UNCUYO
UNIVERSIDAD NACIONAL DE CUYO
FACULTAD DE INGENIERÍA

Licenciatura en Ciencias de la Computación

Arquitectura de Computadoras II

Unidad 3

Modelos y Arquitecturas Escalables

Dr. Ing. Pablo Daniel Godoy (pablo.godoy@ingenieria.uncuyo.edu.ar)

Problemas del paralelismo a nivel de instrucción o hilos en un solo núcleo

- **Aumentar** la profundidad del Pipeline (**k**) o las vías de un superescalar (**N**) aumenta la complejidad de los circuitos para manejar dependencia de datos, conflicto de recursos y saltos. Esto **aumenta el consumo de energía**.
- **Aumentar** la cantidad de **hilos por núcleo** requiere aumentar la cantidad de vías superescalar y la cantidad de unidades de ejecución (idem al anterior). Esto **aumenta el consumo de energía**. Además, la **memoria** es un **límite**.

K ≤ 20 (llegó a 31 en el Pentium 4, año 2000). N ≤ 8. Hilos por núcleo ≤ 4

Por encima de estos valores se consigue **mejor relación performance / consumo de energía** agregando más núcleos.

Regla de Pollack: aumentar N veces la complejidad de un núcleo se traduce en un aumento del $(N)^{1/2}$ en performance.

En términos de cantidad de transistores en el procesador:

Arquitecturas multinúcleo

- Dos o más **procesadores** (llamados **núcleos**) en un mismo chip **compartiendo la memoria principal**.
- Cada núcleo posee todos los elementos típicos de un procesador (Fetch, decodificadores, ALU, hardware superescalar y pipeline, registros, sistemas multihilo, etc).
- Pueden (o no) compartir diferentes niveles de caché (la L1 usualmente no es compartida).
- Pueden (o no) ser fuera de orden o multithreading.

Ejemplo de Arquitecturas multinúcleo

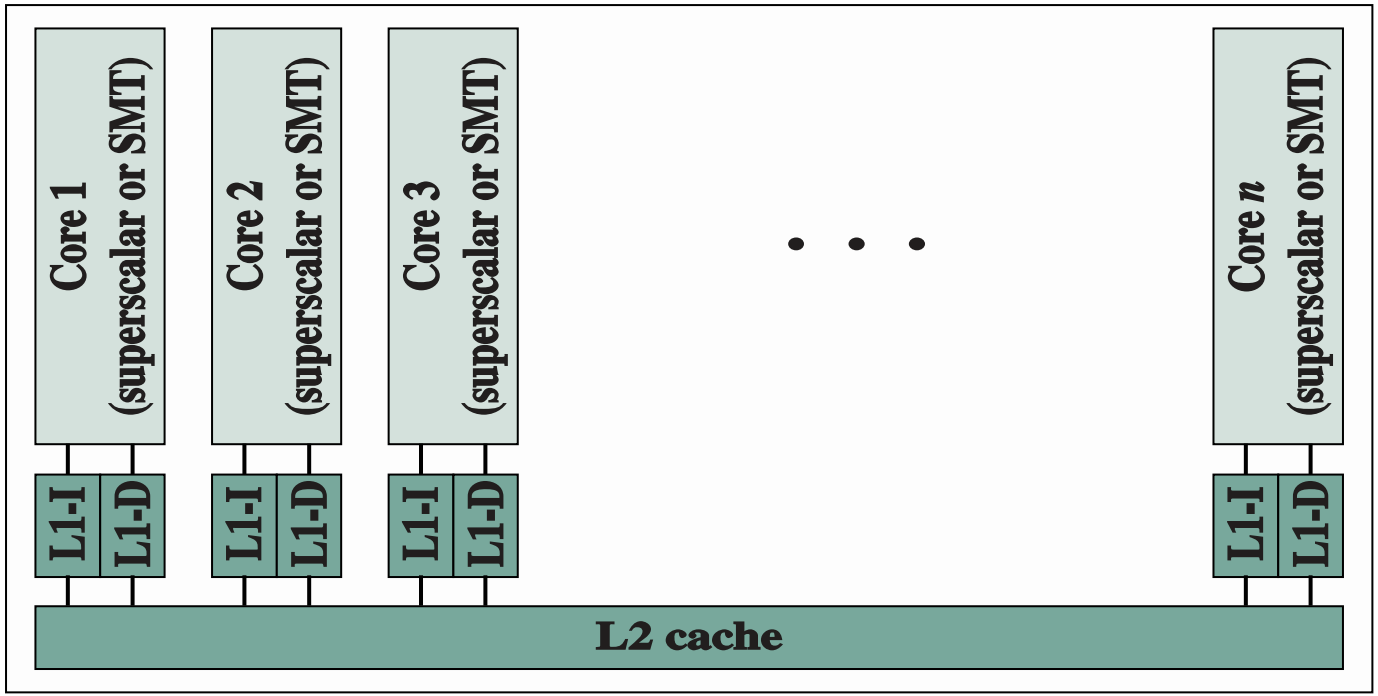


Figura obtenida de “Computer Organization and Architecture”, William Stallings, 10º edición, página 658

Sistemas multicore - Memoria caché

Importancia de la memoria caché

- Si los procesadores comparten la memoria (y el bus), solo uno podrá acceder mientras que los demás esperan.
  - **Mientras más procesadores, cada uno estará mayor cantidad de tiempo esperando poder acceder al bus.**

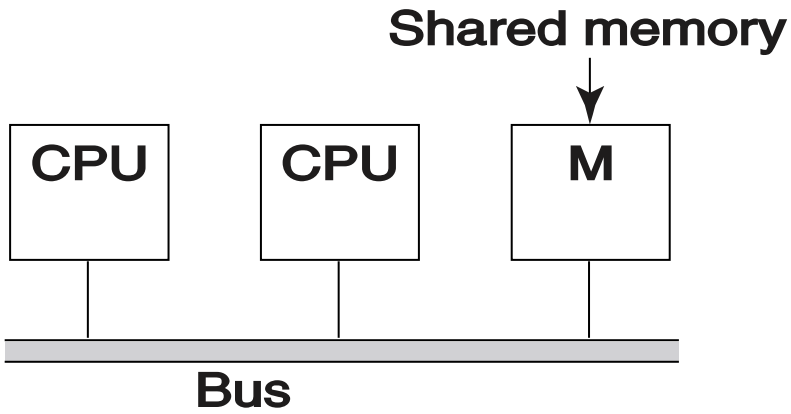


Figura obtenida de: Tanenbaum, Bos, “Modern Operating Systems”, 4° edición, página 521.

Sistemas multicore - Memoria caché

Importancia de la memoria caché

- **Solución: Memoria caché. Varios procesadores pueden trabajar sobre sus cachés al mismo tiempo.**
- Memorias caché grandes disminuyen los accesos a memoria RAM.
- Transistores dedicados a **memoria caché** consume **menos energía** que los dedicados a unidades de ejecución.

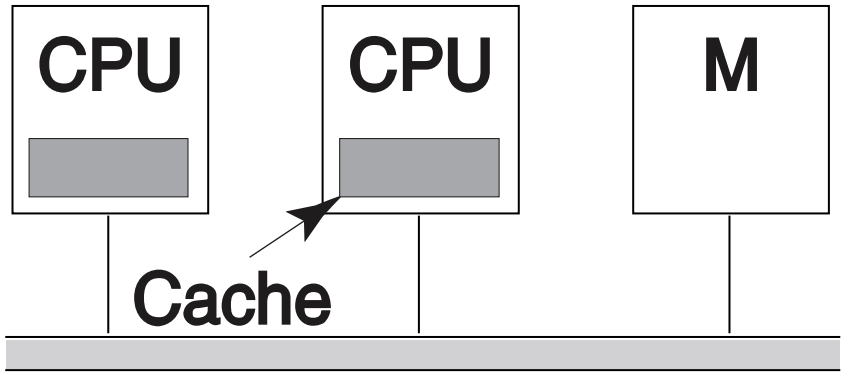


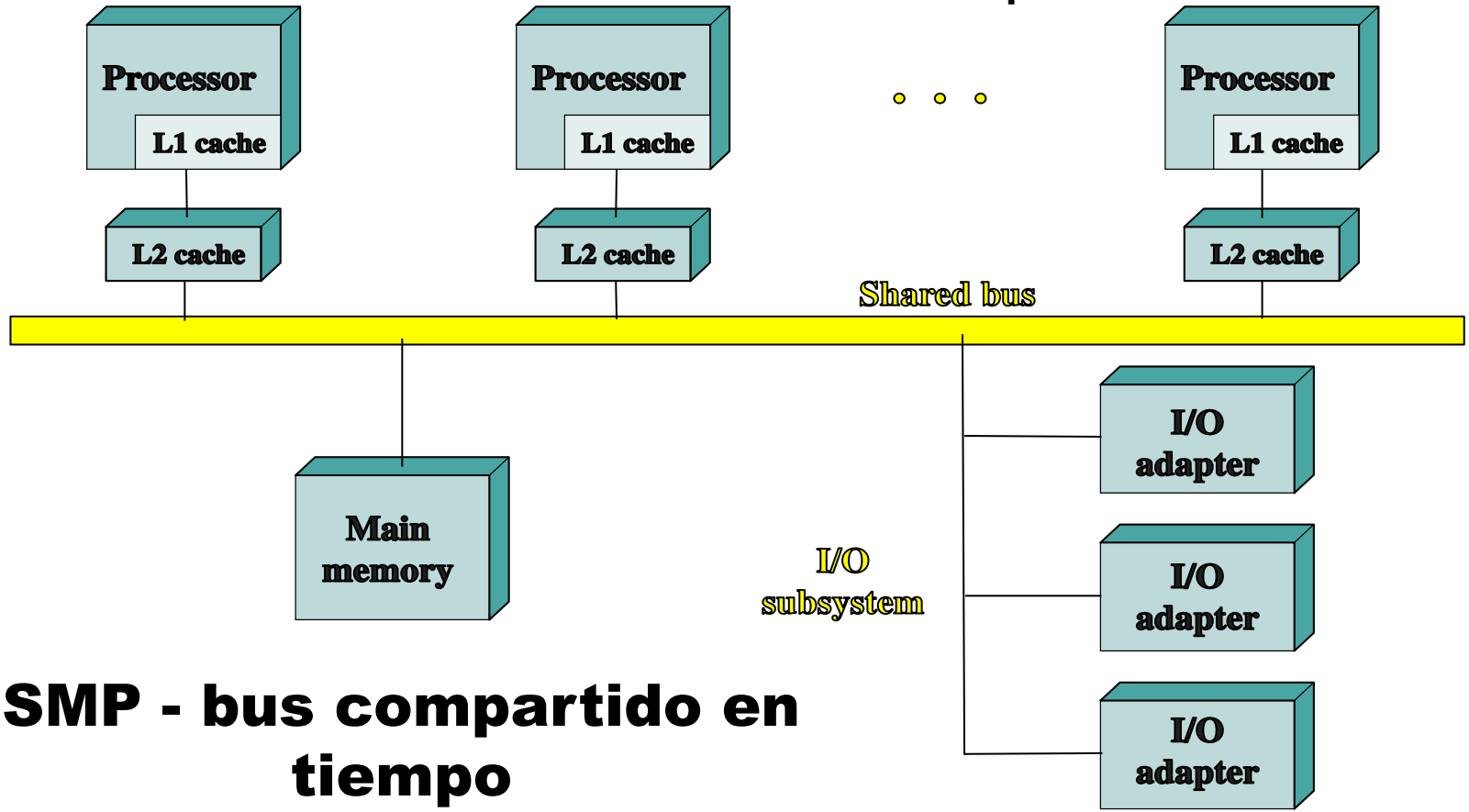
Figura obtenida de: Tanenbaum, Bos, “Modern Operating Systems”, 4° edición, página 521.

Sistemas multi-núcleo - Diferentes arquitecturas

- Chip **multi-núcleo simétrico**: Todos los núcleos tienen igual set de instrucciones y poder de cómputo.
- Chip **multi-núcleo heterogeneo**: Más de un núcleo de diferentes tipos.
  - Multicore heterogéneo con **igual set de instrucciones** (distinto poder de cómputo).
    - Ejemplo: arquitectura big.Little de ARM.
  - Multicore heterogéneo con **distinto set de instrucciones**.
    - CPU/GPU.
    - CPU/DSP (Digital Signal Processors)

SMP

- Los procesadores son **iguales**: **igual set de instrucciones** y **igual poder de cómputo**.
- Los procesadore se conectan a un bus
- Cada procesador posee su
  - ALU
  - Registros
  - Unidades de ejecución (si es superescalar).
  - Uno o más niveles de memoria Caché.
- Problema de **coherencia con la memoria caché**: Un dato puede estar en varias cachés, si un procesador lo modifica, debe actualizarse en todas las cachés y la memoria principal.



SMP - bus compartido en tiempo

Figura basada en Computer Organization and Architecture 9th Edition William Stallings

SMP Ejemplos: ARM núcleos simétricos

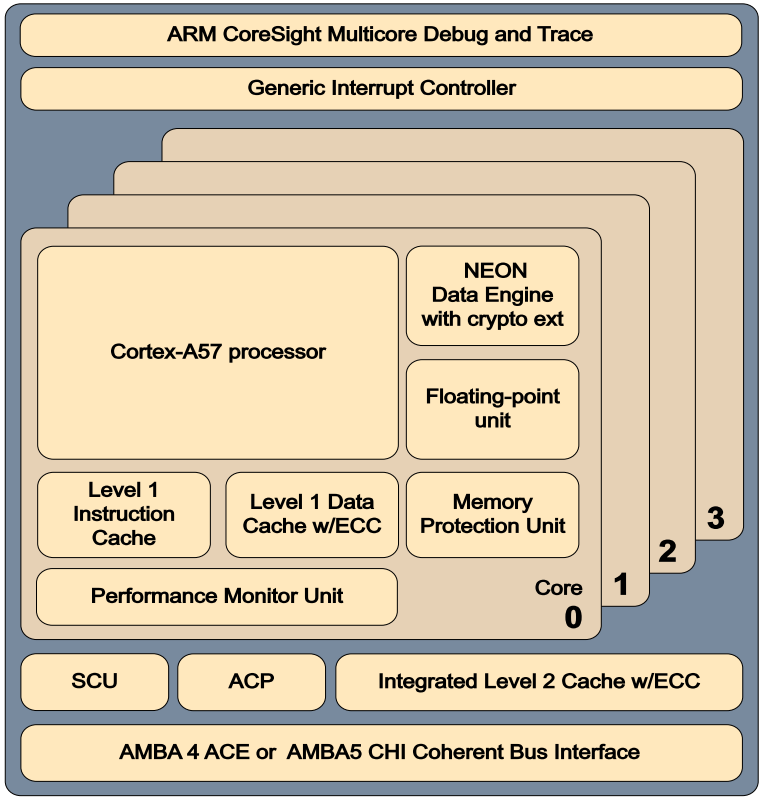


Figura obtenida de ARM Cortex -A Series, Programmer's Guide for ARMv8-A, Version: 1.0, pag 2-7 y 2-8

SMP Ejemplos: Intel i7 - 990x

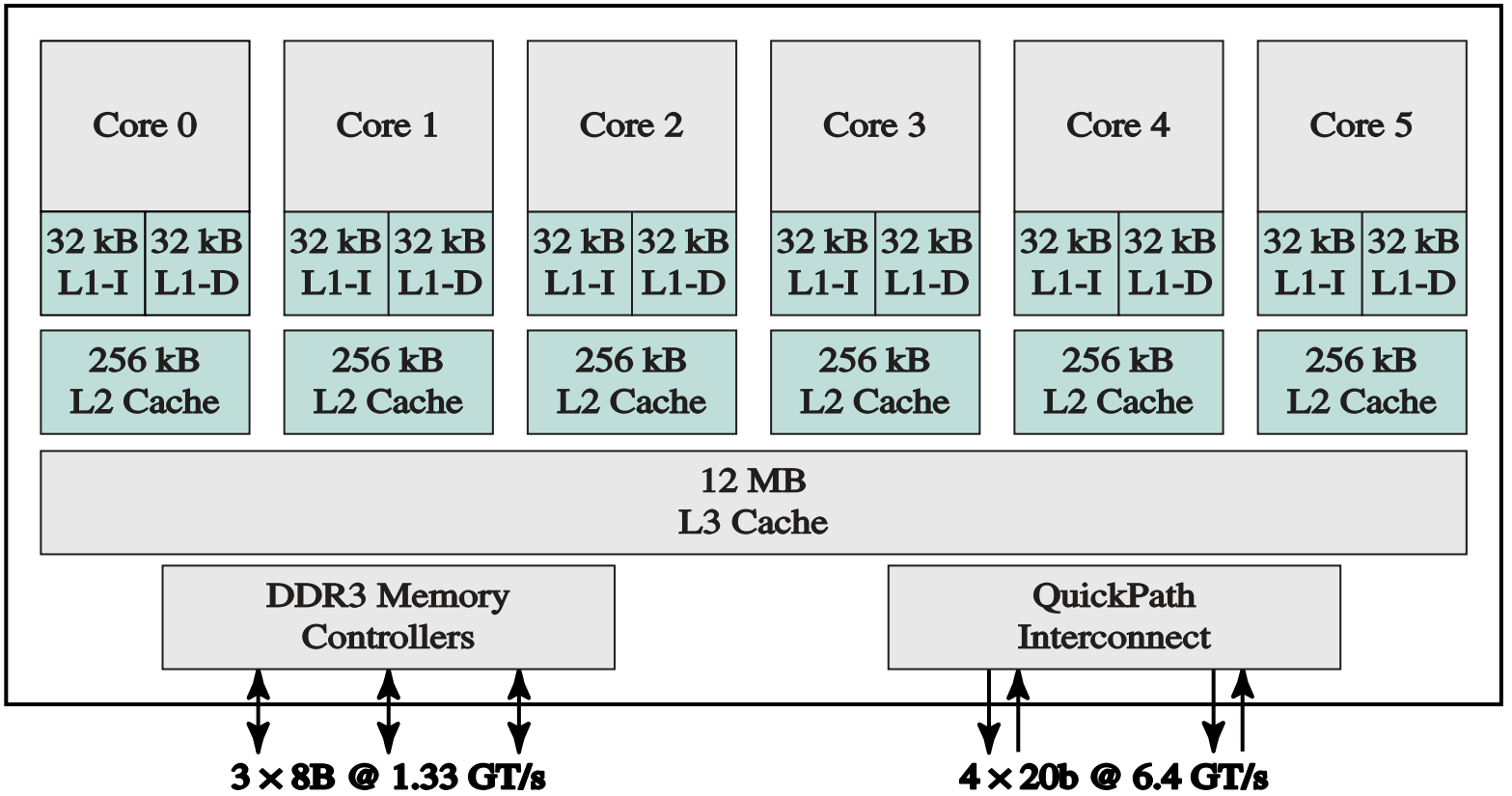


Figura obtenida de Operating Systems Internals and Design Principles - 8 ed - William Stallings

Colección de productos	Intel® Xeon® 6 processors
Nombre de código	Products formerly Granite Rapids
Segmento vertical	Workstation
Número de procesador	638
Litografía	Intel 3
Precio recomendado para el cliente	$899.00

Ejemplo hoja de datos de Intel Xeon 638

5 nm a 4 nm de TSMC

Especificaciones de la CPU

Cantidad de núcleos	16
Cantidad de Performance-cores	16
Cantidad de Efficient-cores	0
Total de subprocesos	32
Frecuencia turbo máxima	4.8 GHz

2 hilos por núcleo

https://www.intel.la/content/www/xl/es/products/sku/244334/intel-xeon-638-processor-72m-cache-3-20-ghz/specifications.html

Colección de productos	Intel® Xeon® 6 processors
Nombre de código	Products formerly Granite Rapids
Segmento vertical	Workstation
Número de procesador	698X
Litografía	Intel 3
Precio recomendado para el cliente	$7699.00

Especificaciones de la CPU

Cantidad de núcleos	86
Cantidad de Performance-cores	86
Cantidad de Efficient-cores	0
Total de subprocesos	172
Frecuencia turbo máxima	4.8 GHz

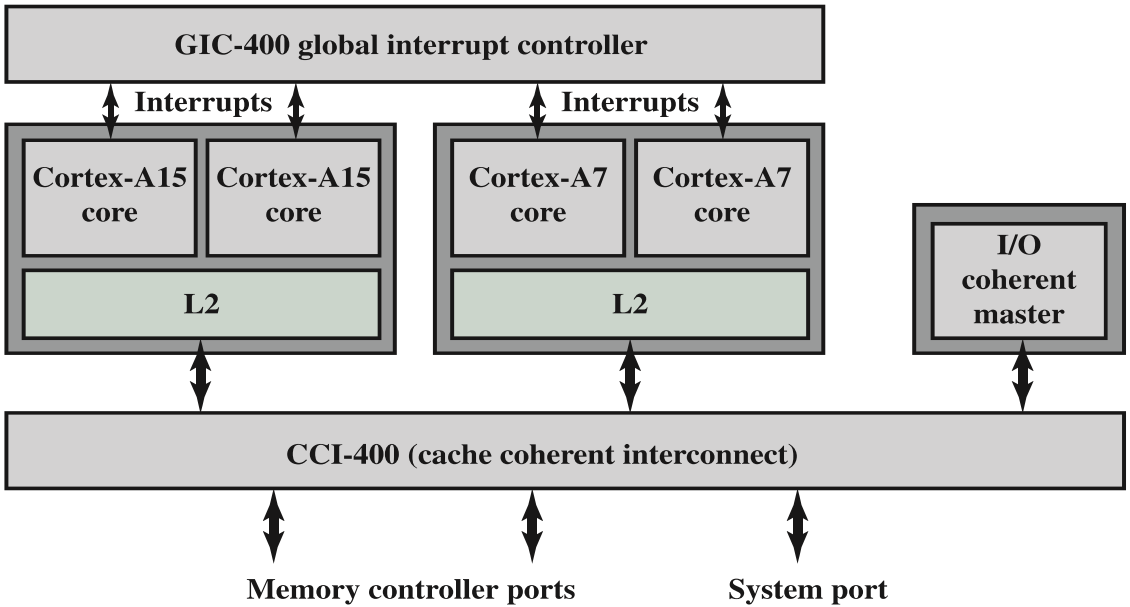
2 hilos por núcleo

https://www.intel.la/content/www/xl/es/products/sku/243857/intel-xeon-698x-processor-336m-cache-2-00-ghz/specifications.html

Sistemas multicore heterogéneo con set de instrucciones equivalente

- **Núcleos diferentes** con **igual set de instrucciones**.
- Un programa puede ejecutarse en uno u otro núcleo indistintamente (desde el punto de vista lógico).
- Ejemplos:
  - Arquitectura **big.Little de ARM**:
    - Combinan núcleos de alto poder de procesamiento con núcleos de bajo poder de procesamiento pero menor consumo de energía.
    - Teléfonos celulares, notebooks, etc.
  - Núcleos **E-cores** y **P-cores** de Intel:
    - E-cores (Efficiency Cores): Núcleos bajo consumo de energía (menos transistores, no son multihilo, menores frecuencias).
    - P-cores (Performance Cores): Núcleos de alto poder de procesamiento, multihilos, mayores frecuencia, mayores cachés.

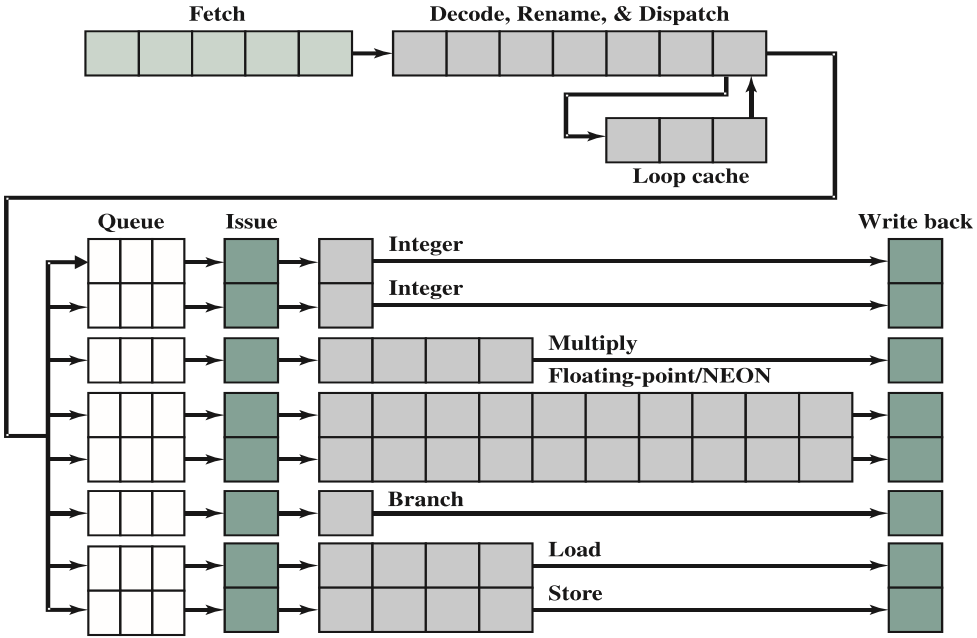
Sistemas multicore heterogéneo set de instrucciones equivalente
Arquitectura big.Little de ARM



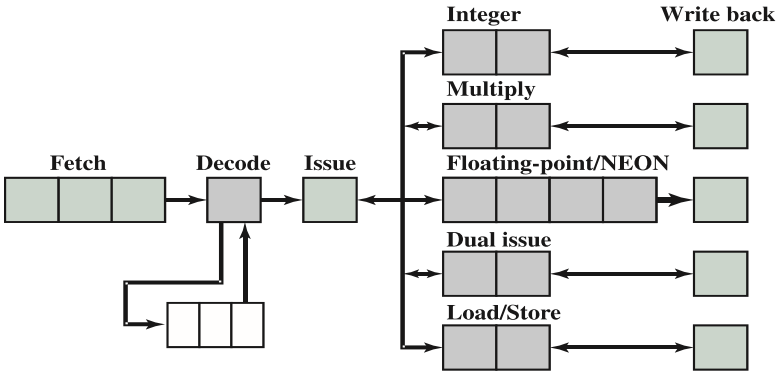
- El Cortex-A15 consume el triple de energía que el Cortex-A7
- El Cortex-A15 es el doble de potente que el Cortex-A7

Figura obtenida de “Computer Organization and Architecture”, William Stallings, 10º edición, página 671

Sistemas multicore heterogéneo set de instrucciones equivalente



Pipeline ARM Cortex A-15 (superescalar fuera de orden de 3 vías)

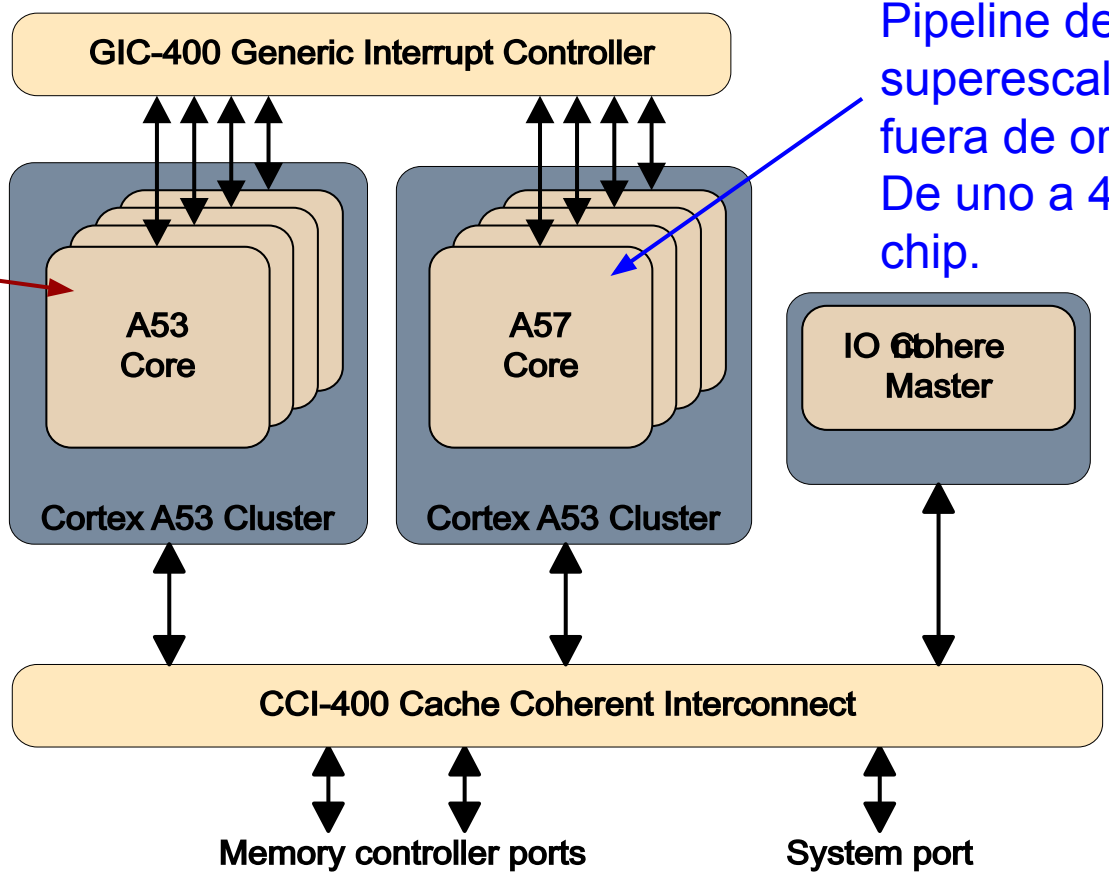


Pipeline ARM Cortex A-7 (superescalar en orden 2 vías)

Figura obtenida de “Computer Organization and Architecture”, William Stallings, 10º edición, página 671

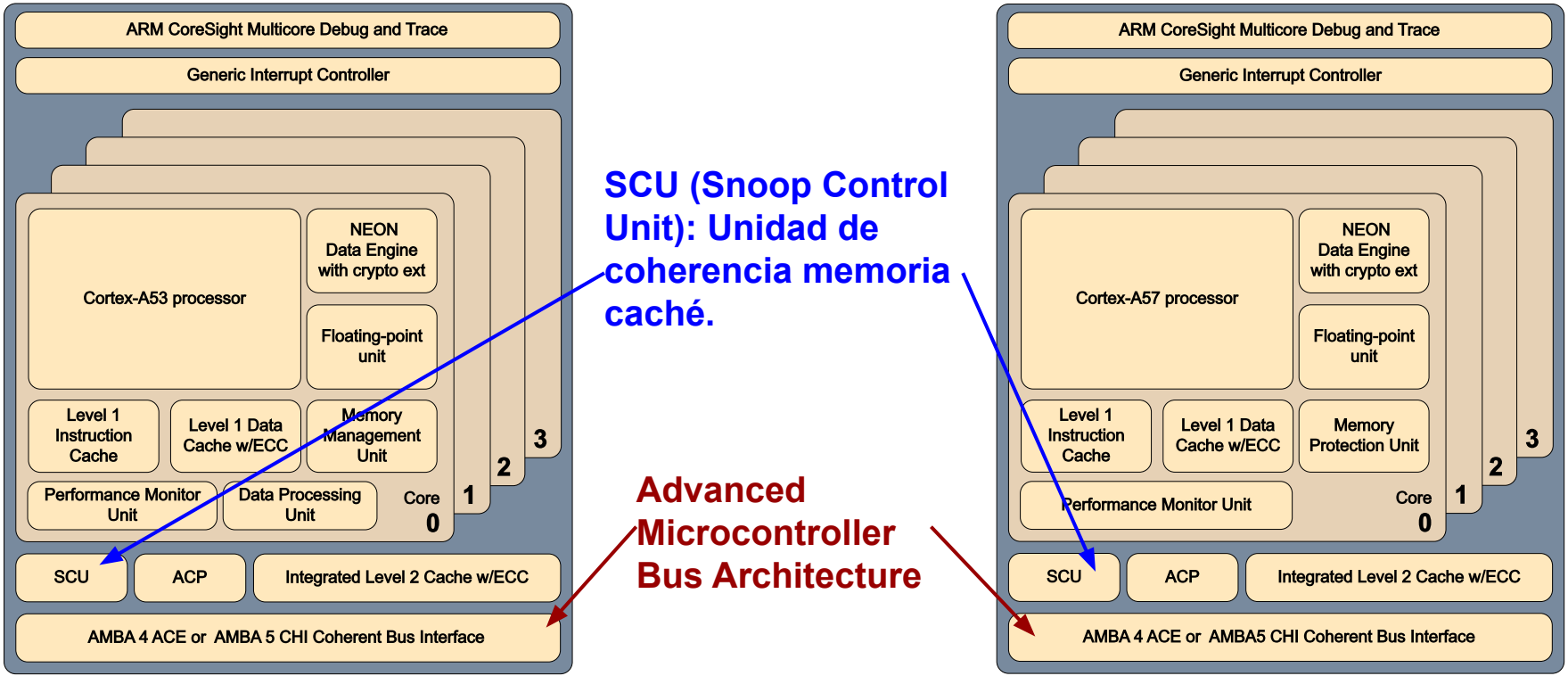
Pipeline de 8 etapas, superescalar de 2 vías, en orden. 1 a 4 núcleos por chip.

Pipeline de 15 etapas, superescalar de 3 vías, fuera de orden.
De uno a 4 núcleos por chip.



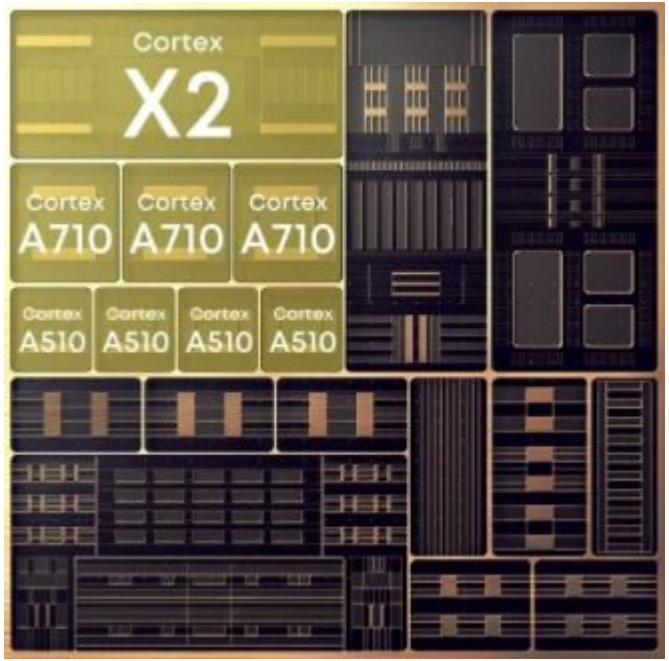
Fuente: ARM Cortex -A Series, Programmer's Guide for ARMv8-A, Version: 1.0. Pag 233

Procesadores ARM Cortex A-53 y Cortex A-57 (Snapdragon 810)



ARM Cortex-A Series Programmer's Guide for ARMv8-A Version: 1.0, página 2-6

Qualcomm Snapdragon 8 Gen 1



Procesador ARM usado en teléfonos celulares.

Cortex X2: 3 GHz, L2 1MB
Cortex A710: 2.5 GHz.
Cortex A510: 1.8 GHz
Caché L3 compartida de 6 MB.

Comandos útiles Linux:
lscpu
cat /proc/cpuinfo
powertop

Colección de productos	Intel® Core™ processors (Series 1)
Nombre de código	Products formerly Raptor Lake
Segmento vertical	Mobile
Número de procesador	100U

Especificaciones de la CPU

Cantidad de núcleos	6
Cantidad de Performance-cores	2
Cantidad de Efficient-cores	4
# of Low Power Efficient-cores	0
Total de subprocesos	8
Frecuencia turbo máxima	4.7 GHz
Frecuencia turbo máxima del Performance-core	4.7 GHz
Frecuencia turbo máxima de Efficient-core	3.3 GHz
Frecuencia base de Performance-core	1.2 GHz
Frecuencia base de Efficient-core	900 MHz

ıra en Ciencias de la
ıión

https://www.intel.la/content/www/xl/es/products/sku/236776/intel-core-3-processor-100u-10m-cache-up-to-4-70-ghz/specifications.html

Colección de productos	Intel® Core™ Ultra Series 3 processors
Nombre de código	Products formerly Panther Lake
Segmento vertical	Mobile
Número de procesador	378H

Especificaciones de la CPU

Cantidad de núcleos	16
Cantidad de Performance-cores	4
Cantidad de Efficient-cores	8
# of Low Power Efficient-cores	4
Total de subprocesos	16

https://www.intel.la/content/www/xl/es/products/sku/246128/intel-core-ultra-x9-processor-378h-18m-cache-up-to-5-00-ghz/specifications.html

Sistemas multicore heterogéneo con set de instrucciones equivalente

- **Monitoreo de carga de un núcleo**:
  - Tiempo de uso del núcleo vs tiempo inactivo.
  - Tamaño de las colas de tareas pendientes del núcleo.
- **Acciones para adaptar el consumo de energía**:
  - Aumentar o disminuir la **frecuencia** (y **voltaje**) del núcleo.
    - Intel le llama **Turbo Boost** y **SpeedStep**.
    - AMD le llama **Precision Boost** y **Cool'n'Quiet**.
    - ARM le llama **DynamIQ**.
  - Migrar tareas entre núcleos:
    - Intel le llama **Thread Director**.
    - ARM le llama **big.LITTLE**.
- Sistemas operativos: **C-states** (combinaciones de frecuencias y núcleos).

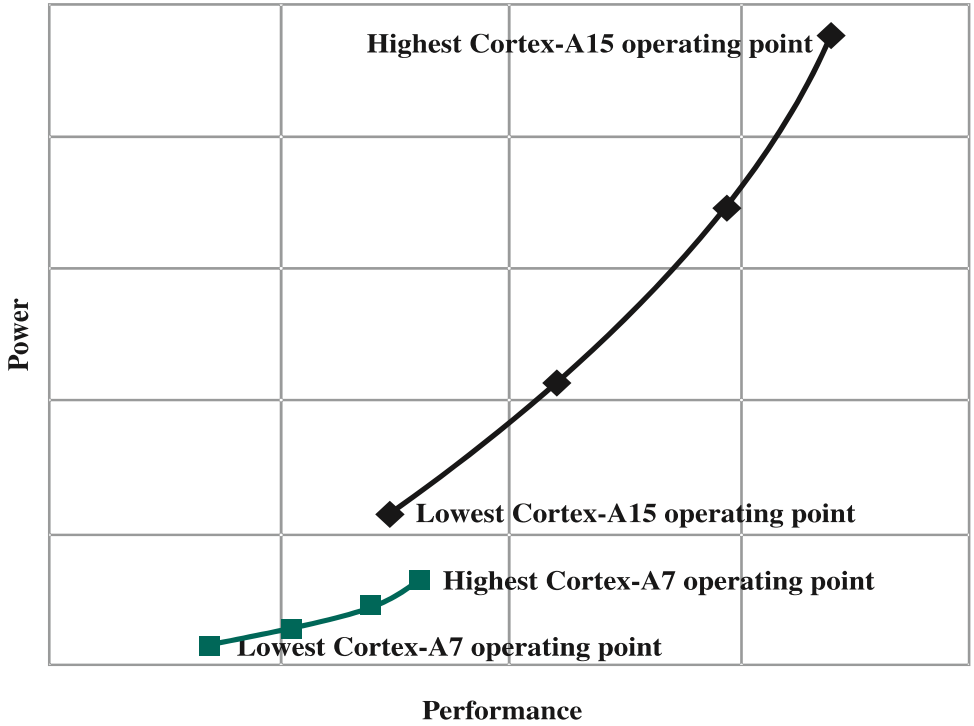
Frecuencia turbo máxima	4.8 GHz
Frecuencia de la Tecnología Intel® Turbo Boost Max 3.0 ‡	4.8 GHz
Frecuencia de la Tecnología Intel® Turbo Boost 2.0‡	4.6 GHz
Frecuencia básica del procesador	3.2 GHz
Caché	72 MB
Velocidad de Intel® UPI	0 GT/s
Cantidad de enlaces UPI	0
Velocidad del bus	0.000 GT/s
Potencia base del procesador	180 W
Potencia turbo máxima	216 W

Ejemplo hoja de datos de Intel Xeon 638

Dos tecnologías de Intel para ajustar la frecuencia (GHz) al poder de procesamiento necesario

https://www.intel.la/content/www/xl/es/products/sku/244334/intel-xeon-638-processor-72m-cache-3-20-ghz/specifications.html

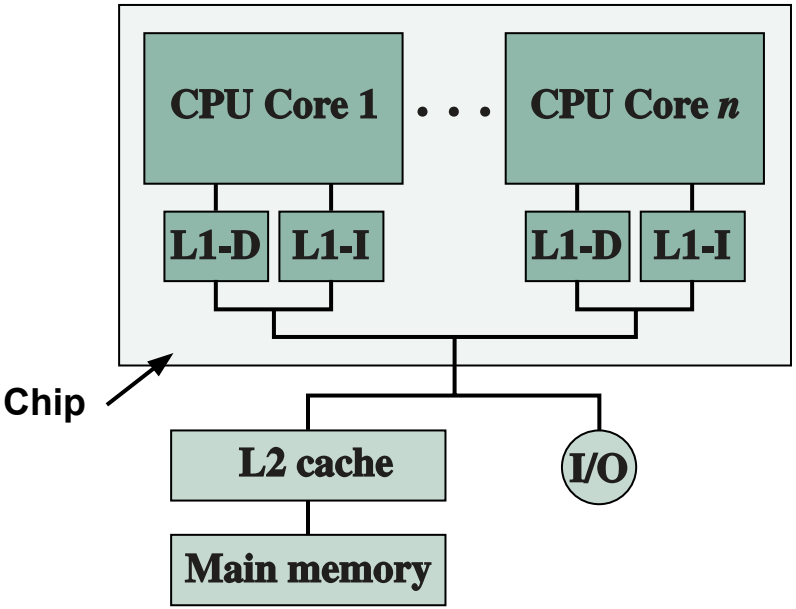
Sistemas multicore heterogéneo set de instrucciones equivalente



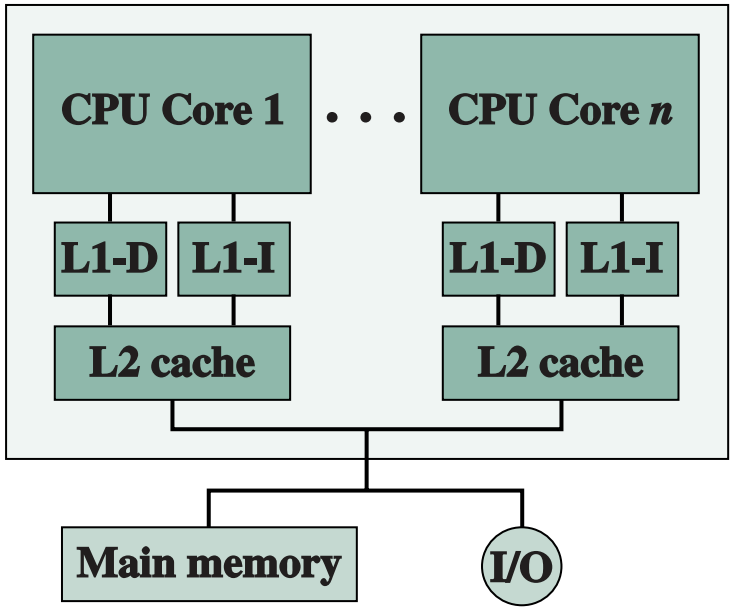
Comandos útiles Linux:
lscpu
cat /proc/cpuinfo
powertop

Figura obtenida de “Computer Organization and Architecture”, William Stallings, 10º edición, página 673

Sistemas multicore - diferentes arquitecturas de memoria caché



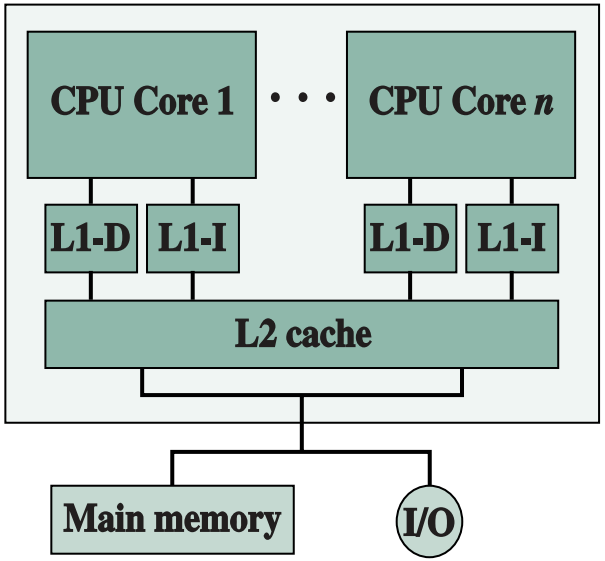
L1 dedicada on chip, L2 compartida (primeros procesadores. ARM11)



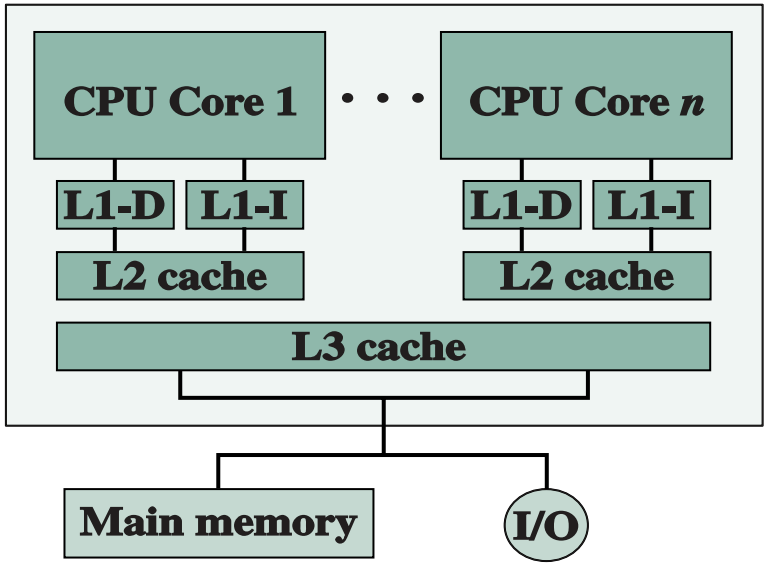
L1 y L2 dedicadas on chip (AMD Opteron)

Figura obtenida de “Computer Organization and Architecture”, William Stallings, 10º edición, página 666

Sistemas multicore - diferentes arquitecturas de memoria caché



L1 dedicada, L2 compartida on chip (Intel Core Duo)



L1 y L2 dedicada, L3 compartida on chip (Intel Core i7)

Figura obtenida de “Computer Organization and Architecture”, William Stallings, 10º edición, página 666

Sistemas multicore - diferentes arquitecturas de memoria caché

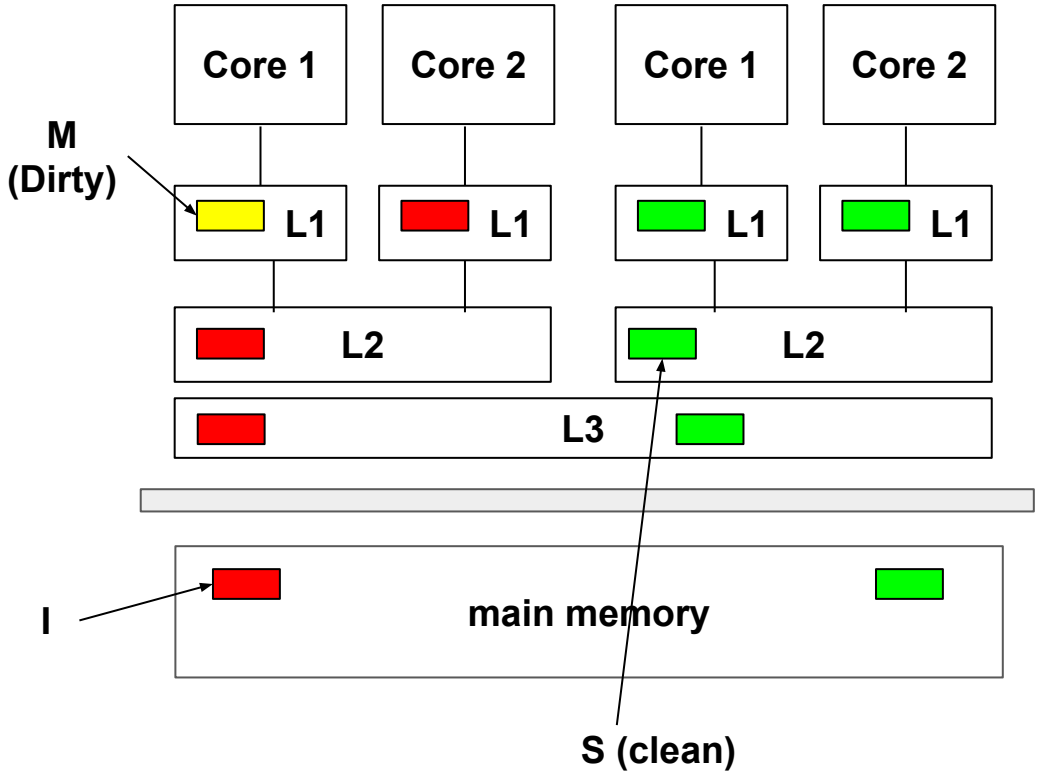
- Necesidad: incremento de la **diferencia de velocidad entre el procesador y la memoria principal** hace necesario más niveles de caché.

Beneficios de usar memoria caché compartida dentro del chip.

- Aplicaciones **multihilos** a los que el scheduler asigna **varios núcleos**: Disminuir los fallos de caché (accesos a datos que no están en caché)
- **Balance dinámico**: Si un núcleo usa poca caché, otros pueden usar mayor cantidad.
- **Confina** los **problemas de coherencia de caché** entre procesadores a cachés dedicadas. Esto disminuye los fallos de caché.

Protocolos de Coherencia de Caché

- Un dato puede estar en una o varias cachés y en memoria principal.
- **Dirty**: En dato ha sido modificado en una caché y no ha sido actualizado a memoria principal.
- **Clean**: El dato está actualizado en todas las cachés y la memoria principal.

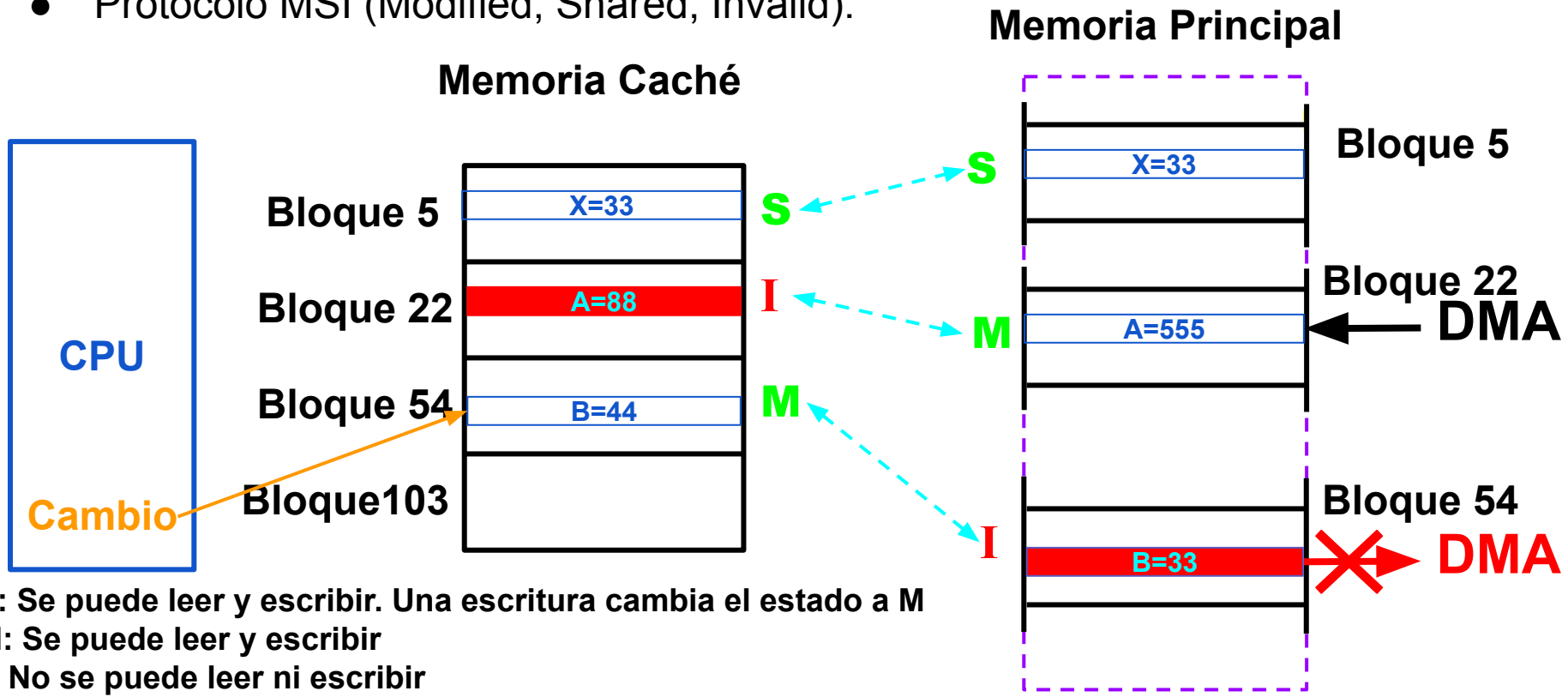


Protocolos de Coherencia de Caché

- Soluciones por hardware:
  - Protocolos **Snoopy** (“fisgón”):
    - Actúa de forma **distribuida** (todos los controladores de caché contribuyen a resolver el problema).
    - Si el controlador de caché **detecta un cambio** en el contenido su caché **genera señales por broadcast**.
    - Todos los controladores **escuchan esas señales todo el tiempo**.
    - Dos tipos:
  - Protocolos **directorio**:
    - Utiliza mecanismo **centralizado**

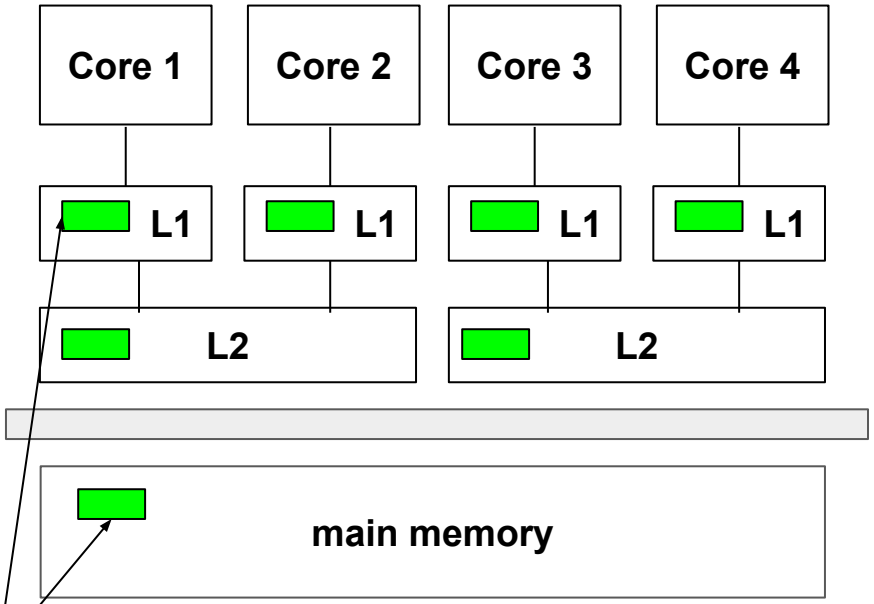
Algoritmos de coherencia de memoria caché

- Protocolo MSI (Modified, Shared, Invalid).

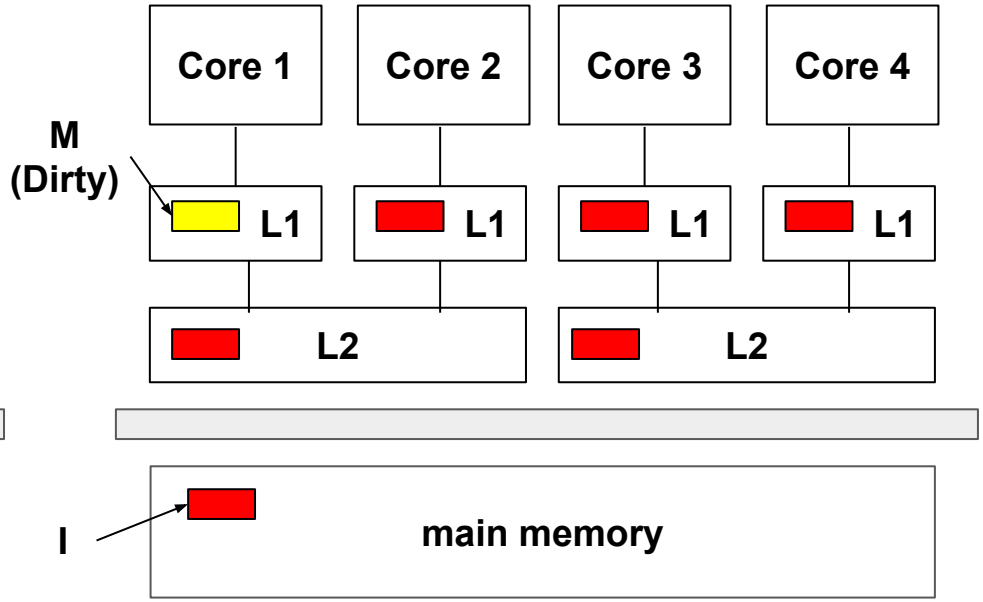


S: Se puede leer y escribir. Una escritura cambia el estado a M
M: Se puede leer y escribir
I: No se puede leer ni escribir

Sistemas multicore - Coherencia memoria caché

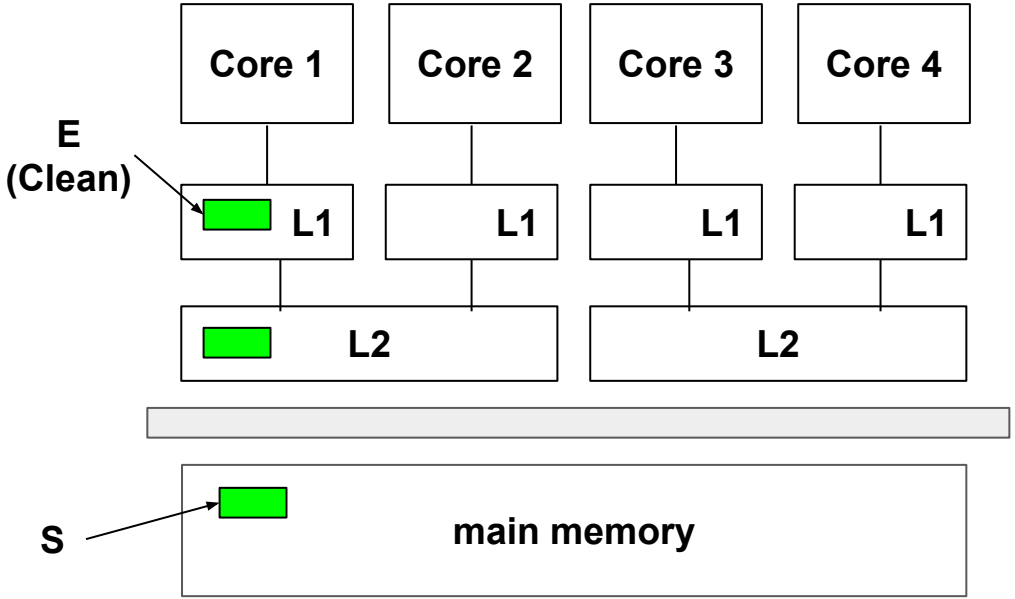


Cualquier Core puede modificar el dato, pero **debe enviar señales de invalidación** a los demás.



El Core 1 puede modificar el dato sin enviar señales. Cualquier otro core, debe enviar una petición de actualización.

Sistemas multicore - Coherencia memoria caché: Protocolo MESI

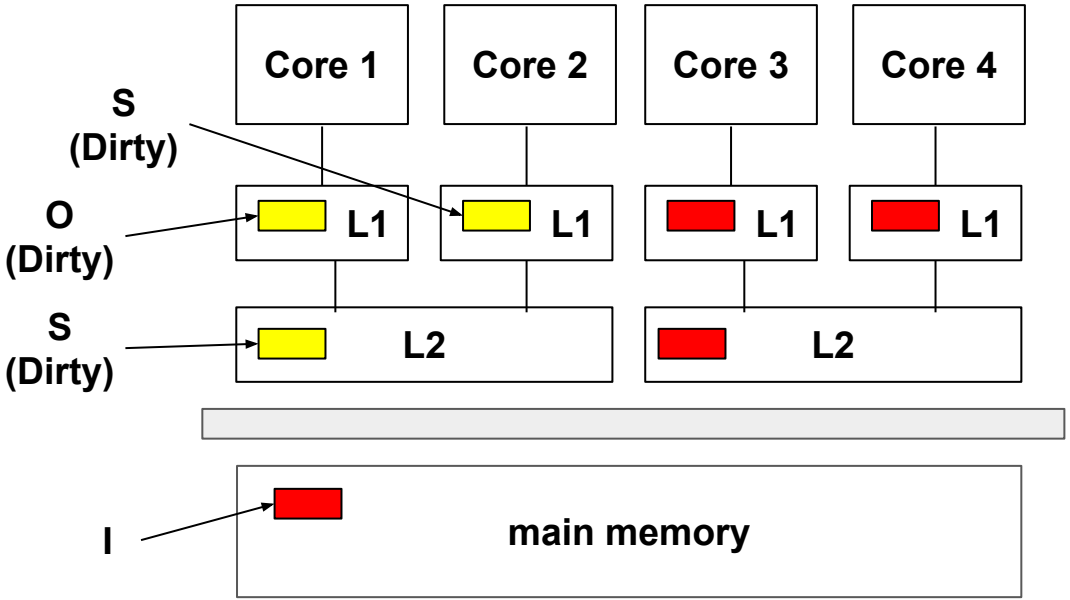


El Core 1 puede modificar el dato sin enviar señales, pero debe cambiar el estado a M.

Si el dato está solo en la caché del core 1:

- MSI: Estará en estado S. Cuando el Core 1 lo modifique, enviará señales de invalidación a la memoria principal y también enviará innecesariamente señales de invalidación a los demás núcleos.
- MESI: Estará en estado E. Cuando el Core 1 lo modifique, enviará señales de invalidación solo a la memoria principal.

Sistemas multicore - Coherencia memoria caché: Protocolo MOESI



Si el núcleo 3 o 4 envía una petición de actualización:

- MESI: Los núcleos 1 y 2 responderán.
- MOESI:
  - Solo el núcleo 1 responderá.
  - El dato va de L1 a L1.

Sistemas multicore - Coherencia memoria caché

- En arquitecturas heterogéneas, los procesadores se agrupan **conjuntos**, y cada conjunto puede tener caché compartida, pero no compartida con otros grupos.
  - Arquitectura big.Little, los procesadores más potentes pueden tener su L2 compartida, y los menos potentes otra L2 compartida.
  - CPU/GPU, los CPUs pueden tener una L3 compartida, y los GPU otra L3 compartida.
- Protocolo **MOESI**, se agrega otro estado: **Owned**
  - Un dato está en varias cachés y no ha sido actualizado a la memoria principal (Dirty).
  - Solo una de esas cachés tiene permiso de actualizar el dato a la memoria principal ante una señal miss (ese procesador está en estado Owned, los demás en shared).

Mainframe

- Computadora completa de alta confiabilidad.
- Objetivo: **Cero fallos** (disponibilidad permanente).
- Optimizado para gran cantidad de operaciones (millones por segundo) **confiables**.
- Aplicaciones: Sistemas bancarios, pagos, etc.
- Arquitectura: Gran cantidad de núcleos, memoria altamente coherente, sistema de I/O especializado.

Ejemplo: mainframe IBM zEnterprise EC12

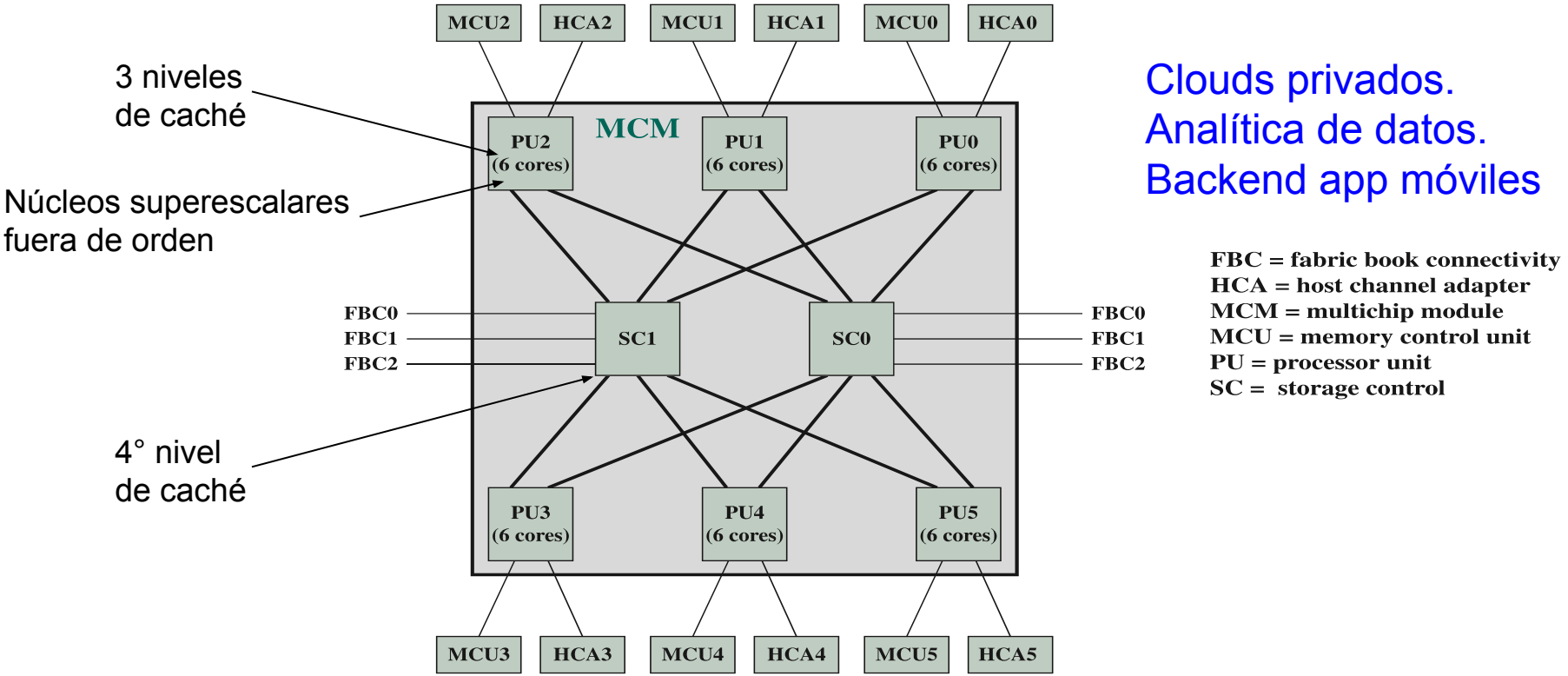


Figura obtenida de “Computer Organization and Architecture”, William Stallings, 10º edición, página 682

Ejemplo: mainframe IBM zEnterprise EC12

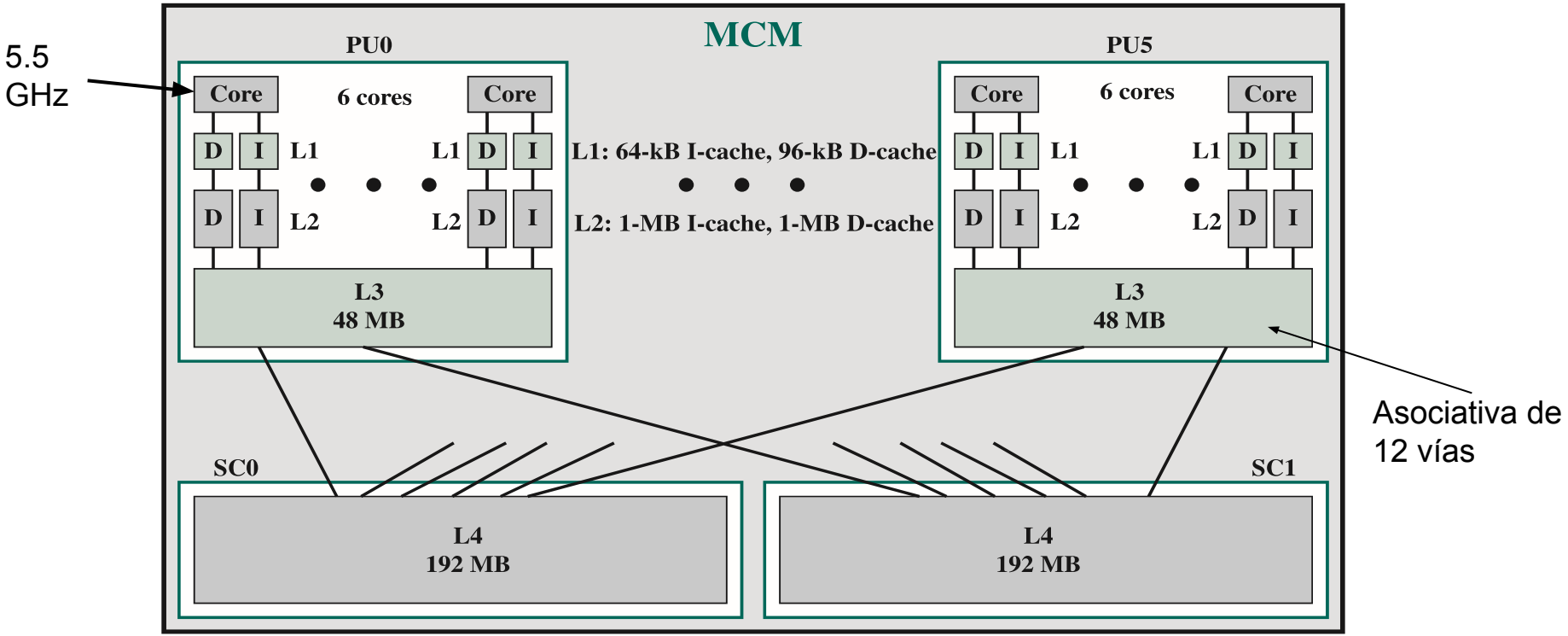


Figura obtenida de “Computer Organization and Architecture”, William Stallings, 10º edición, página 682

Multicore heterogéneo con diferentes set de instrucciones - GPU y GPGPU

- GPU: Unidad de procesamiento gráfico.
  - **Gran paralelismo a nivel de hilos**. Operaciones simples en paralelo.
  - Inicialmente pensadas para computación gráfica.
    - Renderización: Generar imagen (o video) a partir de datos (provenientes de modelos).

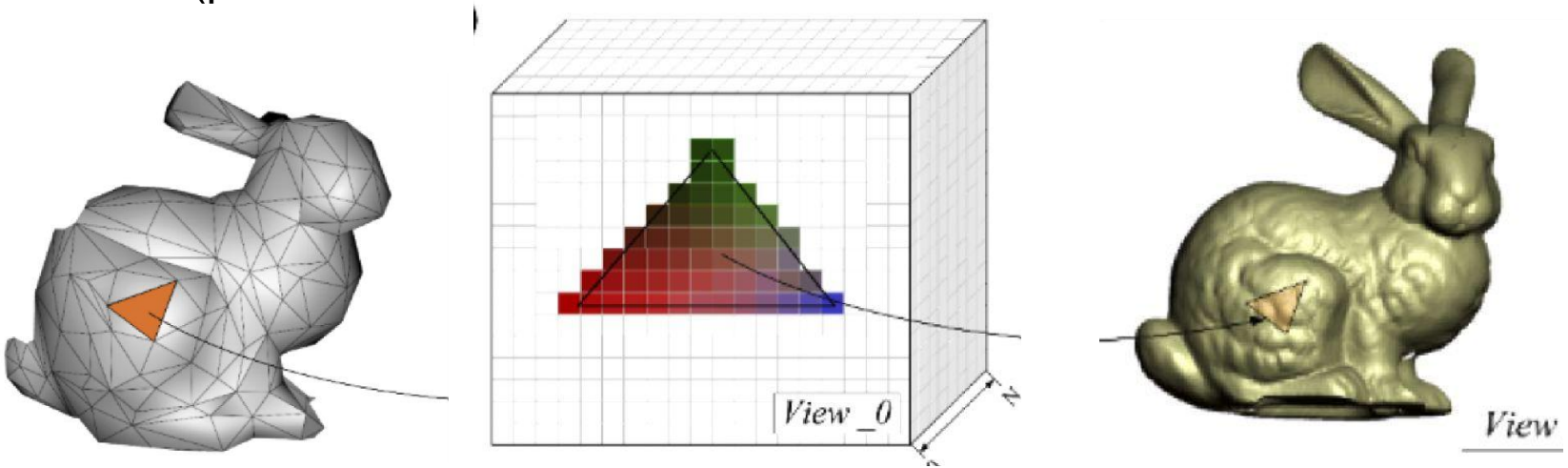


Imagen obtenida de https://opg.optica.org/oe/fulltext.cfm?uri=oe-28-23-34406&id=442048

Renderización:

- Utilizada en **animación por computadora**, los **videojuegos**, **diseño asistido por computadora** (arquitectura, piezas mecánicas, etc.), **realidad virtual**, diseño gráfico, etc.
- Convertir modelos o imágenes 3D en imágenes 2D.
  - Entrada: conjunto de polígonos 3D, texturas, fuentes de luz y posición de la cámara.
  - Salida: imagen 2D.

Varias técnicas:

- Rasterización: Proyectar vértices de los polígonos en una superficie 2D.
- Trazado de rayos (ray tracing): trazar rayos de luz y calcular cómo se reflejarán o refractaria.

- **GPU**: Actualmente poseen **miles de núcleos**
  - Cada uno con **pocas unidades de ejecución específicas** (operaciones enteras y/o con punto flotante) y gran cantidad de registros.
- **GPGPU**: (General Purpose GPU) GPU para procesamiento general.
  - Se aprovecha la elevada capacidad de procesamiento paralelo de las GPU.
  - Implementaciones:
    - GPU como co-procesadores.
    - Máquinas con GPU como procesador central + un CPU para tareas auxiliares (clúster).
  - Aplicaciones:
    - Simulaciones en física
    - Procesamiento de señales, etc.

Multicore heterogéneo - CPU/GPU

- Comparación GPU - CPU (AMD A10 5800K)

	CPU	GPU
Clock frequency (GHz)	3.8	0.8
Cores	4	384
FLOPS/core	8	2
GFLOPS	121.6	614.4

Tabla obtenida de “Computer Organization and Architecture”, William Stallings, 10º edición, página 669

- NVIDIA Tesla V100

	CPU	GPU
Núcleos	24	2688
Clock (GHz)	2.7	1.38
GFlops	1037	7419

	GeForce RTX 3090 Ti	GeForce RTX 3090	GeForce RTX 3080 Ti	GeForce RTX 3080	GeForce RTX 3070 Ti	GeForce RTX 3070
Núcleos CUDA de NVIDIA	10752	10496	10240	8960 / 8704	6144	5888
Frecuencia Modificada (GHz)	1.86	1.70	1.67	1.71	1.77	1.73
Tamaño de la Memoria	24 GB	24 GB	12 GB	12 GB / 10 GB	8 GB	8 GB
Tipo de Memoria	GDDR6X	GDDR6X	GDDR6X	GDDR6X	GDDR6X	GDDR6

GDDR (Graphics Double Data Rate): **mayor cantidad de canales de acceso**, **mayor ancho de bus**, **mayor ancho de banda** (bps), mayor latencia.

Fuente: https://www.nvidia.com/

	CPU	GPU
Unidades de punto flotante	Algunas.	Miles.
Unidades de salto	Varias por núcleo. Complejas (predicción de salto, optimización de bifurcaciones, etc.).	Pocas. Simples. Externas a los núcleos.
Acceso y gestión de memoria	Compleja (paginación y segmentación).	Muy simple.
Encriptación	Muchas (RSA, AES, SHA, etc.)	No poseen.
Manejo de cadenas de caracteres	Muchas y complejas	Poco comunes.

Multicore - CPU/GPU

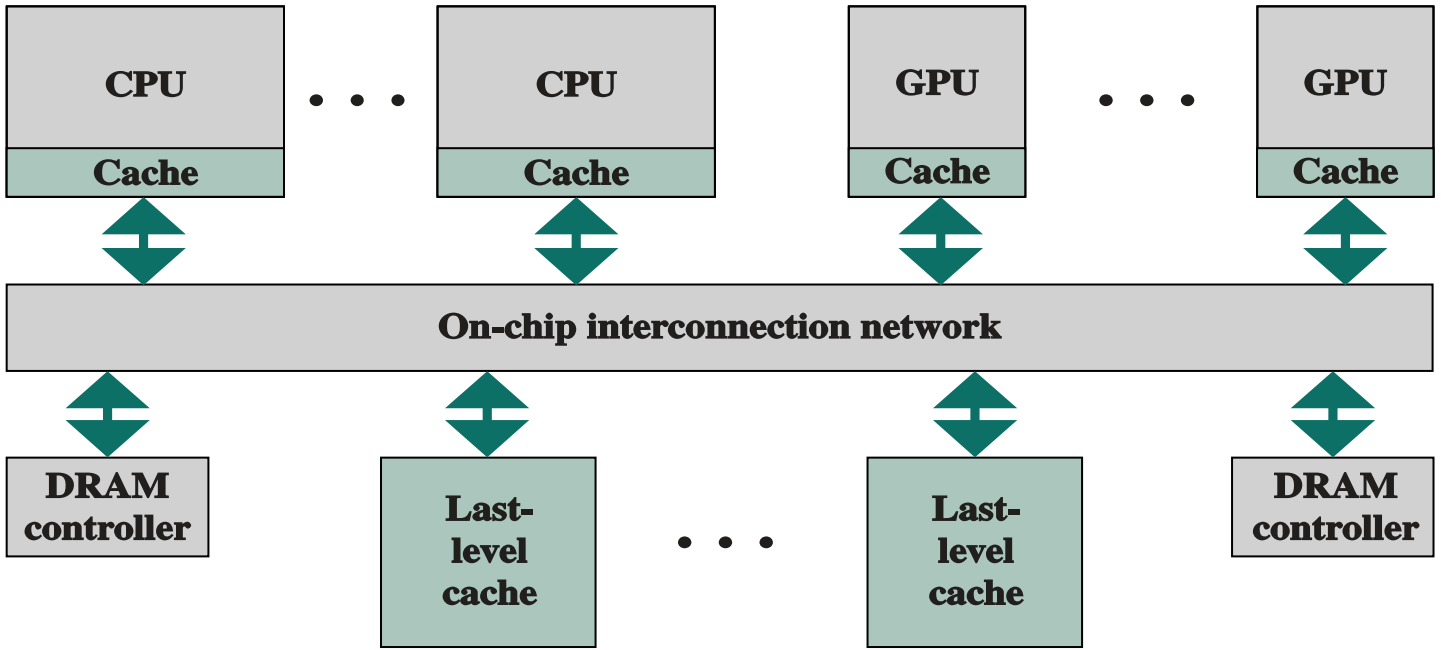


Figura obtenida de “Computer Organization and Architecture”, William Stallings, 10º edición, página 668

CPU vs GPU

- CPU: Mucha más caché y lógica de control.
- GPU: Mucha más área dedicada a unidades aritméticas

Lógica de control: fuera de orden, predicción de saltos, dependencia de datos, etc.

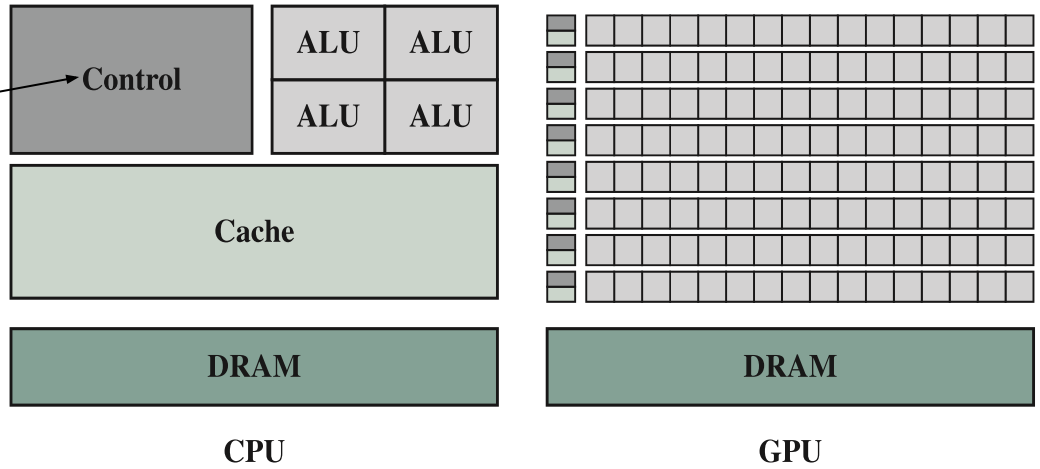
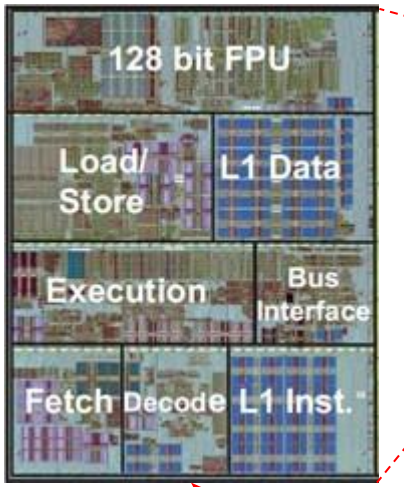
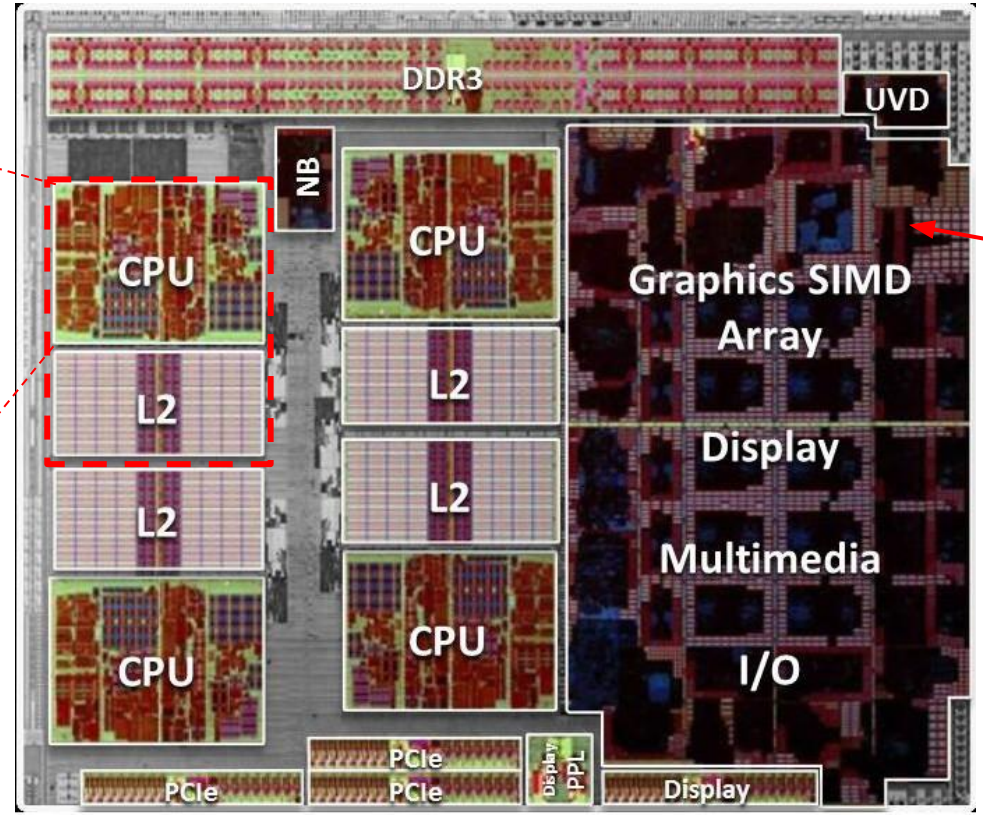


Figura obtenida de “Computer Organization and Architecture”, William Stallings, 10º edición, página 692

Ejemplo de procesador AMD: Llano APU



CPU microarquitectura k10 (multi-hilo, superescalar fuera de orden)



GPU Terascale

Ejemplo arquitectura GPU (NVIDIA Fermi architecture)

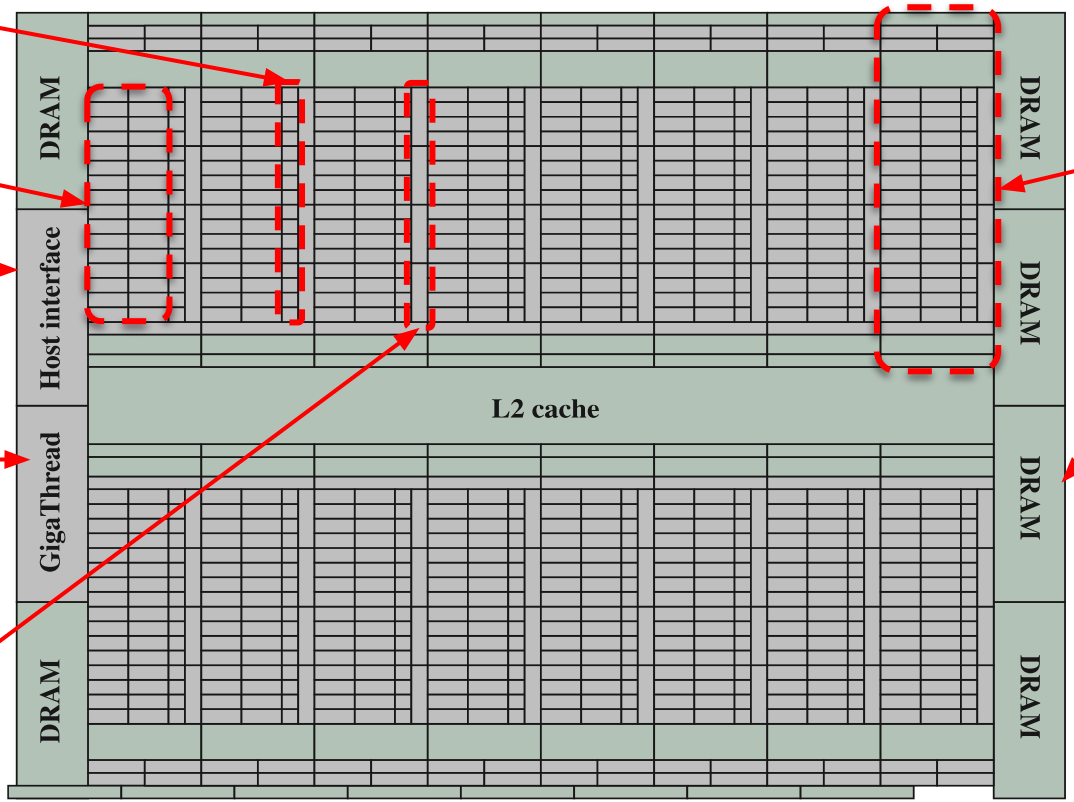


Figura obtenida de “Computer Organization and Architecture”, William Stallings, 10º edición, página 694

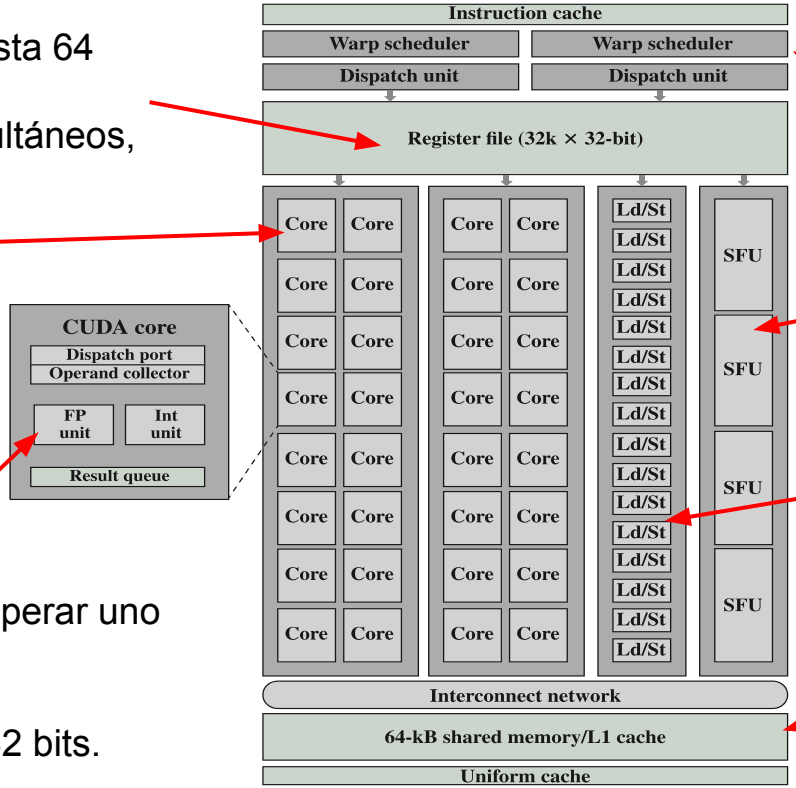
Ejemplo arquitectura GPU (NVIDIA Fermi architecture)

- 512 núcleos.
  - 32 núcleos (streaming processors) por streaming Multiprocessors, 16 multiprocesadores de flujos.
- Los hilos se dividen en:
  - **Bloques**: El GigaThread (scheduler global de hilos) asigna los bloques de hilos a los streaming processors.
  - **Warp**: bloque de 32 hilos que comienzan en la misma dirección y poseen thread IDs consecutivos.
    - Los “dual warp scheduler” dividen los bloques de hilos se dividen en wraps.
    - Cada hilo posee su contador de programa y registros asignados.

GPUs arquitectura: Multiprocesador de flujos (NVIDIA Fermi architecture)

A cada hilo se le asignan hasta 64 registros dedicados (no compartidos). 48 warps simultáneos, cada uno con 32 hilos.

Cada core ejecuta un hilo



Divide los grupos de hilos enviados por el scheduler global de hilos (GigaThread) en waps (grupo de 32 hilos)

Unidades de funciones especiales.

Calcula direcciones en memoria principal (distintos modos de direccionamiento).

Compartida por todos los hilos (peligro de dependencias RAW)

2 pipelines. Solo puede operar uno a la vez.
Int Unit, 32 o 64 bits.
FP (punto flotante) unit, 32 bits.

Figura obtenida de “Computer Organization and Architecture”, William Stallings, 10º edición, página 682

GPUs arquitectura: scheduler y dispatch (NVIDIA Fermi architecture)

Unidades de procesamiento:

- Cores: Pueden realizar una operación con enteros (32 o 64 bits) o con operandos en punto flotante de simple precisión (32 bits) en un clock del reloj.
- Unidades de funciones especiales: realizan operaciones trascendentales (seno, coseno, raíz cuadrada, etc.) en un ciclo de reloj.
- Load store unit: acceden a memoria (utilizan los métodos de direccionamiento conocidos).

GPUs arquitectura: (NVIDIA Fermi) organización de memoria.

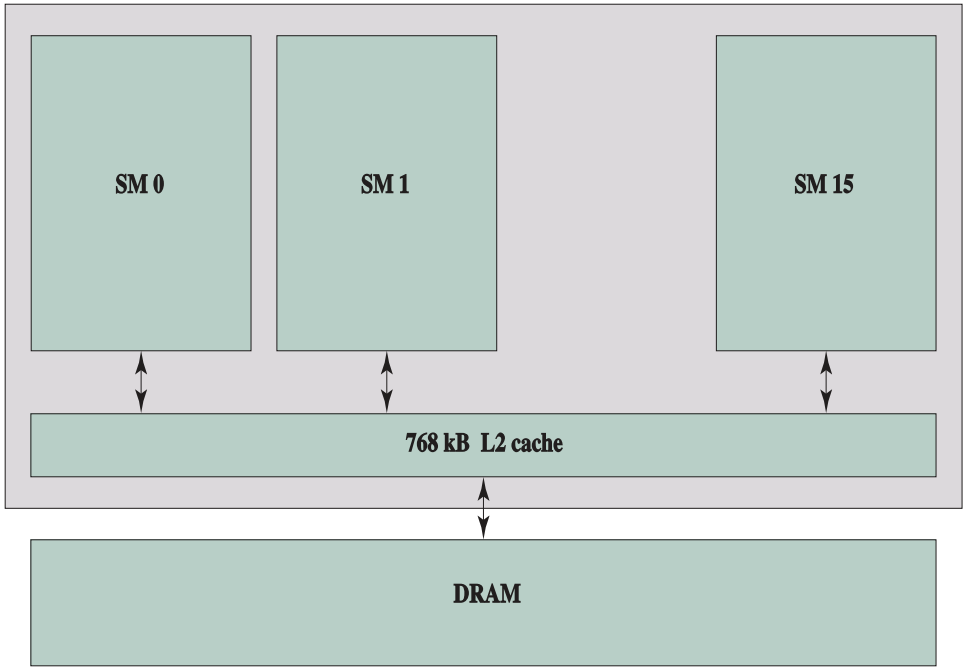
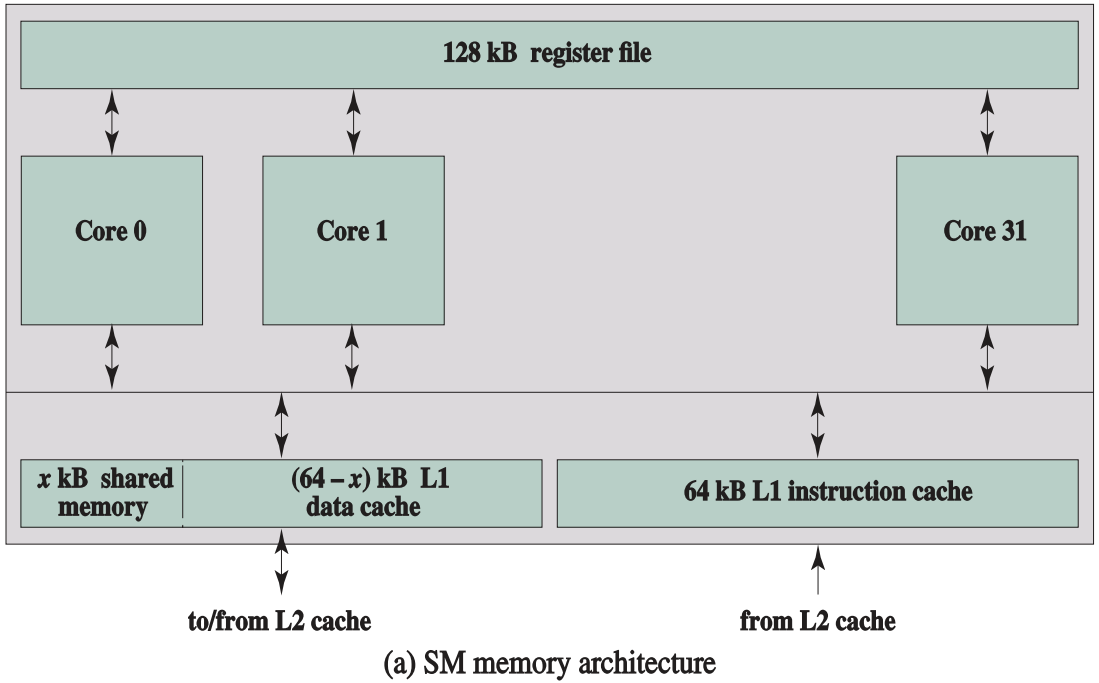


Figura obtenida de “Computer Organization and Architecture”, William Stallings, 10º edición, página 699

GPUs arquitectura: (NVIDIA Fermi) organización de memoria.

Memoria de un multiprocesador de flujos



(a) SM memory architecture

- A cada hilo se le asigna un conjunto de registros.
- Shared memory: Memoria compartida por los hilos. Facilita la comunicación entre hilos.

Figura obtenida de “Computer Organization and Architecture”, William Stallings, 10º edición, página 699

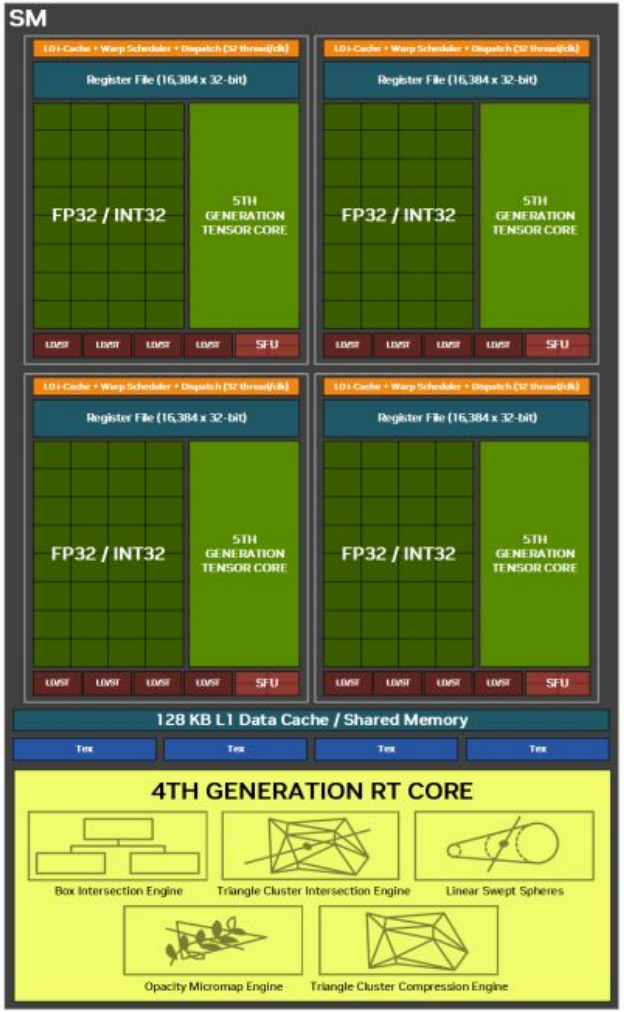
UNCUYO
UNIVERSIDAD NACIONAL DE CUYO

FACULTAD DE INGENIERÍA

Licenciatura en Ciencias de la Computación

Ejemplo: Arquitectura Nvidia Blackwell

Fuente: https://images.nvidia.com/aem-dam/Solutions/geforce/blackwell/nvidia-rtx-blackwell-gpu-architecture.pdf



ACULTAD
E INGENIERÍA

Licenciatura en Ciencias de la Computación

Arquitectura Nvidia Blackwell

- Lanzada al mercado en 2024 (última arquitectura NVidia al 2026).
- 128 CUDA Cores.
- 4 Tensor Cores. Realizan operaciones matriciales (muy utilizadas en deep learning)
- 1 RT core (Ray Tracing core). Complementan a los Cuda Cores

Fuente:
https://images.nvidia.com/aem-dam/Solutions/geforce/blackwell/nvidia-rtx-blackwell-gpu-architecture.pdf

Graphics Card	GeForce RTX 3090	GeForce RTX 4090	GeForce RTX 5090
GPU Codename	GA102	AD102	GB202
GPU Architecture	NVIDIA Ampere	NVIDIA Ada Lovelace	NVIDIA Blackwell
GPCs	7	11	11
TPCs	41	64	85
SMs	82	128	170
CUDA Cores / SM	128	128	128
CUDA Cores / GPU	10496	16384	21760
Tensor Cores / SM	4 (3rd Gen)	4 (4th Gen)	4 (5th Gen)
Tensor Cores / GPU	328 (3rd Gen)	512 (4th Gen)	680 (5th Gen)
GPU Boost Clock (MHz)	1695	2520	2407
RT Cores	82 (2nd Gen)	128 (3rd Gen)	170 (4th Gen)

GPUs comerciales Nvidia con Arquitecturaa Nvidia Blackwell

RTX 5090

100 TFLOPs
575 W

Tarjeta De Vídeo Inno3d Geforce Rtx 5090 X3 32gb

$7.239.479

Mismo precio en 12 cuotas de $ 603.289[92]

Precio sin impuestos nacionales: $ 6.551.565

Combiná medios de pago
Pagá con tarjetas de crédito, dinero disponible y más en la misma compra.

Ver los medios de pago

GPU Intel GEN 8: Unidad de ejecución

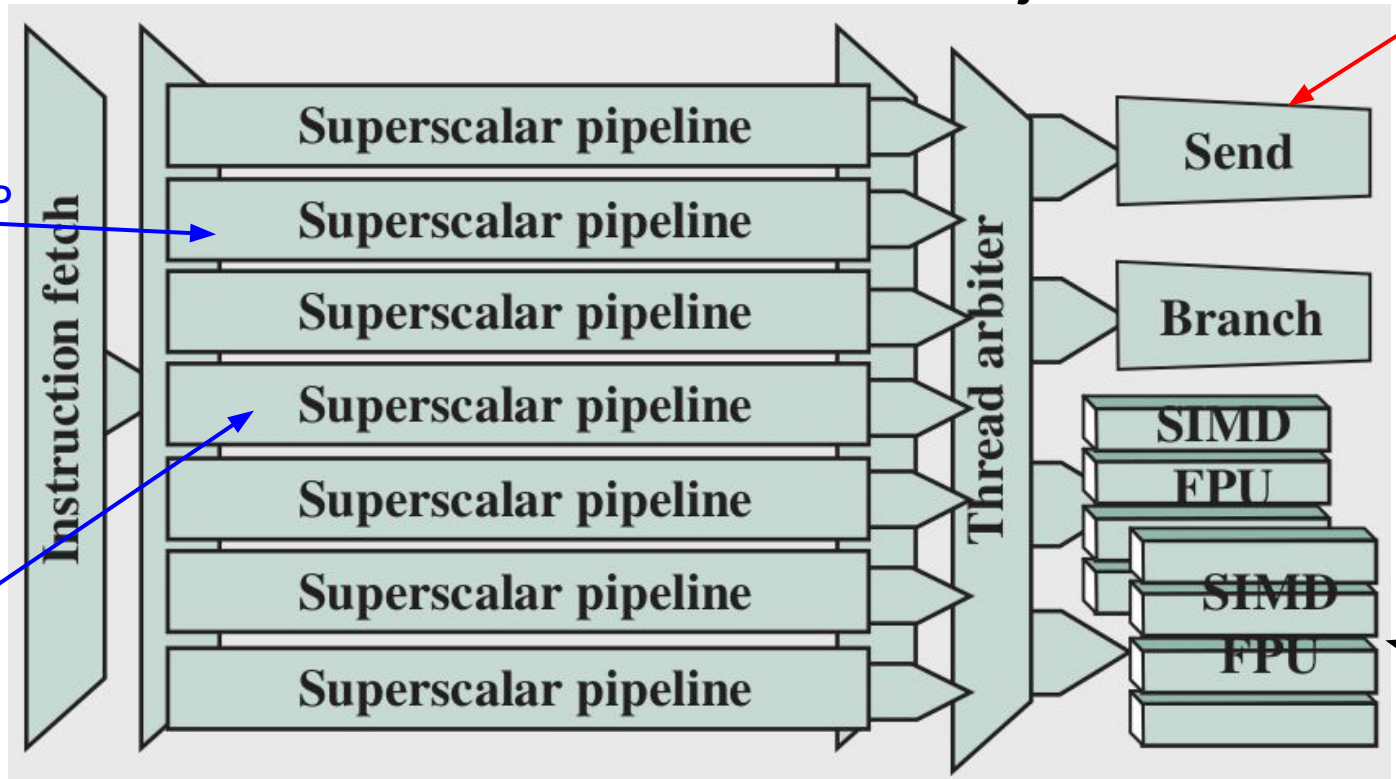


Figura obtenida de “Computer Organization and Architecture”, William Stallings, 10º edición, página 701

GPU Intel GEN 8: Unidad de ejecución

- **Multithreading simultáneo** de 7 hilos.
- **128 registros por hilo** de 32 bytes.
  - Cada registro almacena 8 datos de 32 bits.
  - Pueden agruparse para almacenar datos de mayor longitud.
- SIMD FPU (floating-point units): operaciones SIMD con flotantes o enteros.
  - Cada EU Puede ejecutar operaciones SIMD sobre 16 datos.
- 4 operaciones al mismo tiempo (2 SIMD FPU, 1 Send y 1 Branch).
- Procesadores Intel i3, i5, i7, core M, Atom (moviles), Xeon (servidores), etc.

GPU Intel GEN 8: Subslice

- 8 unidades de ejecución por subslice (56 hilos).
- Sampler: operaciones sobre imágenes (formatos, etc.)

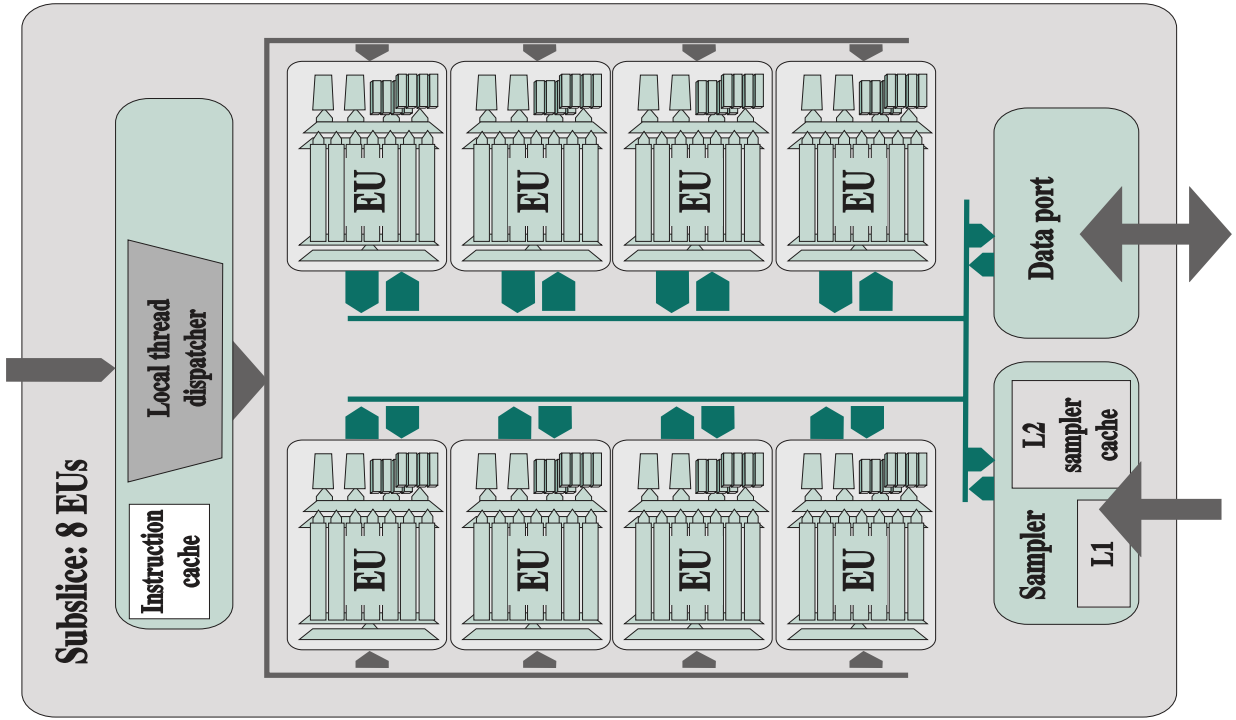


Figura obtenida de “Computer Organization and Architecture”, William Stallings, 10º edición, página 702

GPU Intel GEN 8: Slice

- 3 subslices por slice.
- L3 dividida en bancos para permitir varios accesos simultáneos (1 por banco).
- Un GPU GEN 8 puede incluir 1 o varios slices.
- Shared local memory: memoria compartida. Permite a las UE compartir variables temporales.

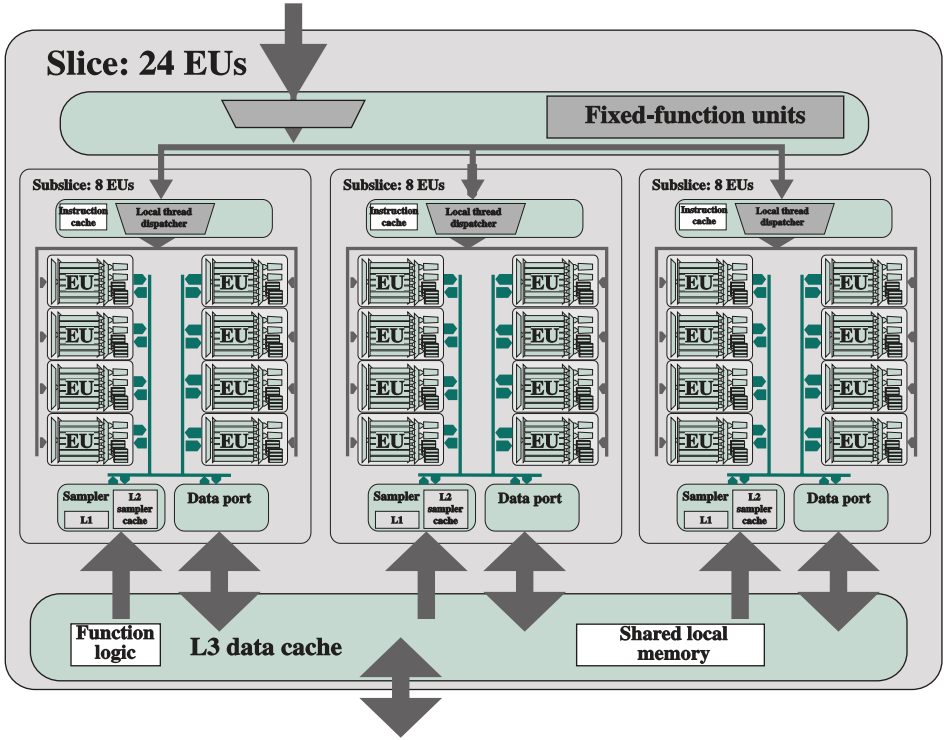


Figura obtenida de “Computer Organization and Architecture”, William Stallings, 10º edición, página 702

GPU Intel GEN 8: Ejemplo Procesador Intel Core M

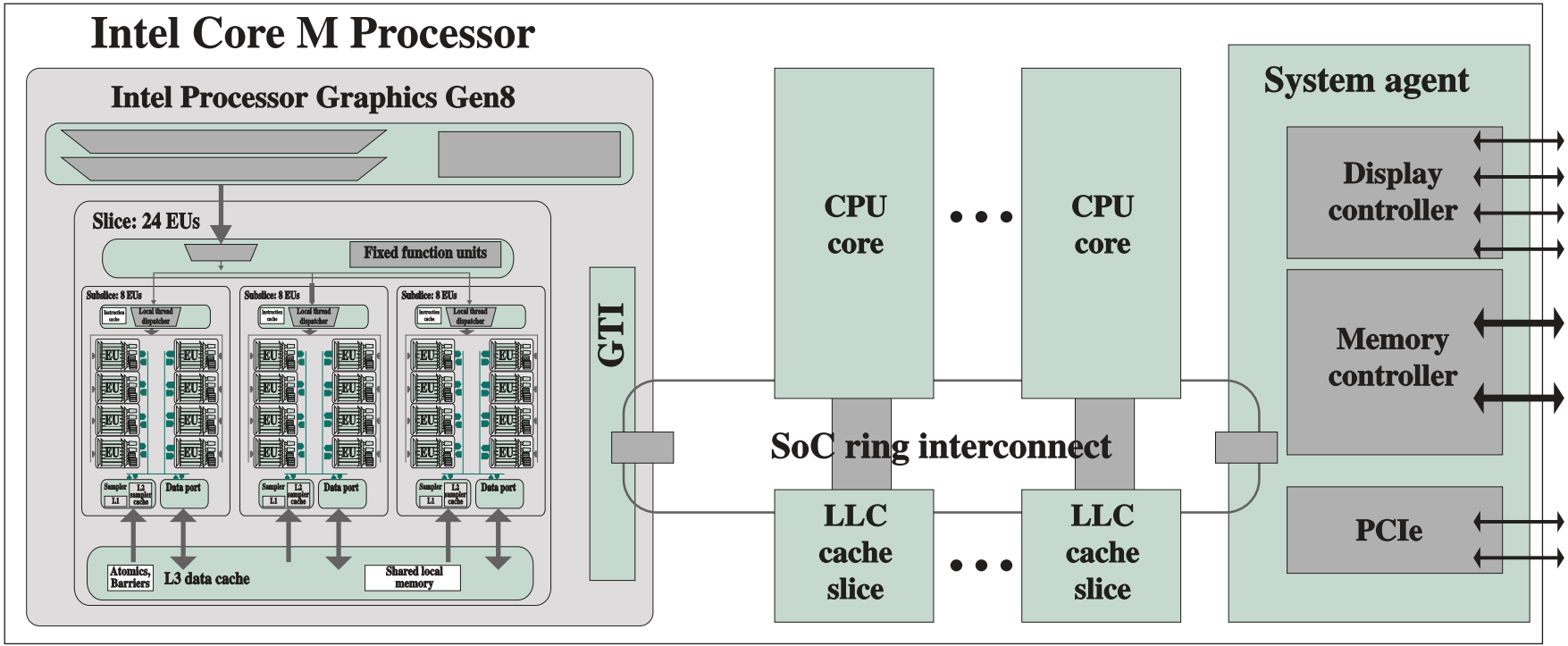


Figura obtenida de “Computer Organization and Architecture”, William Stallings, 10º edición, página 705

GPU Intel Gen 8.
GPU del procesador Intel Core i7-11370

Processor Graphics

Processor Graphics ‡	Intel® Iris® Xe Graphics
Graphics Max Dynamic Frequency	1.35 GHz
Graphics Output	eDP 1.4b, MIPI-DSI 2.0, DP 1.4, HDMI 2.0b
Execution Units	96
Max Resolution (HDMI)‡	4096x2304@60Hz
Max Resolution (DP)‡	7680x4320@60Hz
Max Resolution (eDP - Integrated Flat Panel)‡	4096x2304@60Hz
DirectX* Support	12.1
OpenGL* Support	4.6
OpenCL* Support	3.0

672 hilos

Información obtenida de:
https://ark.intel.com/content/www/us/en/ark/products/196655/intel-core-i711370h-processor-12m-cache-up-to-4-80-ghz-with-ipu.html

Herramientas de desarrollo

- CUDA: creado por Nvidia (2007).
  - Funciona desde la serie de GPUs Nvidia G8x (2006) en adelante (productos compatibles: https://developer.nvidia.com/cuda-gpus).
- OpenCL (Open Computing Language): Lenguaje libre. Pensado para ser utilizado con todas las GPU (AMD, Intel, Nvidia) y también CPU.
  - Creado por Apple. Mantenido por Grupo Khronos.
- Otros ejemplos:

https://armkeil.blob.core.windows.net/developer/Files/pdf/product-brief/arm-gpu-processor-comparison-table.pdf

Conceptos básicos de CUDA

- CUDA (Compute Unified Device Architecture): Modelo de programación creado por NVidia,
  - Lenguaje de programación basado en C++/C.
    - Wrappers para otros lenguajes como Java, Python, etc.
  - Compilador.
  - Herramientas de desarrollo.

Multicore - CPU/DSP

DSP (Digital Signal Processors)

- Procesadores que permiten procesamiento rápido de **señales digitalizadas inicialmente analógicas** (voz, video, etc.).
  - Datos que llegan continuamente donde el tiempo es importante (audio y video en tiempo real).
  - Filtrado digital (eliminar o reducir ruido, reconocimiento de voz, etc).
  - Extracción y procesamiento de datos y de las señales de datos 4G y 5G.
  - IA relacionado con audio y video.
- Operaciones típicas:
  - Desplazamiento y suma, multiplicación y suma, transformada rápida de Fourier, comparación, multiplicación de matrices, etc.
    - Unidad de performance: **GMACS** (Giga Multiply-Accumulate operations per second)
- CPU/DSP: teléfonos celulares, placas de sonido, cámaras digitales, modems, etc.

Ejemplo: Snapdragon780G

Lanzamiento:
765: 3/2020
768: 12/2020
780: 3/2021

Qualcomm Snapdragon Premium SoCs			
SoC	**Snapdragon 765 Snapdragon 765G**	**Snapdragon 768G**	**Snapdragon 780G**
CPU	1x Cortex-A76 @ 2.3GHz (non-G) @ 2.4GHz (765G) 1x Cortex-A76 @ 2.2GHz 6x Cortex-A55 @ 1.8GHz	1x Cortex-A76 @ 2.8GHz 1x Cortex-A76 @ 2.4GHz 6x Cortex-A55 @ 1.8GHz	**1x Cortex-A78** @ 2.4GHz **3x Cortex-A78** @ 2.2GHz 4x Cortex-A55 @ 1.9GHz
GPU	Adreno 620	Adreno 620 *+15% perf over 765G*	Adreno 642 *+50% perf over 768G*
DSP / NPU	Hexagon 696 HVX + Tensor *5.4TOPS AI* *(Total CPU+GPU+HVX+Tensor)*		**Hexagon 770** Scalar+Tensor+Vector *12TOPs AI* *(Total CPU+GPU+DSP)*
Memory Controller	2x 16-bit CH @ 2133MHz LPDDR4X / 17.0GB/s		

Neural Processing Unit

TOPS: Tera Operations Per Second

Encode/ Decode	2160p30, 1080p120 H.264 & H.265 10-bit HDR pipelines	
Integrated Modem	Snapdragon X52 Integrated (LTE Category 24/22) DL = 1200 Mbps 4x20MHz CA, 256-QAM UL = 210 Mbps 2x20MHz CA, 256-QAM (5G NR Sub-6 4x4 100MHz + mmWave 2x2 400MHz) DL = 3700 Mbps UL = 1600 Mbps	**Snapdragon X53** **Integrated** (LTE Category 24/22) DL = 1200 Mbps 4x20MHz CA, 256-QAM UL = 210 Mbps 2x20MHz CA, 256-QAM (5G NR Sub-6 4x4 100MHz) DL = 3300 Mbps UL = ? Mbps
Mfc. Process	Samsung **7nm (7LPP)**	Samsung **5nm (5LPE)**

Cortex A-55: Superescalar de dos vías en orden (sucesor del Cortex A-53)
Cortex A-76: Superescalar de cuatro vías fuera de orden
Cortex A-78: Superescalar de cuatro vías fuera de orden
Fuente: Twitter de Snapdragon

Sistemas Operativos para procesadores multinúcleos

Evolución:

- SO master-slave.
  - Un núcleo ejecuta el SO y el resto de núcleos procesos de usuario.
  - Problema: el núcleo master es un cuello de botella (debe manejar llamadas al sistema y scheduler).
  - Primeros procesadores y sistemas embebidos simples.
- Sistema operativo SMP (Symmetric Multiprocessing):
  - El sistema operativo puede ejecutarse en cualquier núcleo.

La sigla SMP se refiere a diferentes conceptos cuando se habla de:

- **Procesadores SMP: Procesadores multinúcleos simétricos.**
- **Sistemas Operativos SMP: Los componentes del SO pueden ejecutarse en cualquier núcleo.**
  - **Un Sistema Operativo SMP puede ejecutarse en un procesador heterogéneo con igual set de instruciones.**

Sistemas Operativos para procesadores multinúcleos

Sistemas operativos actuales:

- Sistemas operativos **divididos en bloques** (scheduler, sistema de archivos, gestión de memoria, pila TCP/IP, etc.).
- SO dividido en partes (bloques) independientes. Cada bloque es ejecutado en un núcleo diferente (por ejemplo, 1- scheduler, 2 - gestión del sistema de archivos, 3 - manejo de páginas, pila TCP/IP, etc.).
  - **Difícil de diseñar (deadlock)**
- Cualquier núcleo puede ejecutar código del SO (kernel o de usuario).
  - Varias partes del kernel pueden **ejecutarse en paralelo** en varios núcleos, pero usando mecanismos de sincronismo (bloqueos).

a

```
top - 17:57:44 up  6:30,  1 user,  load average: 0,54, 0,83, 0,84
Tareas: 253 total,   1 ejecutar,  252 hibernar,    0 detener,    0 zombie
%Cpu(s):  1,8 usuario,  0,7 sist,  0,0 adecuado, 96,7 inact,  0,8 en espera,  0,0 hardw int,
MiB Mem :   7838,3 total,   1757,2 libre,   2753,3 usado,   3327,8 búfer/caché
MiB Intercambio:  19073,0 total,  19071,2 libre,      1,8 usado.   4357,9 dispon Mem
```

PID	USUARIO	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	HORA+	ORDEN	P
285	root	19	-1	298496	211316	209532	S	0,7	2,6	2:15.65	systemd-journal	2
187	root	0	-20	0	0	0	I	0,3	0,0	0:01.10	kworker/2:1H-kblo+	2
14239	root	20	0	0	0	0	I	0,3	0,0	0:01.17	kworker/1:1-events	1
1	root	20	0	169872	12936	8312	S	0,0	0,2	0:02.58	systemd	1
2	root	20	0	0	0	0	S	0,0	0,0	0:00.02	kthreadd	1
3	root	0	-20	0	0	0	I	0,0	0,0	0:00.00	rcu_gp	0
4	root	0	-20	0	0	0	I	0,0	0,0	0:00.00	rcu_par_gp	0
5	root	0	-20	0	0	0	I	0,0	0,0	0:00.00	netns	0
9	root	0	-20	0	0	0	I	0,0	0,0	0:00.00	mm_percpu_wq	0
10	root	20	0	0	0	0	S	0,0	0,0	0:00.00	rcu_tasks_rude_	0
11	root	20	0	0	0	0	S	0,0	0,0	0:00.00	rcu_tasks_trace	0
12	root	20	0	0	0	0	S	0,0	0,0	0:00.89	ksoftirqd/0	0
13	root	20	0	0	0	0	I	0,0	0,0	0:18.56	rcu_sched	1
14	root	rt	0	0	0	0	S	0,0	0,0	0:00.16	migration/0	0
15	root	-51	0	0	0	0	S	0,0	0,0	0:00.00	idle_inject/0	0
17	root	20	0	0	0	0	S	0,0	0,0	0:00.00	cpuhp/0	0
18	root	20	0	0	0	0	S	0,0	0,0	0:00.00	cpuhp/1	1
19	root	-51	0	0	0	0	S	0,0	0,0	0:00.00	idle_inject/1	1
20	root	rt	0	0	0	0	S	0,0	0,0	0:00.35	migration/1	1
21	root	20	0	0	0	0	S	0,0	0,0	0:00.75	ksoftirqd/1	1
23	root	0	-20	0	0	0	I	0,0	0,0	0:00.00	kworker/1:0H-even+	1
24	root	20	0	0	0	0	S	0,0	0,0	0:00.00	cpuhp/2	2

Comando **top** Linux. f: agregar columna. u: elegir usuario.

Ultimo núcleo utilizado

Multiprocesadores - Cuestiones relacionadas con los SO
Schedulers

- Los schedulers sobre sistemas multinúcleo son bidimensionales, se debe decidir:
  - **Qué tarea ejecutar a continuación** (procesos o hilos).
  - **En qué núcleo ejecutar la tarea**.
  - Basados en **hilos**.
- Los schedulers buscan:
  - **Afinidad de tarea con núcleos**: Si una tarea k se ejecuta en un núcleo n, es deseable que siga ejecutándose en el **mismo núcleo**, para aprovechar bloques cargados en caché.
  - **Balanceo de carga**: Migrar tareas de núcleos muy ocupados a núcleos menos ocupados.
  - **Space sharing**: Ejecutar grupos de hilos relacionados en grupos de núcleos, para permitir la ejecución de **tareas paralelas**.

Multiprocesadores - Cuestiones relacionadas con los SO
Schedulers

Para lograr **afinidad de hilos con núcleos** se utilizan schedulers de dos niveles:

- **Scheduler global**.
- **Schedulers locales para cada núcleo**.

- Cuando se crea una tarea, se agenda en un scheduler global.
- Los núcleos toman hilos primero de los schedulers locales.
  - Un hilo se ejecutará en el mismo núcleo por estar en un scheduler local.
  - Si los núcleos no poseen tareas que ejecutar, toman hilos del scheduler global.
- **Migración de tareas**:
  - Un núcleo ocupado puede pedir mover tareas a otro núcleo.
  - Un núcleo libre puede pedir tareas (llamado “robar” tareas).
- Ventaja
  - Aprovechar la afinidad de caché.
  - Distribución de carga entre núcleos.

Multiprocesadores - Cuestiones relacionadas con los SO

Schedulers

Space sharing: agendar varios hilos al mismo tiempo sobre varios CPUs. La asignación se mantiene hasta que los hilos finalicen su tarea.

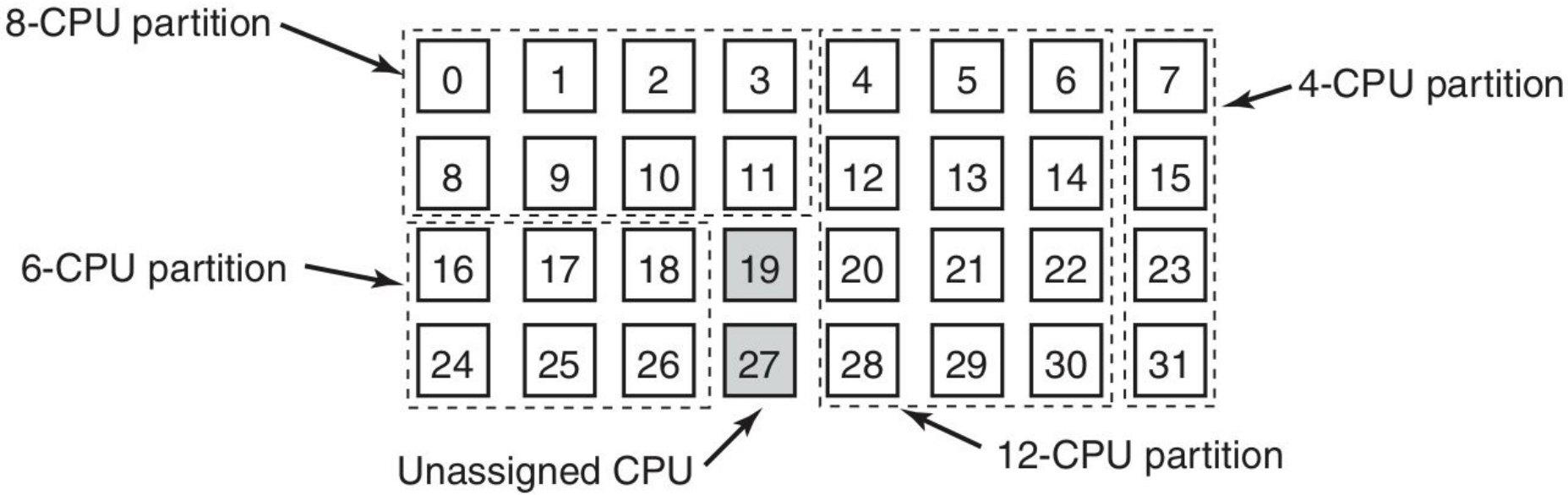


Figura obtenida de: Tanenbaum, Bos, “Modern Operating Systems”, 4° edición, página 542

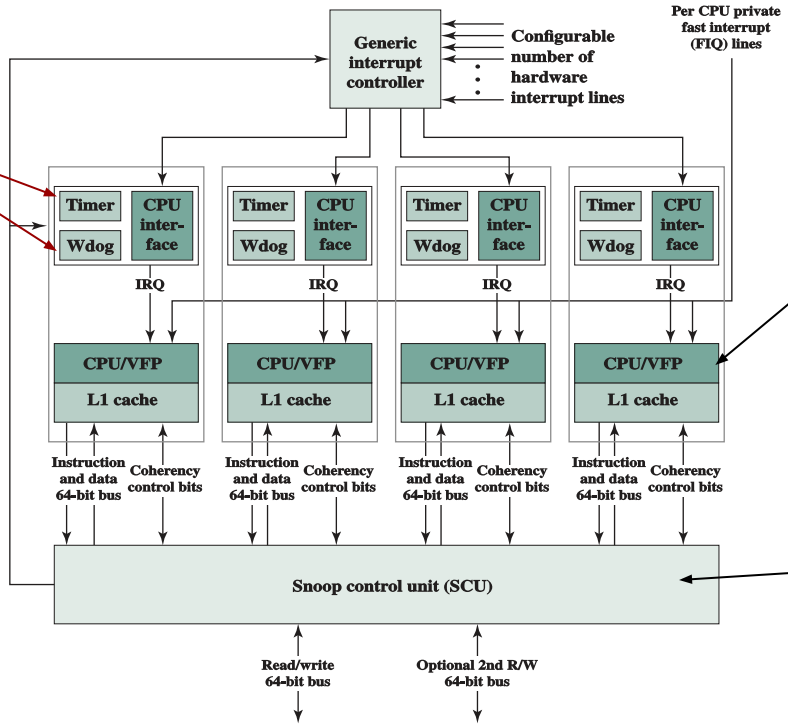
Multiprocesadores - Cuestiones relacionadas con los SO
Schedulers

Cuestiones de diseño:

- La asignación es en base a **bloques de hilos**. Si los hilos son creados al mismo tiempo (generalmente por un mismo proceso) se asignan a bloques de núcleos.
- Al momento de crear los hilos, el algoritmo busca que haya varios núcleos libres. Cuando hay varios núcleos disponibles, los los hilos se asignan al grupo.
- Los núcleos son divididos en grupos, para asignar los grupos de núcleos a hilos de un mismo proceso.
- **Grupos dinámicos**: Las particiones de núcleos pueden cambiar a medida que los procesos entran y salen de ejecución y que la carga total de de todos los núcleos trabajo varía.

Interrupciones en procesadores multinúcleos: Controlador de Interrupciones

Pueden generar interrupciones.



Coprocesador Vector Floating-Point

Coherencia entre cachés

Figura obtenida de “Computer Organization and Architecture”, William Stallings, 10º edición, página 677

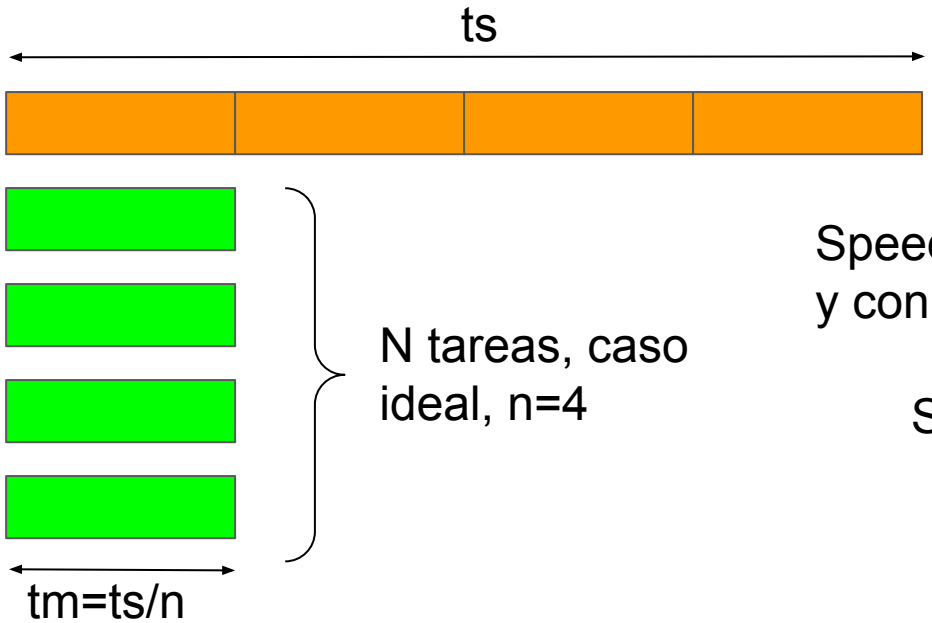
Controlador de Interrupciones

- **Recibe** interrupciones y las **distribuye** entre núcleos.
- Maneja las **prioridades** y enmascaramiento de interrupciones.
- Cuando se produce una interrupción, decide que núcleo la manejará.
  - Una interrupción puede ser dirigida a un núcleo específico o a cualquier núcleo (lo decide el SO).
  - Una interrupción dirigida a cualquier núcleo (usualmente interrupciones por hardware) es asignada por el GIC al núcleo con menos trabajo.
- Permite a los núcleos generar interrupciones (por software).
  - Señalizar eventos o sincronizar tareas (semáforos, mutex).
- Pueden atenderse varias interrupciones al mismo tiempo, aunque tengan diferentes prioridades.
- En procesadores x88: **APIC** (Advanced Programmable Interrupt Controller).
- En ARM: **GIC** (Generic Interrupt Controller)

Medidas de performance: Speedup

Speedup ideal: S(n)= $\frac{ts}{tm}$

ts=Tiempo total n tareas ejecución secuencial.
tm=Tiempo de cada tarea ejecutándose en paralelo.



Speedup ideal, sin overhead de comunicación y con código 100% paralelizable

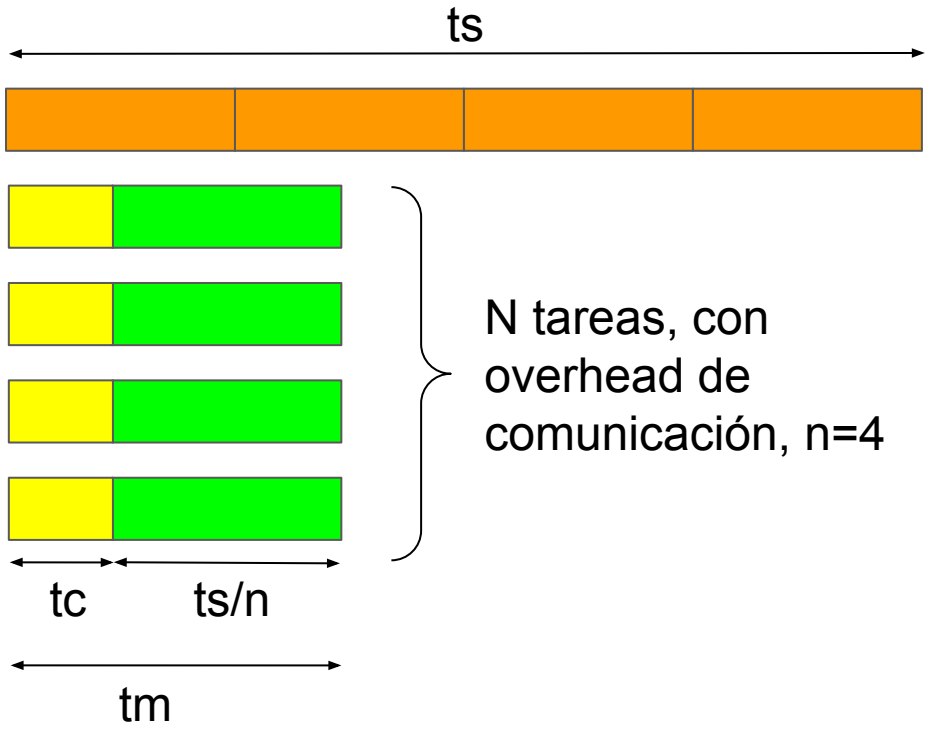
$$\text{Speedup: } S(n)= \frac{ts}{tm} = \frac{ts}{ts/n} = n$$

Speedup Ideal

Speedup considerando overhead de comunicación

$$tm = \frac{ts}{n} + tc$$

Tiempo usando por cada procesador para tareas de comunicación



Speedup considerando overhead de comunicación

$$tm = \frac{ts}{n} + tc$$

$$\text{Speedup: } S(n) = \frac{ts}{\frac{ts}{n} + tc} = \frac{n}{1 + n\frac{tc}{ts}}$$

tc << ts → S = n

$n \to \infty$

- Caso ideal (tc=0): $S \to \infty$
- Caso real (tc > 0): $S = ts/tc$

Por esto es importante que el tiempo de comunicación sea lo más pequeño posible.

Eficiencia o escalabilidad

$$\xi = \frac{S}{n}$$ Speedup por núcleo

Ideal: $S = n \longrightarrow \xi = 1$ ← Significa que si se aumenta k veces el número de procesadores, aumenta k veces el speedup

Con tc>0:

$$S = \frac{n}{1 + n\frac{tc}{ts}}$$

$$\xi = \frac{1}{1 + n\frac{tc}{ts}}$$

Si tc/ts = 0 , ξ =1
Si tc/ts > 0 , ξ <1

Ejecución paralela con código no paralelizable (serial)

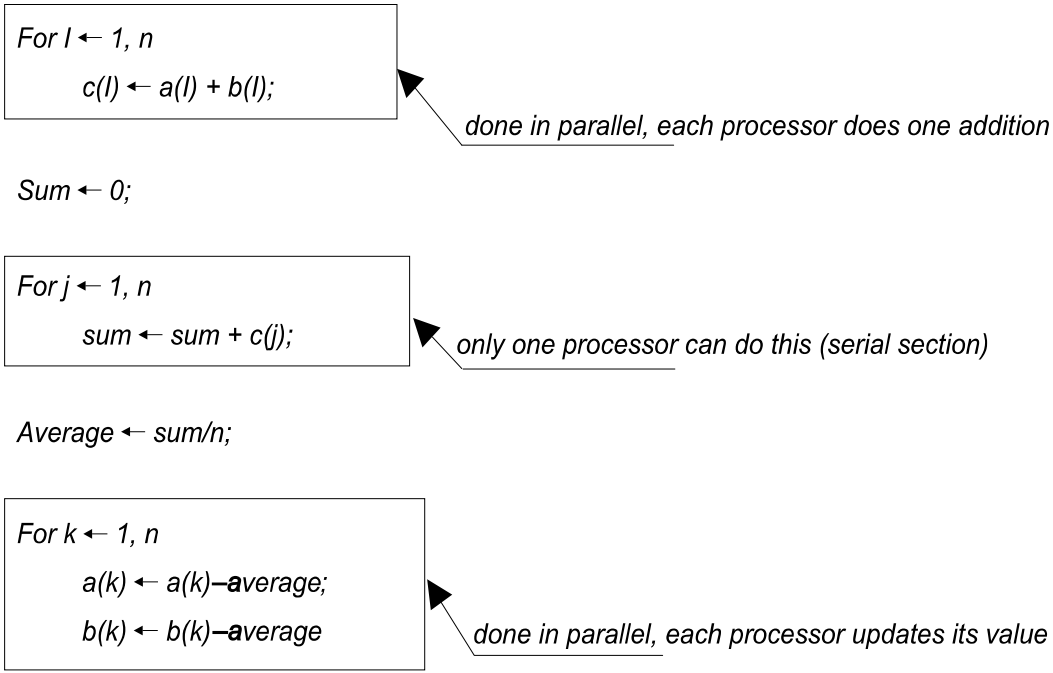


Figura obtenida de: Hesham and Mostafa, “Advanced Computer Architecture and Parallel Processing”, 1° edición, página 53

Ejecución paralela con código no paralelizable

- Para analizar el efecto de la fracción de código no paralelizable, debemos considerar el **tipo de problema** a paralelizar.
  - Modelo de **Amdahl** o problema de **tamaño fijo**.
    - Problemas donde el tiempo para resolver el problema es el parámetro crítico.
  - Modelo de **Gustafson-Barsis** o problema de **tamaño variable proporcional** a la cantidad de tareas que pueden ejecutarse en paralelo.
    - Problemas donde se adapta la precisión del resultado o tamaño de los modelos a los recursos disponibles.
  - Modelo de **Sun y Ni** o de **memoria total limitada**.
    - Problemas con grandes cantidades de datos.

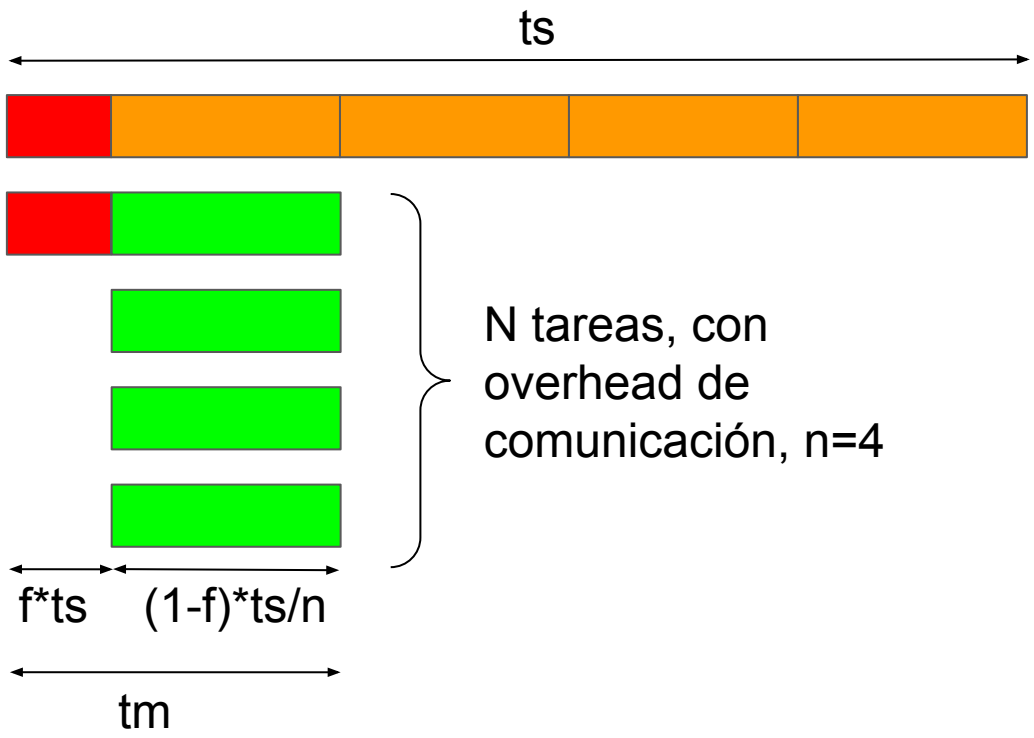
No confundir modelo de Amdahl con Ley de Amdahl. La Ley de Amdahl (“La mejora global que se consigue al mejorar un componente de un sistema de cómputo depende de la fracción de tiempo que se use dicho componente”) está planteada para problemas que siguen el modelo de Amdahl.

Ejecución paralela con código no paralelizable según modelo de Amdahl

- Sin importar el overhead

$$tm = f * ts + (1-f) * \frac{ts}{n}$$

f: Fracción de código no paralelizable (serial)



Ejecución paralela con código no paralelizable según modelo de Amdahl

- Sin importar el overhead de comunicación

$$tm = f * ts + (1-f) * \frac{ts}{n}$$

Fracción de código no paralelizable (serial)

$$S = \frac{ts}{f * ts + \frac{(1-f) * ts}{n}} = \frac{n}{1 + (n-1) * f}$$

f = 1 (ninguna parte del código es paralelizable) ⇒ S = 1

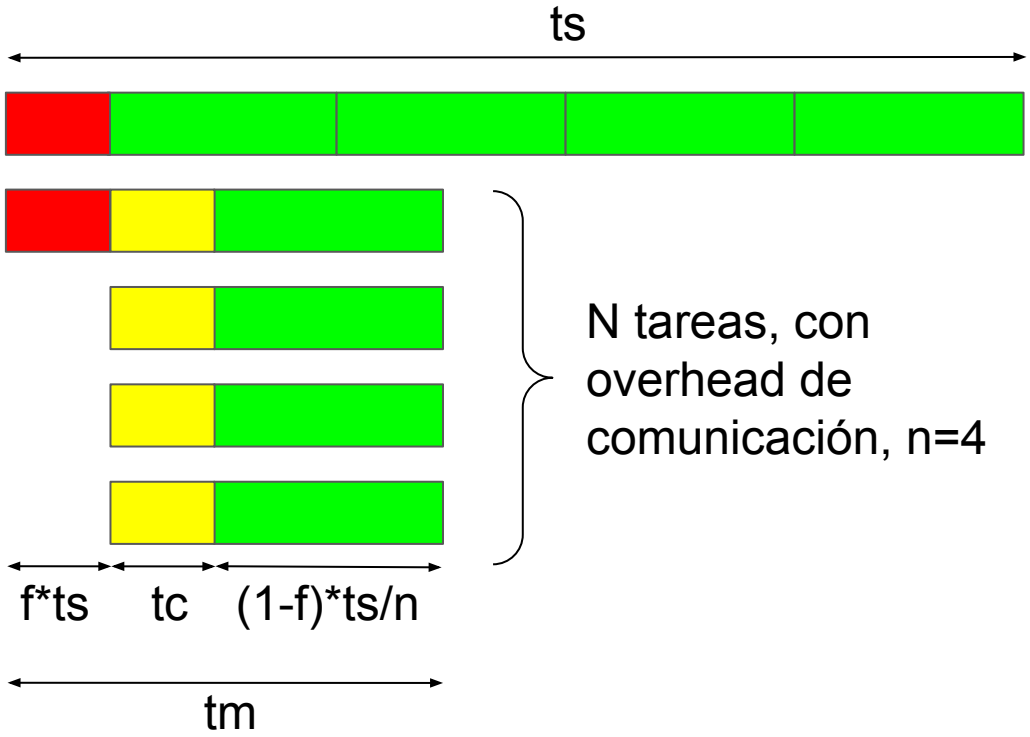
$n \rightarrow \infty \Rightarrow S = 1/f$ ← **Sin importar la arquitectura ni n, el S máximo está limitado por la porción de código no paralelizable**

Ejecución paralela con código no paralelizable según modelo de Amdahl

- Considerando overhead y código no paralelizable

$$tm = f * ts + \frac{(1-f) * ts}{n} + tc$$

tc=Tiempo de comunicación (overhead)
f=Fracción de código no paralelizable



Ejecución paralela con código no paralelizable según modelo de Amdahl

- Considerando overhead y código no paralelizable

$$tm = f * ts + \frac{(1-f) * ts}{n} + tc$$

$$S = \frac{ts}{f * ts + \frac{(1-f) * ts}{n} + tc} = \frac{n}{1 + (n-1) * f + n * \frac{tc}{ts}}$$

$$\text{Si } n \rightarrow \infty \Rightarrow S = \frac{1}{f + tc/ts}$$

Ejecución paralela con código no paralelizable según modelo de Amdahl

- Sin Considerar overhead y considerando código paralelizable

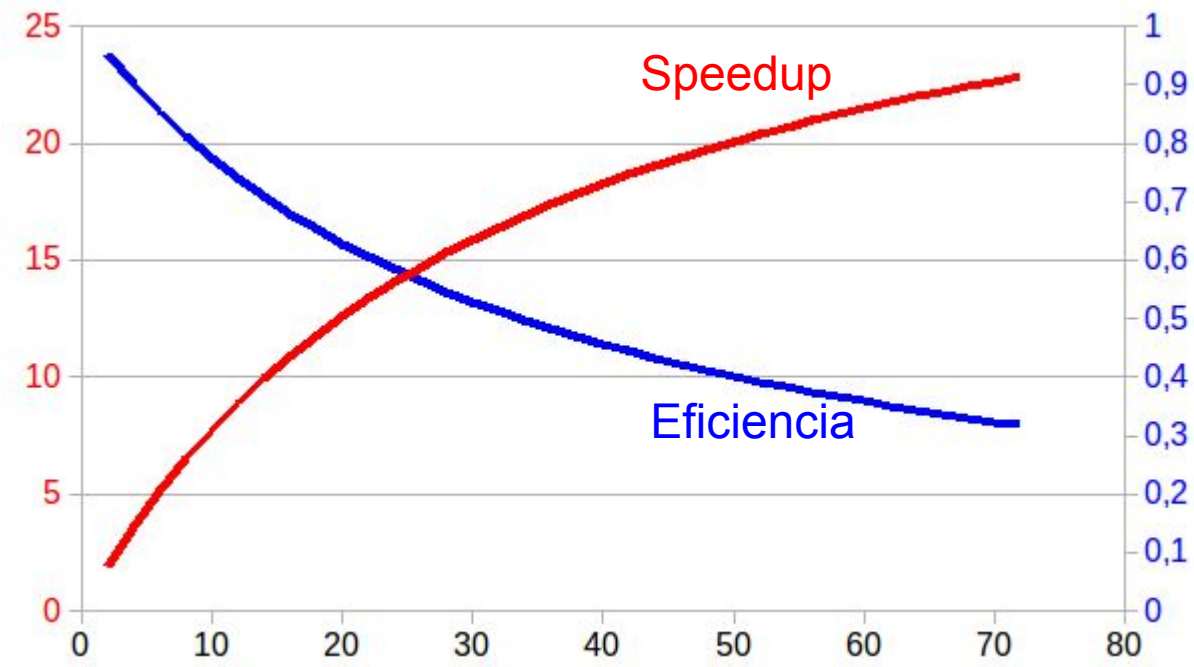
$$\xi = \frac{S}{n} = \frac{1}{1 + (n-1) * f}$$

Ejemplo: f=0.2
n=4; ξ=0.625
n=100; ξ=0,0481

- Con overhead y código paralelizable

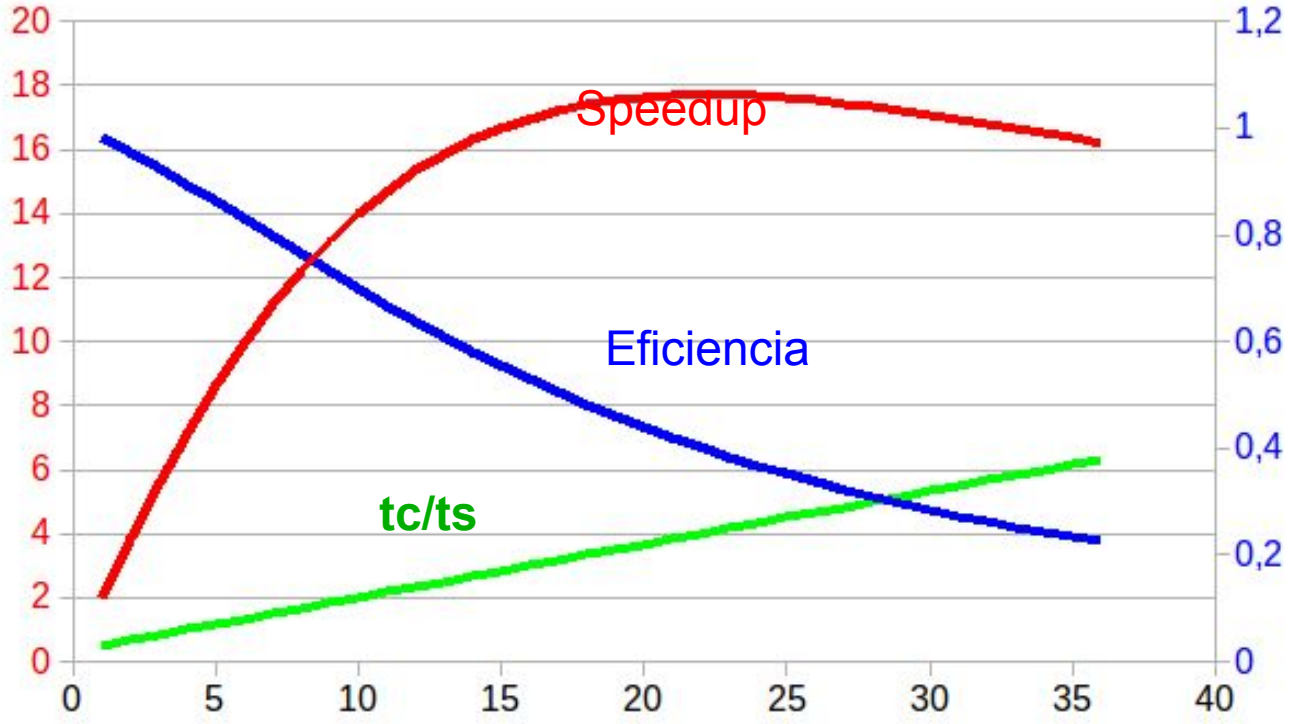
$$\xi = \frac{S}{n} = \frac{1}{1 + (n-1) * f + n * \frac{tc}{ts}}$$

Ejemplo con (tc/ts) = 0,02 y f = 0,04 (S_{max}=25)



Ejemplo con (tc/ts) variable en función de n y f = 0,01

Nota: tc depende de la cantidad de tareas a ejecutar en paralelo, del hardware de comunicaciones y del software.

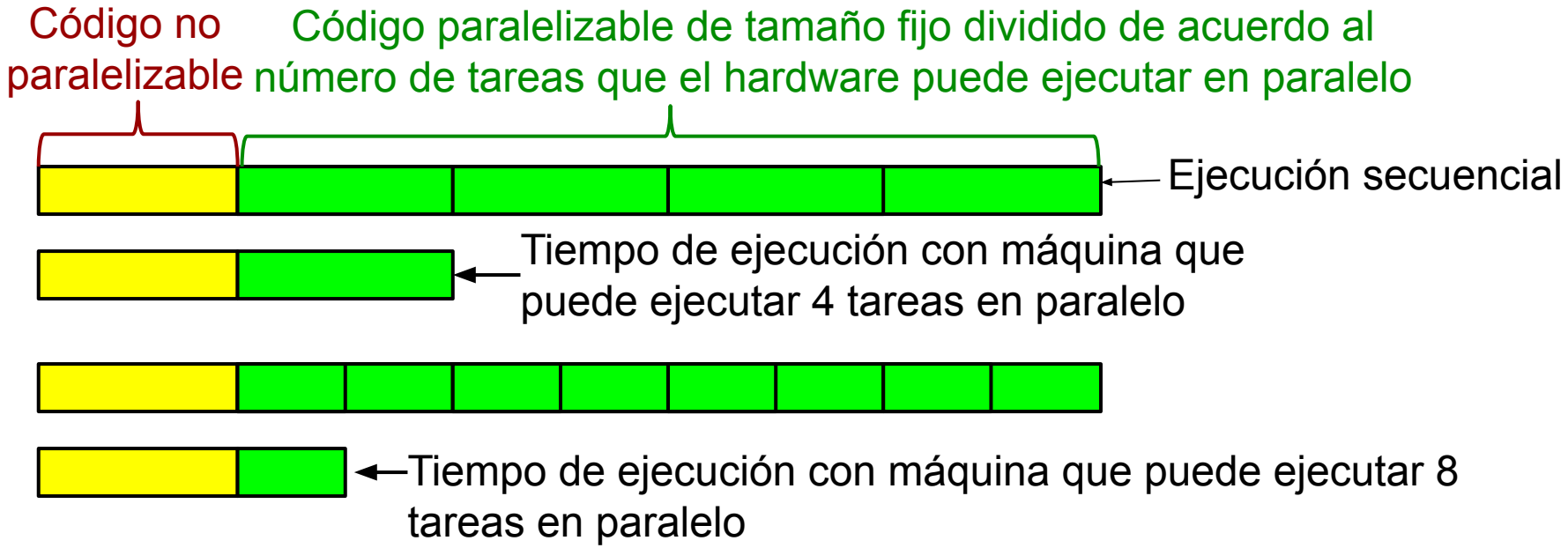


Ejecución paralela con código no paralelizable según modelo de Amdahl

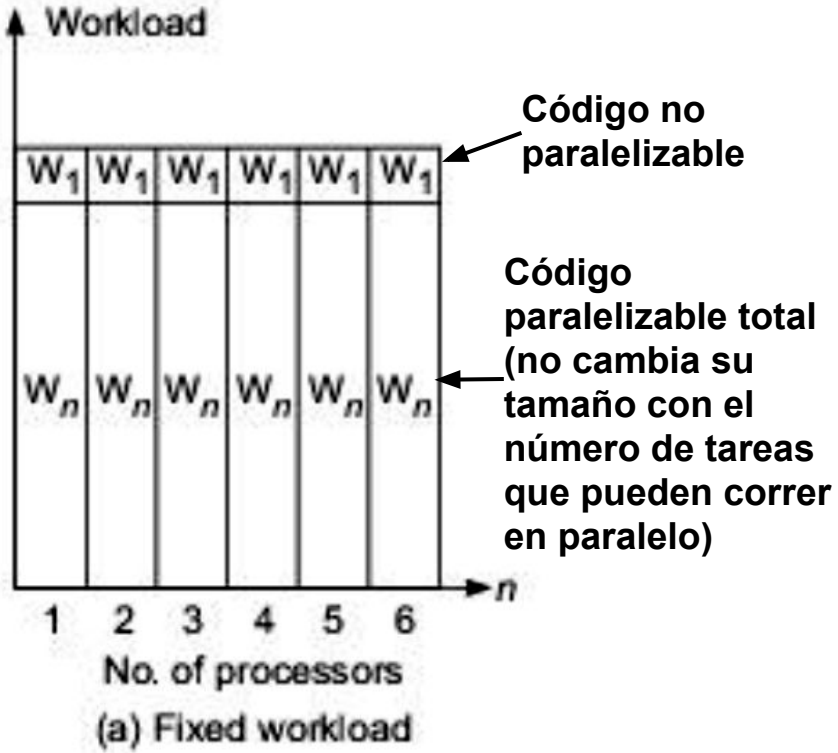
- A medida que agregamos más procesadores (o computadoras) el speedup **no aumenta linealmente**.
- Mientras mayor la porción de código no paralelizable, menor la ganancia de speedup al agregar más procesadores (disminuye la eficiencia).
- El **overhead de comunicación** (tiempo de comunicación entre procesadores) tiene un impacto importante sobre el speedup.
    - Si el tiempo de comunicación entre procesadores depende del número de procesadores n (lo cual ocurre en la realidad) puede imponer un máximo a la curva speedup en función de n.

Speedup según Ley de Amdahl

- Modelo de **Amdahl** supone que el tiempo total o **tamaño total del problema es fijo**.
  - Si aumenta n, disminuye el tiempo de ejecución en cada procesador.
  - Ejemplo: Problemas donde se busca el menor tiempo de ejecución.

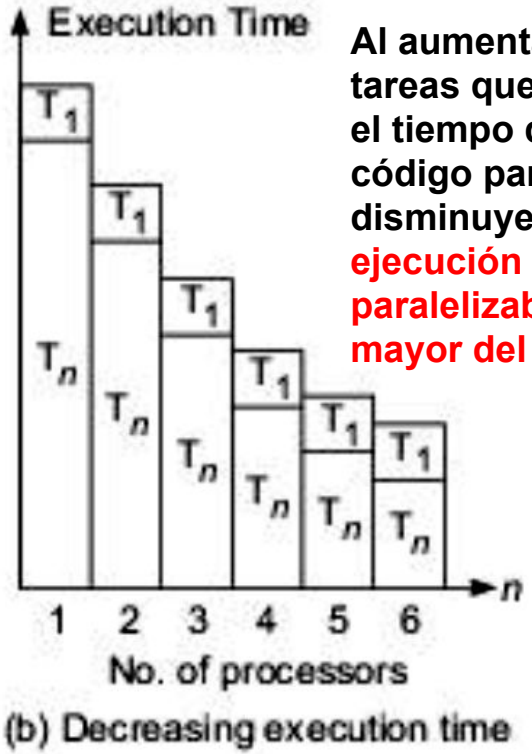


Speedup según Ley de Amdahl



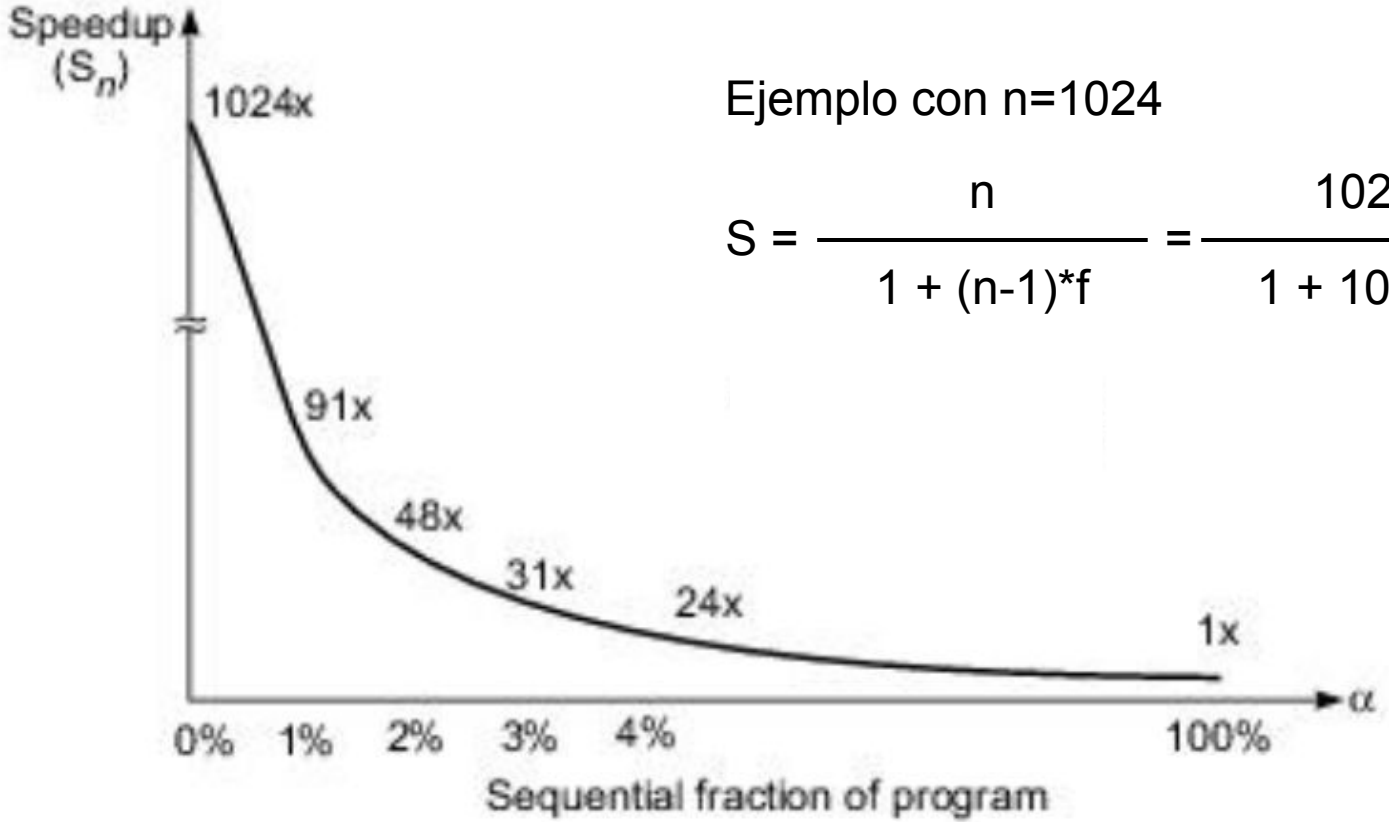
Código no paralelizable

Código paralelizable total (no cambia su tamaño con el número de tareas que pueden correr en paralelo)



Al aumentar el el número de tareas que corren en paralelo, el tiempo de ejecución del código paralelizable disminuye, y el tiempo de ejecución del código no paralelizable es un porcentaje mayor del tiempo total.

Figura obtenida de Kai Hwang, “Advanced Computer Architecture Parallelism Scalability Programmability”, 2º edición, página 110



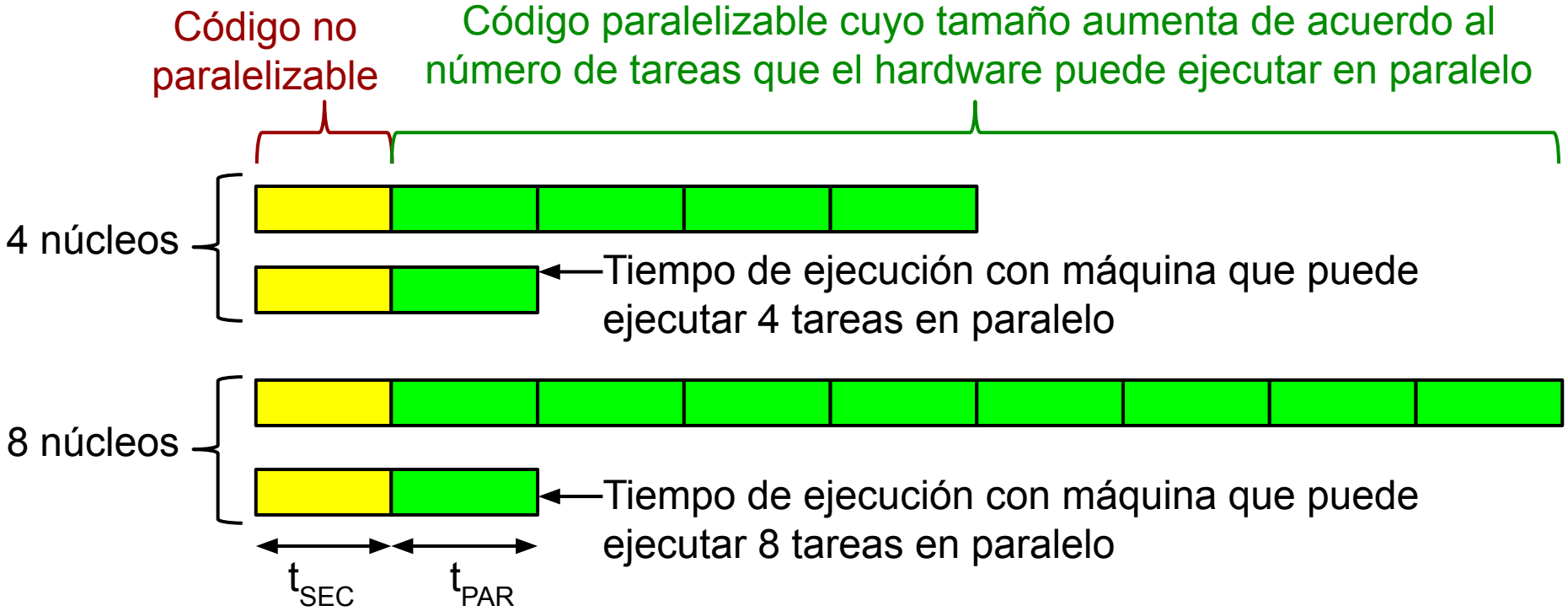
Ejemplo con n=1024

$$S = \frac{n}{1 + (n-1)*f} = \frac{1024}{1 + 1023*f}$$

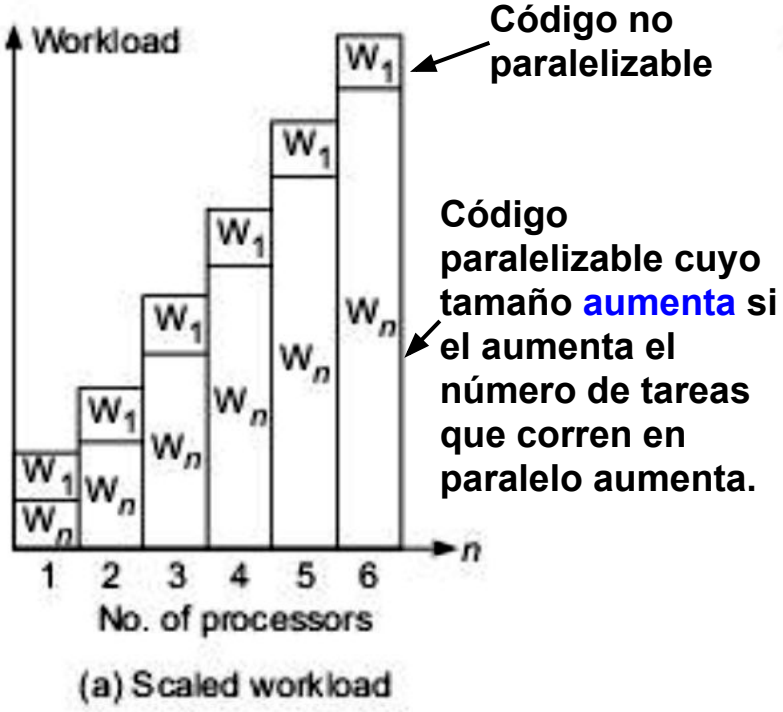
Figura obtenida de Kai Hwang, “Advanced Computer Architecture Parallelism Scalability Programmability”, 2º edición, página 110

Speedup según Gustafson-Barsis

- Supone que los tiempos t_{SEC} y t_{PAR} son constantes, y el tiempo total o **tamaño del problema aumenta con el valor de n**.

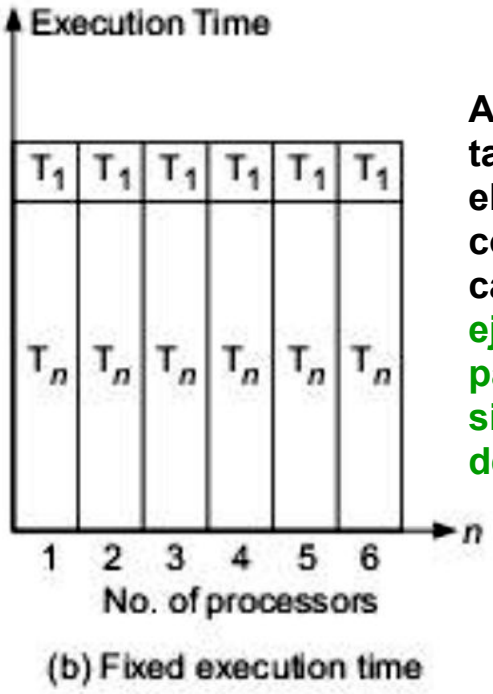


Speedup según Ley de Gustafson-Barsis



Código no paralelizable

Código paralelizable cuyo tamaño aumenta si el aumenta el número de tareas que corren en paralelo aumenta.



Al aumentar el el número de tareas que corren en paralelo, el tiempo de ejecución del código paralelizable no cambia, y el tiempo de ejecución del código no paralelizable representa siempre el mismo porcentaje del tiempo total.

Figura obtenida de Kai Hwang, “Advanced Computer Architecture Parallelism Scalability Programmability”, edición 2001, página 110

Speedup: Ley de Gustafson-Barsis

$$S = \frac{t_{SEC} + t_{PAR} * n}{t_{SEC} + t_{PAR}}$$

(Ver figura filmina anterior)

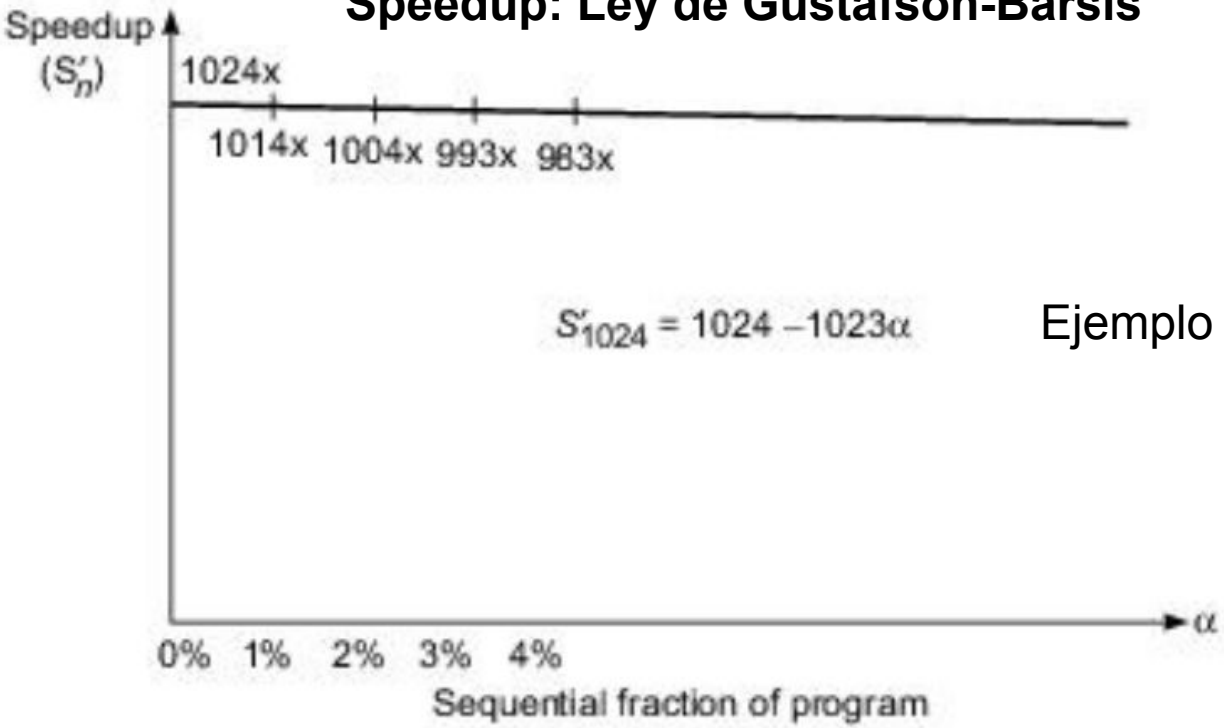
$$a = \frac{t_{SEC}}{t_{SEC} + t_{PAR}} \Rightarrow (1 - a) = 1 - \frac{t_{SEC}}{t_{SEC} + t_{PAR}} = \frac{t_{SEC} + t_{PAR} - t_{SEC}}{t_{SEC} + t_{PAR}} = \frac{t_{PAR}}{t_{SEC} + t_{PAR}}$$

$$S = \frac{t_{SEC} + t_{PAR} * n}{t_{SEC} + t_{PAR}} = \frac{t_{SEC}}{t_{SEC} + t_{PAR}} + \frac{t_{PAR} * n}{t_{SEC} + t_{PAR}} = a + (1 - a)*n$$

a<1

Si $n \rightarrow \infty \Rightarrow S \rightarrow \infty$

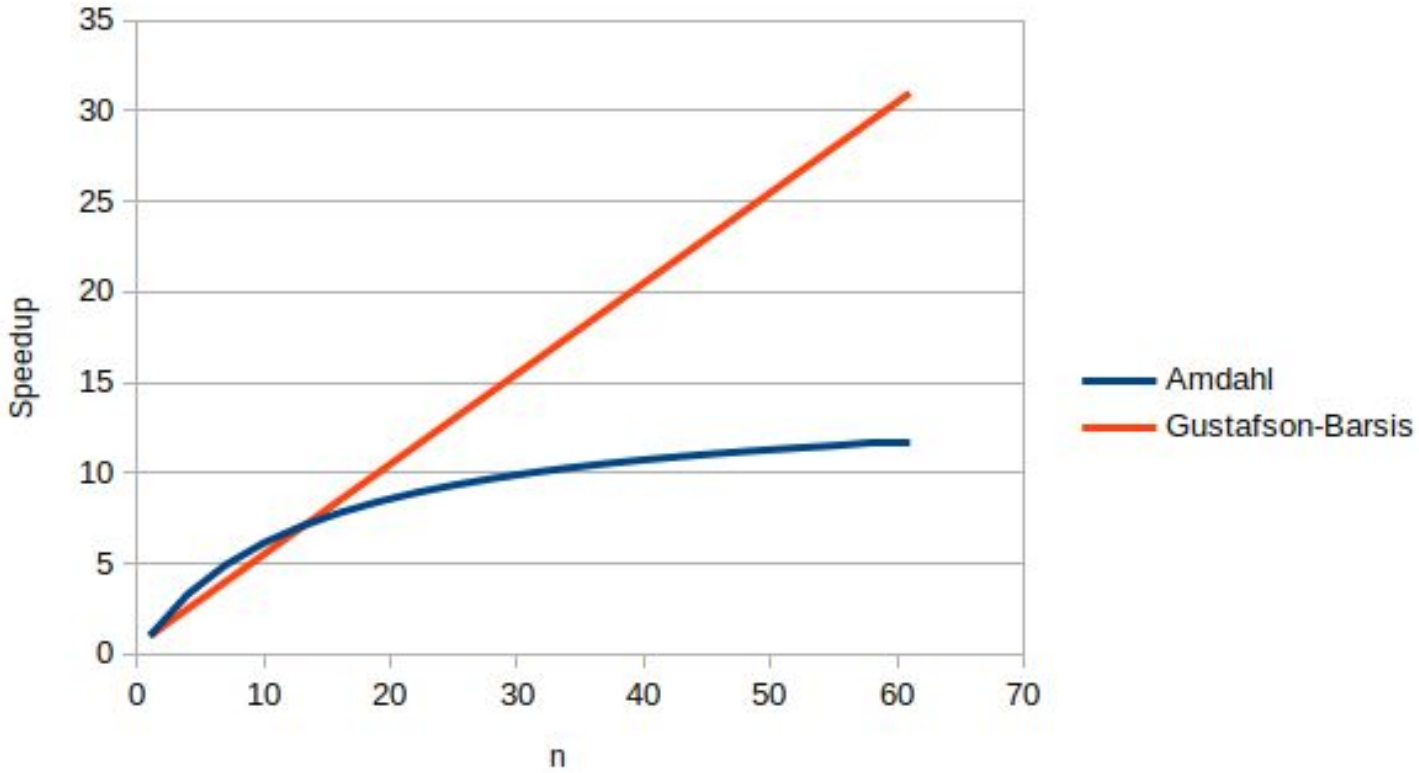
Speedup: Ley de Gustafson-Barsis



Ejemplo con n=1024

Figura obtenida de Kai Hwang, “Advanced Computer Architecture Parallelism Scalability Programmability”, 2º edición, página 110

f=0,07


Speedup
35
30
25
20
15
10
5
0
0
10
20
30
40
50
60
70
n
Amdahl
Gustafson-Barsis

Speedup según Gustafson-Barsis

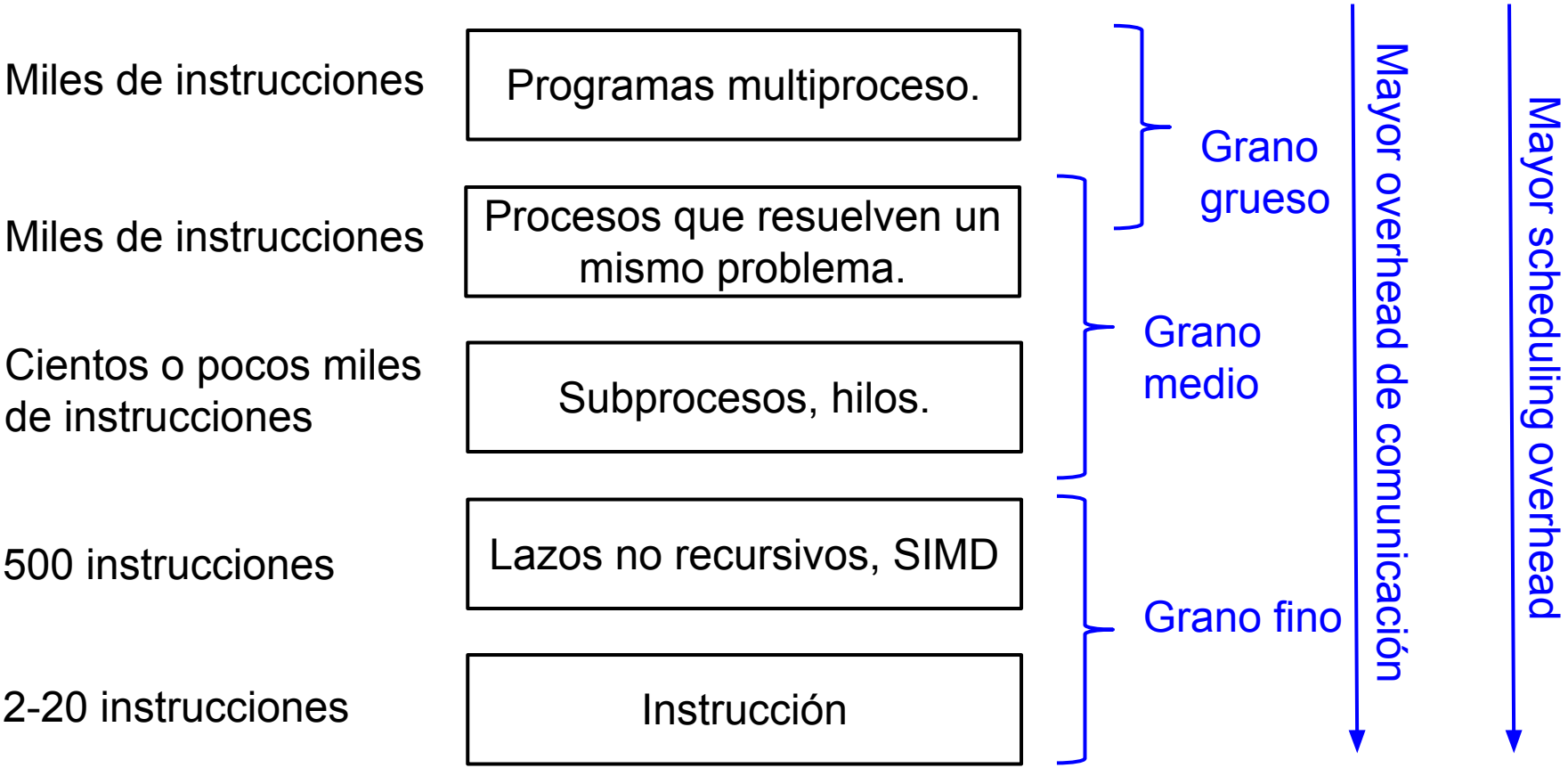
- La **Ley de Amdahl** indica que **el speedup tiene un límite superior** para **problemas de tamaño fijo**.
  - Conclusión muy negativa, ya que indica que “no vale la pena” incrementar la cantidad de procesadores dado un valor de f para conseguir mayores speedup.
- La **Ley de Gustafson-Barsis** indica que **el speedup no posee límite** superior si el **tamaño del problema crece linealmente** con el número de procesadores.
  - **Conclusión muy positiva, ya que indica que si vale la pena incrementar el número de procesadores para conseguir mayores speedup**.
- Ambos suponen modelos ideales, pero los problemas reales se acercan más al modelo de la **Ley de Gustafson** ya que:
  - Los problemas requieren la mayor precisión posible (mayor número de decimales, menor tamaño de grilla), lo cual se logra incrementando n.
  - El tamaño de los problemas se adapta al n disponible, y no al revés.

Paralelismo en el software: Granularidad

- **Tamaño de grano**: Tamaño de los segmentos o porciones de código (número de instrucciones) que se ejecutan en paralelo para resolver una tarea.
- Varios niveles. Cada nivel de granularidad posee diferentes características (latencias, mecanismos de comunicación, herramientas de desarrollo, plataformas de hardware adecuadas, etc.).
- Mientras mayor el tamaño de los segmentos de código o granos (mayor número de instrucciones), mayor cantidad de problemas puede resolver un solo segmento de código o grano, y menor la cantidad de comunicaciones.
- Fuertemente relacionado con la latencia en la comunicación.
- La decisión del tamaño de grano depende del problema a resolver y de la o las computadoras disponibles (su hardware, sistema operativo, etc.).
- Las aplicaciones usualmente combinan varios niveles de granularidad.

Bibliografía: Kai Hwang. Advanced Computer Architecture. Parallelism, Scalability, Programmability. Pag 52 en la 2° ed, o Pag 61 1° ed.

Paralelismo en el software: Granularidad



Paralelismo en el software: Granularidad

	Plataforma típica	Quién decide que paralelizar
Programas multiproceso.	Multicomputadora débilmente acoplada, computadora multitarea.	SO.
Procesos de una misma tarea.	Multinúcleo, multicomputadoras fuertemente acopladas (clústers).	Usuario (procesos).
Subprocesos, hilos.	Multinúcleo, procesadores con multihilo simultáneo. GPU	Usuario (hilos), compilador, SO.
Lazos no recursivos, SIMD	Procesadores SIMD, vectoriales, GPU.	Compilador (lazos) usuario (SIMD GPU).
Instrucción	Pipeline, superescalar, fuera de orden.	Procesador, Compilador (VLIW).

Paralelismo en el software: Granularidad

	Comunicación	Herramientas de desarrollo
Programas multiproceso.	Mensajes (poca comunicación).	
Procesos de una misma tarea.	Mensajes.	MPI, RPC.
Subprocesos, hilos.	Memoria compartida.	Librerías progr multi-hilo, OpenCL, CUDA, OpenMP.
Lazos no recursivos, SIMD	Registros, memoria caché, memoria compartida.	CUDA, OpenCL, instrucciones SIMD.
Instrucción	Registros, memoria caché.	Lo realiza el procesador.

Bibliografía:

- William Stallings, "Computer Organization and Architecture”, 10º edición, editorial Pearson, año 2016.
- Tanenbaum and Bos, “Modern Operating Systems”, 4° edición, editorial Pearson, año 2015.
- Kai Hwang, “Advanced Computer Architecture, Parallelism, Scalability, Programmability”, 2º edición, editorial Mc Graw-Hill, año 2011.
- Hesham and Mostafa, “Advanced Computer Architecture and Parallel Processing”, 1° edición, editorial Wiley, año 2005.
- ARM, "ARM Cortex-A Series Programmer’s Guide for ARMv8-A", Version 1.0, año 2015