}, o_i , s_j);
  if COST($S_{current}$)- COST(S_{work}) > *MaxGain* **then**
   MaxGain = COST($S_{current}$)- COST(S_{work}) ;
   BestOp = o_i;
   BestStep = s_j;
  endif
 endfor
ednfor
 $S_{current}$ = SCHEDULE_OP($S_{current}$, *BestOp*, *BestStep*);
 ADJUST_DISTRIBUTION($S_{current}$, *BestOp*, *BestStep*);
endwhile

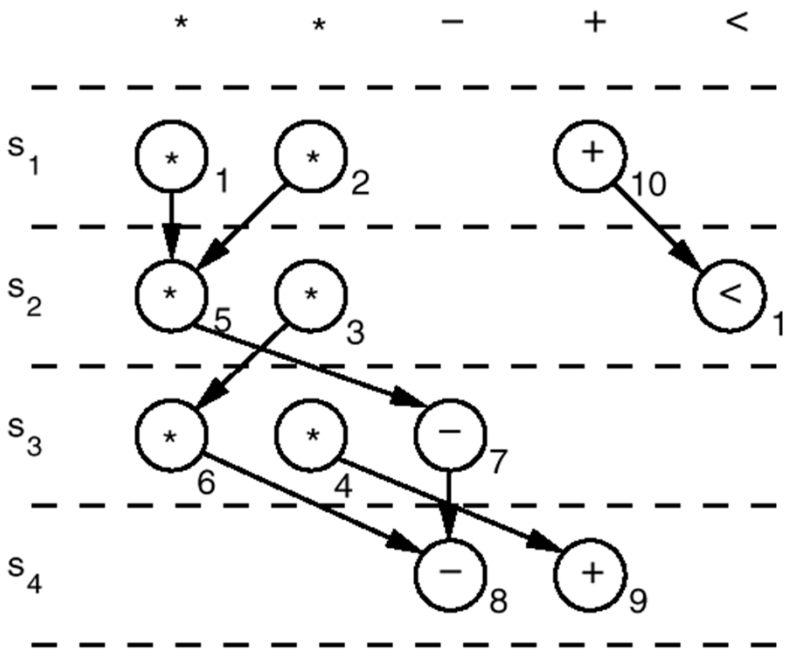
Métodos básicos: planificación con límite de área

- Planificación basada en listas
  - Nodos listos: los que tienen planificados todos sus predecesores inmediatos
  - Lista de prioridad: Determina el orden en que se planifican los nodos listos
    - Aspecto clave del método
    - Variantes
      - Operaciones con menor movilidad
      - Operaciones con el camino más largo a nodos de salida
      - Operaciones con más sucesores inmediatos
  - Idea básica: Planificar los nodos listos tan pronto como sus dependencias lo permitan, respetando la restricción de que en ningún *cstep* se usen más recursos de los disponibles
  - Efecto de la planificación de nodos
    - Modificar conjunto de nodos listos
    - Cálculo dinámico de las listas de prioridades

Métodos básicos: planificación con límite de área

- Planificación basada en listas: Aplicación al ejemplo de trabajo





Recursos hardware

$Resources_{*}$: 2
$Resources_{+}$: 1
$Resources_{-}$: 1
$Resources_{<}$: 1

Listas de prioridad

$PList_{*}$: 1<0>, 2<0>, 3<1>, 4<2>
$PList_{+}$: 10<2>
$PList_{-}$: NIL
$PList_{<}$: NIL

Métodos básicos: planificación con límite de área

- Planificación basada en listas
  - Definiciones
    - Una lista de nodos listos, $PList_k$, para cada tipo k de UF
    - Un número de UFs disponibles, N_k, para cada tipo k de UF
    - INSERT_READY_OPS, crea listas de nodos listos
  - SCHEDULE_OP, asigna una operación a un *cstep*

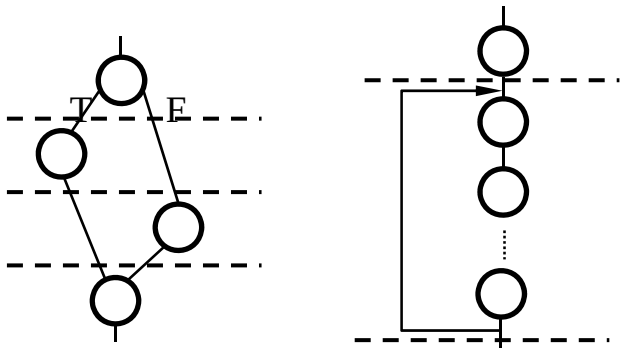
Métodos básicos: planificación con límite de área

- Planificación basada en listas
  - Algoritmo

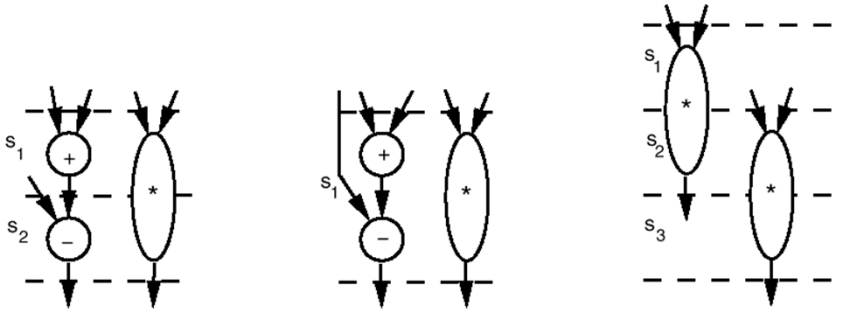
INSERT_READY_OPS (V, $PList_1$, $PList_2$, ... , $PList_m$);
  Cstep = 0;
  while (($PList_1 \neq \emptyset$) **or** ... **or** ($PList_m \neq \emptyset$)) **do**
    Cstep = Cstep +1;
    for k = 1 **to** m **do**
      for funit = 1 **to** N_k **do**
        if $PList_k \neq \emptyset$ **then**
          SCHEDULE_OP($S_{current}$, FIRST($PList_k$), Cstep);
          $PList_k$ = DELETE($PList_k$, FIRST($PList_k$));
        endif
      enfor
    endfor
    INSERT_READY_OPS (V, $PList_1$, $PList_2$, ... , $PList_m$);
  endwhile

Técnicas de planificación avanzadas

- Eliminación de suposiciones simplificadoras.
- Los GFDC pueden presentar condicionales y lazos
  - Fronteras a la movilidad de las operaciones



- Cada UF tiene un tiempo de operación que depende al menos de su tipo
  - Alternativas de diseño



Multiciclo **Encadenamiento** **Segmentación**

- Una UF puede ser capaz de ejecutar varios tipos de operaciones (ALUs)
- La mayor parte de las técnicas básicas pueden modificarse para enfrentarse a los anteriores problemas

Técnicas avanzadas

- Ejemplo: Slicer (B.M. Pangrle)
  - Basado en listas con encadenamiento y segmentación
  - Calcular: ASAP - ALAP- Movilidades ⇒ Hallar camino crítico (usa UFs más rápidas)
  - Objetivo: En un *cstep* dado es importante encadenar cuando las posibles operaciones a anticipar están en el camino crítico ⇒ puede reducir el tiempo total de ejecución
  - Persistencia en la acción de encadenar
    - Si logra eliminar un *cstep* ⇒ persiste. Caso contrario desiste.
    - Gestión: Mediante un FLAG
      - (FLAG = .T.) **y** (Todas las operaciones críticas están ya asignadas a HW) **y** (la operación que se está tratando es la 1ª no crítica) ⇒
        ⇒ Ejecutar procedimiento de encadenamiento
      - (FLAG = .F.) ⇒ No encadenar
    - Efecto: Intentar encadenar las operaciones críticas del *cstep* siguiente, antes que asignar recursos HW a las no críticas del *cstep* actual

Slicer (algoritmo)

```
Procedure SLICE
/* NOTACIÓN: mo ≡ operación; partition ≡ cstep  */
partition := 1
while all mo.s are not allocated do {
    N := # of mo.s in partiton
    U := 0   /* U # of unallocated mo.s in partition */
    flag := TRUE
    SORT (mo.s in partiton) {1 sola lista ordenada por movilidad}
    mo := fisrt mo in partition
    i := 0
    while i < N do {
        if (flag = TRUE & first mo is on a critical path
        & mo is not on a critical path
        & all previous mo.s in partition are allocated) then {
            CHAIN (partition + 1)
            if chaining eliminates last partition
                then RE_SORT (mo.s in partition)
            else flag := FALSE }                                  1)
        else {
            if a suitable unit is available then {
                place mo in partition
                if unit takes longer than time estimated then
                    ADJUST_GRAPH↓(mo)}
            else {
                U := U+1
                place mo on unalocated_list }
            i := i+1
            mo := next mo in partition }}                         2)
    if U = 0 & N > 0 then                                         3)
        while flag = TRUE do {
            CHAIN (partition + 1)
            if chaining does not eliminate last partition then
                flag := FALSE }
    mo := first mo on unallocated_list
    for j = 1 to U do {
        ADJUST_GRAPH↓(mo)
        mo := next mo on unallocated_list}                        4)
    partition := partition + 1}
```

Comentarios

1) Si se dan las condiciones, intenta hacer encadenamiento
2) Planifica operaciones en el cstep partition mientras haya UFs
3) Todas las op. eran críticas y se han asignado
4) Mover las op. no planificadas en el cstep actual al cstep siguiente

Slicer (algoritmo)

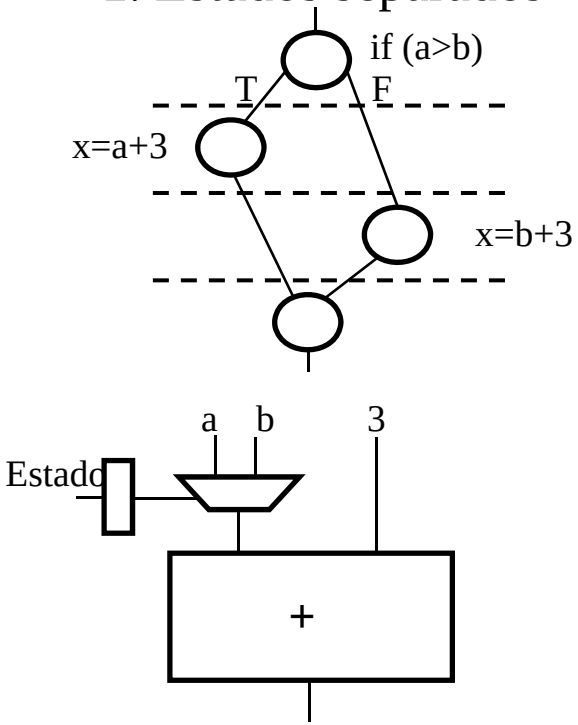
Procedure CHAIN (*partition*)

/* NOTACIÓN: mo ≡ operación; partition ≡ cstep */

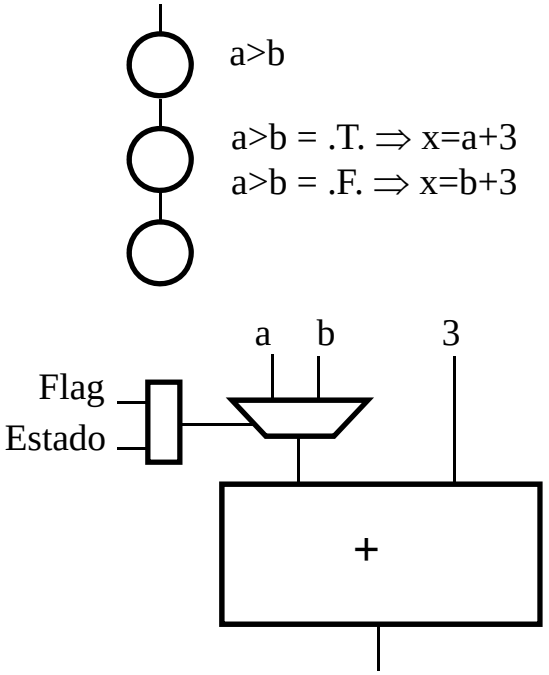
N := # of mo.s in *partition*
SORT (mo.s in *partition*)
mo := first mo in *partition*
while i < N & ***mo*** is on critical path **do** {
    find longest input delay for ***mo***
    if a suitable unit is available **then** {
        place ***mo*** in partition *partiton*-1
        ADJUST_GRAPH↑ (***mo***)
        if a partition can be deleted **then return**}
    i := i+1
    mo := next mo in partition }

Técnicas avanzadas

- Tratamiento de condicionales
  - Idea básica: extraer ventaja de la exclusión mutua
- Alternativas:
  - 1. Estados separados



  - 2. Fusión de estados

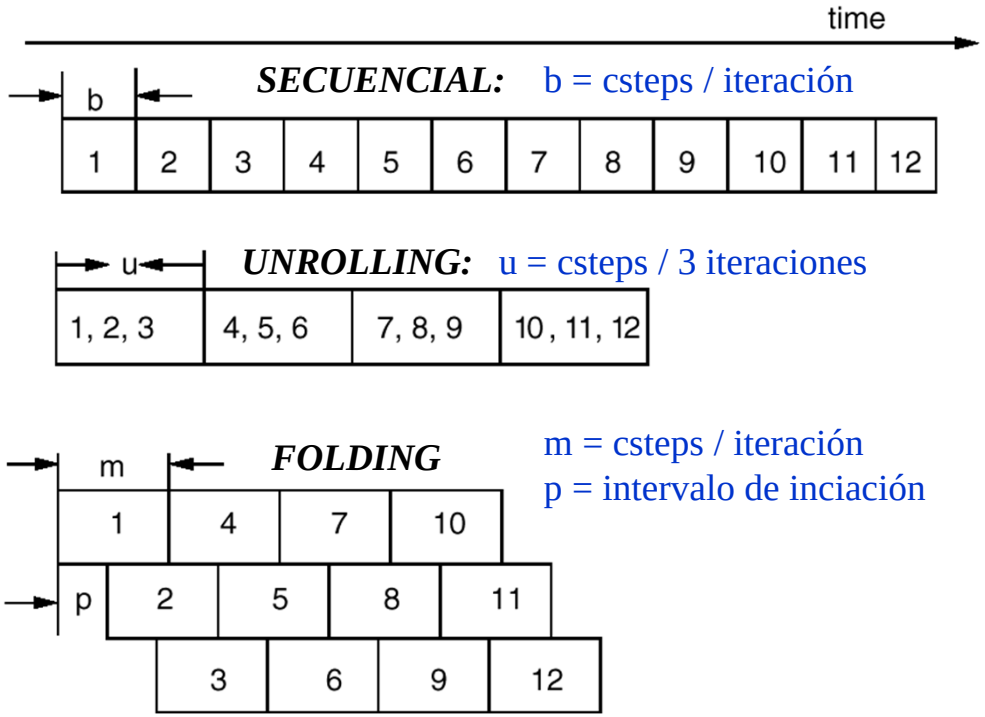


- Mayor libertad para la planificación
- Reducción de la FSM de control

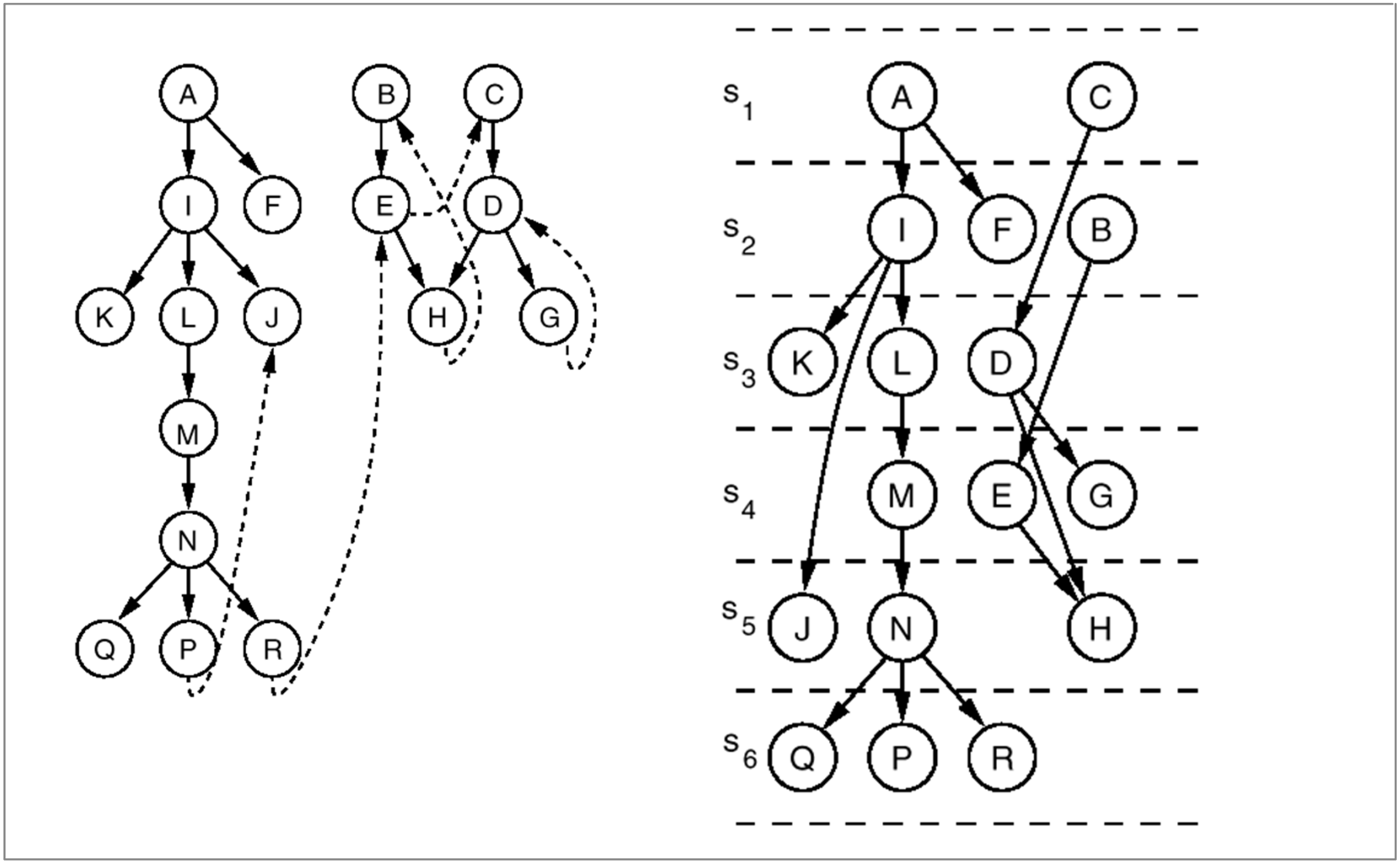
Técnicas avanzadas

- Tratamiento de bucles
  - Idea básica: considerar paralelismo potencial en sucesivas iteraciones
  - Alternativas:
    - Ejecución Secuencial
    - Desarrollo de bucles (loop unrolling)
    - Solapamiento de bucles (loop folding)
  - Las dos últimas alternativas pueden acelerar la ejecución a costa de requerir más hardware

- Ejemplo

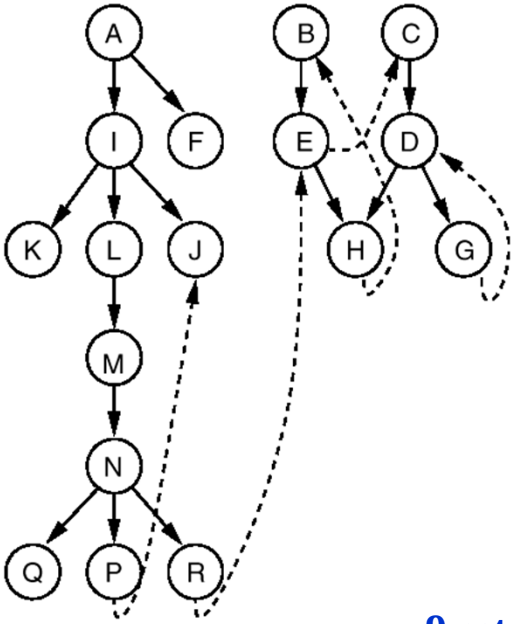


Tratamiento de bucles: secuencial



Tratamiento de bucles: desarrollo

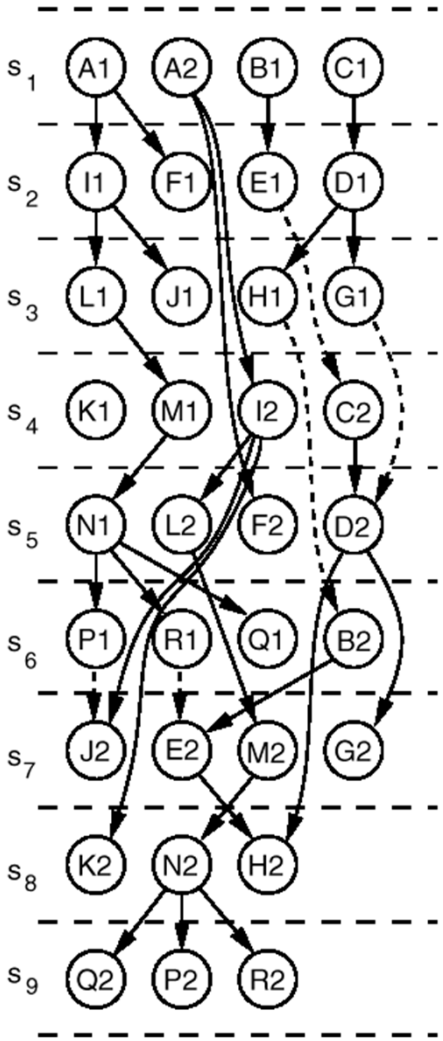
- 2 iteraciones, 4 UFs



9 csteps / 2 iteraciones

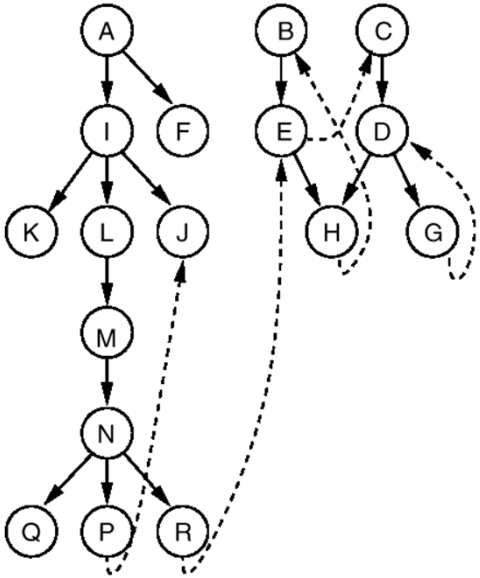
⇓

4.5 ciclos / iteración



Tratamiento de bucles: plegado

- Solapamiento de iteraciones (cada tres ciclos, 6 UFs)



6 csteps / iteración

⇓

I. iniciación = 3 ciclos