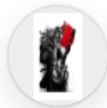


# WUOLAH



Bigbounze

[www.wuolah.com/student/Bigbounze](http://www.wuolah.com/student/Bigbounze)



24313

## Tema 1.pdf

*Resúmenes Por Temas*



2º Arquitectura de Computadores



Grado en Ingeniería Informática



Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación  
UGR - Universidad de Granada



## MÁSTER EN FINANZAS

¿Quieres alcanzar el **éxito profesional**?

■ Título Oficial

■ Prácticas Profesionales



Semana de Formación en Londres



[www.cunef.edu](http://www.cunef.edu)

# Tema 1

## - Lección 1

### . Niveles de paralelismo implícito en una aplicación.

Existen 4 niveles. De mayor a menor granularidad son, a nivel de programa (grano grueso), de funciones (grano medio), bucle (grano medio-fino) y operaciones (grano fino).

- Programas: Los diferentes programas que intervienen en una aplicación o en diferentes aplicaciones se pueden ejecutar en paralelo. Es poco probable que exista dependencia entre ellos.
- Funciones: En un nivel de abstracción más bajo, un programa puede considerarse constituido por funciones. Las funciones de un programa pueden ejecutarse en paralelo, siempre que no haya dependencias entre ellas inevitables (RAW).
- Bucle (bloques): Una función puede estar basada en la ejecución de uno o varios bucles. Se puede ejecutar en paralelo las iteraciones de estos siempre que no se de dependencia RAW.
- Operaciones: Las operaciones independientes se pueden ejecutar en paralelo. En este nivel se pueden detectar la posibilidad de usar instrucciones compuestas, que van a evitar la penalización por dependencia RAW.

### . Dependencia de datos.

Son condiciones que se deben cumplir para que un bloque de código presente dependencia respecto a otro bloque. Los tipos de dependencia de datos son:

- RAW (Read After Write) o dependencia verdadera:

○ ...

$a = b + c$

$d = a + c$

...

- WAW (Write After Write) o dependencia de salida:

○ ...

$a = b + c$

...

$a = d + e$

...

- WAR (Write After Read) o antidependencia:

# 2x1

*carné universitario*



# FOSTER'S HOLLYWOOD

\*Consulta las condiciones de la promoción en [fostershollywood2x1universitario.com](http://fostershollywood2x1universitario.com)

- ...
- $b = a + c$
- ...
- $a = d + e$
- ...

## . Paralelismo implícito en una aplicación

- Paralelismo de tareas (task parallelism o TLP – Task Level Parallelism)
  - Se encuentra extrayendo la estructura lógica de funciones de una aplicación. Está relacionado con el paralelismo a nivel de función.
- Paralelismo de datos (data parallelism o DLP-Data Level Parallelism)
  - Se encuentra implícito en las operaciones con estructuras de datos (vectores y matrices)
  - Se puede extraer de la representación matemática de la aplicación.
  - Está relacionado principalmente con el paralelismo a nivel de bucle.

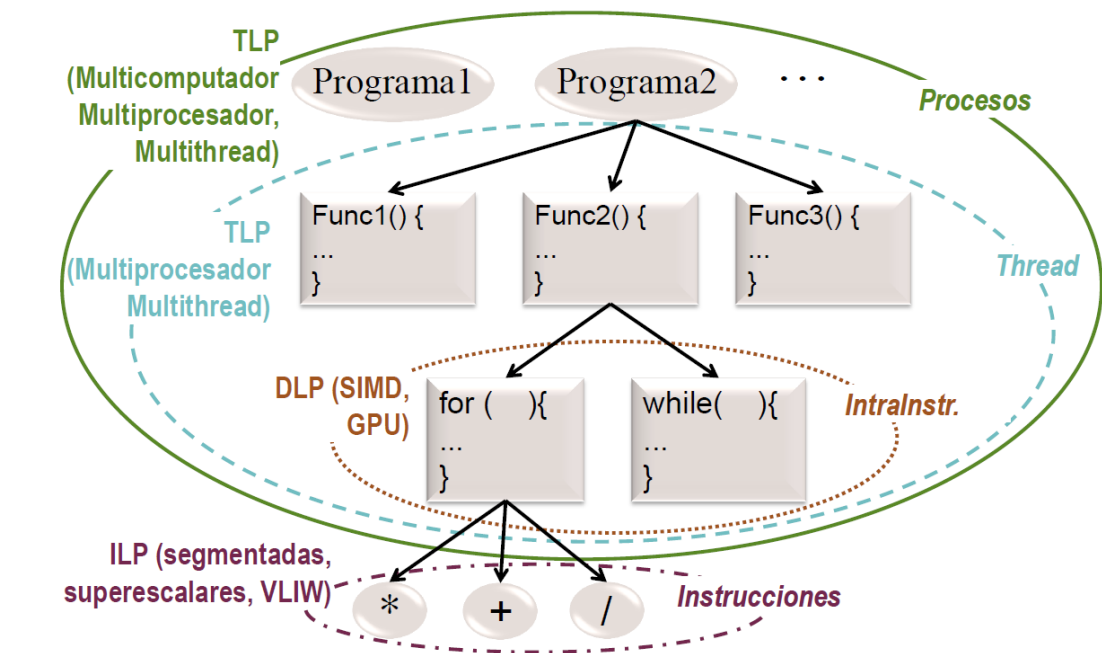
## . Paralelismo implícito, explícito y arquitecturas paralelas

El paralelismo entre programas se utiliza a nivel de procesos. En el momento en el que se ejecuta un programa, se crea el proceso asociado al programa.

El paralelismo disponible entre funciones se puede extraer también a nivel de proceso o de hebras.

Se puede aumentar la granularidad asociando un mayor número de iteraciones del bucle a cada unidad a ejecutar en paralelo. El paralelismo dentro de un bucle también se puede hacer explícito dentro de una instrucción vectorial para que sea aprovechado por arquitecturas SIMD.

El paralelismo entre operaciones se puede aprovechar en arquitecturas con paralelismo a nivel de instrucción (ILP) ejecutando en paralelo las instrucciones asociadas a estas operaciones independientes.



## . Unidades en ejecución en un computador

- Instrucciones
  - o La unidad de control de un core o procesador gestiona la ejecución de instrucciones por la unidad de procesamiento.
- Thread
  - o Es la menor unidad de ejecución que gestiona el SO.
  - o Menor secuencia de instrucciones que se pueden ejecutar en paralelo o concurrentemente.
- Proceso
  - o Mayor unidad de ejecución que gestiona el SO.
  - o Un proceso consta de uno o varios thread.

## . Thread vs. Proceso

Un proceso comprende el código del programa y todo lo que hace falta para su ejecución:

- o Datos en pila, segmentos (variables globales y estáticas) y en heap.
- o Contenido de los registros.
- o Tabla de páginas.
- o Tabla de ficheros abiertos.



Para comunicar procesos hay que usar llamadas al SO.

Un proceso puede constar de múltiples flujos de control llamados threads (hebras).  
Cada thread contiene:

- Su propia pila.
- Contenido de los registros, en particular el contador de programa y el puntero de pila.

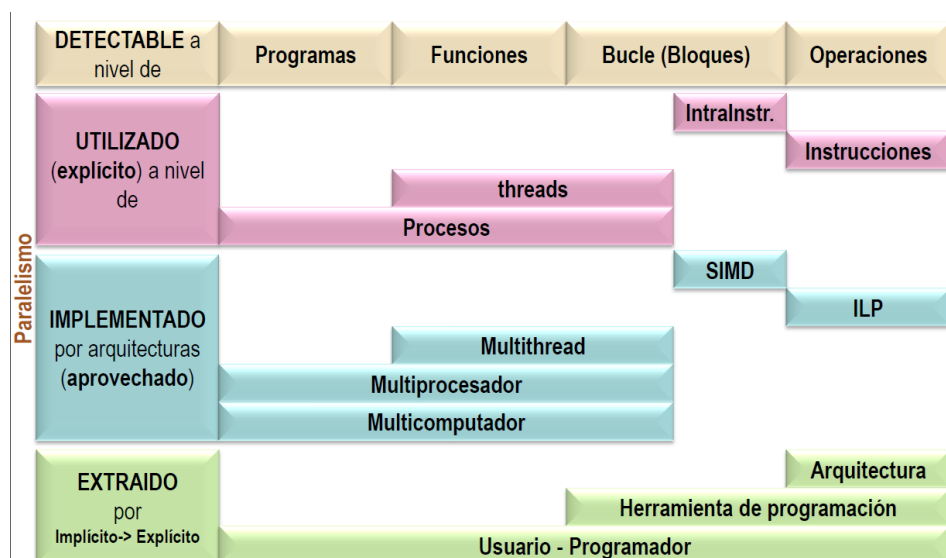
Para comunicar hebras de un proceso se usa la memoria que comparten.

Hay una menor granularidad para threads. La creación y destrucción de hebras se realiza en menor tiempo, al igual que la conmutación y la comunicación entre ellas.

### . Detección, utilización, implementación y extracción del paralelismo.

En los procesadores ILP superescalares o segmentados la arquitectura extrae paralelismo. Para ello, eliminan dependencias de datos falsas entre instrucciones y evitan problemas debidos a dependencias de datos, de control y de recursos. En estos procesadores, la arquitectura extrae paralelismo implícito. El grado de paralelismo de las instrucciones se puede incrementar con ayuda del compilador y del programador.

Podemos definir el grado de paralelismo de un conjunto de entradas a un sistema, como el máximo número de entradas del conjunto que se pueden ejecutar en paralelo. Para los procesadores las entradas son las instrucciones



## **- Lección 2**

### **. Computación paralela – Computación distribuida**

La computación paralela estudia aspectos hardware y software relacionados con el desarrollo y ejecución de aplicaciones en un sistema de cómputo compuesto por múltiples cores/procesadores/computadores que es visto externamente como una unidad autónoma.

La computación distribuida estudia los aspectos hardware y software relacionados con el desarrollo y ejecución de aplicaciones en un sistema distribuido, es decir, en una colección de recursos autónomos situados en distintas localizaciones físicas.

### **. Computación distribuida a gran escala: Computación Grid**

La computación distribuida a baja escala estudia los aspectos relacionados con el desarrollo y ejecución de aplicaciones en una colección de recursos autónomos de un dominio administrativo situados en distintas localizaciones físicas conectados a través de infraestructura de red local.

La computación grid estudia los aspectos relacionados con el desarrollo y ejecución de aplicaciones en una colección de recursos autónomos de múltiples dominios administrativos geográficamente distribuidos conectados con infraestructuras de telecomunicaciones.

### **. Computación en la nube**

Comprende los aspectos relacionados con el desarrollo y ejecución de aplicaciones en un sistema cloud.

El sistema cloud ofrece servicios de infraestructura, plataforma y/o software por los que se paga cuando se necesitan y a los que se accede típicamente a través de una interfaz de auto-servicio.

Consta de recursos virtuales que:

- Son una abstracción de los recursos físicos.
- Parecen ilimitados en número y capacidad y son reclutados/liberados de forma inmediata sin interacción con el proveedor.
- Soportan el acceso de múltiples clientes.
- Están conectados con métodos estándar independientes de la plataforma de acceso.

### **. Criterios de clasificación de computadores.**

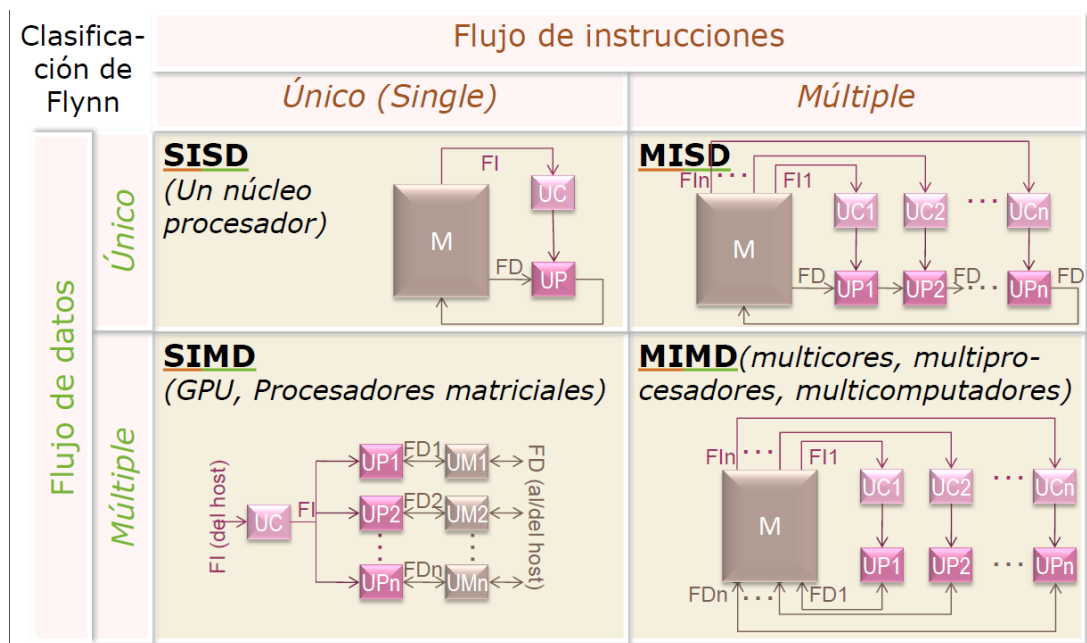
- Comercial:

- Se tiene en cuenta el segmento del mercado (de mayor a menor segmento):



- Mercado de computadores empujados
- PC/WS
- Servidores de gama baja
- Servidores de gama media
- Servidores de gama alta
- Supercomputadores
- Educación, investigación:
  - Clasificación de Flynn
  - Sistemas de memoria.
  - Flujos de control.
  - Nivel de paralelismo aprovechado.

## . Clasificación de Flynn de arquitecturas.



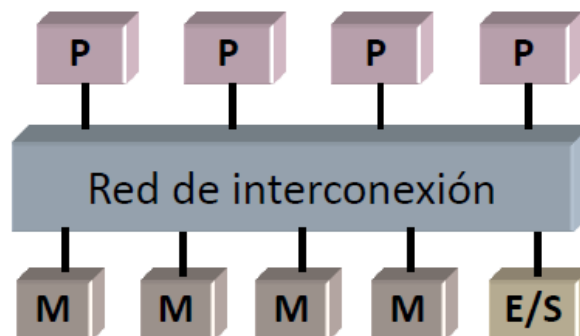
- SISD: un único flujo de instrucciones procesa operandos y genera resultados, definiendo un único flujo de datos. Corresponde a una ejecución secuencial.
- SIMD: un único flujo de instrucciones procesa operandos y genera resultados, definiendo múltiples flujos de datos, dado que cada instrucción codifica realmente varias operaciones iguales, cada una actuando sobre operadores distintos. Aprovecha paralelismo de datos.



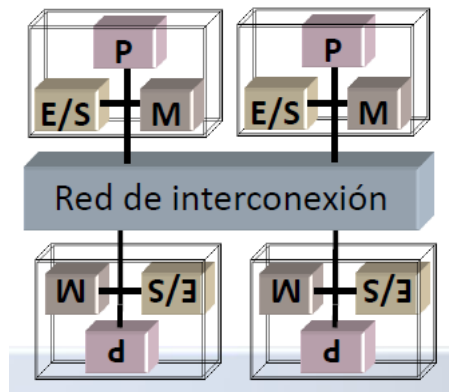
- MISD: se ejecutan varios flujos distintos de instrucciones, aunque todos actúan sobre el mismo flujo de datos.
- MIMD: el computador ejecuta varias secuencias o flujos distintos de instrucciones, y cada uno de ellos procesa operandos y genera resultados definiendo un único flujo de instrucciones, de modo que existen también varios flujos de datos uno por cada flujo de instrucciones.

## . Clasificación de Computadores Paralelos MIMD según el sistema de memoria

- Multiprocesadores (SMP):
  - Todos los procesadores comparten el mismo espacio de direcciones.
  - El programador no necesita conocer donde están almacenados los datos haciéndola más sencilla.
  - Mayor latencia – Poco escalable.
  - Comunicación implícita mediante variables compartidas. Datos no duplicados en memoria principal.



- Multicomputador:
  - Cada procesador tiene su espacio de direcciones propio.
  - El programador necesita conocer donde están almacenados los datos.
  - Menor latencia – Más escalable.
  - Comunicación explícita mediante software para paso de mensajes. Datos duplicados en memoria principal, copia de datos. Hay que distribuir código y datos entre procesadores.



Nota: Un sistema es escalable si al añadir más recursos al sistema éste incrementa sus prestaciones de forma proporcional al número de recursos utilizados. Idealmente, la productividad debería ser proporcional y la latencia o tiempo por operación, mantenerse constante.

### . Comunicación uno a uno en un multiprocesador

En un sistema con múltiples procesadores, un procesador puede producir un dato que necesite otro. Es necesario pues que los procesadores puedan acceder a datos que se han generado. Dado que los procesadores comparten el espacio de direcciones, pueden acceder a contenidos de direcciones de memoria que otros han producido. De este modo la comunicación se realiza implícitamente con instrucciones de carga (load) y almacenamiento (store) del procesador. Se debe garantizar que el flujo de control consumidor del dato lea la variable compartida cuando el productor haya escrito en la variable dato.

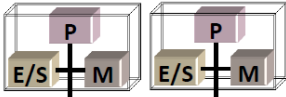
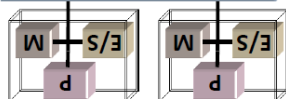
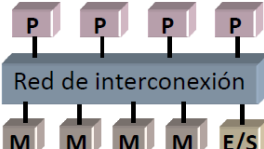
### . Comunicación uno a uno en multicomputador

En este caso el mismo dato quedará duplicado en el sistema tras la transferencia. Se pueden emplear dos funciones para realizar la comunicación. Una que permite enviar a un destino un dato (send), y otra que permita recibirlo (receive).

### . Incremento de escalabilidad en multiprocesadores y red de interconexión

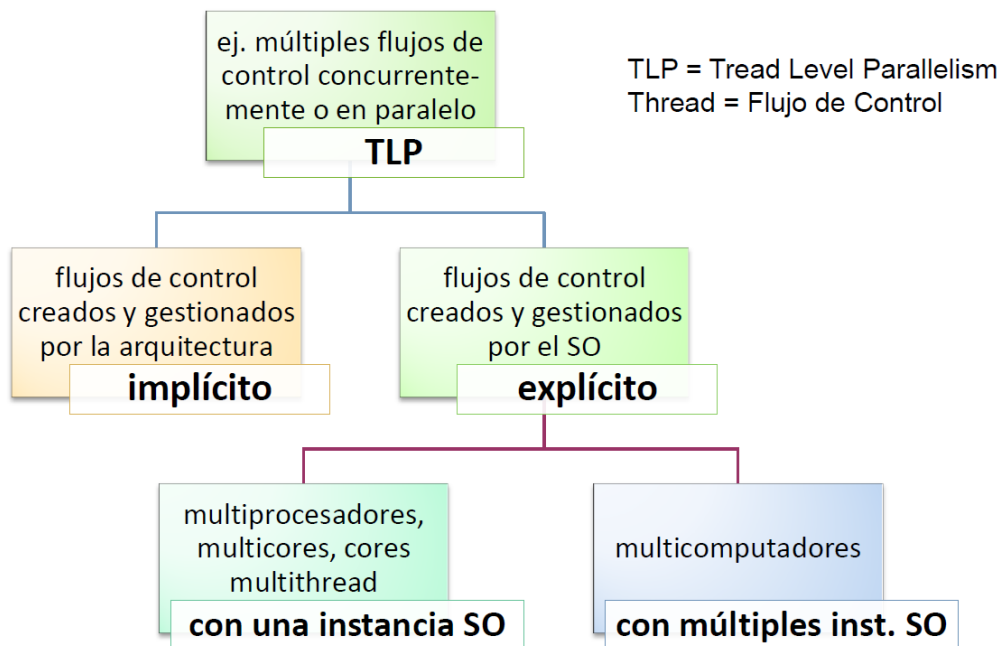
- Aumentar cache del procesador
- Usar redes de menor latencia y mayor ancho de banda que un bus.
- Distribuir físicamente los módulos de memoria entre los procesadores. Se sigue compartiendo espacio de direcciones.

## . Clasificación completa de computadores según el sistema de memoria

<b>Multi-computadores</b> Memoria no compartida	<b>NORMA</b> <i>No Remote Memory Access</i>	<i>ej. cluster, red de computadores</i>	Memoria físicamente distribuida	<div>+</div> <div>+</div> <div>Escalabilidad</div> <div>Nivel de empaquetamiento y conexión</div> <div>-</div> <div>-</div>
<b>Multi-procesadores</b> Memoria compartida Un único espacio de direcciones	<b>NUMA</b> <i>Non-Uniform Memory Access</i>	NUMA		
		CC-NUMA		
		COMA		
	<b>UMA</b> <i>Uniform Memory Access</i>	<b>SMP</b> <i>Symmetric MultiProcessor</i>	Memoria físicamente centralizada	
				

- NUMA: no incorporan hardware para evitar problemas por incoherencias entre cachés de distintos nodos. Esto hace que los datos modificables compartidos no se puedan trasladar a caché de nodos remotos; hay que acceder a ellos individualmente a través de la red. Se puede hacer más tolerable la latencia utilizando precaptación de memoria y procesamiento multihebra.
- CC-NUMA: arquitectura con acceso a memoria no uniforme y con caché coherente. Tienen hardware para mantener coherencia entre cachés de distintos nodos, que se encarga de las transferencias de datos compartidos entre nodos. El hardware añadido introduce un retardo que hace que estos sistemas escalen en menor grado que un NUMA.
- COMA: arquitecturas con acceso a memoria sólo caché. El sistema de mantenimiento de coherencia se encarga de llevar dinámicamente el código y los datos a los nodos donde se necesiten. Permite replicación y la migración de bloques de memoria en función de su frecuencia de uso en los nodos. El coste de este sistema y su retardo es mayor que el de CC-NUMA.

## . Clasificación con múltiples flujos de control o threads



### . Arquitecturas con DLP, ILP y TLP

- DLP (Data Level Parallelism): Ejecutan las operaciones de una instrucción concurrentemente o en paralelo. Emplea unidades funcionales vectoriales o SIMD.
- ILP (Instruction Level Parallelism): Ejecutan múltiples instrucciones concurrentemente o en paralelo. Emplea Cores escalares segmentados, superescalares o VLIW/EPIC.
- TLP (Thread Level Parallelism):
  - TLP explícito y una instancia de SO: Ejecutan múltiples flujos de control concurrentemente o en paralelo. Emplea Cores que modifican la arquitectura escalar segmentada, superescalar o VLIW/EPIC para ejecutar threads concurrentemente o en paralelo. También usa Multiprocesadores que ejecutan threads en paralelo en un computador con múltiples cores.
  - TLP explícito y múltiples instancias SO: Ejecutan múltiples flujos de control en paralelo. Emplea Multicomputadores que ejecutan threads en paralelo en un sistema con múltiples computadores.

## - Lección 3

### . Tiempo de respuesta de un programa en una arquitectura

- Tiempo de CPU de usuario: tiempo en ejecución en espacio de usuario.
- Tiempo de CPU de sistema: tiempo en el nivel del kernel del SO.

- Tiempo I/O: tiempo asociado a las esperas debidas a las entradas/salidas o asociados a la ejecución de otros programas

### . Tiempo de CPU

$$T_{CPU} = NI \times CPI \times T_{CICLO}$$

$$\text{Ciclos por Instrucción (CPI)} = \frac{\text{Ciclos\_del\_Programa}}{\text{Numero\_de\_Instrucciones(NI)}}$$

Hay procesadores que pueden lanzar para que empiecen a ejecutarse varias instrucciones al mismo tiempo.

- CPE: número mínimo de ciclos transcurridos entre los instantes en que el procesador puede emitir instrucciones
- IPE: instrucciones que pueden emitirse cada vez que se produce dicha emisión

$$T_{CPU} = NI \times \underbrace{(CPE / IPE)}_{CPI} \times T_{ciclo}$$

Hay procesadores que pueden codificar varias operaciones en una instrucción.

- Noper: número de operaciones que realiza un programa
- Op\_instr: número de operaciones que puede codificar una instrucción.

$$T_{CPU} = \underbrace{(Noper / Op\_instr)}_{NI} \times CPI \times T_{ciclo}$$

### . MIPS



MIPS = Millones de Instrucciones por segundo

$$\text{MIPS} = \frac{\text{NI}}{T_{\text{CPU}} \times 10^6} = \frac{F(\text{frecuencia})}{\text{CPI} \times 10^6}$$

- Depende del repertorio de instrucciones.
- Puede variar con el programa
- Puede variar inversamente con las prestaciones

#### . MFLOPS

MFLOPS: millones de operaciones en coma flotante por segundo

$$\text{MFLOPS} = \frac{\text{Operaciones\_en\_Coma\_Flotante}}{T_{\text{CPU}} \times 10^6}$$

- No es una medida adecuada para todos los programas
- El conjunto de operaciones en coma flotante no es constante en máquinas diferentes y la potencia de las operaciones en coma flotante no es igual para todas las operaciones
- Es necesaria una normalización de las instrucciones en coma flotante

#### . Benchmarks

El benchmark es una técnica utilizada para medir el rendimiento de un sistema o componente del mismo. Las medidas de prestaciones que realicemos o que nos ofrezcan terceros deben ser fiables y aceptadas por los interesados:

- Para confiar en las medidas de tiempo de respuesta, productividad y escalabilidad que se dan de un sistema, éstas se deben obtener procesando un conjunto de entradas que sean representativas del funcionamiento habitual del sistema.
- Deben permitir comparar diferentes realizaciones de un sistema o diferentes sistemas.

Podemos diferenciar distintos tipos de Benchmarks:

- De bajo nivel o microbenchmark: evalúan las prestaciones de distintos aspectos de la arquitectura o del software (el procesador, la memoria, E/S,

comunicación send/receive, mecanismos de sincronización, el SO o las herramientas de programación.

- Kernels: Son trozos de código muy utilizados en diferentes aplicaciones. Junto con los de bajo nivel son útiles para encontrar los puntos fuertes de cada máquina.
- Sintéticos: También son trozos de código, pero no pretenden obtener, pero no pretenden obtener un resultado con significado.
- Programas reales: Utilizados en el sector de computadores que pretenden evaluar como por ejemplo bases de datos, servidores web...
- Aplicaciones diseñadas: Se diseñan aplicaciones que pretenden representar a aquellas para las que se utiliza el computador.

### . Mejora o ganancia de prestaciones

Si en un computador se incrementan las prestaciones de un recurso haciendo que su velocidad sea  $p$  veces mayor. El incremento de velocidad que se consigue en la nueva situación con respecto a la previa se expresa mediante la ganancia de velocidad o speed-up  $s_p$ .

$$S_p = \frac{V_p}{V_1} = \frac{T_1}{T_p}$$

$V_1$  -> Velocidad

base

$V_p$  -> Velocidad mejorada

$T_1$  -> Tiempo base

$T_p$  -> Tiempo mejorado

### . Ley de Amdahl

La mejora de velocidad,  $S$ , que se puede obtener cuando se mejora un recurso de una máquina en un factor  $p$  está limitada por:

$$S \leq \frac{p}{1 + f(p - 1)}$$

Donde  $f$  es la fracción del tiempo de ejecución en la máquina sin la mejora durante el que no se puede aplicar esa mejora