4.2 Tipos de computación paralela Paralelismo

a nivel de bit.

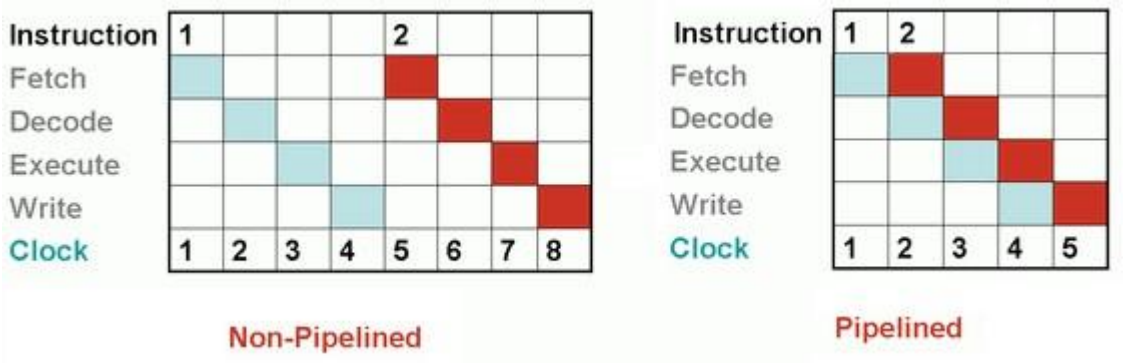
Desde el advenimiento de la integración a gran escala (VLSI) como tecnología de fabricación de chips de computadora en la década de 1970 hasta alrededor de 1986, la aceleración en la arquitectura de computadores se lograba en gran medida duplicando el tamaño de la palabra en la computadora, la cantidad de información que el procesador puede manejar por ciclo. El aumento del tamaño de la palabra reduce el número de instrucciones que el procesador debe ejecutar para realizar una operación en variables cuyos tamaños son mayores que la longitud de la palabra. Por ejemplo, cuando un procesador de 8 bits debe sumar dos enteros de 16 bits, el procesador primero debe adicionar los 8 bits de orden inferior de cada número entero con la instrucción de adición, a continuación, añadir los 8 bits de orden superior utilizando la instrucción de adición con acarreo que tiene en cuenta el bit de acarreo de la adición de orden inferior, en este caso un procesador de 8 bits requiere dos instrucciones para completar una sola operación, en donde un procesador de 16 bits necesita una sola instrucción para poder completarla.

Históricamente, los microprocesadores de 4 bits fueron sustituidos por unos de 8 bits, luego de 16 bits y 32 bits, esta tendencia general llegó a su fin con la introducción de procesadores de 64 bits, lo que ha sido un estándar en la computación de propósito general durante la última década.

Este método está “estancado” desde el establecimiento de las arquitecturas de 32 y 64 bits.

Paralelismo a nivel de instrucción.

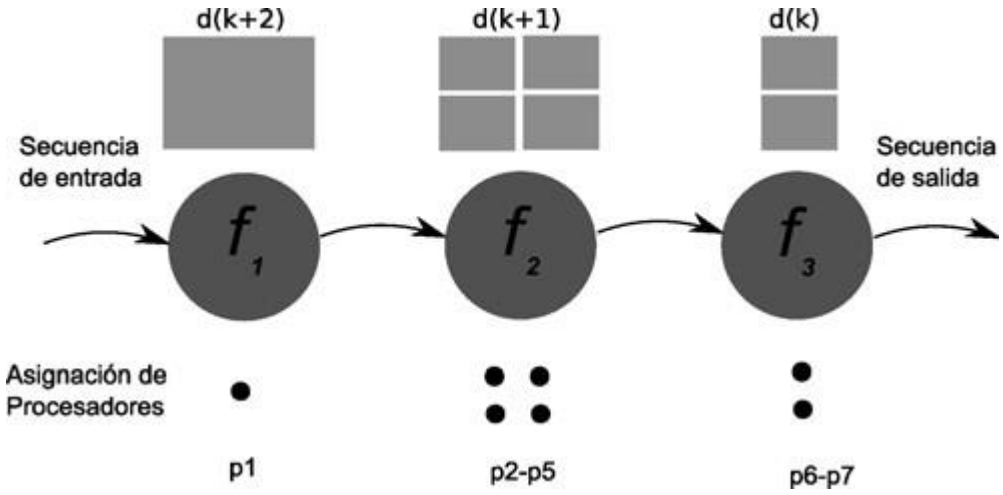
Los procesadores modernos tienen "pipeline" de instrucciones de varias etapas. Cada etapa en el pipeline corresponde a una acción diferente que el procesador realiza en la instrucción correspondiente a la etapa; un procesador con un pipeline de N etapas puede tener hasta n instrucciones diferentes en diferentes etapas de finalización. El ejemplo canónico de un procesador segmentado es un procesador RISC, con cinco etapas: pedir instrucción, decodificar, ejecutar, acceso a la memoria y escritura. El procesador Pentium 4 tenía un pipeline de 35 etapas.



Además del paralelismo a nivel de instrucción del pipelining, algunos procesadores pueden ejecutar más de una instrucción a la vez. Estos son conocidos como procesadores superescalares. Las instrucciones pueden agruparse juntas sólo si no hay dependencia de datos entre ellas. El scoreboarding y el algoritmo de Tomasulo — que es similar a scoreboarding pero hace uso del renombre de registros— son dos de las técnicas más comunes para implementar la ejecución fuera de orden y la paralelización a nivel de instrucción. **Paralelismo de datos.**

El paralelismo de datos es el paralelismo inherente en programas con ciclos, que se centra en la distribución de los datos entre los diferentes nodos computacionales que deben tratarse en paralelo. "La paralelización de ciclos conduce a menudo a secuencias similares de operaciones —no necesariamente idénticas— o funciones que se realizan en los elementos de una gran estructura de datos". Muchas de las aplicaciones científicas y de ingeniería muestran paralelismo de datos.

Una dependencia de terminación de ciclo es la dependencia de una iteración de un ciclo en la salida de una o más iteraciones anteriores. Las dependencias de terminación de ciclo evitan la paralelización de ciclos.



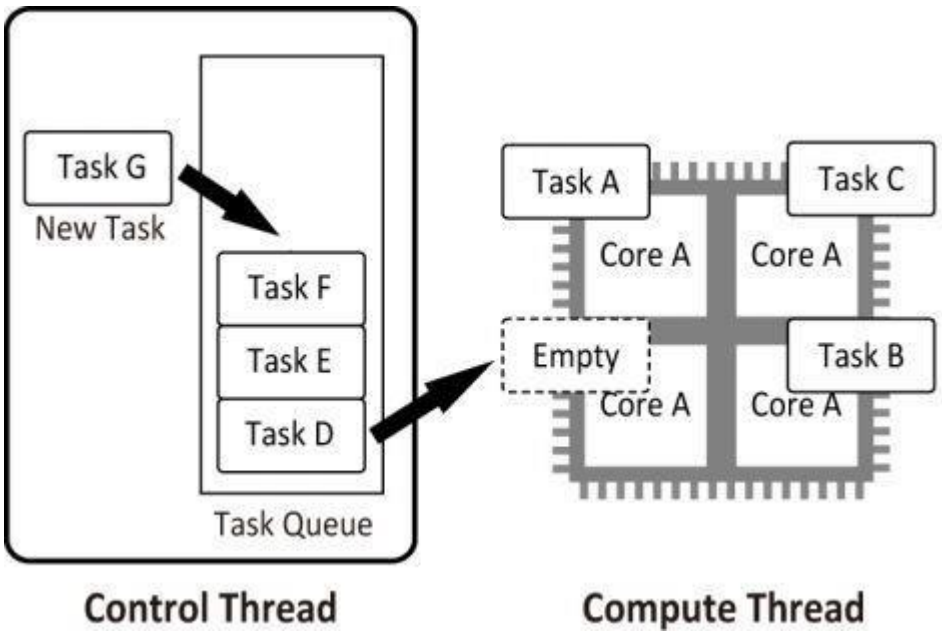
Cascada de paralelismo de datos. En este ejemplo se observan 7 procesadores ejecutando una cascada. La primera etapa corre en un procesador, la segunda etapa en 4 procesadores, y la tercera etapa en 2 procesadores.

Paralelismo de tareas.

Paralelismo de tareas es un paradigma de la programación concurrente que consiste en asignar distintas tareas a cada uno de los procesadores de un sistema de cómputo. En consecuencia, cada procesador efectuará su propia secuencia de operaciones.

En su modo más general, el paralelismo de tareas se representa mediante un grafo de tareas, el cual es subdividido en subgrafos que son luego asignados a diferentes procesadores. De la forma como se corte el grafo, depende la eficiencia de paralelismo resultante. La partición y asignación óptima de un grafo de tareas para ejecución concurrente es un problema NP-completo, por lo cual en la práctica se dispone de métodos heurísticos aproximados para lograr una asignación cercana a la óptima.

Sin embargo, existen ejemplos de paralelismo de tareas restringido que son de interés en programación concurrente. Tal es el caso del paralelismo encauzado, en el cual el grafo tiene forma de cadena, donde cada nodo recibe datos del nodo previo y sus resultados son enviados al nodo siguiente. El carácter simplificado de este modelo permite obtener paralelismo de eficiencia óptima.



Cada hilo realiza una tarea distinta e independiente de las demás.