**Tipo/s de paralelismo usado:**

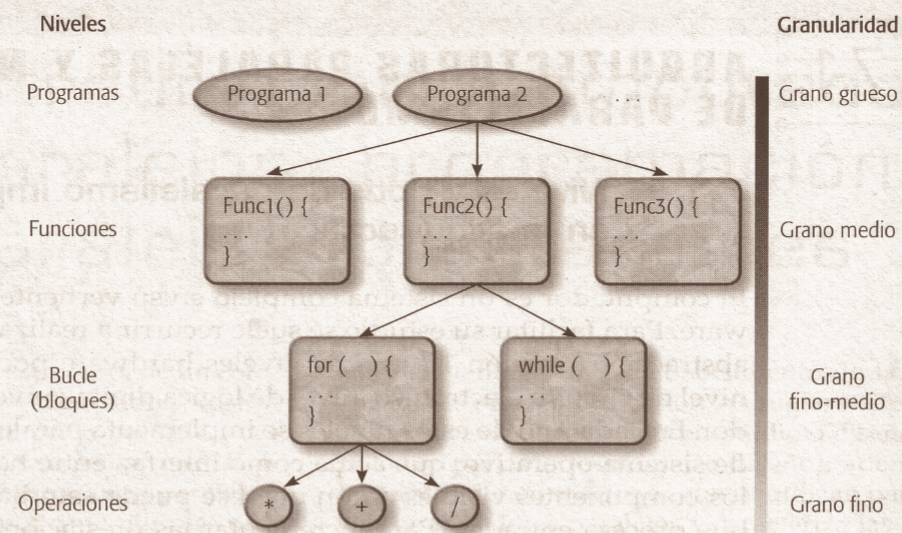
A nivel de paralelismo podemos conseguir optimizaciones mediante el reordenamiento de las operaciones para permitir que múltiples cálculos se produzcan en paralelo, ya sea en la instrucción, la memoria o a nivel de hilo.

Antes de tratar el tipo de paralelismo usado en la aplicación que hemos paralelizado, introduciremos brevemente los tipos de paralelismo que podemos encontrarnos:

* **Paralelismo funcional:** Se consigue gracias a una reorganización de la estructura lógica del código. Este es aprovechado cuando las funciones, bloques, instrucciones, etc. sean iguales o distintas se ejecuten en paralelo.

Existen diferentes niveles de paralelismo funcional los cuales trataremos brevemente:

1. Nivel de instrucciones u operación: Se produce cuando se ejecutan en paralelo las instrucciones de un programa. Compone el nivel de granularidad más fina.
2. Nivel de bucle: Se produce cuando se ejecutan en paralelo distintas iteraciones de un bucle o secuencias de instrucciones. Compone un nivel de granularidad fina-media.
3. Nivel de Funciones: Se produce cuando los distintos procesos que componen un programa se ejecutan simultáneamente. Compone el nivel de granularidad medio.
4. Nivel de programas: Se produce cuando se ejecutan simultáneamente programas que pueden ser o no de una misma aplicación. Compone el nivel de granularidad grueso.

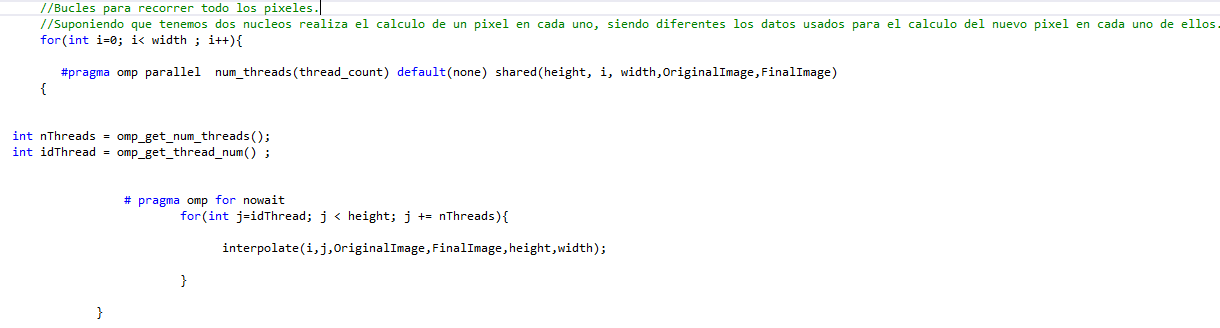


* **Paralelismo de datos:** Se obtiene mediante la división del conjunto de datos de entrada de la aplicación, asignando a cada procesador un conjunto de estos datos. Cada procesador realiza la misma operación sobre un conjunto de datos distintos. El paralelismo de datos se explota cuando una misma instrucción o función, se ejecuta repetidas veces en paralelo con datos diferentes.

Es por lo anteriormente citado que este tipo de paralelismo es usado muy frecuentemente en los casos donde tenemos una taxonomía de Flynn de la clase SIMD (una instrucción y múltiples datos) y en los casos de clase MIMD (múltiples instrucciones y datos), cuando la aplicación a implementar sea un programa paralelo SPMD (Simple Program Múltiple Data) como es nuestro caso.

* **Paralelismo explícito:** Este tipo de paralelismo no es inherente en las estructuras de programación y es el programador el que mediante código debe cambiar este hecho.
* **Paralelismo implícito**: Paralelismo presente en las estructuras de datos y aplicaciones, queda dentro del procesador, no trasciende al exterior y, por tanto, no afecta a la programación.

Una vez vistos y definidos los tipos de paralelismo veremos claro los tipos de paralelismo usados en nuestra aplicación:



Como vemos en la imagen las estructuras que hemos paralelizado se tratan de dos bucles **for,** uno contenido en otro. Como también podemos observar la paralelización de estas estructuras nos origina un paralelismo función a nivel de bucle en nuestra aplicación. Este tipo de paralelismo se origina cuando se ejecutan en paralelo distintas iteraciones de un bucle y posee un nivel de granularidad fina-media.

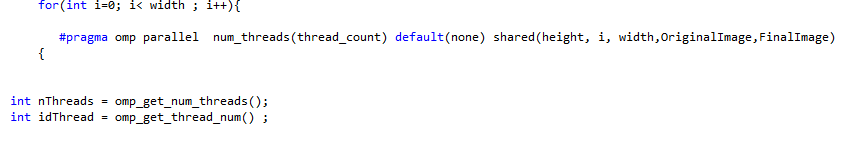
Para ver más claro este hecho explicaremos que realiza la ejecución de estos bucles (cuál es su resultado):

Suponiendo que tenemos dos núcleos, nuestro programa realiza el cálculo de un pixel en cada uno, siendo diferentes los datos usados para el cálculo del nuevo pixel en cada uno de ellos, es decir, realizamos siempre la misma operación pero sobre un conjunto de datos distintos cada vez. Este hecho nos indica que nuestro programa sigue un tipo de programación paralela del tipo SPMD (un solo programa, múltiples datos), lo cual se tratara en el apartado siguiente. Este modo de programación paralela es casualmente el indicio que necesitamos para saber que nuestro programa también presenta un paralelismo de datos. El paralelismo de datos es obtenido mediante la división del conjunto de datos de entrada de la aplicación, asignando a cada procesador un conjunto de estos datos. Cada procesador realiza la misma operación sobre un conjunto de datos distintos. Esta definición encaja perfectamente con el objetivo que tienen las estructuras paralelizadas: realizar el cálculo de los pixeles en los diferentes procesadores, de modo que realizan siempre la misma operación pero con distintos datos.

Para aclara bien lo citado con anterioridad explicaremos las estructuras paralelizadas:

Antes de comenzar debemos añadir que se ha realizado la paralelización mediante directivas de la biblioteca **OpenMP.**

El primer bucle simplemente itera hasta el final de la imagen a modo de filas.



Ahora explicaremos las clausulas:

Primeramente hacemos uso de la cláusula **parallel:** Mediante la cláusula **parallel** indicamos que la estructura que viene a continuación tendrá una ejecución paralela, es decir, la siguiente estructura se verá afectada por el **pragma** y **parallel** puede tomar los núcleos de manera automática y paralelizara la siguiente estructura. Para ello la directiva crea un conjunto de hebras que se obtienen mediante **OMP\_NUM\_THREADS**.

La cláusula **num\_threads(numero\_hilos)** asigna el número de procesadores que usaremos en la paralelización.

La cláusula **Default(none)** fuerza al programador a explícitamenteespecificar como se comparten los atributos de todas las variables en la región paralelizada. Esta es la razón del uso de la cláusula **shared** que de otra manera vendría por defecto y deja como públicas las variables siguientes: **height**, **i**, **width**, **OriginalImage** y **FinalImage**.

Comentaremos ahora las asignaciones que usamos dentro del primer bucle para obtener el número de núcleos de nuestro computador y obtener un identificador para cada hilo.

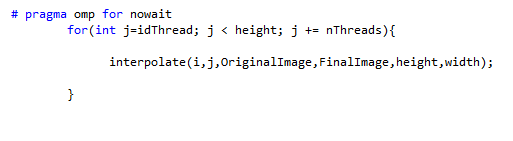
**int nThreads = omp\_get\_num\_threads();**

Como ya habíamos citado obtenemos el número de núcleos del computador y para ellos usamos la **omp\_get\_num\_threads()** que nos devuelve el número de hilos y lo guardamos en la variable **nThreads** para su posterior uso.

**int idThread = omp\_get\_thread\_num() ;**

Es mediante **omp\_get\_thread\_num()** que obtenemos el identificador de cada hilo que será necesario en el momento de asignar los datos a cada hilo en la ejecución del bucle **for.**

El segundo bucle es el que itera **j** a la cual se le asigna el identificador del primer hilo e itera para asignar a cada hilo la ejecución de la misma operación pero con datos distintos.



La cláusula **# pragma omp for nowait** localiza la estructura iterativa **for** y hace que se distribuyan las iteraciones del bucle entra las hebras creadas. La cláusula **nowait** simplemente deja asíncrona la operación anterior.

**Modos de programación paralela:**

* **SPMD (Un solo programa con múltiples datos):** Varios procesadores ejecutan el mismo programa, de manera simultánea, con diferentes entradas de datos. Este modo de programación también es denominado **paralelismo de datos.**
* **MPMD (Múltiples programas con múltiples datos):** Se ejecutan programas independientes con diferentes entradas de datos en cada procesador. En este caso la aplicación a ejecutar se divide en unidades independientes que se asignan a procesadores distintos y trabajan con conjuntos de datos distintos. Este modo de programación también es denominado **paralelismo de funciones.**

* **Modo Mixto SPMD-MPMD:** Programas que requieren o pueden ser usados ambos modelos. Algunos ejemplos son los programas que usan estrategias como divide y vencerás, map-reduce, programas que hacen uso de sistema de tuberías para comunicarse o streaming(difusión). También debemos destacar la variante **SPMD-MPMD** que se obtiene mediante el uso de la programación **dueño-esclavo** al crear esclavos dinámicamente y con el mismo código.

El más usado, y que usaremos en nuestra práctica, es el modo **SPMD** ya que evita, en los sistemas con memoria distribuida, la necesidad de tener que distribuir el código entre nodos y solo teniendo que distribuir los datos. Un indicativo de este hecho es que nuestra aplicación tiene paralelismo a nivel de datos y como hemos explicado en el apartado anterior nuestro programa itera la misma operación pero con datos distinto, hecho que encaja perfectamente con la definición de un programa **SPMD**.

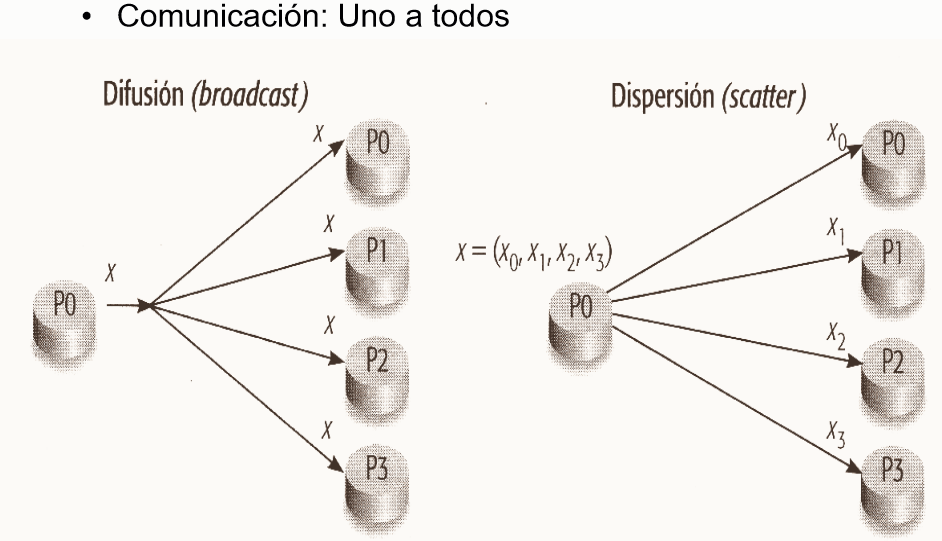
**Qué alternativas de comunicación (explícitas o implícitas emplea su programa).**

Mediante el uso de herramientas (Bibliotecas como OPENMP y compiladores paralelos) podemos conseguir que exista comunicación entre los distintos procesos. Existen casos en los que pueden ser usados para la comunicación explicita de datos entre procesos del grupo que colabore en el mismo código y otros casos para sincronizar un grupo de procesadores.

Antes de comenzar a hablar de nuestro programa, consideramos oportuno una breve explicación sobre las alternativas de comunicación (funciones colectivas) que podría emplear nuestro programa:

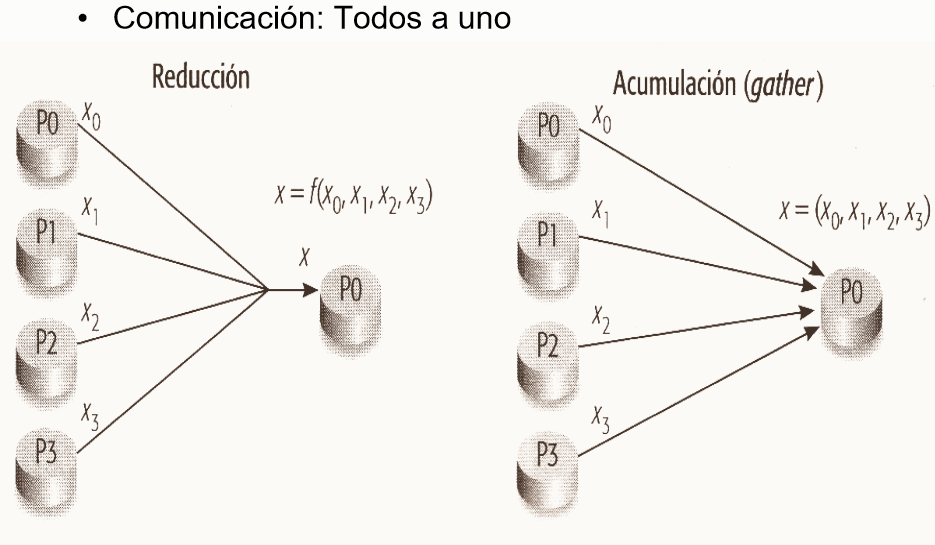
1. **Comunicación múltiple uno-a-uno:** Sucede cuando un proceso envía un dato o estructura de datos y otro lo recibe. Si todos los procesos envían o reciben es necesario implementar una **permutación.**

Podemos ver unos ejemplos en las siguientes imágenes:



1. **Comunicación uno-a-todos:** Sucede cuando un proceso es el que envía y todos los demás son los que reciben. Podemos distinguir dos casos en este tipo de comunicación:
   * + 1. **Difusión:** En el que todos los procesos reciben el mismo mensaje.
       2. **Dispersión:** En el que cada proceso recibe un mensaje diferente.

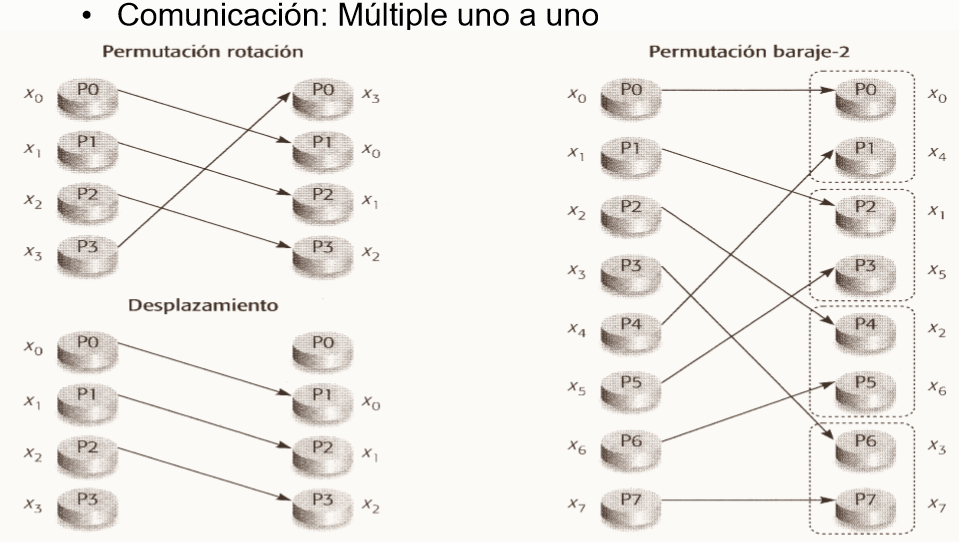
Podemos ver unos ejemplos en las siguientes imágenes:



1. **Comunicación todos a uno:** Sucede cuando todos los procesos envían a un único proceso. Podemos encontrar de dos tipo:

* **Reducción**: En el que os mensajes enviados se combinan en uno solo.
* **Acumulación**: Sucede cuando el receptor recibe los mensajes de forma concatenada (en orden).

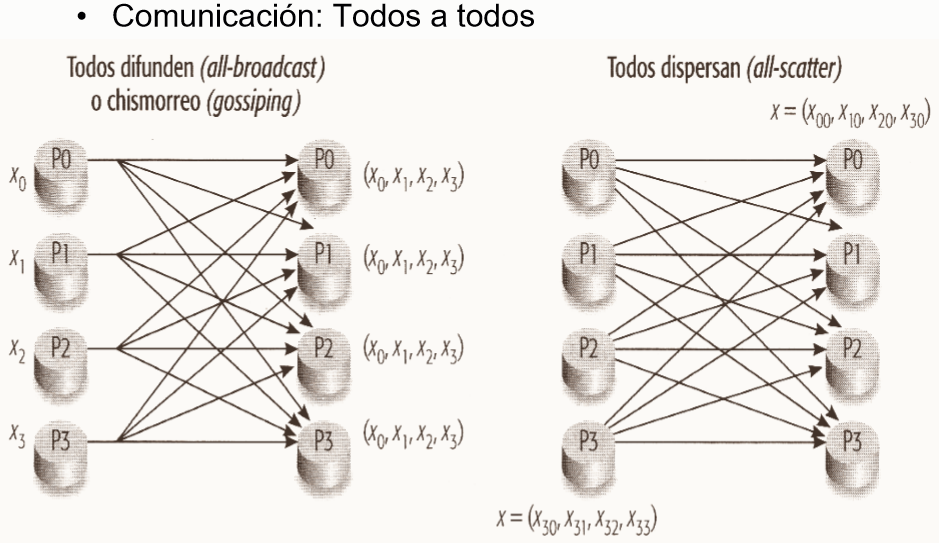
Podemos ver unos ejemplos en las siguientes imágenes:



1. **Comunicación todos-a-todos:** Sucede cuando todos los procesos realizan una comunicación **uno a todos.** Podemos encontrar de dos tipos:

* **Todos difunden:** En el que todos los procesos realizan una difusión (todos reciben lo mismo).
* **Todos dispersan:** En este caso los procesos concatenan diferentes transferencias.

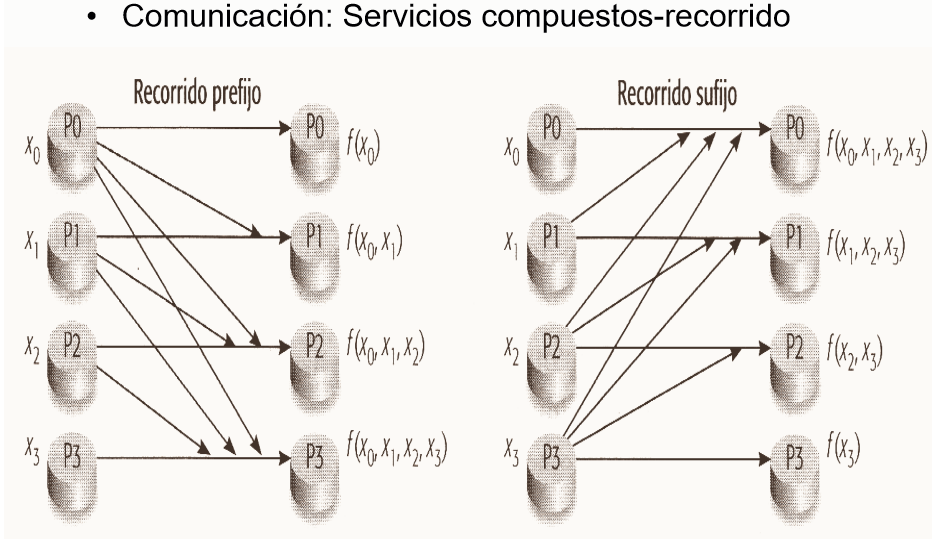
Podemos ver unos ejemplos en las siguientes imágenes:



1. **Comunicaciones colectivas compuestas:** Son el resultado de la unión de algunas de las alternativas anteriores. Podemos encontrar de tres tipos:

* **Todos combinan o reducción y extensión:** El resultado de una reducción y una difusión o realizando una reducción en todos los procesos.
* **Barrera:** Detiene los procesos hasta que estén todos los de un grupo para conseguir su sincronización.
* **Recorrido:** Todos los procesos envían y reciben el resultado de reducir el conjunto de que se envió por todos.

Podemos ver unos ejemplos en las siguientes imágenes:



* **Estilo de programación paralela.**

Diferenciaremos tres estilos de programación que se usan alternativamente según cual favorezca más la implementación por hardware.

1. **Paso de mensajes (para multicomputadores):** Se caracteriza en que cada procesador tiene un espacio de direcciones propio.

Para el uso de este tipo de programación se dispone de diversas herramientas como: Ada, Occam y bibliotecas como MPI. Todas estas herramientas deben cumplir el requisito de poder realizar una comunicación **uno-a-uno.** Para esto último hacen uso de dos funciones:

* + - Send(proceso\_destino, datos): para enviar datos.
    - Receive(proceso\_fuente, datos): para recibir datos.

Las transmisiones pueden ser síncronas o asíncronas según si existe bloque entre él envió y la recepción o no. Aunque existen herramientas para que esto no pase.

1. **Variables compartidas (para multiprocesadores):** Se caracteriza en que cada procesador comparte el mismo espacio de direcciones. Para el uso de este tipo de programación se dispone de diversas herramientas como: Ada 95º java y bibliotecas como **OPENMP** la cual usamos en nuestra aplicación. La comunicación entre los procesos se realiza con el acceso a variables compartidas. Las herramientas de este estilo proporcionan recursos como semáforos, monitores, secciones críticas para sincronía, etc.
2. **Paralelismo de datos (para procesadores matriciales (SIMD)):** Como ya hemos comentado con anterioridad se trata de ejecutar en paralelo una misma operación para un conjunto de datos distinto. Su paralelismo solo es soportado a nivel de bucle. Este estilo trata de aprovechar el paralelismo inherente de las estructuras, para ello el programador debe crear **construcciones** (funciones de la biblioteca **OPENMP** en nuestro caso)que permitirán aprovecharlo.

Para finalizar hay que añadir que existen herramientas de programación (como la usada **OPENMP,** compiladores con opciones de paralelización, etc.**)** quepermiten la implementación de varios estilos programación al mismo tiempo.

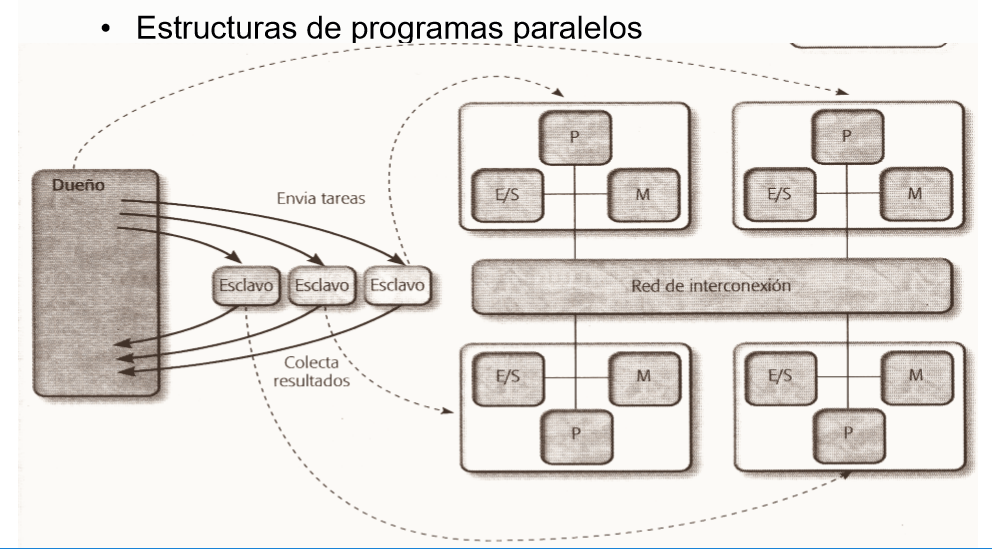
Tratando la pregunta que nos ocupa podemos encontrar dos tipos de estilo de programación paralela en nuestro código:

El primero de ellos es el estilo de **paralelismo de datos,** este paralelismo es soportado solo a nivel de bucle (como ocurre en nuestro código) y aprovecha el paralelismo inherente de las estructuras (modelos de matrices o vectores como en nuestro caso). Para aprovechar el paralelismo es necesario que el programador cree construcciones propias mediante el uso de herramientas como **OpenMP.**

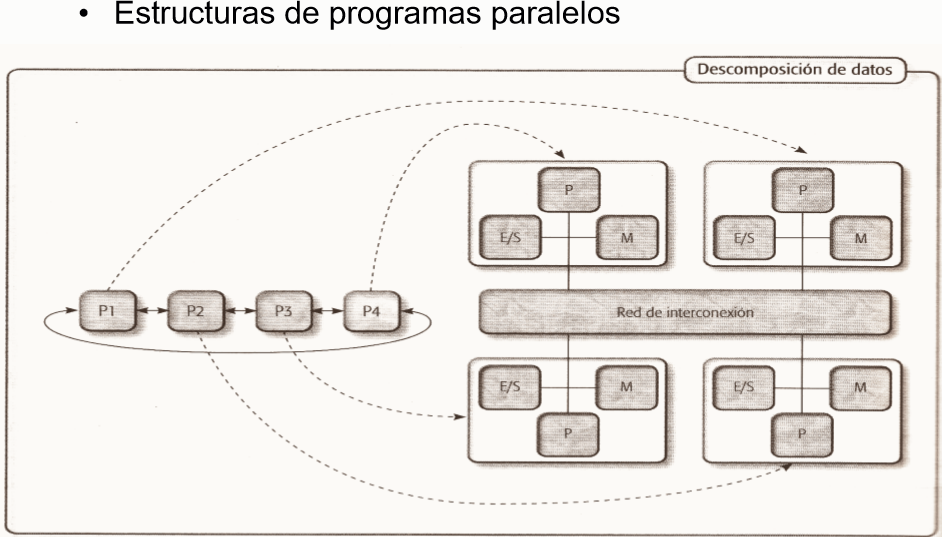
* **Tipo de estructura paralela del programa.**

Podemos tratar estos como ciertos patrones de estructuras que se repiten en los programas paralelos. Podemos agrupar estas estructuras en 4 tipos:

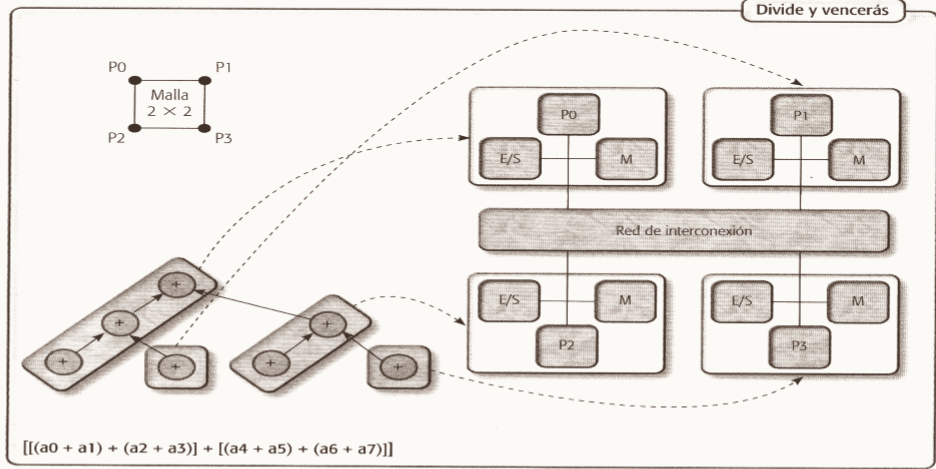
* + **Dueño-esclavo**: En él, el proceso dueño distribuye las tareas a los procesos esclavos y toma de ellos los resultados con los que el proceso dueño dará el resultado final. El reparto de estas tareas se puede producir de forma dinámica (en ejecución) o estática (realizada por el programador).



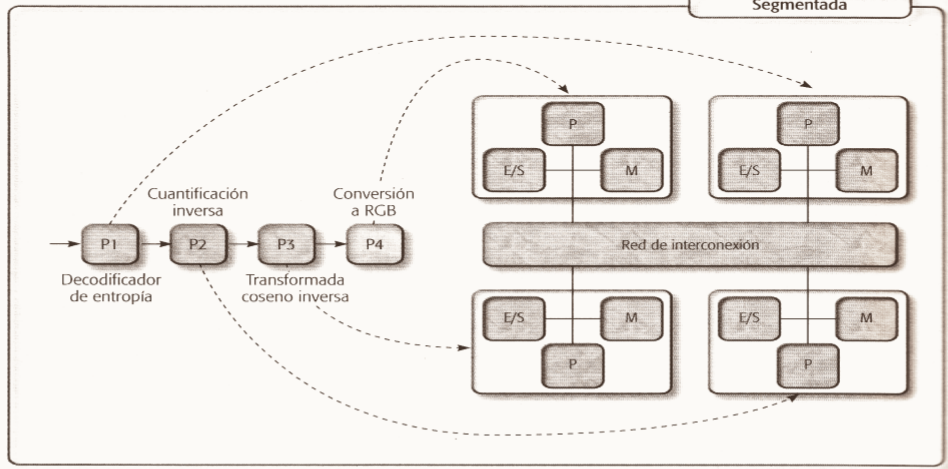
* + **Paralelismo de datos o descomposición de datos (Usada en estructuras de proceso SPMD)**: En él, la estructura de entrada y/o salida de datos se divide en partes que realizan operaciones similares paralelamente. Este tipo de paralelismo es usado en algoritmos que hacen uso de imágenes, al igual que realiza nuestra aplicación.



* + **Divide y vencerás**: Al igual que en algoritmo de mismo nombre, **divide y vencerás** trata de dividir un problema en varios subproblemas independientes cuyos resultados deben ser unificados para obtener el resultado final. Este tipo de estructura solo es posible si es posible realizar esta división. Suelen verse en **algoritmos** **secuenciales recursivos.**



* + **Segmentada o flujo de datos (Usada en estructuras MPMD):** En el cada proceso ejecuta distinto código, la estructura de procesos que se da en este tipo es la de un cauce segmentado (paralelismo a nivel de función o tarea).



Respondiendo a la pregunta de este apartado, nuestra aplicación usa un tipo de estructura **paralelismo de datos o descomposición de datos** que, es usada en modos de programación SPMD (un solo programa, múltiples datos) como lo es nuestro programa. Otro indicio es que este tipo de estructura es usada en algoritmos que hacen uso de imágenes, al igual que realiza nuestra aplicación. Su descripción es similar a la del tipo de **paralelismo a nivel de datos** el cual también posee nuestro programa.