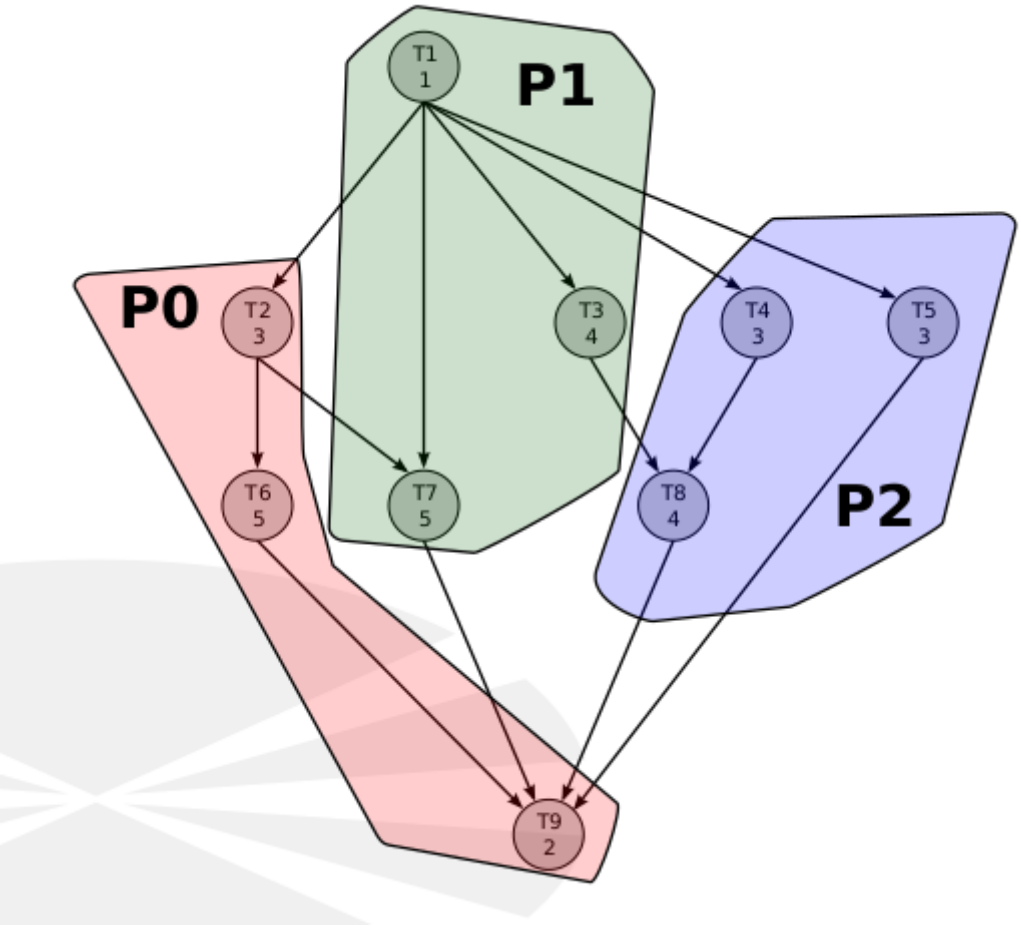
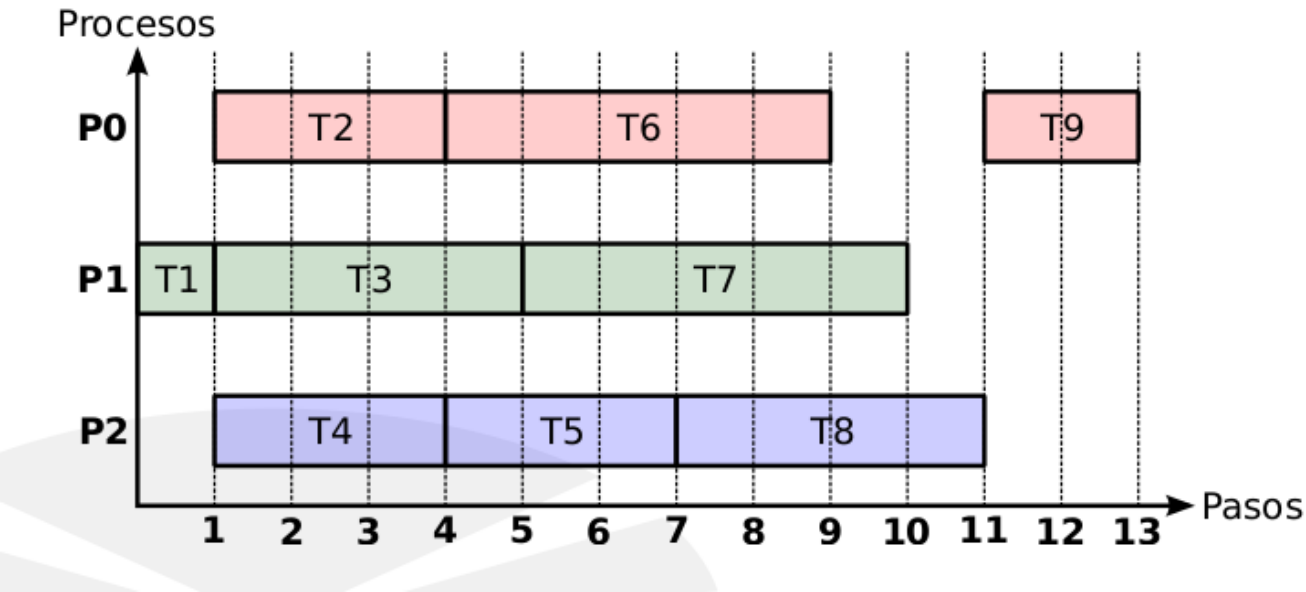
Tema 4-Metodología de programación paralela

Tras la descomposición el siguiente paso es definir una correspondencia entre las tareas y los procesos que las van a ejecutar. Por ejemplo:





Se busca minimizar el tiempo de computación, de comunicaciones y de inactividad entre los procesos (desequilibrios de carga y dependencias).

Esquemas de asignación estática

Cuando se conoce el grafo de tareas antes de la ejecución se pueden tomar antes las decisiones de asignación de tareas. Hay dos tipos:

- Para descomposición de dominio->distribución por bloques
- Para descomposición funcional o recursiva->grafos de dependencias

Distribución por bloques

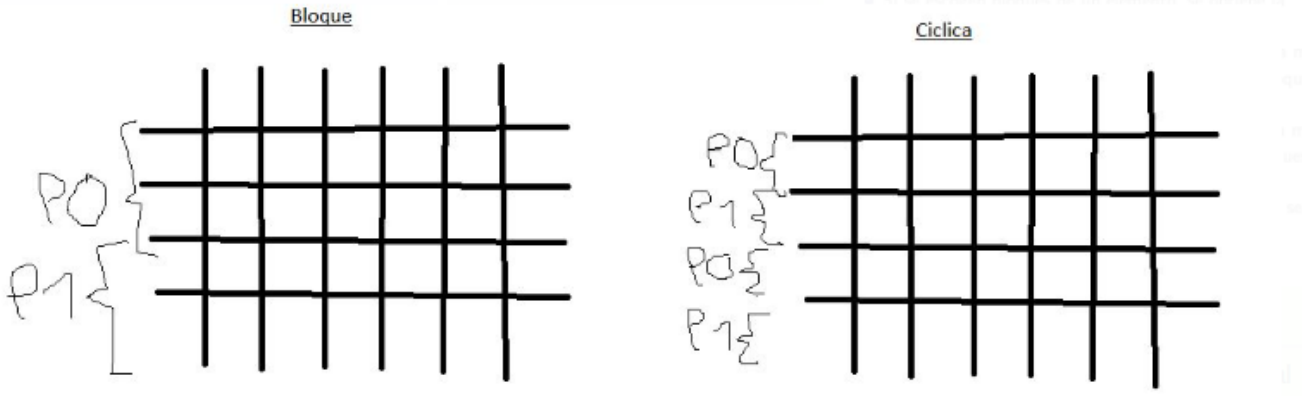
Si tenemos p procesos, a cada proceso se le asignará un bloque de tamaño n/p redondeado hacia arriba. Por ejemplo, si tenemos 20 bloques y 3 procesos:
P1->0-6
P2->7-13
P3->14-19 (al último se le asignan los bloques sobrantes)

Esto mismo se puede hacer por columnas o por bloques de columnas y filas. Generalizando esto en k dimensiones, el tamaño de una submatriz en la dimesión d sería:

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

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

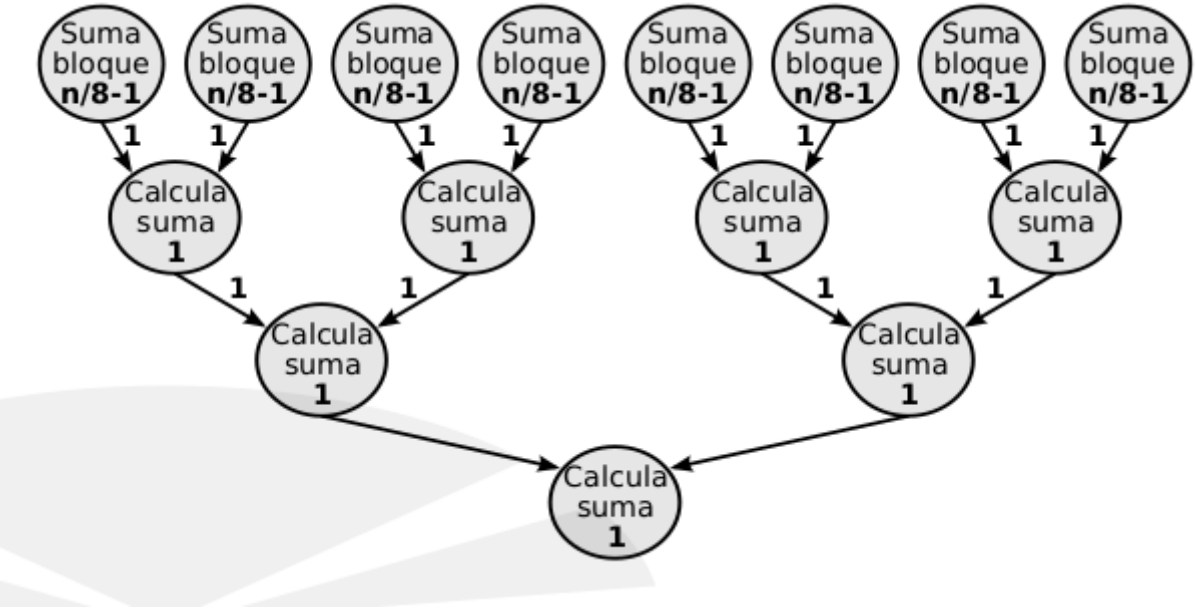
Explicación de Antón de distribución cíclica:



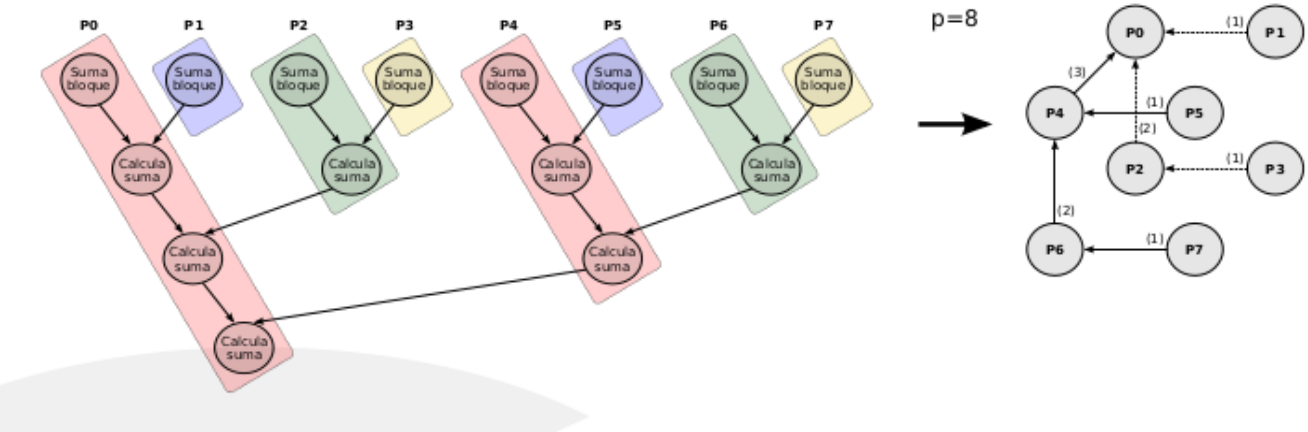
El objetivo de la distribución cíclica por bloques es balancear la carg de forma equitativa entre los procesos, independientemente de la regularidad de los cálculos sobre la matriz.

Grafos de dependencias estáticos

Existen muchas situaciones donde el cálculo no se define como una matriz, sino como un un grafo de dependencias. Por ejemplo, en el producto escalar:



En este caso forma un árbol binario de orden k. Para aplicar una reducción a k datos necesitamos log2(k) pasos.



Esquemas de asignación dinámica centralizados

En un esquema centralizado, todos los problemas o tareas se mantienen en una única ubicación centralizada. Un proceso maestro se encarga de administrar y distribuir la carga de trabajo a los procesos esclavos. El proceso maestro asigna trabajos repetidamente a cada uno de los procesos esclavos, que realizan el cómputo. A medida que los esclavos procesan los trabajos asignados, pueden devolver resultados y, en algunos casos, generar nuevos

subproblemas. Estos subproblemas se envían de vuelta al proceso maestro para que los agregue a la colección centralizada de problemas.

Este enfoque es efectivo cuando se trabaja con un número moderado de procesos esclavos y el costo de ejecutar subproblemas es alto en comparación con el costo de obtenerlos. En otras palabras, si la ejecución de cada subproblema lleva mucho tiempo y el tiempo dedicado a las comunicaciones con el proceso maestro es relativamente pequeño en comparación, entonces este esquema es beneficioso.

Sin embargo, si el costo de ejecutar los subproblemas es bajo en comparación con el tiempo que lleva comunicarse con el proceso maestro, entonces las comunicaciones con el proceso maestro se vuelven un cuello de botella. Esto significa que la eficiencia general del sistema puede verse afectada debido a la sobrecarga de comunicación entre el proceso maestro y los esclavos.

Algunos esquemas son:

- **Planificación por bloques**:es una estrategia de asignación de tareas en sistemas distribuidos donde los procesos esclavos capturan tareas en bloques de subproblemas. En lugar de recibir una tarea a la vez, los esclavos toman un bloque de subproblemas para procesar. Esto reduce la contención en las comunicaciones con el proceso maestro, ya que los esclavos no tienen que comunicarse constantemente para solicitar nuevas tareas. Sin embargo, es importante tener en cuenta el tamaño de los bloques de subproblemas. Si los bloques son demasiado grandes, puede haber desequilibrios de carga. Esto significa que algunos esclavos pueden terminar su bloque de subproblemas antes que otros, lo que puede generar una carga desigual hacia el final de la ejecución. Por lo tanto, es necesario encontrar un equilibrio adecuado en el tamaño de los bloques para evitar desequilibrios significativos.
- **Colecciones locales de subproblemas**:cada esclavo almacena localmente algunos de los nuevos subproblemas generados para su resolución. Esto también reduce la contención en las comunicaciones con el proceso maestro, ya que los esclavos no tienen que enviar cada subproblema generado inmediatamente al maestro. Sin embargo, al igual que con la planificación por bloques, el almacenamiento local de subproblemas puede causar desequilibrios de carga si los esclavos almacenan demasiados subproblemas sin enviarlos al maestro. Si un esclavo acumula demasiados subproblemas sin compartirlos con el maestro, puede haber una carga desigual en la distribución de tareas y recursos.
- **Captación anticipada**: es una estrategia en la que los esclavos superponen la recepción de nuevos subproblemas con el cálculo de los subproblemas anteriores. En lugar de esperar a que un subproblema se resuelva por completo antes de recibir el siguiente, los esclavos pueden recibir continuamente nuevos subproblemas mientras trabajan en los

anteriores. Esto mejora el grado medio de concurrencia, lo que significa que los esclavos pueden realizar cálculos en paralelo de manera más eficiente.

El algoritmo finaliza cuando la colección de sobproblemas está vacía o cuando todos los procesos esclaves han solicitado nuevos subproblemas.

Esquemas de asignación dinámica descentralizados

En un esquema completamente descentralizado, no hay un maestro central que controle la asignación de subproblemas. En su lugar, los subproblemas se distribuyen entre los repositorios locales de cada proceso. Cada proceso es responsable de gestionar los subproblemas en su propio repositorio y contribuye a la solución general del problema.

El equilibrado de la carga es una parte importante en los esquemas de asignación dinámica. Puede ser iniciado por el receptor o por el emisor. Cuando es iniciado por el receptor, es el proceso que necesita trabajos el que solicita subproblemas al resto de los procesos. Por otro lado, cuando es iniciado por el emisor, un proceso que tiene un gran número de subproblemas en su repositorio local decide distribuirlos entre otros procesos para equilibrar la carga de trabajo.

Además, existe la posibilidad de utilizar esquemas mixtos, donde se combinan ambas estrategias dependiendo de la carga global del sistema. Si se puede estimar la carga global, se puede decidir cuándo aplicar el equilibrado de la carga iniciado por el receptor o por el emisor para optimizar el rendimiento del sistema.

Los modos de seleccionar el proceso al que se le asigan el trabajo son:

- **Sondeo aleatorio**: el proceso que solicita trabajo elige al proceso receptor de manera aleatoria. Esto significa que no hay un patrón predefinido o criterio específico para seleccionar al proceso receptor. La elección aleatoria tiene la ventaja de equilibrar las solicitudes de trabajo recibidas por cada proceso, ya que no hay preferencia sistemática hacia un proceso en particular. Sin embargo, también puede haber cierta variabilidad en la carga de trabajo asignada a cada proceso debido a la aleatoriedad.
- **Sondeo cíclico**: cada proceso mantiene una variable, llamada "x", que se inicializa a un valor específico, por ejemplo, (i + 1), donde "i" es el identificador del proceso. Esta variable indica el proceso al que el proceso actual debe solicitar trabajo. En cada petición de trabajo, la variable "x" se incrementa cíclicamente, lo que significa que salta al siguiente proceso en secuencia. La ventaja del sondeo cíclico es que, en caso de no obtener trabajo de un determinado proceso, se solicita a otro proceso diferente. Esto ayuda a evitar la dependencia excesiva de un proceso en particular y distribuye las solicitudes de trabajo de manera más equitativa. Sin embargo, un inconveniente potencial es que si existe un alto acoplamiento entre los procesos, es posible que varios procesos realicen peticiones

simultáneas al mismo proceso receptor, lo que puede generar cuellos de botella y afectar negativamente al rendimiento del sistema. Una posible solución para mitigar este inconveniente en el sondeo cíclico es centralizar la variable "x", lo que significa que un proceso centralizado controla y actualiza el valor de "x" para todos los procesos. Esto puede ayudar a evitar las peticiones simultáneas al mismo proceso receptor, ya que se garantiza que la asignación de trabajo sea controlada y equilibrada. Sin embargo, esta solución también incrementa los costos de interacción, ya que se requiere una mayor comunicación entre los procesos y el proceso centralizado encargado de gestionar la variable "x".

En un sistema que trabaja en forma de anillo:
Cada proceso puede estar en un estado activo, resolviendo problemas, o en un estado de espera, cuando se queda sin trabajo y ningún otro proceso puede cederle tareas.

Cuando el proceso 0 pasa al estado de espera porque ha agotado su trabajo, envía un mensaje especial llamado "testigo de detección de fin" al proceso anterior en el anillo, que en este caso es el proceso p-1 (donde p es el número total de procesos en el anillo). Este mensaje de testigo indica que el proceso 0 ha finalizado su trabajo y está esperando la señal de finalización de los demás procesos.

Cuando un proceso j (distinto de 0) recibe el testigo de detección de fin, se evalúa su estado actual. Si el proceso j está en estado de espera, retransmite el mensaje al proceso anterior en el anillo, es decir, al proceso j-1. Esto se hace para que el testigo de detección de fin continúe circulando por el anillo y se propague a los procesos que aún están activos.

Por otro lado, si el proceso j está en estado activo, retiene el testigo de detección de fin hasta que termine su trabajo y pase al estado de espera. En ese momento, el proceso j también retransmite el mensaje al proceso j-1 para que el testigo siga circulando.

Este proceso de retransmisión continúa hasta que el testigo de detección de fin llega nuevamente al proceso 0. Cuando el proceso 0 recibe nuevamente el testigo, sabe que todos los procesos han pasado al estado de espera, lo que indica que todos han finalizado su trabajo.

En el algoritmo de terminación de Dijkstra:

1. Inicialización: Todos los procesos y el testigo se establecen inicialmente en color blanco.
2. Cuando un proceso i envía un mensaje de trabajo a otro proceso j, donde j > i, el proceso i cambia su color a negro. Esto indica que el proceso i ha enviado trabajo y está en un estado activo.
3. Si un proceso en estado de espera recibe el testigo para reenviarlo, y ese proceso tiene color negro, reenvía el testigo manteniendo el color negro. Después de reenviar el testigo,

el proceso cambia su color nuevamente a blanco. Esto asegura que el testigo mantenga el color negro cuando haya procesos en espera que hayan enviado trabajo recientemente.

4. Si el proceso 0, que es el proceso inicial en el algoritmo de Dijkstra, recibe un testigo de color negro, cambia su color a blanco. Esto se hace para garantizar que el proceso 0 pueda recibir el testigo con color blanco solo cuando todos los procesos hayan terminado.
5. El algoritmo termina cuando el proceso 0, estando en color blanco, recibe un testigo con color blanco. Esto indica que todos los procesos han finalizado y que se ha completado la ejecución del algoritmo.

Al introducir los colores y las reglas de cambio de color, se evita el problema de que un proceso reactivado por otro proceso después de haber enviado trabajo pueda interrumpir incorrectamente la terminación del algoritmo. Los colores negro y blanco se utilizan como marcas para indicar si un proceso ha enviado trabajo recientemente o no, y el testigo con su color correspondiente se propaga y se verifica adecuadamente durante el proceso de terminación.