# LA PROGRAMACIÓN PARALELA

La programación paralela es el uso de múltiples recursos computacionales para resolver un problema. Se distingue de la computación secuencial en que varias operaciones pueden ocurrir simultáneamente.

El paralelismo clásico, o puesto de otra manera, el clásico uso del paralelismo, es el de diseño de programas eficientes en el ámbito científico. La simulación de problemas científicos es un área de gran importancia, los cuales requieren de una gran capacidad de procesamiento y de espacio de memoria, debido a las complejas operaciones que se deben realizar.
En el sentido más simple, la programación paralela es el uso simultáneo de múltiples recursos computacionales para resolver un problema computacional:

- Un problema se divide en partes discretas que se pueden resolver simultáneamente
- Cada parte se descompone en una serie de instrucciones
- Las instrucciones de cada parte se ejecutan simultáneamente en diferentes procesadores
- Se emplea un mecanismo global de control/coordinación

**Ventajas**

- Resuelve problemas que no se podrían realizar en una sola CPU
- Resuelve problemas que no se pueden resolver en un tiempo razonable
- Permite ejecutar problemas de un orden y complejidad mayor
- Permite ejecutar código de manera más rápida (aceleración)
- Permite ejecutar en general más problemas
- Obtención de resultados en menos tiempo
- Permite la ejecución de varias instrucciones en simultáneo
- Permite dividir una tarea en partes independientes

**Desventajas**

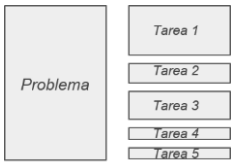
- Mayor consumo de energía
- Mayor dificultad a la hora de escribir programas
- Dificultad para lograr una buena sincronización y comunicación entre las tareas
- Retardos ocasionados por comunicación ente tareas
- Número de componentes usados es directamente proporcional a los fallos potenciales
- Condiciones de carrera
- Múltiples procesos se encuentran en condición de carrera si el resultado de los mismos depende del orden de su llegada
- Si los procesos que están en condición de carrera no son correctamente sincronizados, puede producirse una corrupción de datos

**Conceptos Clave**

Conceptos acerca de Tareas

*Tarea:*

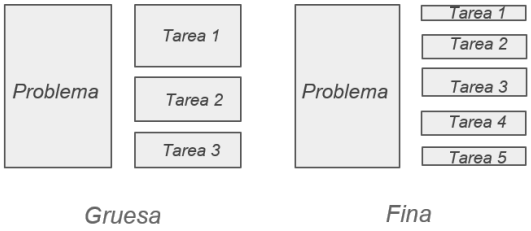
Un problema complejo se subdivide en una **cantidad discreta** de tareas que representan trabajo computacional. Una tarea esta compuesta de un **conjunto de instrucciones** que seran ejecutadas por un procesador.




*Granularidad:*

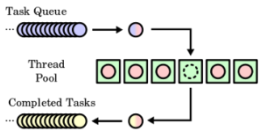
Se refiere al tamaño de cada tarea y a la independiencia de las demás tareas, se dividen en dos categorías.

- **Gruesa:** Cantidad relativamente grande de trabajo, alta independencia entre tareas y poca necesidad de sincronización.
- **Fina:** Cantidades pequeñas de trabajo, poca independencia entre tareas, y una alta demanda de sincronización.




*Scheduling:*

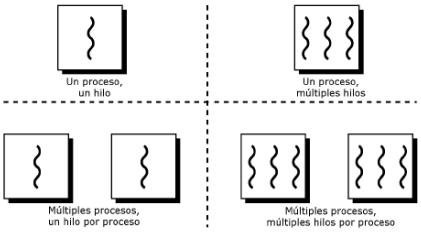
Scheduling es el proceso en el que **las tareas son asignadas a los procesos o hilos**, y se les da un orden de ejecución. Este puede ser especificado en el código, en tiempo de compilación o dinámicamente en tiempo de ejecución.




El proceso de scheduling debe tener en cuenta la dependencia entre tareas, ya que, aunque muchas pueden ser independientes, otras pueden requerir los datos producidos por otras tareas.

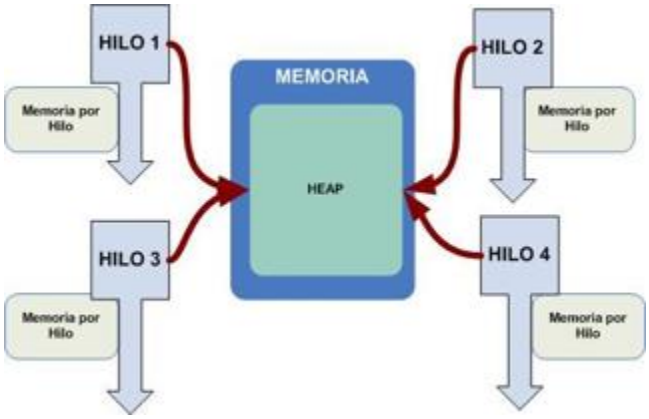
Conceptos acerca de Hilos

*Hilo:*



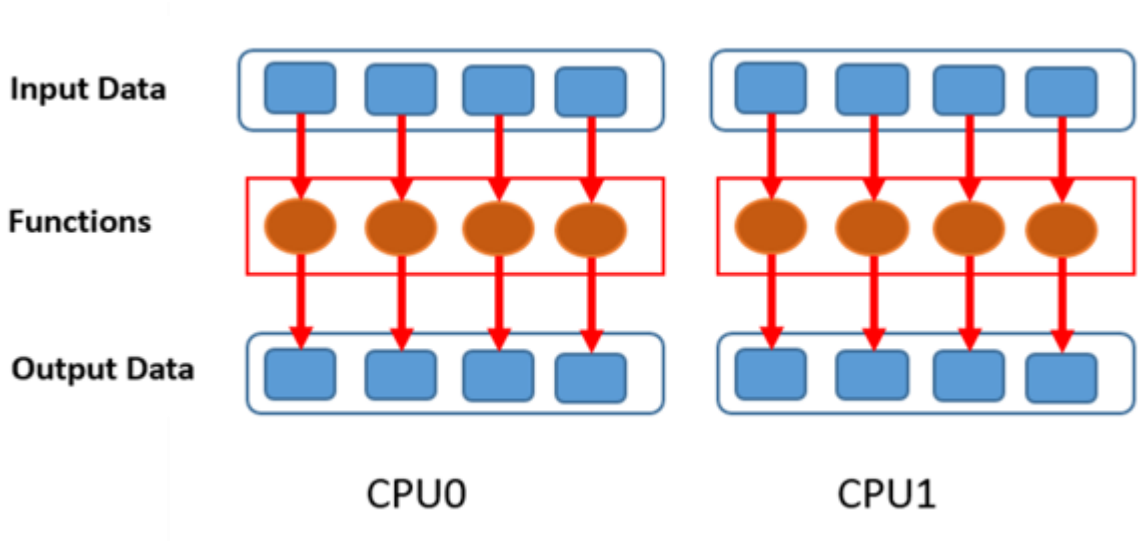

Un proceso pesado padre puede convertirse en varios **procesos livianos hijos**, ejecutados de manera concurrente. Cada uno de estos procesos livianos se conoce como hilo. Estos se comunican entre ellos a través de la memoria global.

*Sincronizacion:*




Los programas en paralelo necesitan la **coordinación de procesos e hilos, para que haya una ejecución correcta**. Los métodos de coordinación y sincronización en la programación paralela están fuertemente asociados a la manera en que los procesos o hilos intercambian información, y esto depende de cómo está organizada la memoria en el hardware.

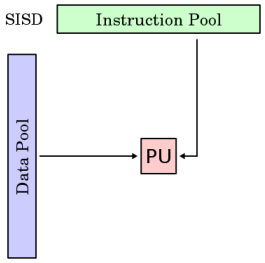
*Mapping:*



Mapping en el proceso de **asignación de procesos e hilos a unidades de procesamiento**, procesadores o núcleos. Usualmente el mapping se hace por el sistema en tiempo de ejecución, aunque en ocasiones puede ser influenciado por el programador.

**Taxonomia de Flynn**

Single Instruction, Single Data (SISD)



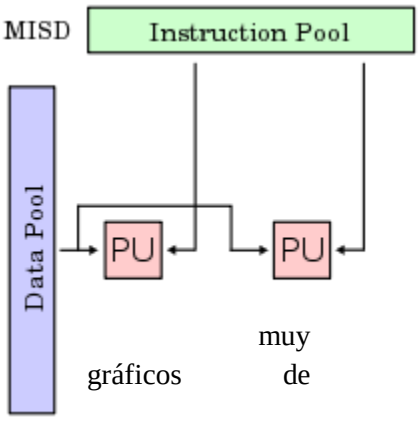

hay un elemento de procesamiento, que tiene acceso a un único programa y a un almacenamiento de datos. En cada paso, el elemento de procesamiento carga una instrucción y la información correspondiente y ejecuta esta instrucción. El resultado es guardado de vuelta en el almacenamiento de datos. Luego SISD es el computador secuencial convencional, de acuerdo al modelo de von Neumann.

Multiple Instruction, Single Data (MISD)

hay múltiples elementos de procesamiento, en el que cada cual tiene memoria privada del programa, pero se tiene acceso común a una memoria global de información. En cada paso, cada elemento de procesamiento de obtiene la misma información de la memoria y carga una instrucción de la memoria privada del programa. Luego, las instrucciones posiblemente diferentes de cada unidad, son ejecutadas en paralelo, usando la información (idéntica) recibida anteriormente. Este modelo es muy restrictivo y no se ha usado en ningún computador de tipo comercial.

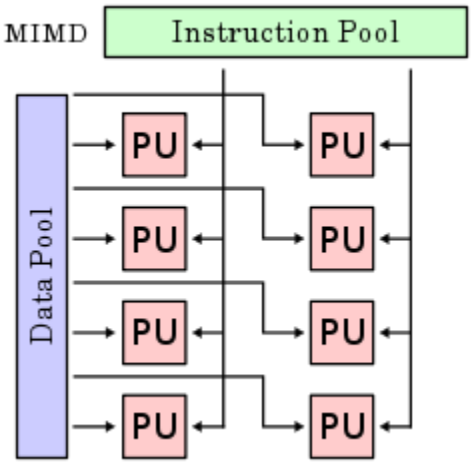
Single Instruction, Multiple Data (SIMD):

Hay múltiples elementos de procesamiento, en el que cada cual tiene acceso privado a la memoria de información (compartida o distribuida). Sin embargo, hay una sola memoria de programa, desde la cual una unidad de procesamiento especial obtiene y despacha instrucciones. En cada paso, cada unidad de procesamiento obtiene la misma instrucción y carga desde su memoria privada un elemento de información y ejecuta esta instrucción en dicho elemento. Entonces, la instrucción es síncronamente aplicada en paralelo por todos los elementos de proceso a diferentes elementos de información. Para aplicaciones con un grado significante de paralelismo de información, este acercamiento puede ser muy eficiente. Ejemplos pueden ser aplicaciones multimedia y algoritmos de gráficos de computadora.



Multiple Instruction, Multiple Data (MIMD):

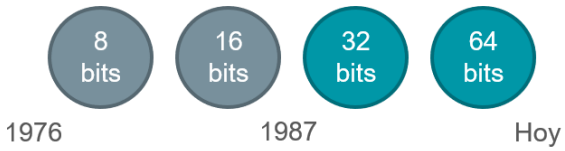
hay múltiples unidades de procesamiento, en la cual cada una tiene tanto instrucciones como información separada. Cada elemento ejecuta una instrucción distinta en un elemento de información distinto. Los elementos de proceso trabajan asíncronamente. Los clusters son ejemplo son ejemplos del modelo MIMD.




**Tipos de paralelismo**

Paralelismo a nivel de bit:

Se habla de paralelismo al nivel de bit, cuando se **aumenta el tamaño de la palabra del procesador** (tamaño de la cadena de bits a procesar). Este aumento reduce el número de instrucciones que tiene que ejecutar el procesador en variables cuyos tamaños sean mayores a la longitud de la cadena.
**Ejemplo:** En un procesador de 8-bits sumar dos números de 16bits tomaría dos instrucciones. En un procesador de 16-bits esa operación requiere solo una instrucción.
3



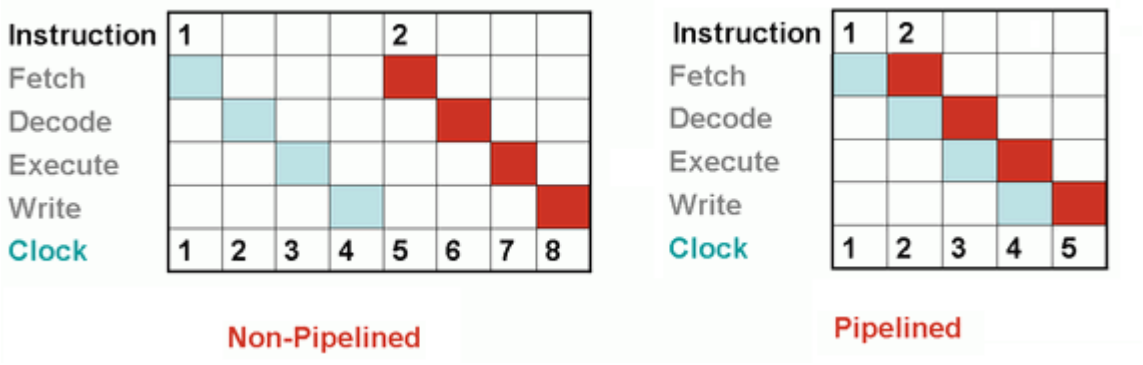

**Nota:** este método está "estancado" desde el establecimiento de las arquitecturas de 32 y 64 bits.

Paralelismo a nivel de instrucción

Este tipo de paralelismo consiste en **cambiar el orden de las intrucciones** de un programa y juntarlas en grupos para posteriormente ser ejecutados en paralelo **sin alterar el resultado final** del programa.

*Pipelining*

El pipelining proviene de la idea de que en una tubería no es necesario esperar a que todo el agua dentro salga, para que pueda entrar más. Los procesadores modernos tienen un 'pipeline' que separa las instrucciones en varias etapas, donde **cada etapa corresponde a una acción diferente** que necesita la salida de la anterior.
**Ejemplo:** Un pipeline de 5 etapas: fetch (buscar la instrucción), decode (decodificarla), execute (ejecutarla), write (escribir en memoria el resultado de la operación).




En el gráfico anterior se observa el procesamiento de dos instrucciones sin pipeline, tomando un tiempo de 8 ciclos, y con pipeline reduciendo este tiempo a solo 5 ciclos.

Paralelismo a nivel de datos

**Cada procesador** realiza **la misma tarea** sobre un subconjunto independiente de datos.
**Ej:** Dos granjeros se dividen el área de césped a podar.

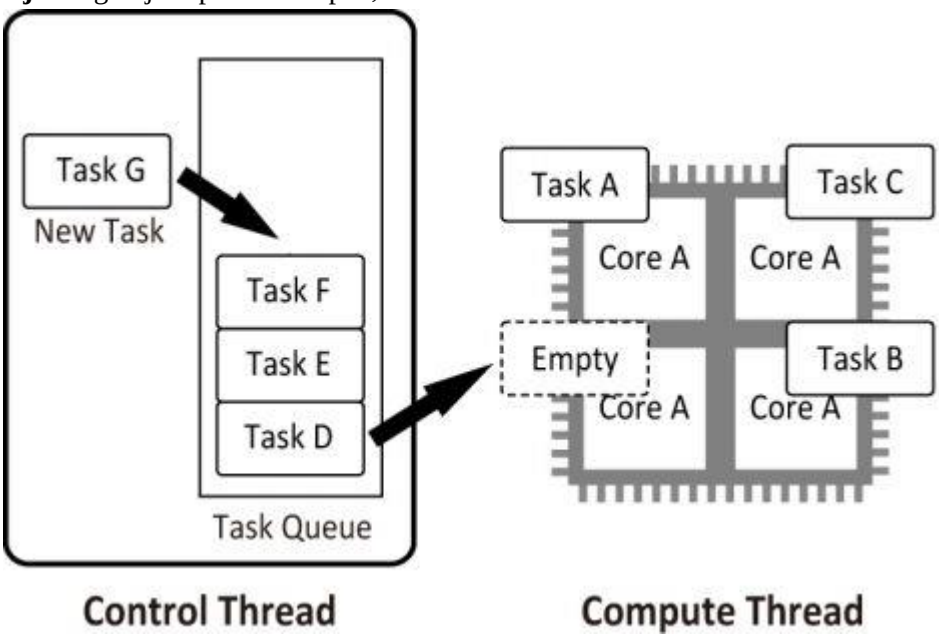
El caso clásico de paralelismo de datos, es el cálculo de pi por partes usando el método de monte carlo:

```
np = multiprocessing.cpu_count() # Nummber of CPUs

n = 10000000 # Nummber of points to use for the Pi estimatio

part_count = [n/np for i in range(np)] # list of points in e

pool = Pool(processes=np) # Create the worker pool

count = pool.map(monte_carlo_pi_part, part_count) # Get the

print "Esitmated value of Pi:: ", sum(count)/(n*1.0)*4
```

Ejemplo hecho en python

Paralelismo a nivel de tareas

**Cada hilo** realiza **una tarea distinta** e independiente de las demás.
**Ej:** Un granjero poda el césped, el otro cosecha.



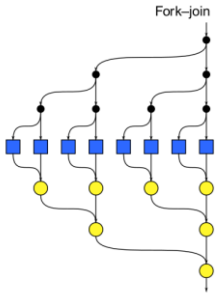

**Patrones de control paralelo**

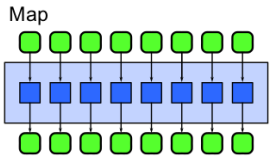
Los patrones se han establecido como buenas practicas a la hora de realizar ingeniería de software.

Los patrones de control en el caso de la programación paralela son maneras de combinar la distribución de los procesos y el acceso a los datos para la solucion de un problema.

Fork-Join

Fork-Join es un patrón que nos permite partir el flujo de un programa en múltiples flujos paralelos para reunirlos luego.

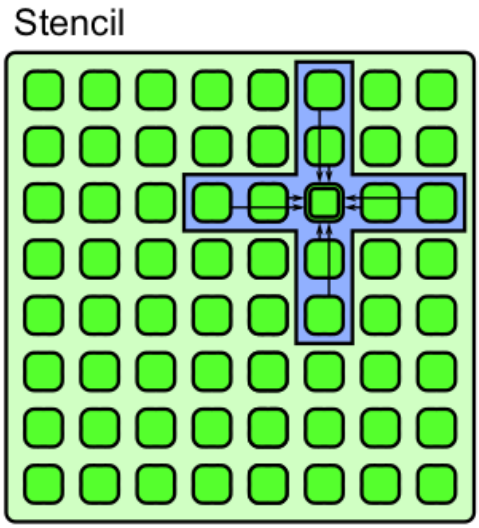


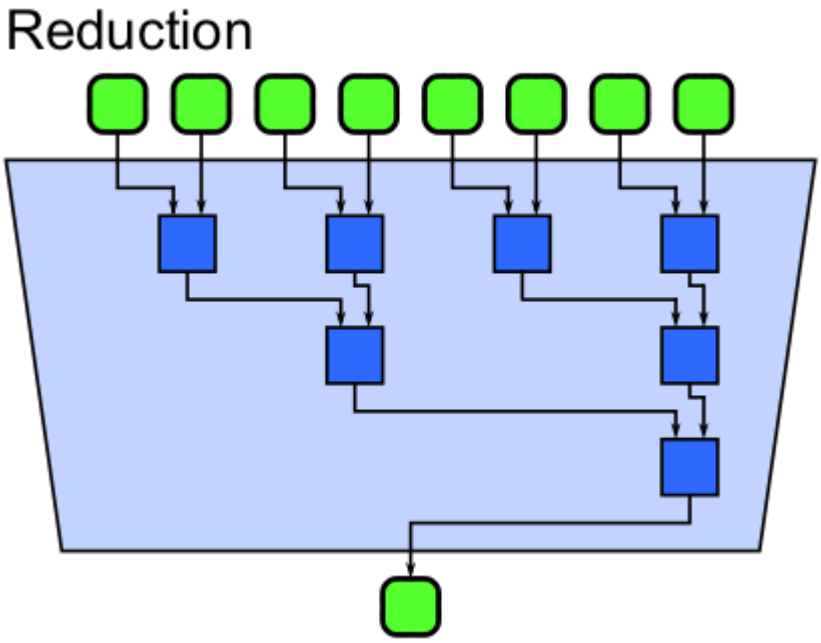




Map

Map es un patrón que replica una función sobre todos los elementos de un conjunto de entrada. La función que está siendo replicada se llama **función elemental**, dada que la misma se aplica a una coleccion real de datos.
Stencil

Stencil es un generalización del patron de Map, en el cual una **función elemental** tiene acceso no solo a un elemento del conjunto de entrada sino también a un conjunto de "vecinos"
Como la estructura de datos no es infinita se deben tener en cuenta el manejo de excepciones para los bordes de la misma.



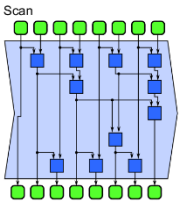

Reducción

Una reducción combina cada elemento de una colección en uno solo utilizado una función asociativa conocida como **función combinatoria**. Como es asociativa las tareas se pueden distribuir de muchas maneras y si la función resultara ser también conmutativa el número de posibilidides aumentaría aún más.
Scan

Scan realiza las reducciones de cada elemento perteneciente a una estructura. En otras palabras cada elemento de salida es la reduccion de un elemento de entrada. A traves de una **función sucesora** se avanza de un estado al otro haciendo *folds* en el proceso asociativo.

**Arquitecturas de memoria de computación paralela**

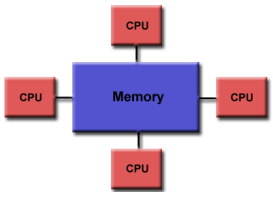
Memoria compartida




- Los procesos comparten un espacio de memoria común
- Escriben y leen de manera asíncrona
- No es necesario especificar cómo se comunican los datos entre las tareas
- Se usan semáforos o locks para controlar el acceso a la memoria compartida

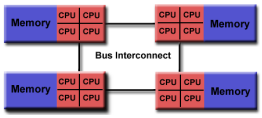
**Uniform Memory Access (UMA):**

- Lo más comúnmente representado hoy por las máquinas Symmetric Multiprocessor (SMP)
- Procesadores idénticos
- Igual acceso y tiempos de acceso a la memoria
- Si un procesador actualiza una ubicación en memoria compartida, todos los demás procesadores saben sobre la actualización, esto es llamado coherencia del caché




**Non-Uniform Memory Access (NUMA)**

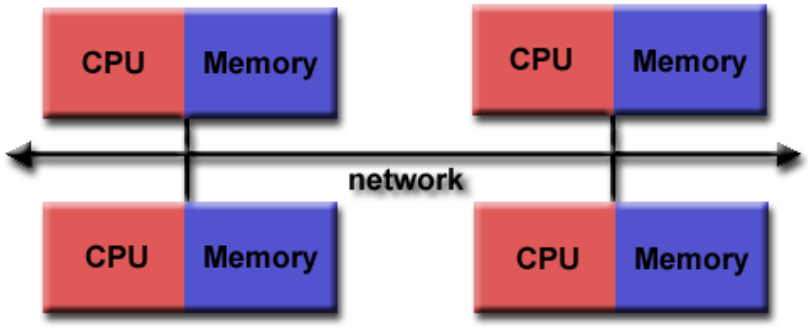
- Hecho mediante la vinculación física de dos o más SMP
- Un SMP puede acceder directamente a la memoria de otro SMP
- No todos los procesadores tienen igual tiempo de acceso a toda la memoria
- El acceso a la memoria es más lento
- Si se mantiene la coherencia del caché




Memoria distribuida

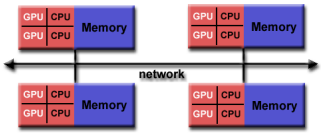
- También llamado modelo de paso de mensajes
- requieren una red de comunicación para conectar la memoria entre procesadores
- Las tareas intercambian datos por medio del paso y recepción de mensajes
- Los procesadores tienen su propia memoria local. Las direcciones de memoria en un procesador no se asignan a otro procesador, por lo que no hay concepto de espacio de direcciones global en todos los procesadores.
- Debido a que cada procesador tiene su propia memoria local, funciona independientemente. Los cambios que hace en su memoria local no tienen ningún efecto en la memoria de otros procesadores. Por lo tanto, el concepto de coherencia de caché no se aplica.

- Cuando un procesador necesita acceso a los datos de otro procesador, suele ser la tarea del programador definir explícitamente cómo y cuándo se comunican los datos. La sincronización entre tareas también es responsabilidad del programador.



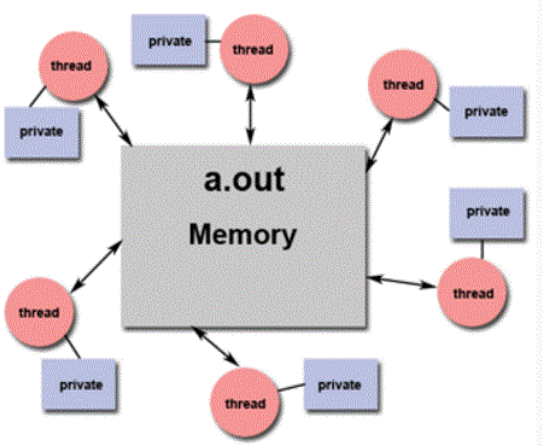

Hibrido memoria distribuida-comopartida

- Es la combinación entre memoria compartida y memoria distribuida, con sus ventajas en común.
- Su principal ventaja es su escalabilidad.
- Su principal desventaja es que la complejidad de programación aumenta.




Hilos

- Un proceso pesado puede convertirse en varios procesos livianos ejecutados de manera concurrente.
- Se pueden describir como una subrutina dentro del programa principal.
- Se comunican entre ellos a través de la memoria global.




Datos en paralelo

- También conocido como PGAS (Partitioned Global Address Space)
- Una serie de tareas trabajan de manera colectiva en la misma estructura de datos
- Las tareas realizan la misma operación, pero cada una en su partición pero cada tarea trabaja en una partición diferente de ésta