PARALELISMO Y ALGORITMOS PARALELOS

PROGRAMACIÓN PARALELA

CÓMPUTO EN LA NUBE

MAESTRÍA MNA

Dr. Eduardo Cendejas

CONCURRENCIA

- En ciencias de la computación, **concurrencia** se refiere a la habilidad de distintas partes de un programa, algoritmo, o problema de ser ejecutado en desorden o en orden parcial, sin afectar el resultado final.
- Los cálculos (operaciones) pueden ser ejecutados en múltiples procesadores, o ejecutados en procesadores separados físicamente o virtualmente en distintos hilos de ejecución

PARALELISMO

- El paralelismo es una técnica de computación basada en principios aparentemente simples:
- "Divida un gran problema en varios pequeños y resuélvalos al mismo tiempo"

PARALELISMO

- Esto permite ejecutar más instrucciones en menos tiempo. Pero cuando se pone en práctica, se trata de un tema muy complejo y varios grupos científicos de todo el mundo lo están investigando.
- La computación 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.

COMPUTACIÓN PARALELA

En el sentido más simple, la computación 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

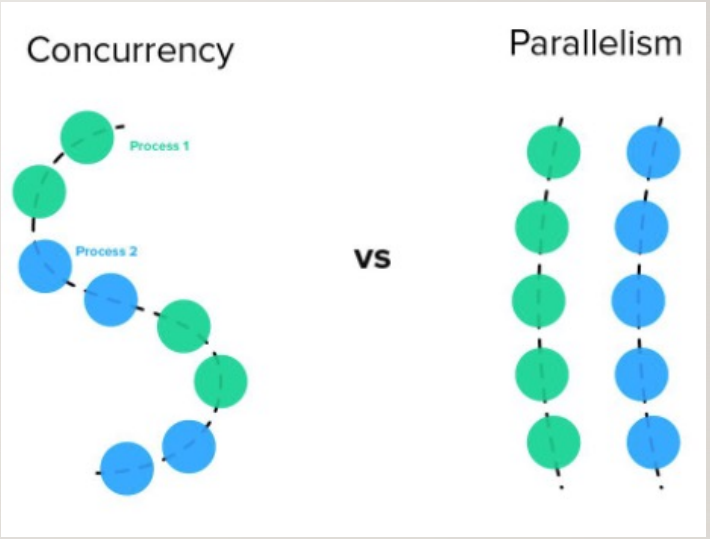
- Resuelve problemas que no se podrían realizar en una sola CPU
- Resuelve problemas que no se pueden resolver en un tiempo razonable
- Permite ejecutar problemas de un orden y complejidad mayor
- Permite ejecutar código de manera más rápida (aceleración)
- Permite ejecutar en general más problemas
-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

DESVENTAJAS

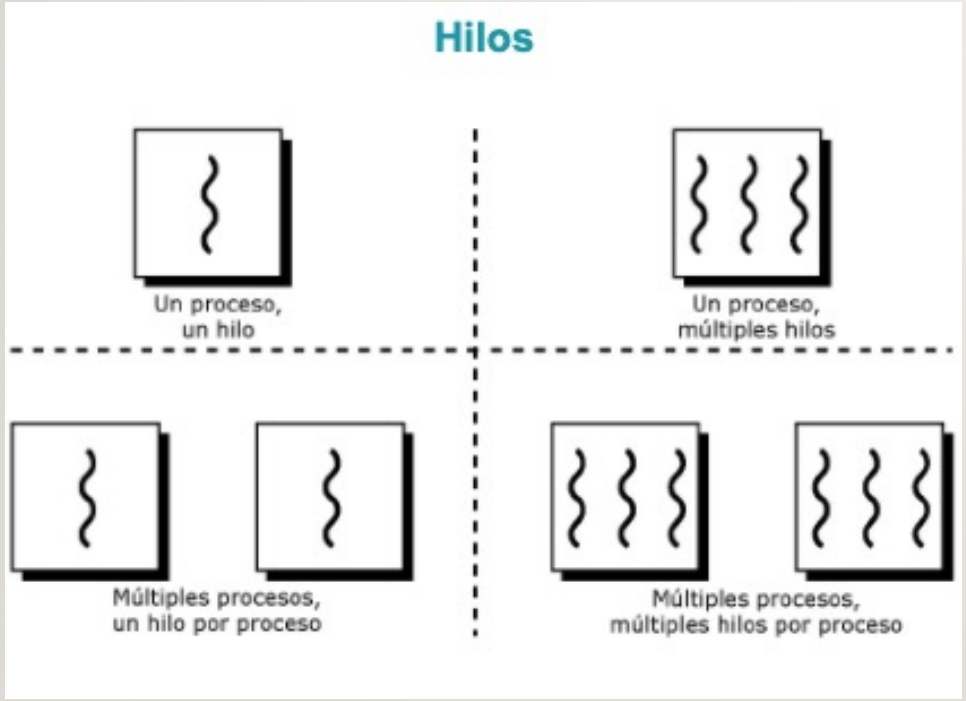
- Mayor consumo de energía
- Mayor dificultad a la hora de escribir programas
- Dificultad para lograr una buena sincronización y comunicación entre las tareas
- Retardos ocasionados por comunicación ente tareas
- Número de componentes usados es directamente proporcional a los fallos potenciales
- Altos costos por producción y mantenimiento
- 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

CONCURRENCIA VS PARALELISMO VARIOS PROCESOS

- **Concurrencia**
- Capacidad de operar actividades al mismo tiempo. Es decir se pueden tener varios procesos corriendo cada uno en un procesador o puede haber varios proceso que corran solo en un procesador
- **Paralelismo**
- Son muchas actividades teniendo lugar al mismo tiempo, "la cualidad o el estado de ser paralelo". El hecho de ser paralelo implica que solo se pueden tener varios procesos corriendo cada uno en un procesador.



HILOS



- Un proceso pesado padre puede convertirse en varios **procesos livianos hijos**, ejecutados de manera concurrente/paralela. Cada uno de estos procesos livianos se conoce como **hilo**. Estos se comunican entre ellos a través de la memoria global.
- Una thread es un único flujo de control dentro de un programa. Algunas veces es llamado contexto de ejecución porque cada thread debe tener sus propios recursos, como el program counter y el stack de ejecución, como el contexto de ejecución.

PUNTO CRÍTICO
OVERCLOCKING INFINITO

- El overclokcing tiene un límite a pesar de que existiera una refrigeración perpetúa y adecuada del procesador. Esto es debido a las corrientes parásitas que impiden una velocidad teóricamente infinita a la cual los circuitos pueden cambiar entre estados, o de hecho sus transistores.

PUNTO CRÍTICO
AUTOMATIZACIÓN DEL PARALELISMO

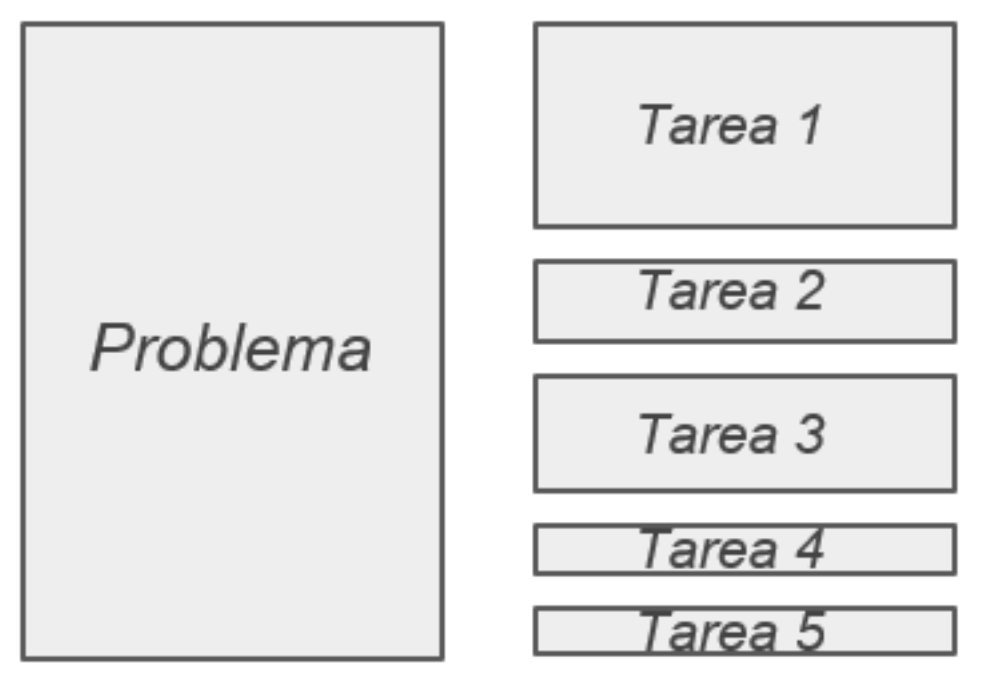
- Se dice en este paradigma que el éxito es inversamente proporcional al número de Cores precisamente porque existen complejidades en el corazón del paralelismo que implican cosas que todavía no se pueden predecir ni con inteligencia artificial, en este mismo artículo de hecho se menciona cuáles son las posibles estrategias para atacar un problema de forma paralela, esto da cuenta de que existe una forma prácticamente determinada de abordarlos pero no se automatizarlos, a pesar de que sí existan algunas partes que son automatizables en el proceso.

PUNTO CRÍTICO
SOLUCIÓN EN EL HARDWARE

- Un diseño adecuado del hardware permitiría que la paralelización siempre estuviera presente con respecto a los procesadores que se están usando de tal modo que alguno los problemas que son inherentes al paradigma pudieran evitarse.
- Esto ha resultado imposible hasta la fecha, de hecho, solo diseñar solamente algo tan efectivo y tradicional como se ha hecho en programación secuencial es algo que no existe hasta ahora.

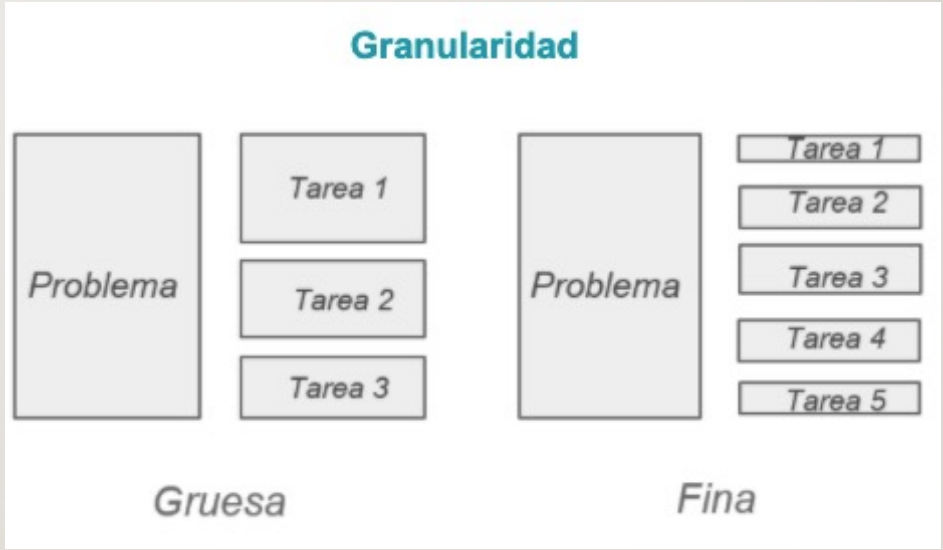
CONCEPTOS CLAVE TAREITAS

- Son **secciones lógicamente discretas** de trabajo computacional. Una tarea está compuesta de un **conjunto de instrucciones** que seran ejecutadas por un procesador.



CONCEPTOS CLAVE GRANULARIDAD

- Se refiere al tamaño de cada tarea y a la independencia 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.



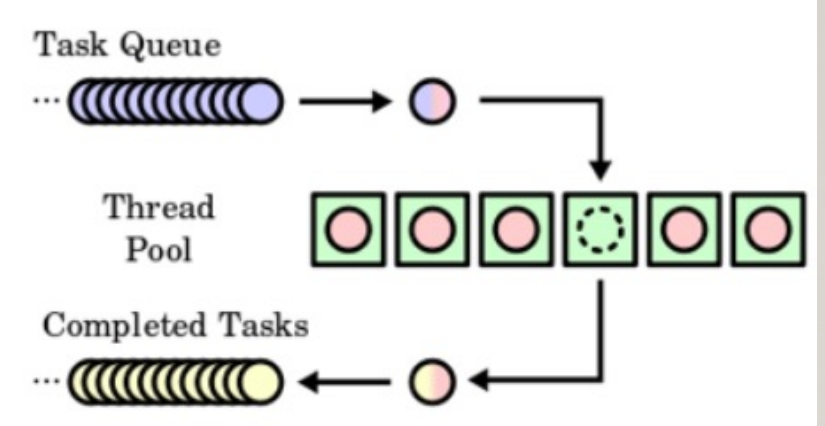
CONCEPTOS CLAVE SCHEDULING

- 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 CLAVE
SCHEDULING – CRITERIOS DE PLANIFICACIÓN

Orientados al usuario:

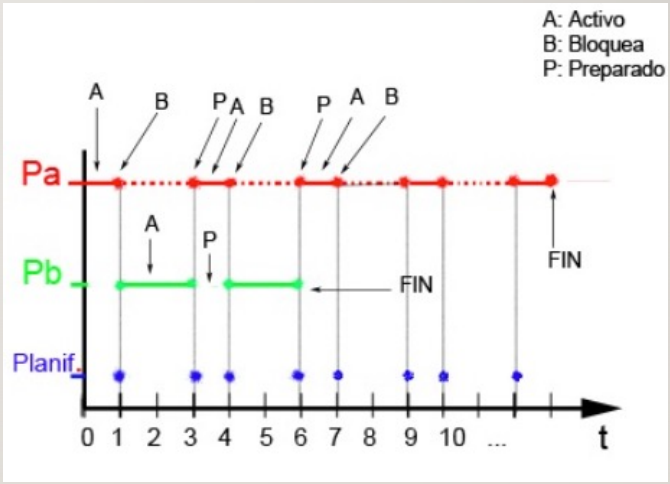
- Tiempo de vuelta: Intervalo de tiempo que transcurre entre la solicitud de ejecución de un proceso y su terminación.
- Tiempo de respuesta: Tiempo transcurrido desde que se hace una solicitud y se empieza el proceso.
- Plazos: Ocurre cuando se pueden dar plazos de ejecución de procesos, Obligando al planificador a subordinar otros procesos.
- Previsibilidad: Un proceso deberia ejecutarse en el mismo tiempo siempre (sin importar la carga que se tenga en el sistema).



CONCEPTOS CLAVE
SCHEDULING – CRITERIOS DE PLANIFICACIÓN

Orientados al sistema:

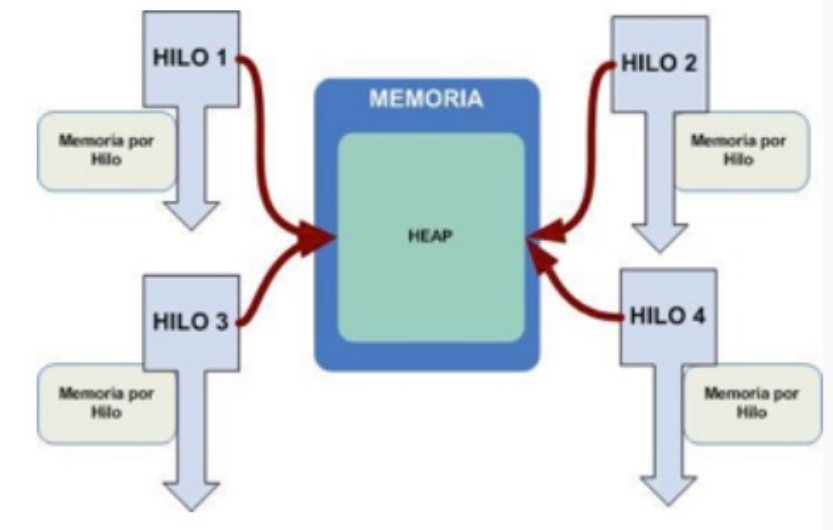
- Tasa de salida: Consiste en el numero de procesos ejecutados por unidad de tiempo.
- Utilización del proceso: Cantidad de tiempo que el procesador permanece ocupado.
- Equidad: Los procesos deben ser tratados todos de igual forma para evitar la inanición de procesos.
- Prioridades: Se debe poder tener una politica de prioridades para poder favorecer a ciertos procesos que se consideren importantes.
- Balanceo de recursos: Se debe balancear la carga del sistema de modo que todos sus componentes se mantengan ocupados.



CONCEPTOS CLAVE
SCHEDULING – ALGORITMOS DE PLANIFICACIÓN

- FCFS (first-come-first-served)
- Round Robin
- "Primero el proceso más corto"
- "Primero el de menor tiempo restante"
- "Primero el de mayor tasa de respuesta"
- realimentación

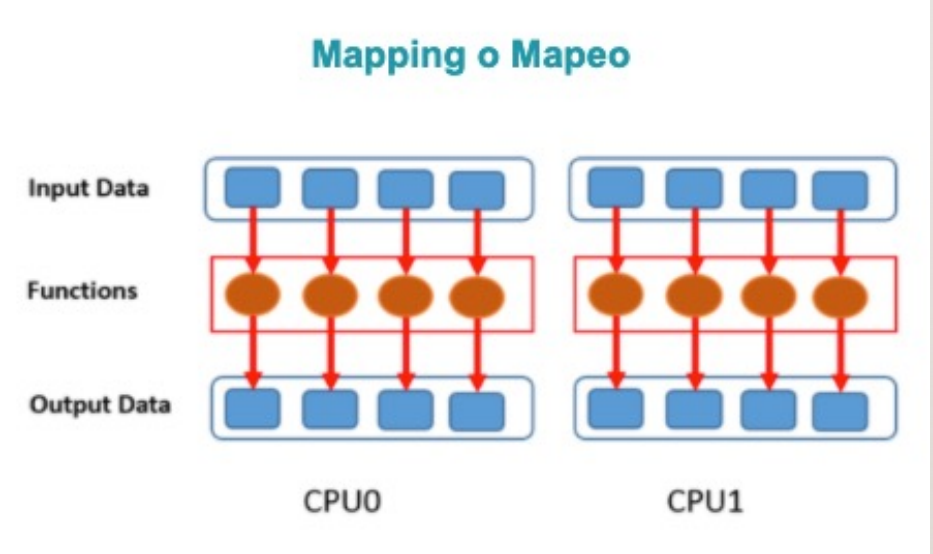
CONCEPTOS CLAVE SINCRONIZACIÓN



- 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.

CONCEPTOS CLAVE
MAPEO

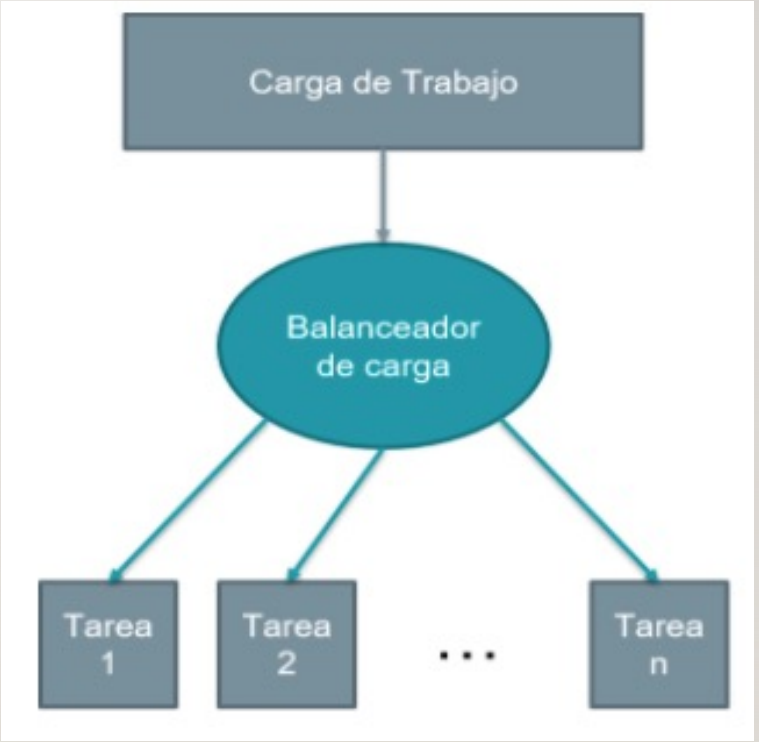
- Mapping en el proceso de **asignación de procesos e hilos a unidades de procesamiento**, procesadores o núcleos. Usualmente el mapping se hace por el sistema en tiempo de ejecución, aunque en ocasiones puede ser influenciado por el programador.



CONCEPTOS CLAVE
BALANCEO DE CARGA

Se refiere a la práctica de distribuir **cantidades equitativas de trabajo** entre las tareas, de modo que todas las tareas se mantengan ocupadas todo el tiempo. Se puede considerar una minimización del tiempo de inactividad de la tarea. Algunos puntos a tener en cuenta con el balanceo de carga son:

- Asignar el trabajo que recibe cada tarea equitativamente
- Puede ser un factor significativo en el desempeño del programa
- A menudo requiere "serialización" de segmentos del programa.



CONCEPTOS CLAVE
SPEEDUP

- Es un proceso para **aumentar el rendimiento** entre dos sistemas procesando el mismo problema. Es la mejora en la velocidad de ejecución de una tarea ejecutada en **dos arquitecturas similares** con diferentes recursos. El SpeedUp representa la ganancia que se obtiene en la versión paralela del programa respecto a la versión secuencial del mismo.

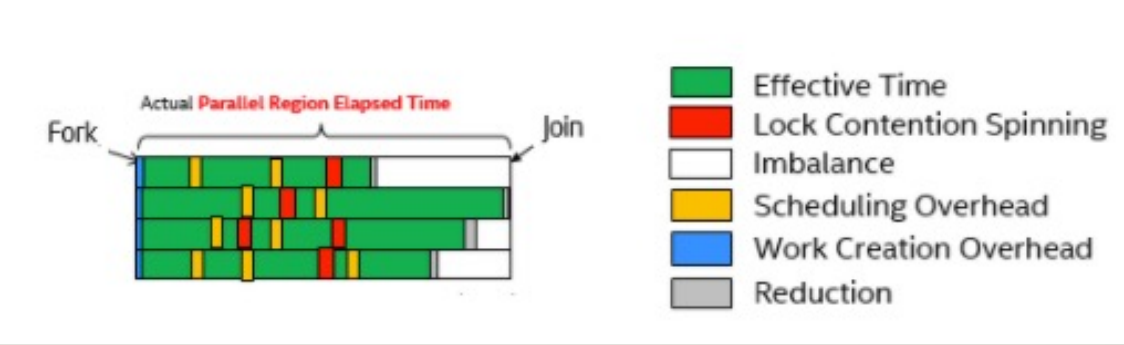
$$SPEED\_UP = \frac{T*(n)}{T_P(N)}$$

$T*(n)$= El mejor tiempo de ejecutarlo secuencialmente

$T_P(N)$= El tiempo que toma la ejecución paralela.

CONCEPTOS CLAVE OVERHEAD

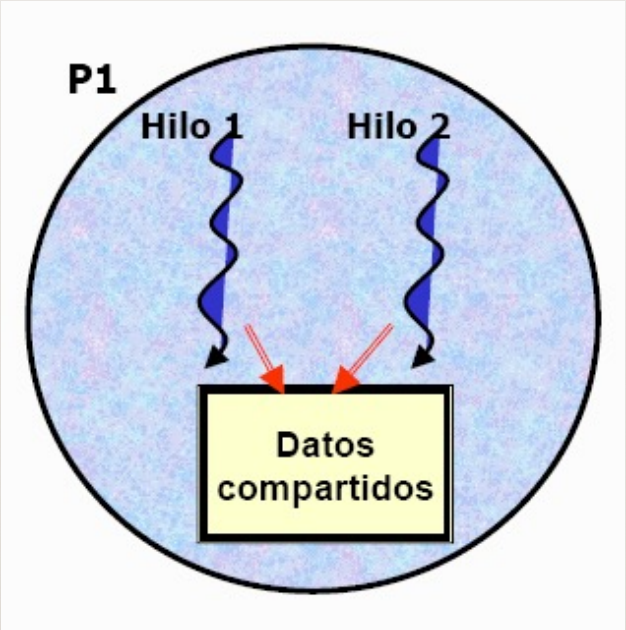
Es la cantidad de **tiempo requerido para coordinar tareas paralelas**, en lugar de hacer un trabajo útil. Incluye factores como:



- Tiempo de inicio de la tarea
- Sincronización
- Comunicaciones de datos
- Sobrecarga de software impuesta por lenguajes paralelos, bibliotecas, sistema operativo, etc.
- Tiempo de terminación de la tarea

CONCEPTOS CLAVE
SECCIÓN CRÍTICA

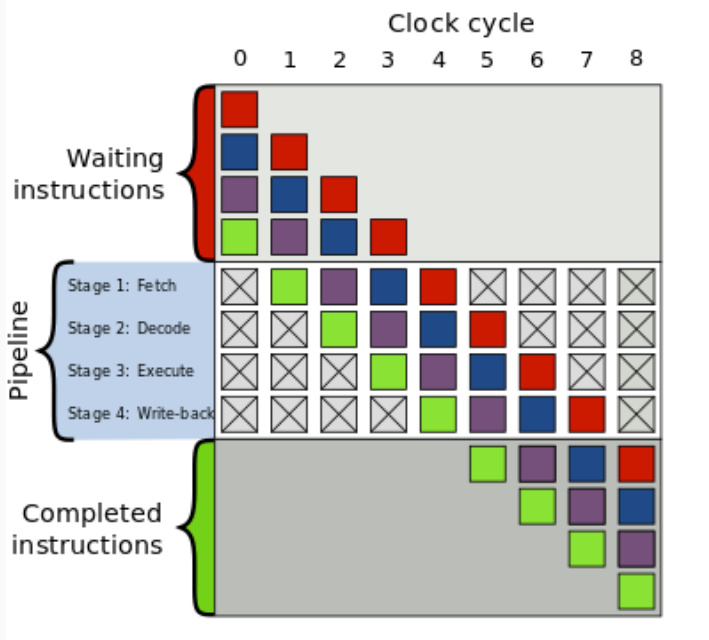
- Es un segmento de código que manipula un recurso y se debe ejecutar de forma **atómica**, es decir, que no debe de ser accedido por más de un hilo en ejecución.



CONCEPTOS CLAVE
CONDICIÓN DE CARRERA

- Es la representación del **acceso de dos o más procesos a un recurso compartido sin control**, este acceso depende del orden de llegada de los procesos

CONCEPTOS CLAVE PIPELINING



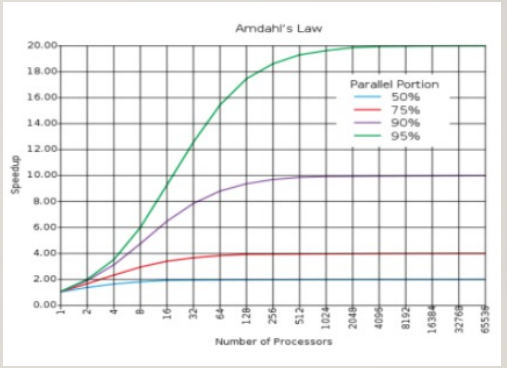
- Es la **ruptura o segmentación** de una tarea en pasos realizados por diferentes unidades de procesador. El pipelining proviene de la idea de que en una tubería no es necesario esperar a que todo el agua dentro salga, para que pueda entrar más. Los procesadores modernos tienen un **pipeline** que separa las instrucciones en **varias etapas**, donde cada etapa corresponde a una acción diferente que necesita la salida de la anterior.

CONCEPTOS CLAVE
COOPERACIÓN

- Es esa característica de los procesos que los orienta **trabajar conjuntamente**

LEYES IMPORTANTES
LEY DE AMDAHL

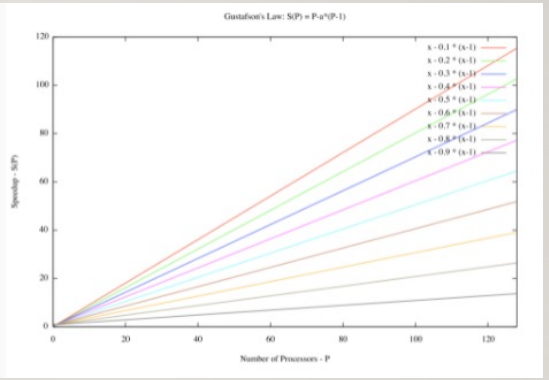
- Ley de la computación formulada por Gene Amdahl y dicta que la mejora obtenida en el rendimiento de un sistema debido a la alteración de uno de sus componentes está limitada por la fracción de tiempo que se utiliza dicho componente. Sea f el porcentaje paralelizado del programa expresado en decimal, la ley de Amdahl dice que llega un punto en el cual sin importar que el numero de procesadores sea muy alto , el speedup se va a comportar de manera lineal; esto de acuerdo al porcentaje que esté paralelizado el código. El speedup de un programa con un fragmento paralelizado se calcula con:
- $$S_p(n) = \frac{1}{f + \frac{1-f}{p}} \leq \frac{1}{f}$$
- Esta ley ayuda a definir si introducir una mejora en el sistema vale o no la pena



LEYES IMPORTANTES
LEY DE GUSTAFSON

- Ley de la computación formulada por John L Gustafson en 1988, también llamada ley de Gustafson-Barsis, es una ley en ciencia de la computación que establece que cualquier problema suficientemente grande puede ser eficientemente paralelizado, ofrece un nuevo punto de vista y así una visión positiva de las ventajas del procesamiento paralelo.

$$S(P) = P - \alpha \cdot (P - 1)$$



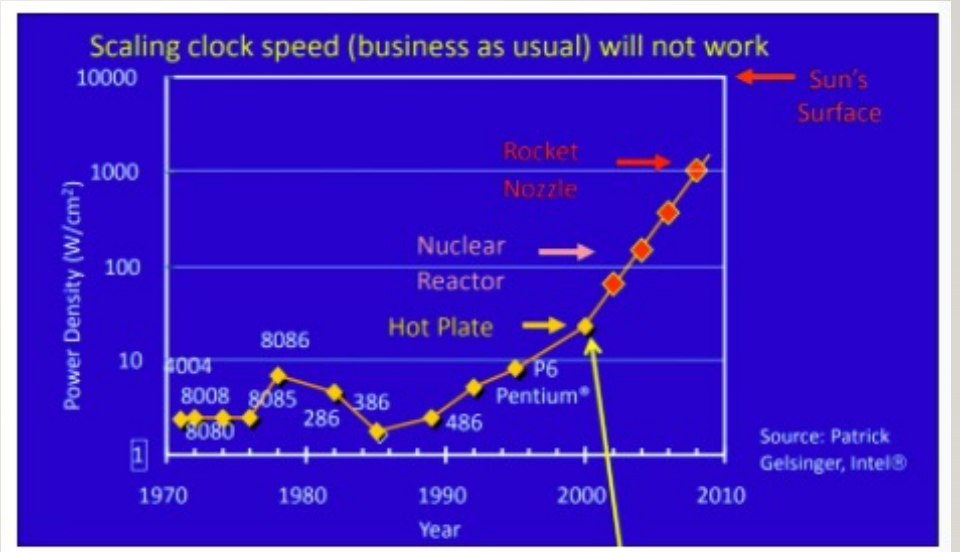
- En su fórmula, P es el número de procesadores, S es el speedup (aceleración), y α la parte no paralelizable del proceso.

LEYES IMPORTANTES
LEY DE MOORE

- Ley propuesta por Gordon E. Moore en 1965, inicialmente decía resumidamente que el número de transistores en un chip determinado se doblaría cada año aunque unos años más tarde, en 1975, modificó su propia ley para aumentar esta cadencia a cada dos años. Esto quiere decir un aumento del rendimiento en los procesadores del alrededor del 50%, esto se traduce en escalar la velocidad de reloj de los procesadores, pero esta ley no es fidedigna desde el 2002 dónde solo ha habido un 20%, lo cual sigue siendo un aumento considerable, sin embargo, para que esto sea posible es necesario reducir el tamaño de los transistores para que todos los avances en computación que se han logrado hasta el día y las necesidades de procesamiento en crecimiento exponencial puedan satisfacerse totalmente.

LEYES IMPORTANTES
LEY DE MOORE

- Veamos en la gráfica adjunta que el problema principal es que la ley no puede continuar indeterminadamente porque esto implica un crecimiento exponencial, el cuál es imposible de mantener por espacio, pero principalmente por el punto de la temperatura mismo, se puede ver que si esto fuese cierto llegaría muy pronto el año en que un solo procesador alcanzara la temperatura de la superficie del sol.



La unidad de medida de transistores en superficie es MTr/mm^2 o millones de transistores por milímetro cuadrado

ARQUITECTURA DE VON NEUMAN

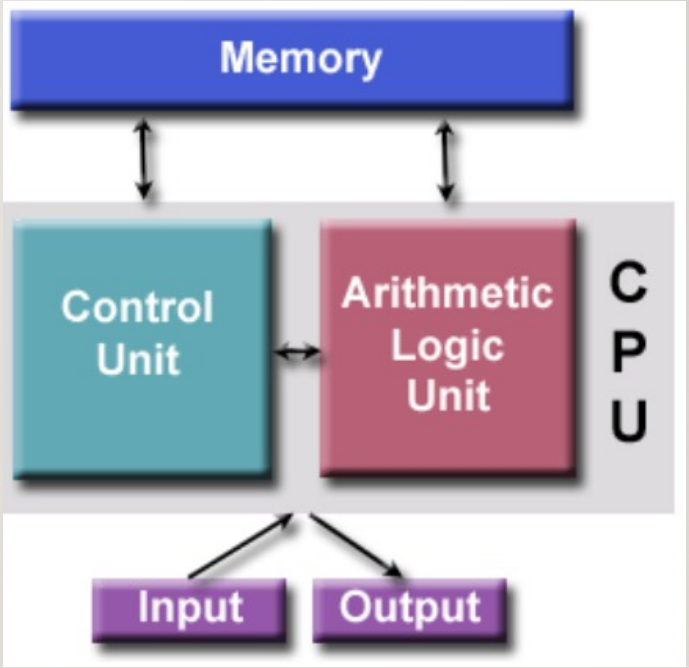
Se caracterizaba por guardar las instrucciones de los procesos y los datos en una memoria electronica, a diferencia de como se modelaban los computadores de la epoca a través de una conexion de cables

Componentes principales

- Memoria
- Unidad de control
- Unidad Aritmetica Logica
- Entradas/Salidas

Memoria de acceso aleatorio

- En la memoria de acceso aleatorio se almacenaban los datos y los programas que estaban siendo ejecutados
- Las instrucciones del programa son datos codificados que le dicen al computador que es lo que tiene que hacer
- Los datos son simplemente informacion que sera usada por el programa

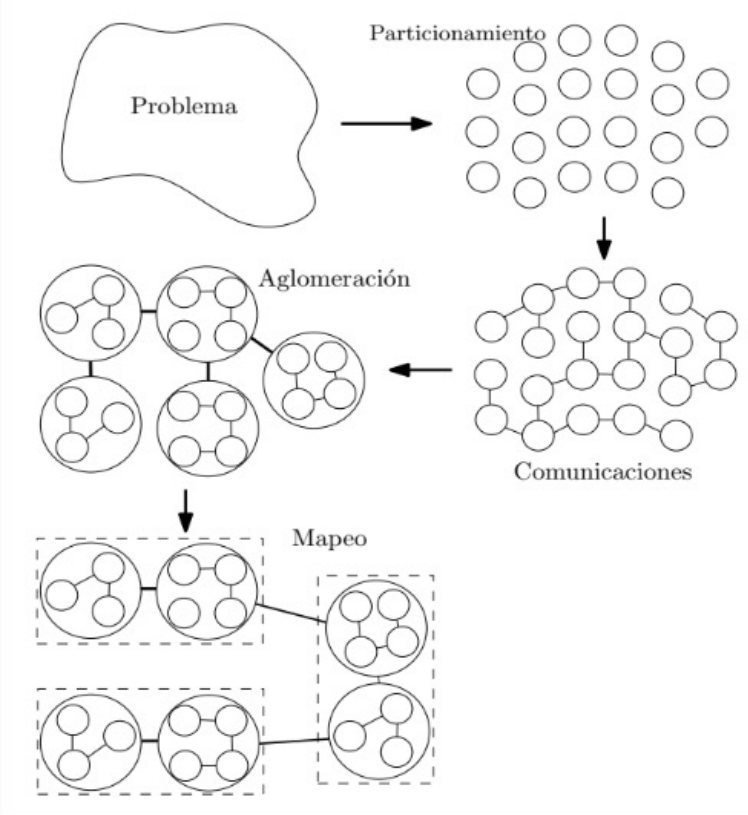


METODOLOGÍAS DE DISEÑO

Cuando se diseña un algoritmo paralelo es necesario tener en cuenta:

1. Los tiempos de las comunicaciones.
2. Maximizar el procesamiento en cada nodo o unidad de procesamiento.
3. Los costes de implementar el algoritmo.
4. Tiempos de planificación (scheduler).

METODOLOGÍA FOSTER



Consiste en cuatro etapas

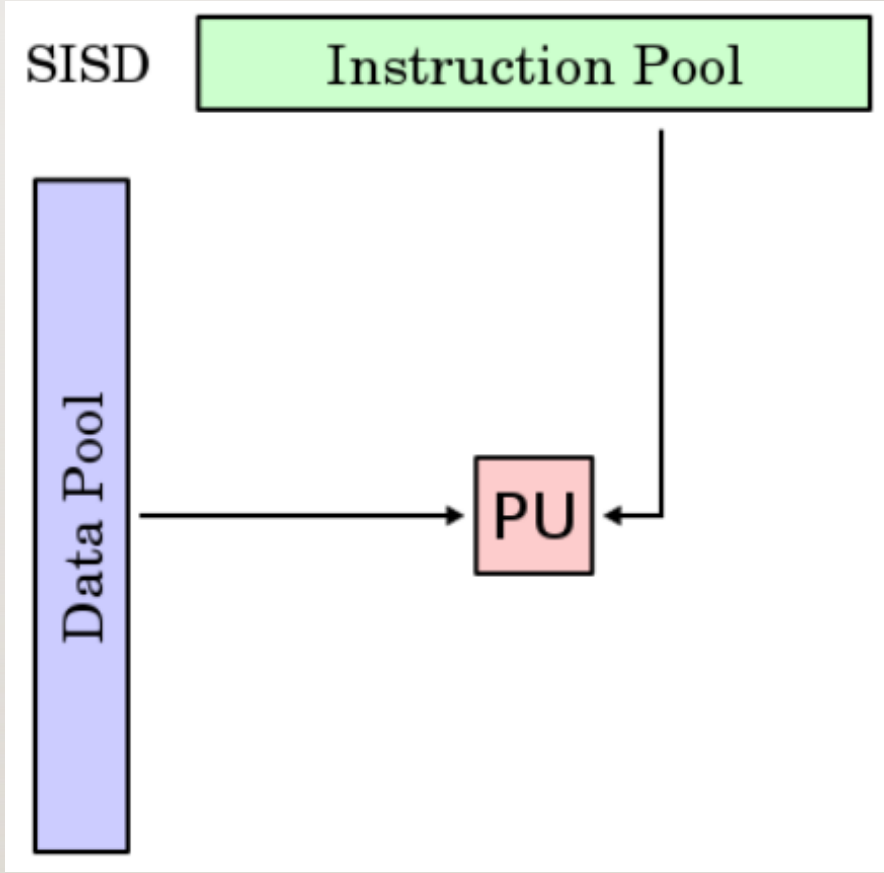
- **Particionamiento:** En el dominio de los datos o de funciones.
- **Comunicaciones:** Se hace por medio de distintos medios o paradigmas tales como la memoria o paso de mensajes
- **Aglomeración:** Las tareas o datos son agrupados teniendo en cuenta posibles dependencias
- **Mapeo:** Los grupos son asignados a una unidad de procesamiento

TAXONOMÍA DE FLYNN

- La **taxonomía de Flynn** es una clasificación para las computadoras con arquitectura paralela, propuesta por el profesor emérito de la Universidad de Stanford Michael J. Flynn, la cual clasifica a las mismas atendiendo a la cantidad de instrucciones y flujo de datos concurrentes en un instante de procesamiento.
- Clasificaciones:
  - SISD
  - MISD
  - SIMD
  - MIMD

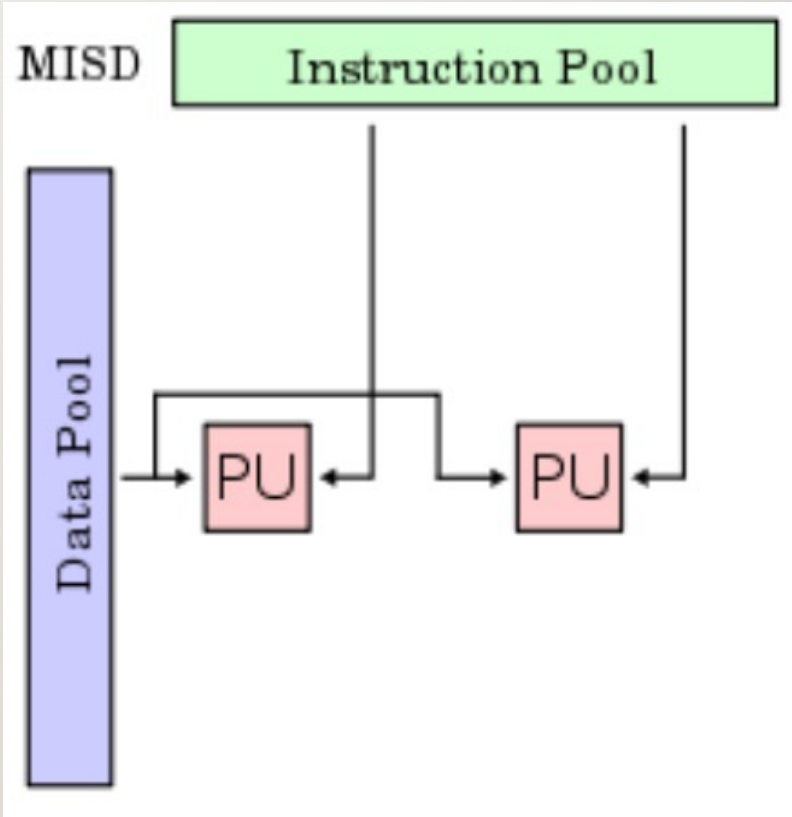
TAXONOMÍA DE FLYNN
SINGLE INSTRUCTION, SINGLE DATA (SISD)

- 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.



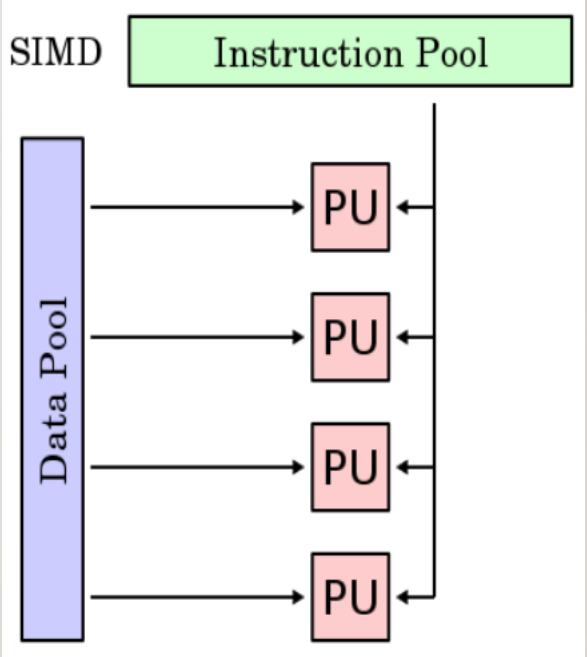
TAXONOMÍA DE FLYNN
MULTIPLE INSTRUCTION, SINGLE DATA (MISD)

- 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.



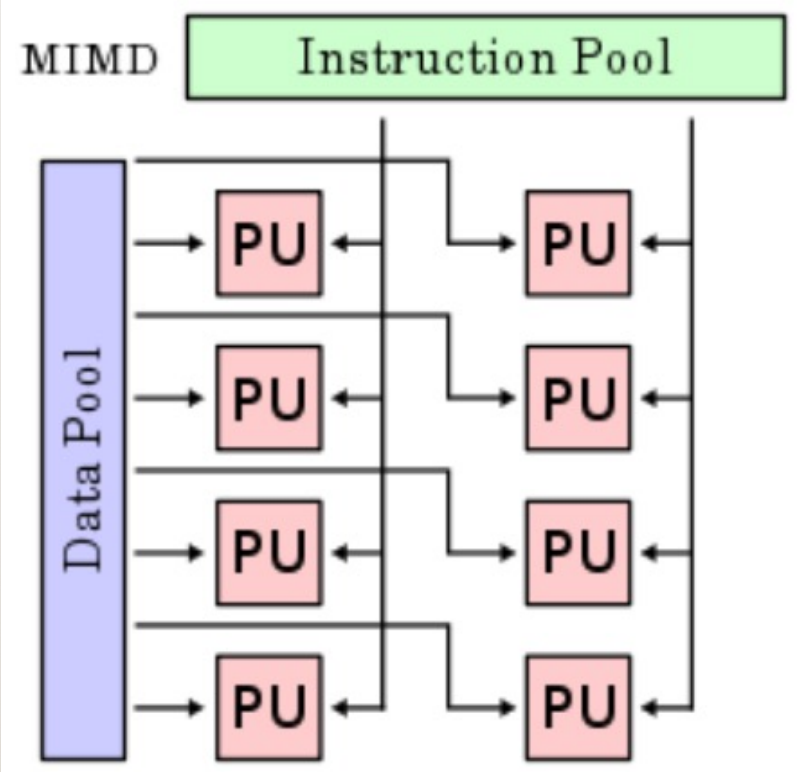
TAXONOMÍA DE FLYNN
SINGLE INSTRUCTION, MULTIPLE DATA (SIMD)

- 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.



TAXONOMÍA DE FLYNN
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.



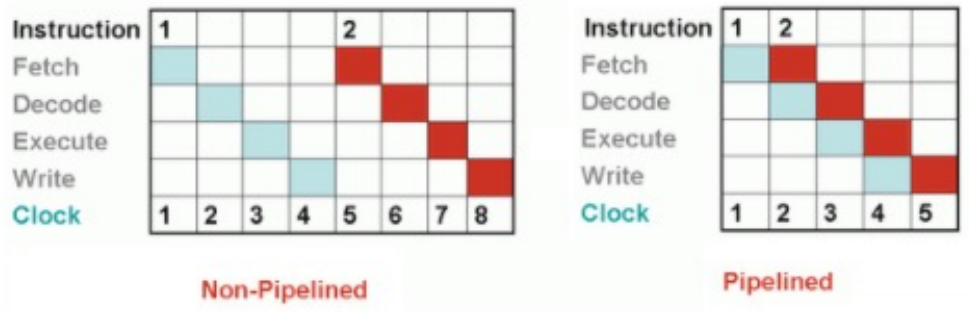
TIPOS DE PARALELISMO
PARALELISMO A NIVEL DE BIT

- 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.

TIPOS DE PARALELISMO
PARALELISMO A NIVEL DE INSTRUCCIÓN

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

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.

TIPOS DE PARALELISMO
PARALELISMO A NIVEL DE DATOS

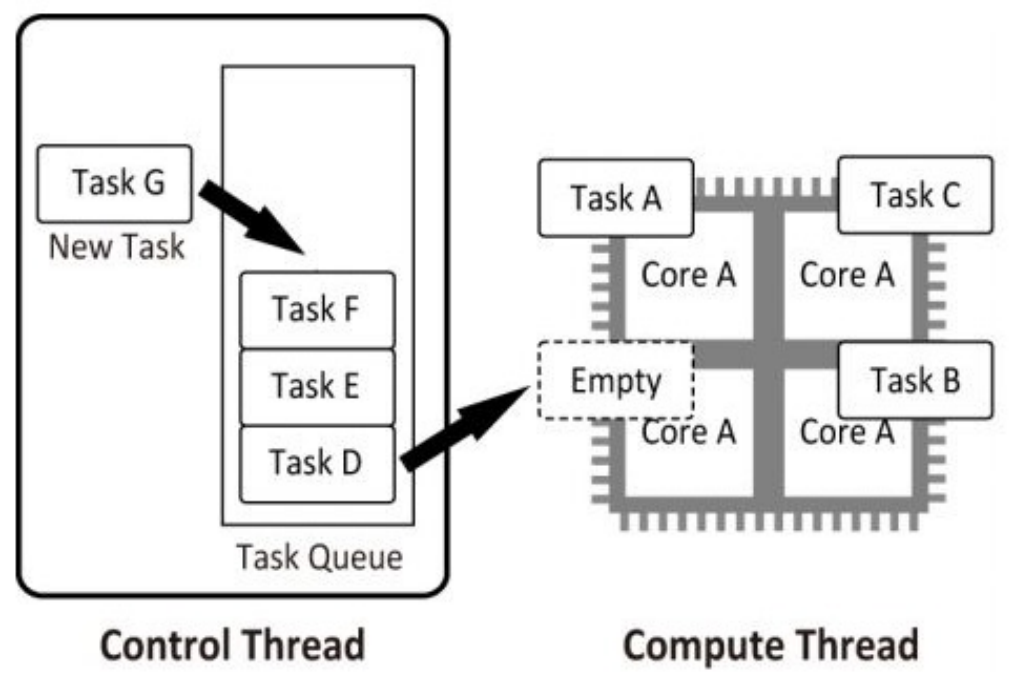
- **Cada procesador** realiza **la misma tarea** sobre un subconjunto independiente de datos.
- **Ej:** Dos granjeros se dividen el área de césped a podar.
- El caso clásico de paralelismo de datos, es el cálculo de pi por partes usando el método de monte carlo:

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

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

part_count = [n/np for i in range(np)] # List of points in each worker

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

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

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

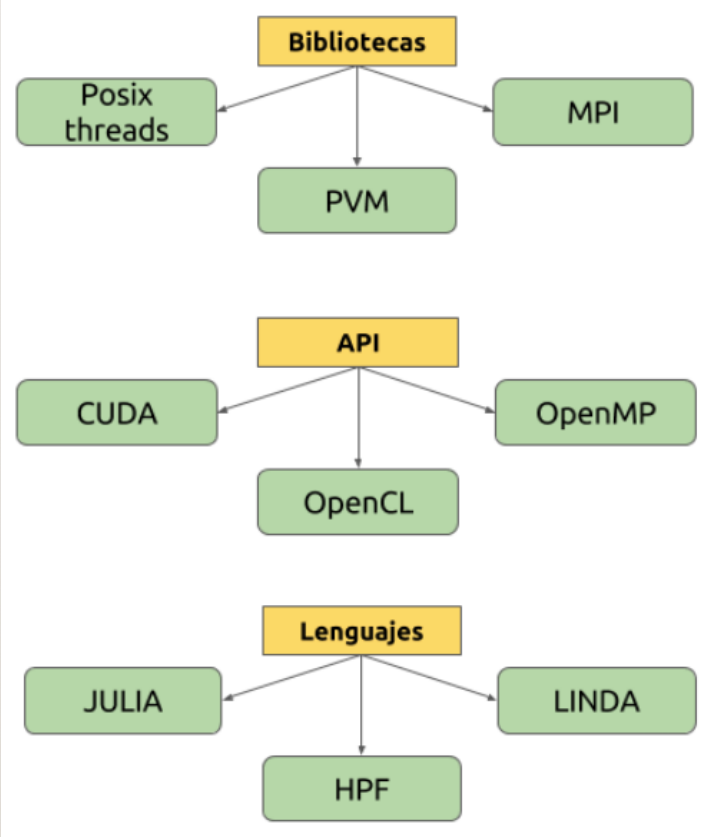
Ejemplo hecho en python

TIPOS DE PARALELISMO
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.



ALGUNAS HERRAMIENTA DE PROGRAMACIÓN PARALELA



LENGUAJES QUE INTEGRAN EL PARADIGMA

Lenguaje	Implementación
C	Posix, MPI, OpenMp
C++	Rogue Wave, Boost, Thread, Dlib, OpenMP, OpenThreads, Parallel Patterns Library, POCO C++ Libraries, POSIX Threads, Qt Qthread, Stapl, TBB, IPP
C#	Task Parallel Library. Parallel Query PLINQ,
Fortran	Co-arrays, MPI, MPICH, OpenMPI, OpenMP, OMP, ADAPTOR, PGI CUDA Fortran compiler, PIPS.
Go	Goroutines y channels
Java	Clase Thread (interfaz **Runnable**)
Javascript	Parallel JS (usando web workers)
Julia	Tareas asíncronas, Multihilos, computación distribuida
Matlab	Parallel computing toolbox
Python	Múltiples librerías
Ruby	Gem Parallel

REFERENCIAS

- Programación Paralela: http://ferestrepoca.github.io/paradigmas-de-programacion/paralela/paralela_teoria/index.html, Fabián Bernal, Camilo Albarracín, Juan Gaona, Luis Giraldo, Camilo Mosquera, Santiago Peña, Yeliana Torres, Juan Ovalle, José Nieto, Diego Chacón, Samael Salcedo, Antonio Suarez, Diego Cortés, Jose Pinzón, Pedro Higuera, Cristian Baquero
- Paralelismo en C++, Luis Miguel Sánchez
- Laboratorio de paralelismo, Dr. Javier Muguerza y Dr. Agustin Arruabarrena