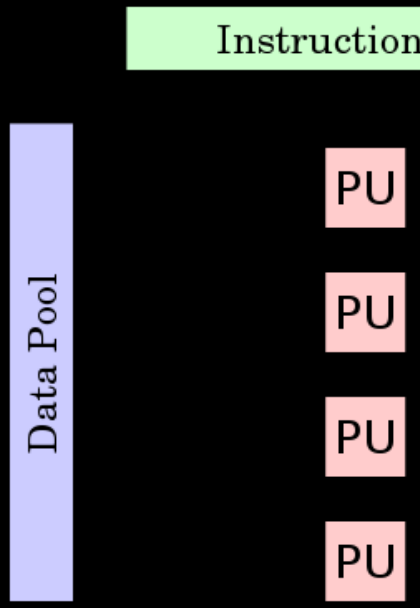
¡Hola!, Ingenieros

Secuencial

Paralelo

Programación Paralela en C (con OpenMP)



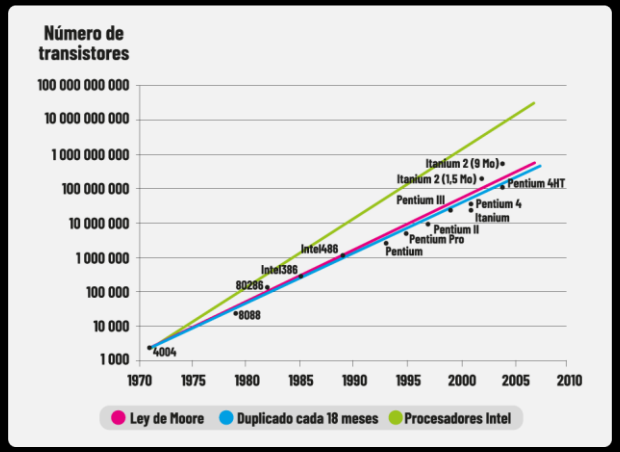
Nombre: Tenorio Martinez Jesus Alejandro

Materia: Sistemas Operativos

Profesor: GUNNAR EYAL WOLF ISZAEVICH

Limitaciones en las computadoras

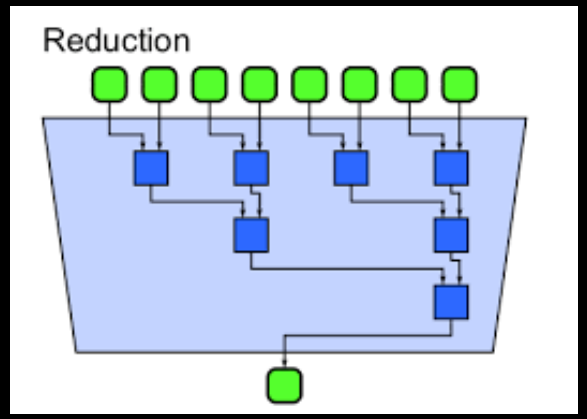
A finales de los 60's ...



Ley de Moore


Transistors
Per Die
1965 Actual Data
MOS Arrays
MOS Logic 1975 Actual Data
1975 Projection
Memory
Microprocessor
1K
4K
16K
64K
256K
1M
4M
16M
64M
128M
256M
512M
1G
2G
4G
4004
8080
8086
80286
386™
486™
Pentium®
Pentium® II
Pentium® III
Pentium® 4
Itanium™
1960
1965
1970
1975
1980
1985
1990
1995
2000
2005
2010

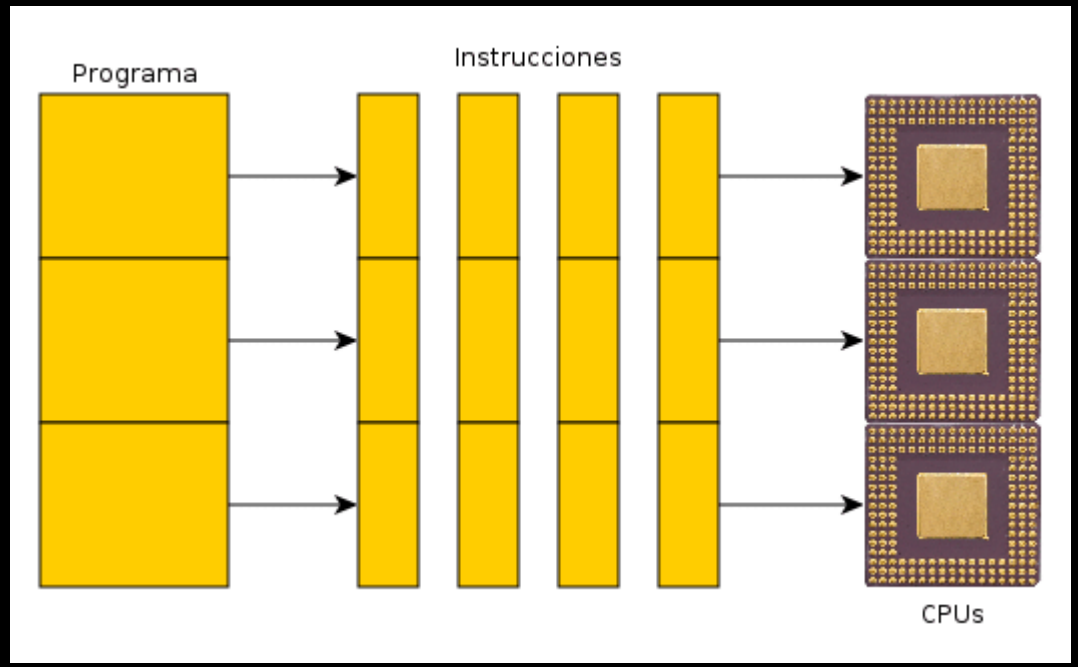
¿Cuándo paralelizar?



No todos los algoritmos son paralelizables

No conviene para pocas instancias

¿Qué pasa cuando se paraleliza?

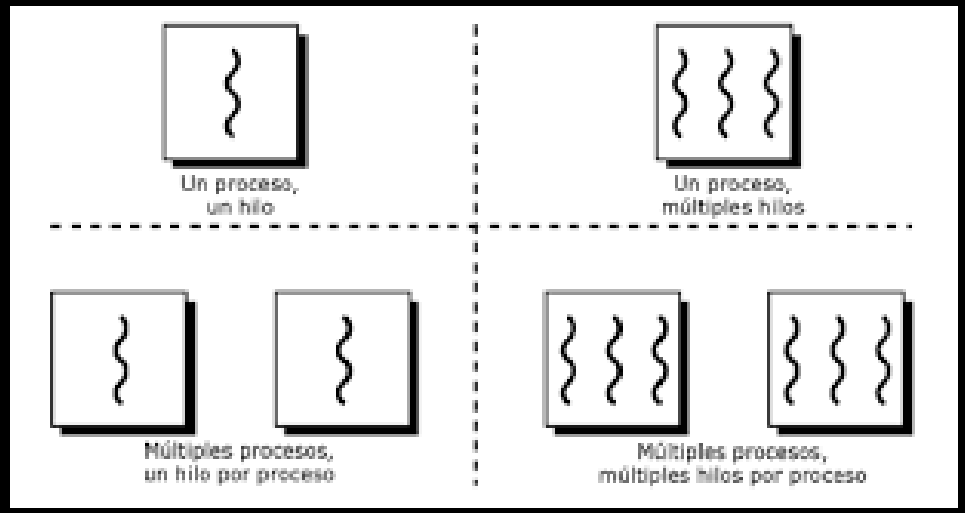


Latencia vs Rendimiento

Latencia

Rendimiento

Prioridad de Hilos



Open MP

OpenMP

OpenMP es un modelo de memoria compartida que permite escribir aplicaciones multihilos, la cual contiene un conjunto de directivas y bibliotecas con funciones y rutinas que permiten desarrollar aplicaciones paralelas

Sintaxis Básica

```
omp.h

#pragma omp

#pragma omp parallel
{

        /Bloque paralelo
}
```

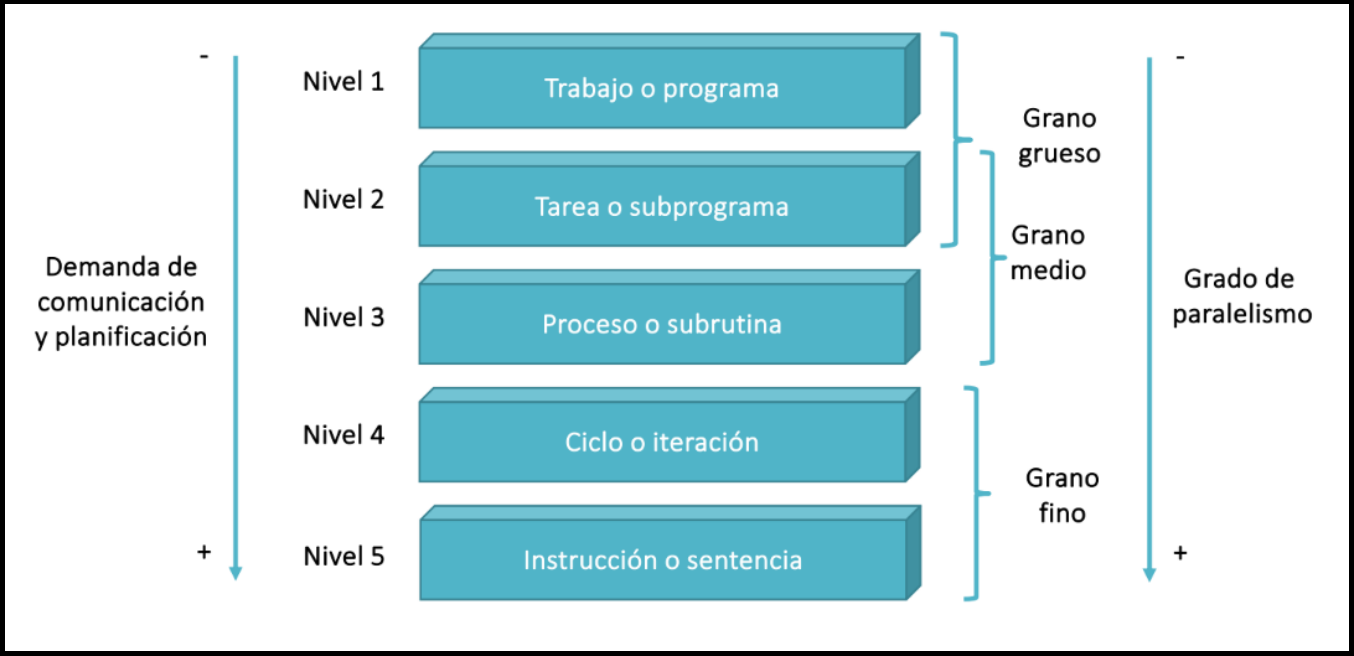
Sintaxis Basica

double omp get_wtime()

int omp_get_thread_num()

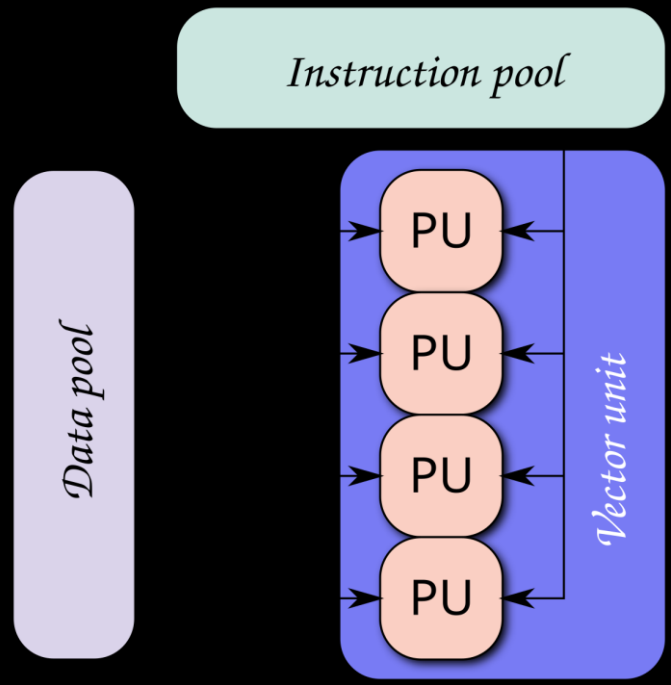
Granularidad

La granularidad es un concepto que mide el grado o nivel de paralelismo que puede aprovechar un sistema, indicando cuántas operaciones pueden realizar los procesadores sin necesitar interacción constante entre ellos. El tamaño del grano refleja la cantidad de procesamiento que se requiere en un proceso, es decir, cuántas instrucciones contiene cada segmento del programa que puede ser ejecutado en paralelo.



SPMD

Single Program Multiple Data (SPMD) es una estrategia para construir algoritmos paralelos, en la cual la información global se particiona, asignando una porción de datos a cada nodo. Este patrón de diseño es muy general y ha sido utilizado como soporte para la mayoría (no todos) de los patrones de estrategias de algoritmos paralelos.



Rutina de entorno de Ejecución

```
double A[1000];

omp_set_num_threads(4);
#pragma omp parallel
{
        // bloque paralelo
}
```

```
double A[1000];

#pragma omp parallel num_threads(4);
{
        // bloque paralelo
}
```

Para saber si se está en una región paralela activa se tiene la función:

omp_in_parallel().

Para saber cuántos procesadores posee el sistema se ocupa la función:

omp_get_num_procs().

Algoritmos con memoria compartida

Dos clases:

- Multiprocesador Simétrico (SMP): Todos los procesadores tienen el mismo acceso a un espacio de memoria compartido, y el sistema operativo asigna el tiempo de acceso de manera equitativa para cada procesador.
- Multiprocesador de Acceso No Uniforme (NUMA): Las diferentes regiones de memoria tienen costos de acceso variables, lo que significa que algunos procesadores pueden acceder más rápidamente a ciertas partes de la memoria que a otras.

En OpenMP, se pueden definir dos tipos de variables:

- Variables compartidas: Son accesibles por cualquier hilo que esté en ejecución. Sin embargo, el acceso no está controlado, lo que puede generar conflictos si varios hilos acceden a la misma variable simultáneamente.
- Variables privadas: Son creadas por cada hilo individualmente y solo existen durante la ejecución de ese hilo. Su valor se pierde una vez que el hilo termina.

Constructor de Ciclo

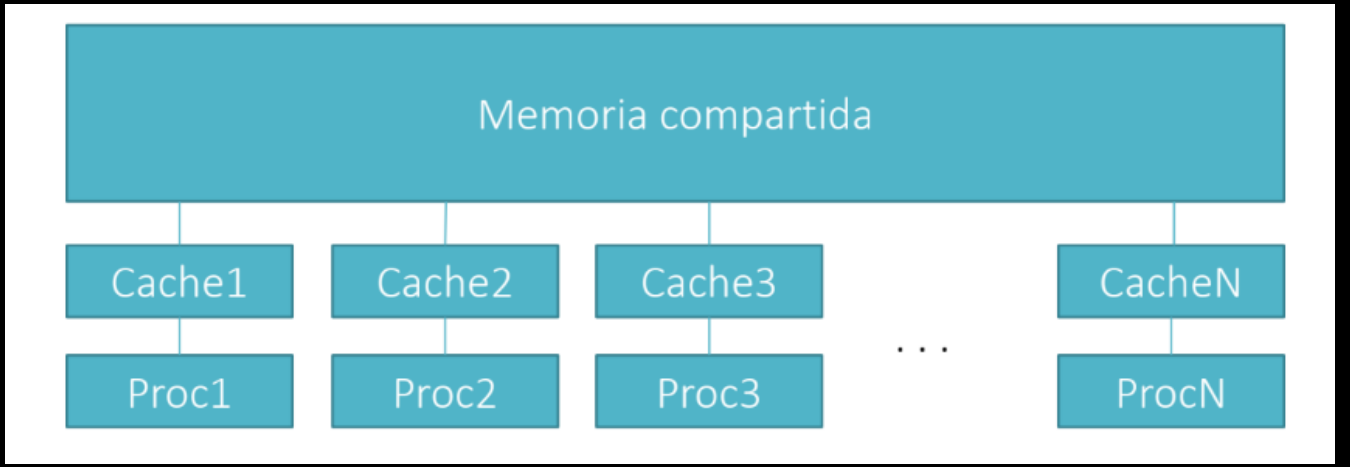
```
#pragma omp parallel
{
        int i, id;
        id = omp_get_thread_num();
        #pragma omp for
        for (i=0; i<ENE ; i++){
                do_it(id);
        }
}
```

```
#pragma omp parallel for
{
        for (i=0; i<MAX ; i++){
                res[i] = do_it(i);
        }
}
```

Sincronización

La sincronización en la programación paralela es el proceso de garantizar que los procesos y hilos se coordinen para que se ejecuten correctamente. Es fundamental para la fiabilidad, el rendimiento y la corrección de los sistemas paralelos

Coherencia y consistencia



Coherencia Se refiere a la relación entre los valores de las variables compartidas en diferentes ubicaciones de memoria. La coherencia de datos es importante para mantener el rendimiento y la escalabilidad de los sistemas de memoria compartida. Consistencia Se refiere a la propiedad de que los valores de los datos son los mismos en todos los nodos de un sistema distribuido. La consistencia de los datos garantiza que si el programador sigue las reglas, la memoria será consistente y el resultado de las operaciones de memoria será predecible

Flush y Reduccion

La directiva flush fuerza que los valores de las variables de un hilo se escriban en la memoria principal (o se sincronicen con la caché), y que otros hilos lean los valores actualizados desde la memoria principal, no desde la caché local de cada procesador.

Cuando, en una estructura de datos, se desea realizar un cálculo entre todos los elementos del conjunto, pero existe una dependencia entre los ciclos y esta dependencia no es posible eliminarla, se puede utilizar un patrón conocido como reducción. La mayoría de los entornos de programación paralelo proporcionan soporte para las operaciones de reducción.

Ya sea dentro de un bloque paralelo o dentro de un constructor de ciclo, al realizar una reducción

Hay más de programación paralela

http://www.openmp.org/wp-content/uploads/Intro_To_OpenMP_Mattson.pdf

http://www.openmp.org/

Introduction to Algorithms. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, McGraw-Hill.

Conclusiones y recomendaciones

La programación paralela en C con OpenMP es una poderosa herramienta que permite optimizar el rendimiento de aplicaciones al distribuir el trabajo entre varios procesadores o núcleos. A medida que la complejidad de los problemas crece, la programación secuencial empieza a mostrar limitaciones, lo que ha llevado al desarrollo de procesadores multinúcleo y la necesidad de aprovechar paradigmas como el paralelismo. OpenMP facilita este proceso mediante directivas sencillas que permiten paralelizar código escrito en C, gestionando el uso de múltiples hilos que trabajan en conjunto para resolver tareas de manera más eficiente.

Al emplear OpenMP, el programador puede dividir tareas complejas en bloques más pequeños y ejecutarlos de forma simultánea en diferentes hilos, optimizando el tiempo de ejecución y el uso de recursos del sistema. Sin embargo, no todos los algoritmos son adecuados para la paralelización, y es importante identificar aquellos en los que el paralelismo será beneficioso. Con OpenMP, las tareas compartidas entre hilos pueden gestionarse fácilmente, pero también es crucial manejar correctamente la sincronización para evitar problemas como condiciones de carrera o bloqueos.