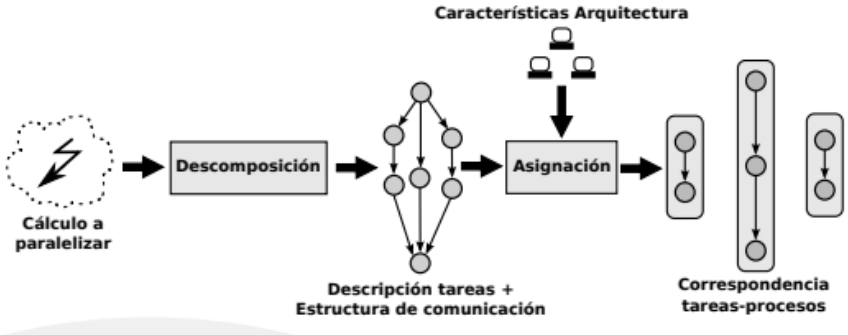
Metodología de la Programación Paralela

Este tema abordara el diseño de algoritmos aprovechando los recursos paralelos disponibles.
Un algoritmo secuencial describe una secuencia de operaciones de cómputo.
El diseño de algoritmos multiprocesador debe tener en cuenta:

- Concurrencia: coordinación de las acciones en los diferentes procesadores.
- Asignación de datos y código a los procesadores.
- Acceso simultáneo a datos compartidos.
- Escalabilidad.



Descomposición:

- Descomponer cálculos en tareas de grano fino.
- Detectar dependencias entre tareas (comunicaciones).
- Se centra en describir el paralelismo del cálculo.

Asignación:

- Establecer qué tareas serán llevadas a cabo por cada proceso.
- Distinción: proceso≠ procesador.
- Dependiente de la arquitectura del sistema.

Descomposición en tareas

Los algoritmos paralelos se basan, generalmente, en trocear un cálculo complejo en partes de menor tamaño.
Cada uno de los trozos resultantes será una unidad de cómputo, de tamaño arbitrario, que denominaremos “tarea’.
El objetivo es que muchas de estas tareas sean independientes entre sí, y por tanto puedan llevarse a cabo de forma concurrente.

Granularidad de la descomposición

Las tareas resultantes de una descomposición pueden tener diferentes costes computacionales.
Granularidad fina: elevado número de pequeñas tareas.
Granularidad gruesa: pocas tareas de gran tamaño.
Límite inferior en la granularidad

- Existe un límite sobre lo fina que puede ser la granularidad de la descomposición de tareas del cálculo

Grafo de dependencias

El objetivo de una descomposición es que las tareas sean independientes, proporcionando oportunidades de paralelización.
No obstante, existirán dependencias entre determinadas tareas (ej. evaluación polinomial y cálculo del mínimo global).
Como consecuencia, ciertas tareas no pueden comenzar mientras las tareas de las que dependen no hayan terminado.
Def. Grafo de dependencias:

- Grafo G = {N, A}, acíclico y dirigido.
- Cada nodo n ∈ N representa una tarea en la descomposición.
    - Opcionalmente puede etiquetarse cada nodo n del grafo con una estimación de su coste, w(n).
- Cada arista a ∈ A es un par ordenado de nodos no , nd ∈ N representando la existencia de una dependencia entre dos tareas: la tarea origen no debe ejecutarse previamente a la tarea destino nd .

Grado de concurrencia: máximo

Grado de concurrencia: medida de cómo de paralelizable es un algoritmo, atendiendo a las dependencias entre las tareas constituyentes.
Def. Máximo grado de concurrencia: mayor número de tareas de una descomposición que pueden ser ejecutadas al mismo tiempo.
En muchos casos puede determinarse de forma sencilla observando el grafo de dependencias.
El máximo grado de concurrencia es un indicador de lo paralelizable que es una determinada descomposición, pero sólo de forma puntual. El grado medio de concurrencia será un mejor indicador global.

Grado medio de concurrencia

Def. Camino crítico: el más largo de los caminos del grafo de dependencias que van desde un nodo de comienzo hasta un nodo de finalización del cómputo.
Sea C ⊂ N el conjunto de nodos del camino crítico, su longitud L puede expresarse matemáticamente como la suma de los costes de cada nodo individual de C:

$$L = \sum_{n \in C} w(n)$$

- **Def.** *Grado medio de concurrencia (M)*:

$$M = \sum_{n \in N} \frac{w(n)}{L}$$

Habitualmente, cuanto menor sea la granularidad de una descomposición mayor será el grado medio de concurrencia. Por ello es deseable buscar descomposiciones del grano más fino posible.

Representación de las comunicaciones

Es posible etiquetar las aristas del grafo de dependencias para incluir información del coste de las comunicaciones.

A la hora de implementar un algoritmo, el grafo de dependencias puede sufrir modificaciones debido a la inclusión de comunicaciones adicionales. P. ej. evaluación polinomial

Técnicas de descomposición

- Descomposición de dominio.
- Descomposición funcional dirigida por el flujo de datos.
- Descomposición recursiva.
- Descomposición especulativa.
- Enfoques mixtos.

Descomposición de dominio

Técnica más usada sobre grandes estructuras de datos.
Dos fases:

- Troceo de datos, centrándose en la estructura más grande, o en la más usada.
- Asociación de la computación a cada subdominio de datos.

Se utiliza cuando es posible resolver un problema aplicando la misma operación sobre partes diferentes de su dominio de datos.

Regla del propietario (owner-computes rule): La tarea a la que se le asigna un determinado dato es responsable de realizar todos los cálculos asociados al mismo
Descomposición centrada en la entrada: todas las computaciones que usen los datos de entrada serán realizadas por la tarea.
D. c. e. l. salida: la salida es computada por el proceso al que se le asigna la misma.
Regla general: no siempre es posible.

Descomposición funcional

Consiste en descomponer el cálculo en tareas atendiendo a las partes diferenciadas del mismo.
Pasos:

- Identificar las fases funcionales del cálculo a paralelizar.
- Asignar una tarea para la realización de cada fase.

Es habitual que el grafo de tareas resultante de este tipo de descomposiciones pueda organizarse como un pipeline.

Descomposición recursiva

También conocida como divide-y-vencerás.
Pasos:

- Si un subproblema ha alcanzado el tamaño crítico, obtener su solución parcial.
- De lo contrario, descomponerlo en subproblemas.
- Combinar los resultados de los subproblemas para obtener la solución al problema original.

El paralelismo surge de la independencia entre las tareas de resolución de los subproblemas.

Descomposición especulativa

En ocasiones un problema no puede dividirse en tareas independientes. No obstante, sí puede dividirse funcionalmente en fases que se ejecutan condicionalmente dependiendo de los resultados de fases anteriores.
La descomposición especulativa consiste en comenzar la ejecución de tareas condicionales excluyentes entre sí sin esperar a la finalización de las tareas de las que depende la selección de una de ellas.
Una vez la tarea de selección haya finalizado, es posible seleccionar el resultado correcto de entre los obtenidos por las tareas ejecutadas de forma especulativa.
Si no es posible comenzar todas las tareas alternativas se seleccionarán aquéllas que se ejecutarán más probablemente. Si la tarea correcta no se ejecuta especulativamente, será necesario cancelar el trabajo realizado e iniciarla.
El coste computacional total es superior, pero puede proporcionar mejoras en tiempo de ejecución.
Aplicación alternativa: lanzar en paralelo diferentes algoritmos de resolución de un mismo problema, quedándose con la solución que se obtenga más rápidamente y cancelando el resto de tareas en ejecución.

Enfoques mixtos

Es común aplicar diferentes esquemas de descomposición paralela a diferentes partes de un problema.
En general, diferentes fases de cómputo pueden operar sobre diferentes estructuras de datos siguiendo diferentes descomposiciones.

Factores a considerar

Creación de tareas

- Creación estática: todas las tareas que participan en el algoritmo se crean al inicio de su ejecución.
- Creación dinámica: las tareas se crean durante la ejecución del algoritmo. Útil cuando no se conoce la estructura del ´ grafo de dependencias de forma estática. Sí se define la política de creación de nuevas tareas.

Número de tareas obtenidas

- Tras la descomposición en tareas, se procede a asignarlas a procesadores.
- Interesa disponer de un número de tareas que permita tener flexibilidad en la asignación. En general, es aconsejable qué #tareas $\gg$ #procesadores
- Es deseable que el número de tareas escale con el tamaño del problema.

Balanceo de carga

- Es deseable que las tareas obtenidas tengan un coste similar, para facilitar la etapa de asignación de tareas.
- Es necesario evitar que se produzcan desequilibrios entre la carga computacional de los diferentes procesadores.
- La descomposición de dominio habitualmente lleva a algoritmos equilibrados. En la descomposición funcional, recursiva y especulativa dependerá del análisis realizado.

Asignación de tareas

Tras descomponer un algoritmo paralelo en tareas, debe definirse una correspondencia entre ´estas y los procesos que las ejecutarán. Denominamos a esta correspondencia “asignación de tareas”.
Es importante distinguir en este nivel entre procesos y procesadores. No usamos el término “procesos” en el sentido técnico comúnmente usado en los sistemas operativos, sino que en nuestro contexto denomina a una unidad lógica de cómputo capaz de ejecutar tareas computacionales (hilos, procesos UNIX).
Este paso supone un paso descendente en el nivel de abstracción. La asignación de tareas se hará teniendo en cuenta algunas de las características de la arquitectura paralela concreta a la que adaptamos el algoritmo (p. ej. crear más procesos que procesadores en una arquitectura que no permite la ejecución eficiente de pequeñas tareas será contraproducente).
La asignación de tareas incluye también la decisión sobre el orden de ejecución de las mismas.

Objetivos

Minimizar el tiempo de computación, a través de la ejecución concurrente de tareas.
Minimizar el tiempo de comunicaciones, asignando tareas que se comuniquen mucho al mismo proceso.
Minimizar el tiempo de inactividad de los procesos. Fuentes de inactividad:

- Desequilibrios de carga.
- Dependencias entre tareas.

Estos tres objetivos suelen estar muy interrelacionados, por lo que será necesario alcanzar un punto de compromiso entre los 3 para lograr una buena asignación.

Esquemas de asignación estática

Cuando el grafo de tareas se conoce con detalle de forma estática (número de tareas, coste, relaciones, comunicaciones), las decisiones de asignación pueden tomarse antes de la ejecución.
Su mayor ventaja es que no añaden sobrecarga en tiempo de ejecución.
El diseño y la implementación es sencillo.
Dos tipos fundamentales:

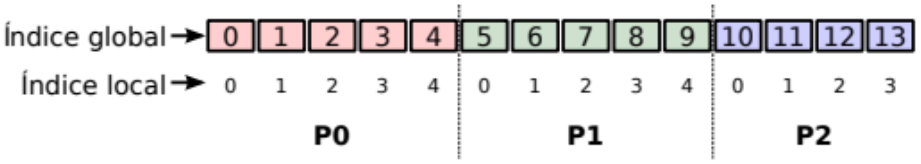
- Esquemas para descomposición de dominio: distribución en bloques.
- Esquemas para descomposición funcional o recursiva: grafos de dependencias estáticos

Distribución de matrices por bloques

En numerosos algoritmos sobre matrices el cálculo de una entrada de la matriz depende de entradas contiguas de la misma (localidad espacial).
La descomposición de dominio de estos problemas dará lugar a tareas que se comunican con las tareas que representan a celdas vecinas.
La asignación de estas tareas contiguas a un mismo proceso permite reducir en gran medida las necesidades de comunicación del algoritmo.
Esto provoca la reducción del número de comunicaciones, así como un mejor aprovechamiento de la jerarquía de memoria (registros CPU, caché, memoria principal de cada procesador).

Matriz unidimensional

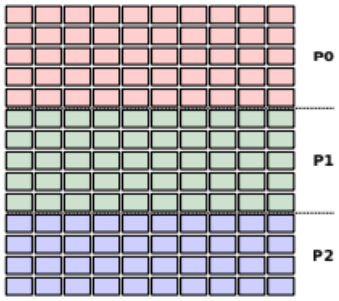
Si tenemos p procesos, se asigna a cada proceso un bloque de tamaño mb = $\binom{n}{p}$:



El proceso p − 1 recibirá entre 1 y mb elementos. Concretamente $\widehat{m_b} = (n - m_b(p-1))$.
A pesar de este pequeño desequilibrio, esta asignación será ´optima cuando todas las celdas de la matriz llevan asociada aproximadamente la misma carga computacional.
El elemento i del vector se asigna al proceso $\lfloor \frac{i}{mb} \rfloor$, ocupando la posición (i mod mb) en su vector local.

Distribución de una matriz bidimensional: distribución por bloques de filas

Cada proceso recibe un bloque de mb =$\binom{n}{p}$ filas de la matriz, a excepción del ´ultimo, que recibirá mˆb = (n − mb(p − 1)).
La fila i se encontrara en el proceso $\lfloor \frac{i}{mb} \rfloor$, ocupando la posici´on (i mod mb) en la submatriz local.
El ´índice de columna j no se ve alterado al transponerse a la submatriz local, pues las filas se reparten enteras.



Distribución de una matriz bidimensional: distribución por bloques de columnas

Cada proceso recibe un bloque de mb = $\binom{m}{p}$columnas de la matriz, a excepci´on del ´ultimo, que recibirá $\widehat{m_b} = (n - m_b(p-1))$.
La columna j se encontrara en el proceso $\lfloor \frac{j}{mb} \rfloor$, ocupando la posición (j mod mb) en la submatriz local.
El ´índice de fila i no se ve alterado al transponerse a la submatriz local, pues las columnas se reparten enteras.

Distribución bidimensional de una matriz bidimensional

Los procesos se organizan lógicamente como una malla bidimensional.
p = $p_x p_y$, siendo pij (i = 0, . . . , px − 1, j = 0, . . . , py − 1) el proceso que ocupa la posici´on (i, j) en dicha malla.
Cada proceso recibe una submatriz con un máximo de $m_{b_x} = \left\lceil \frac{n}{p_x} \right\rceil$filas y $m_{b_y} = \left\lceil \frac{n}{p_y} \right\rceil$ columnas
Los procesos en la fila px−1 de la malla reciben $\widehat{m_{b_x}} = n - m_{b_x}(p_x - 1)$ subfilas.

Los procesos en la columna py−1 de la malla reciben $\widehat{m_{b_y}} = n - m_{b_y}(p_y - 1)$ subcolumnas. El elemento (i, j) de la matriz global se encontrará en el procesador $\left(\left\lfloor \frac{i}{m_{b_x}} \right\rfloor, \left\lfloor \frac{j}{m_{b_y}} \right\rfloor\right)$, donde ocupará la posición (i mod m_{b_x} , j mod m_{b_y}) en la submatriz local.

Ventajas del incremento de dimensionalidad

A mayor dimensionalidad mayor flexibilidad para asignar los elementos de la matriz. En una distribución bloque unidimensional el número de procesos utilizables es igual al tamaño de la dimensión distribuida (en el ejemplo anterior, 8 procesos frente a 16 en la distribución bidimensional).
Aumenta la frecuencia de las comunicaciones (su número se duplica por cada dimensión extra). Sin embargo, el volumen de comunicaciones por proceso disminuye. En la malla de computadores, las comunicaciones se reparten mejor y se aprovecha mejor la localidad de los cálculos.

Generalización multidimensional

En general, siguiendo la misma técnica, una matriz s-dimensional puede distribuirse por bloques sobre p procesos troceando hasta k dimensiones, con k ≤ s, y organizando los procesos como una malla k-dimensional:

- Tamaño de la submatriz local en la dimensión d (0 ≤ d < s):

$$m_{b_d} = \begin{cases} \left\lceil \frac{n_d}{p_d} \right\rceil & \text{si } d < k \\ n_d & \text{en otro caso} \end{cases}$$

$$\hat{m_{b_d}} = \begin{cases} n_d - m_{b_d}(p_d - 1) & \text{si } d < k \\ n_d & \text{en otro caso} \end{cases}$$

Distribución cíclica por bloques

La distribución cíclica por bloques consigue equilibrar mejor la carga de trabajo:

- Definimos un tamaño de bloque en cada una de las dimensiones de la matriz.
- La matriz se trocea en bloques de dicho tamaño.
- Por último, los bloques se asignan a los procesos de la malla de forma cíclica.

Cuando se toman bloques de un elemento, la distribución se denomina distribución cíclica.

El objetivo de esta distribución es tratar de balancear la carga de forma equitativa entre los procesos, independientemente de la regularidad de los cálculos sobre la matriz.
Debe contrastarse esta ganancia frente a la reducción de localidad y el consiguiente incremento en los costes de comunicaciones, especialmente al utilizar bloques de tamaño reducido.

Siguiendo una distribución cíclica por bloques, una matriz s-dimensional puede distribuirse sobre una malla k-dimensional de procesadores usando tamaños de bloque mb0, . . ., mbk−1:

- Sea mbd el tamaño de bloque en la dimensión d (0 ≤ d < k).
- Número de bloques a repartir en la dimensión d (0 ≤ d < k):

$$\#b = \left\lceil \frac{n_d}{m_{b_d}} \right\rceil$$

- Número de bloques recibidos el proceso pj en la dimensión d (0 ≤ d < k):

$$b_d = \begin{cases} \left\lceil \frac{\#b}{p_d} \right\rceil & \text{si } j < (\#b \mod p_d) \\ \left\lfloor \frac{\#b}{p_d} \right\rfloor & \text{en otro caso} \end{cases}$$

La distribución cíclica por bloques es la distribución por bloques más general de una matriz n × m:

- Si se escogen bloques de un elemento, se obtiene la distribución cíclica.
- Si se escogen bloques de tamaño $\left(\left\lceil \frac{n}{p} \right\rceil \times m\right)$ y una malla de procesos p × 1, obtenemos una distribuci´on por bloques de filas.
- Si se escogen bloques de tamaño $\left(n \times \left\lceil \frac{m}{p} \right\rceil\right)$ y una malla de procesos 1 × p se obtiene una distribuci´on por bloques de columnas.
- Por ´ultimo, dada una malla de procesos px × py, si se escogen bloques de tamaño $\left(\left\lceil \frac{n}{p_x} \right\rceil \times \left\lceil \frac{m}{p_y} \right\rceil\right)$ se obtiene una distribución por bloques bidimensionales.

Compromiso

- A menor tamaño de bloque, mejor balanceo de carga.
- A mayor tamaño de bloque, menos coste de comunicaciones.

Para ello, habrá que considerar las características del algoritmo y de la plataforma paralela usada para su ejecución.

Asignación de grafos de dependencias estáticos

En numerosas ocasiones el cálculo a realizar no se define mediante una matriz, por lo que los esquemas de distribución vistos son ineficaces.
La fase de asignación consistirá en encontrar un reparto de las tareas del grafo de dependencias y un orden de ejecución que minimice el tiempo de ejecución.
En el caso general, este problema es NP-completo.
Pero existen situaciones concretas y regulares (y muy comunes) para las que se conocen soluciones ´optimas aplicables como patrones.

Reducciones

Cada proceso calcula la suma de su bloque.
La suma acumulada se obtiene siguiendo una estructura recursiva.
Árbol binario perfecto: log(n) niveles, con n número de tareas hoja en el árbol. Todas las operaciones de reducción tienen un grafo como este.
Sólo se pueden ejecutar de forma concurrente las tareas del mismo nivel, por lo que podemos agrupar las de diferentes niveles sin reducir las oportunidades de paralelismo: ´árbol binomial de orden k.

Árbol binomial de orden ´ k

- Se forma uniendo los nodos raíz de dos ´arboles binomiales de orden (k − 1), y haciendo que uno de ellos sea la raíz del nuevo ´árbol.
- 2 k nodos, profundidad k.
- Permite realizar una operación de reducción sobre k datos en log(k) pasos.
- Ciertos procesos dejan de participar en el cálculo tras un determinado número de pasos.
- Inevitable e impuesto por el grado medio de concurrencia del grafo de dependencias.

Reducción con replicación

Usando el enfoque anterior emplearíamos dos ´arboles binomiales de procesos, llevando en total 2 log(k) pasos realizar la operación para k datos de entrada.
Es posible resolverlo en log(k) pasos usando una estructura en hipercubo.

Esquemas de asignación dinámica

En estos esquemas, el trabajo se reparte durante la ejecución del algoritmo paralelo. Son adecuados cuando:

- Las tareas se generan dinámicamente (no existe ´árbol estático de dependencias).
- El tamaño de las tareas no se conoce a priori (la distribución de la carga no se puede determinar estáticamente).

Su diseño e implementación es más complejo que el de los esquemas estáticos, e imponen una sobrecarga en tiempo de ejecución. Se necesita un algoritmo de detección de terminación.
Las tareas obtenidas en la descomposición se convierten en estructuras de datos que representan subproblemas. Estos se recopilan en una colección desde la que se van asignando a los procesos. En función de cómo se gestione esta colección, los esquemas de asignación pueden ser:

- Centralizados.
- Descentralizados.

Esquemas centralizados

La colección de problemas se mantiene centralizada en una única localización.
Un proceso maestro se encarga de la gestión y reparto de la carga a los procesos de cómputo, denominados esclavos.
El proceso maestro reparte trabajos a cada uno de los (p − 1) procesos esclavos repetidamente. En respuesta, cada esclavo podrá devolver un resultado y/o generar nuevos subproblemas, que se enviarán al maestro para ser añadidos a la colección centralizada.
Estrategia efectiva con número moderado de esclavos y cuando el coste de ejecutar subproblemas es alto comparado con el de obtenerlos.
En caso contrario, las comunicaciones con el proceso maestro se convierten en un cuello de botella.

Planificación por bloques:

- Los procesos esclavos captan las tareas en bloques de subproblemas.
- Se reduce la contención en las comunicaciones con el maestro.
- Si los bloques son demasiado grandes se provocarán desequilibrios de carga, especialmente hacia el final de la ejecución.

Colecciones locales de subproblemas:

- Cada esclavo almacena localmente para su resolución algunos de los nuevos subproblemas generados.
- Reduce la contención en las comunicaciones con el maestro.
- Nuevamente, puede producir desequilibrios de carga si los esclavos almacenan demasiados subproblemas sin enviarlos al maestro.

Captación anticipada: la recepción de nuevos problemas se solapa con el cálculo de los anteriores para mejorar el grado medio de concurrencia.

Se determina que el algoritmo ha finalizado cuando se satisfacen las siguientes condiciones:

- La colección de subproblemas centralizada está vacía.
- Todos los procesos esclavos esta ociosos y han solicitado nuevos subproblemas al maestro.

Esquemas completamente descentralizados

No existe un maestro central, sino que los subproblemas se encuentran distribuidos entre los repositorios locales de cada uno de los procesos.
Todos los procesos participan tanto en la resolución de problemas como en el equilibrado de la carga.
Equilibrado de la carga:

- Iniciado por el receptor: es el proceso que necesita trabajos el que los solicita al resto.
- Iniciado por el emisor: un proceso que mantiene un número elevado de subproblemas en su repositorio local decide repartirlos entre otros procesos.
- Esquemas mixtos: mezclar ambas estrategias dependiendo de la carga global del sistema (si ´esta puede ser estimada).

Modos de seleccionar el proceso al que se solicita trabajo en un esquema iniciado por el receptor:

- Sondeo aleatorio:
    - Se elige aleatoriamente al proceso receptor de la petición.
    - Equilibra las solicitudes de trabajo recibidas por cada proceso.
- Sondeo cíclicos:
    - Cada proceso mantiene una variable x, inicializada a (i + 1), indicando el proceso al que debe solicitar trabajo.
    - En cada petición, la variable x se incrementa cíclicamente, saltando al proceso emisor.
    - Ventaja: en caso de no obtener trabajo de un determinado proceso, se solicita a uno diferente.
    - Inconveniente: puede provocar múltiples peticiones simultáneas al mismo proceso si existe un alto acoplamiento.
    - Solución: centralizar x, lo que incrementa los costes de interacción.

Detección de terminación en esquemas descentralizados sin reactivación

Un proceso puede estar activo, resolviendo problemas, o en espera, cuando se ha quedado sin trabajo y ningún otro proceso ha podido cederle tareas.
Cuando el proceso 0 pasa a espera envía un testigo de detección de fin al anterior proceso en el anillo $(p - 1)$.

Cuando un proceso j recibe el testigo:

- Si está en espera lo retransmite al proceso (j − 1).
- Si está activo lo retiene hasta pasar a espera.

Cuando el proceso 0 recibe nuevamente el testigo, sabe que todos han terminado.

Algoritmo de terminación de Dijkstra

Si se permite la reactivación, el algoritmo anterior puede fallar si el proceso i reenvía el testigo tras quedarse en espera pero es más tarde reactivado por recibir tareas de otro proceso j para j > i.
Se incluye más información en el esquema: cada proceso tendrá un color (blanco o negro), al igual que el testigo.
Inicialmente el testigo y todos los procesos son blancos.
Cuando un proceso i envía un mensaje de trabajo a otro j, siendo j > i, cambia su color a negro.
Un proceso en espera con color negro reenviará el testigo con color negro y, tras ello, cambiará su color a blanco.
Si el proceso 0, siendo blanco, recibe un testigo negro, cambia su color a blanco.
El proceso termina cuando el proceso 0, siendo de color blanco, recibe un testigo blanco.