

# Tema 1. Introducción

## Computación de Altas Prestaciones

Carlos García Sánchez

2 de septiembre de 2022

- “Introduction to High Performance Computing for Scientists and Engineers”, Georg Hager, Gerhard Wellein
- “Parallel Computer Architecture: A Hardware/Software Approach”, David Culler, Jaswinder Pal Singh, Anoop Gupta



# Outline

- 1** Introducción
- 2** Ciencia en ingeniería computacional
- 3** Complejidad, grado de paralelismo y granularidad
- 4** Niveles de Paralelismo
- 5** Evaluación de rendimiento



## Definición

Wikipedia

La computación de alto rendimiento (High performance Computing o HPC en inglés) es la agregación de potencia de cálculo para resolver problemas complejos en ciencia, ingeniería o gestión. Para lograr este objetivo, la computación de alto rendimiento se apoya en tecnologías computacionales como los clusters, los supercomputadores o la computación paralela. La mayoría de las ideas actuales de la computación distribuida se han basado en la computación de alto rendimiento

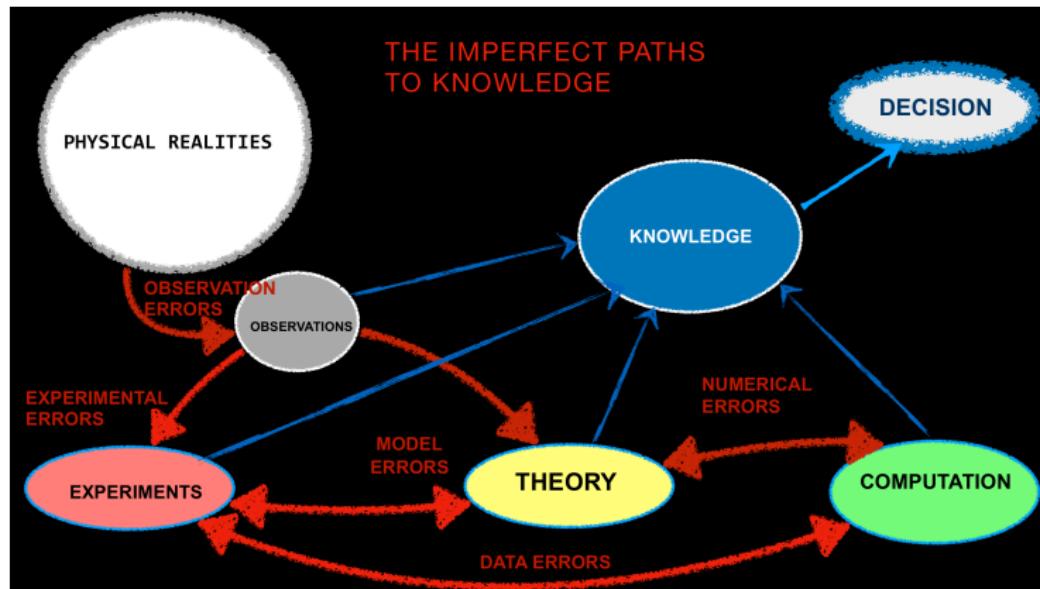


## ¿Para qué usar HPC?

- La humanidad ha ido progresando mediante la observación....
  - ... y ante los conflictos entre seres humanos se han elaborado leyes



## ¿Para qué usar HPC?



# ¿Para qué usar HPC?



## Reducción de gasto de combustible en vuelos comerciales

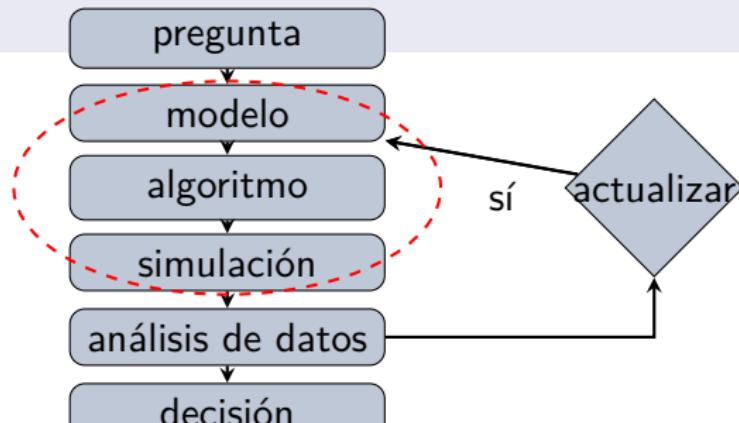
- De acuerdo a la IATA (Asociación de transporte aereo comercial) el número pasajeros se duplicará para el 2036
- El aumento de tráfico aero presenta un nuevo desafío
- Las mejoras en la construcción de las aeronaves ha mejorado significativamente el consumo en los últimos 25 años...
- ... pero se puede ¿hacer más?



# ¿Para qué usar HPC?

## ¿Qué es la computación?

*Computing is a domain of knowledge dealing with the study of Information processing, both what can be computed and how to compute it. **Joseph Sifakis***



# ¿Para qué usar HPC?

## ¿Por qué usar la computación?

- Complementar las teorías y experimentos
- Testear hipótesis
- Procesar y analizar datos
- Predecir comportamientos
- Optimizar el diseño (toma de decisiones)



# ¿Para qué usar HPC?

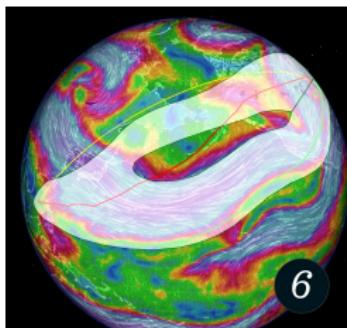
- Reducción de gasto de combustible en vuelos comerciales<sup>1</sup>

## CASE STUDY: TAPEI – NEW YORK

Boeing B777-300ER, April 25, 2017, great circle distance = 12,565 km

	Using typical search space reduction	Using dynamic search space reduction	GAIN
Distance flown (km)	13,385	14,635	-1,250
Flight time (hours)	14:40	13:55	0:45
Fuel burn (kg)	95,524	89,859	5,665 ≈ 17.8 t CO <sub>2</sub>
Overflight fees (USD)	2,291	1,139	1,152
Total costs (USD)*	76,453	71,118	5,335

\*based on: fuel price 500 USD/ton, flight time costs: 1,800 USD/hour



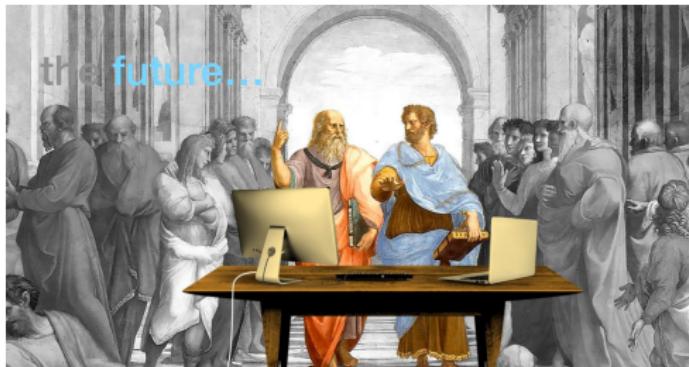
6

<sup>1</sup> Specific fuel consumption for the Lufthansa fleet <https://www.zib.de/features/specific-fuel-consumption-lufthansa-fleet>



# ¿Para qué usar HPC?

- La humanidad ha ido progresando mediante la observación....
- ... y ante los conflictos entre seres humanos se han elaborado leyes
- **Ahora disponemos de computadores para mejorar el progreso de la humanidad**



# Introducción



## ¿Qué tipo de problemas requiere de HPC?

- **Cómputo Intensivo:** un solo problema requiere gran cantidad de operaciones
- **Memoria Intensiva:** un solo problema requiere gran cantidad de memoria para emplazar los datos
- **Datos Intensivos:** un solo problema requiere gran cantidad de datos a procesar. Ej: Big-Data
- **Alto rendimiento o *high throughput*:** muchos problemas deben de ser procesados en lotes



# Problemas del Cómputo intensivo

- Distribuir **trabajo** de un solo problema entre múltiples CPUs para reducir el tiempo de ejecución lo más posible:  
**paralelización**
  - Mismo programa con datos de entrada distintos
  - Cada proceso/hilo realiza una parte del trabajo en su propia CPU simultáneamente
  - Un programa bien paralelizado tardará lógicamente menos:  
**Trabajo en equipo**

## Desafíos

- Las CPU generalmente necesitan intercambiar información rápidamente
  - Hardware de comunicación especializado (Red de interconexión)



# Problemas de Memoria Intensiva

- Requerimientos de mayor cantidad de memoria grande, muchos nodos
  - Memoria (rápida, volátil)
  - Disco (lenta, no volátil).
- La optimización del rendimiento es más difícil
  - El diseño de la memoria en un supercomputador tiende a ser de acceso no uniforme
- Técnicamente más difícil y costoso de escalar en memoria



# Problemas del Datos Intensivos

- Distribuir los datos de un solo problema en múltiples CPU para reducir el tiempo total de ejecución
- Se puede hacer el mismo trabajo en cada subgrupo de datos
- El movimiento rápido de datos con el disco es más importante que comunicación entre CPU

## Problemas de Big Data de gran interés actual

- Hadoop / MapReduce
- Ciencias de la vida (genómica) y en otros lugares



# Problemas del Alto rendimiento

- Distribuir *múltiples problemas independientes* entre varias CPU para reducir el tiempo total de ejecución
- El *workload* contiene paralelismo implícito (*embarrassingly parallel*):
  - La carga de trabajo se divide en trozos independientes
  - Cada porción se realiza mediante en un proceso/hilo separado (concurrentemente)
  - Poca o ninguna comunicación es necesaria entre CPUs
- El **énfasis está en el rendimiento global** no en el de un solo problema



# Resumen

## Poniendo todos estos problemas juntos

- Cada uno de estos problemas requiere la combinación de muchas CPUs y módulos de memoria
- Actualmente, en un servidor se pueden *apilar* varios nodos-CPU y módulos de memoria

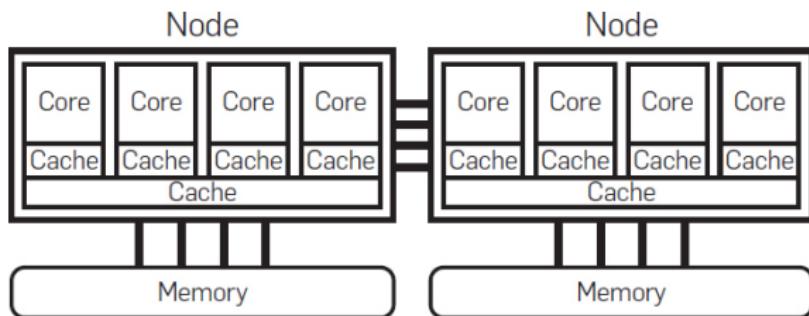


## Resumen: mirando dentro de un computador moderno

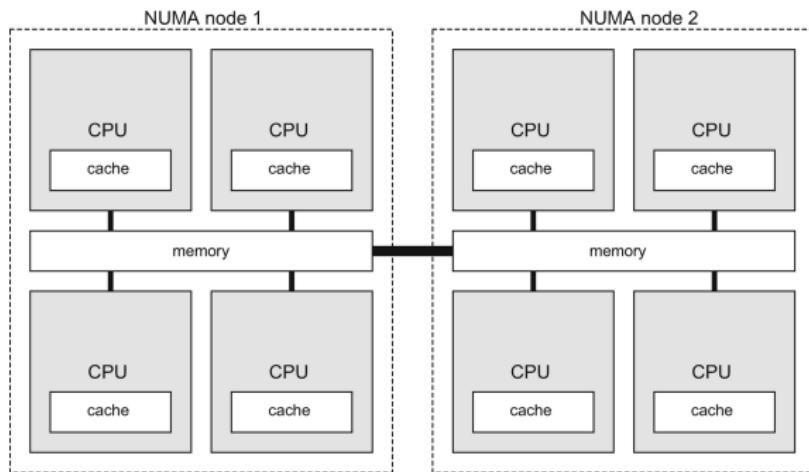
- Los servidores básicos de hoy ya agregan tanto CPUs como memoria para hacer **una sola imagen del sistema**
- Incluso las computadoras *domésticas* ahora tienen múltiples núcleos de CPU por socket
  - ...cada *socket* es un multiprocesador simétrico (SMP)
- Las computadoras más grandes tienen múltiples *sockets* (cada uno con memoria local o **caches**):
  - ... pero todos los núcleos comparten la memoria del nodo
  - ... comportándose como un nodo es un multiprocesador de memoria compartida con arquitectura de memoria no uniforme (NUMA)



# Resumen: mirando dentro de un computador moderno



# Resumen: mirando dentro de un computador moderno



# Paralelismo

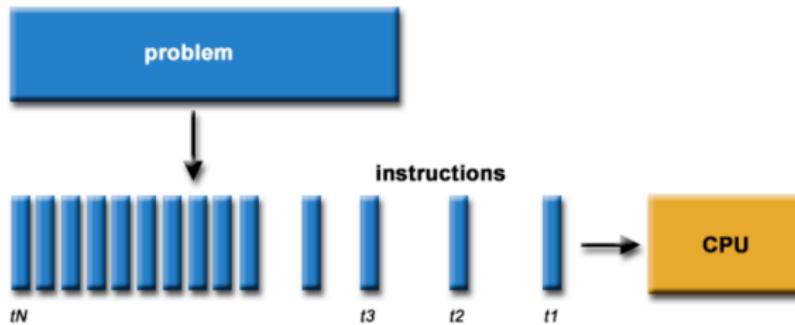
¿Por qué los computadores son paralelos?

- Computador secuencial vs computador paralelo



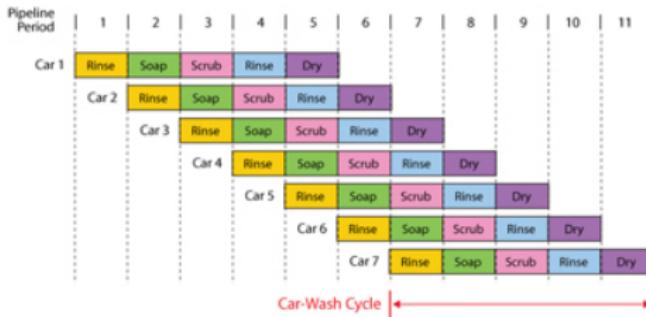
# Paralelismo

- Una **implementación secuencial** de un programa en un mono-procesador
  - El programa ejecuta instrucciones
  - Un instrucción detrás de otra
  - ¿Una instrucción al mismo tiempo?



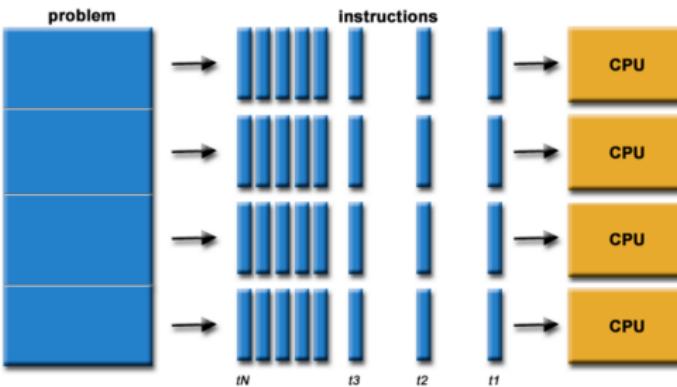
## Paralelismo

- Incremento de rendimiento del procesador: **ILP**



# Paralelismo

- El programa se ejecuta usando múltiples CPUs
  - Un problema se divide en partes discretas que pueden ser resuelto simultáneamente
  - Cada parte se divide en una serie de instrucciones
  - Las instrucciones de cada parte se ejecutan simultáneamente en diferentes máquinas (CPUs)



# Speed-up

- Es una métrica muy habitual para expresar como de **rápido** o lento es la aplicación paralela
  - $T_s$ : tiempo de ejecución secuencial
  - $T_p$ : tiempo de ejecución de la parte paralela

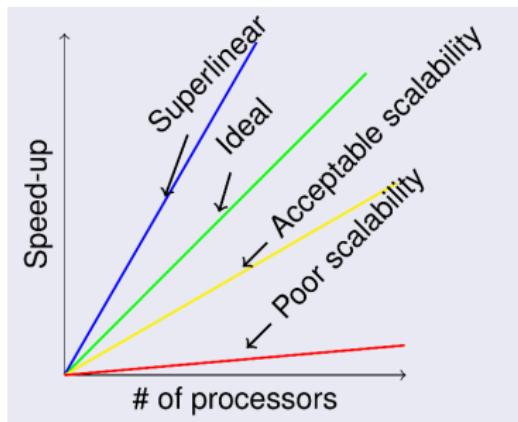
## Aceleración (S)

$$S(p) = \frac{T_s}{T_p}$$



# Escalabilidad

- Expresa como de buena es la eficiencia
- Como se incrementa el rendimiento con el número de procesadores



# ¿Cuánto se puede acelerar una aplicación?

- Conocido como la Ley de Ahmdal
- Expresa el impacto de la parte secuencial de un programa
- ... donde **f** es la fracción del programa que es secuencial
- **Escalabilidad limitada por la parte secuencial**

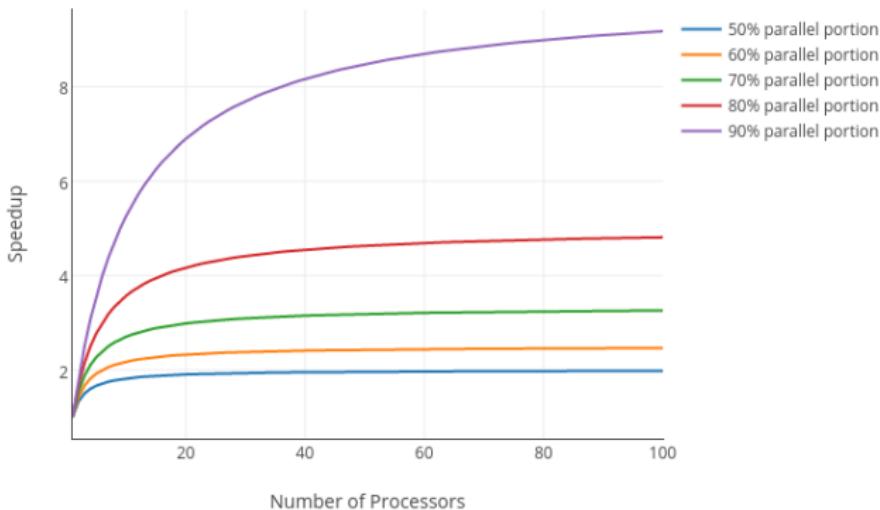
## Ley de Ahmdal

$$S_{max}(p) = \frac{1}{f + \frac{1-f}{p}}$$



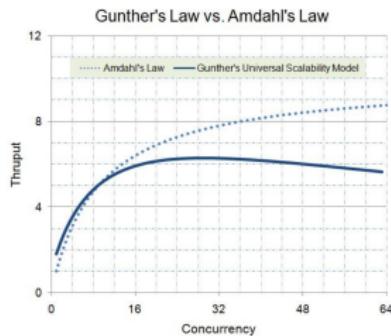
# Ley de Amdahl

Amdahl's Law



# Ley de Gunther

- Un extensión de la ley de Amdahl...
- ... que añade el coste de la comunicación



## Conclusión

- 1 Si se desea escalabilidad hay que reducir la porción secuencial
- 2 Reducir la penalización por coherencia (comunicación)



## Ejemplo Ley de Amdahl

$$TiempoEjec_{antes} = TiempoEjec_{afecta\ mejora} + TiempoEjec_{sin\ mejora}$$

$$TiempoEjec_{mejorado} = \frac{TiempoEjec_{afecta\ mejora}}{Factor\ Mejora} + TiempoEjec_{sin\ mejora}$$

$$Speedup = \frac{\frac{TiempoEjec_{antes}}{TiempoEjec_{sin\ mejora} + \frac{TiempoEjec_{mejorado}}{Factor\ Mejora}}}{TiempoEjec_{sin\ mejora}}$$

¿Qué fracción del código hay que mejorar para lograr una aceleración de 90x con 100 procs?

$$1 \quad S_{max}(p) = \frac{1}{f + \frac{1-f}{p}} \rightarrow 90 = \frac{1}{f + \frac{1-f}{100}}$$

$$2 \quad f = \frac{89}{89,1} = 0,999$$

- Para alcanzar un speedup de 90x con 100 procs. **solo un 0.1% debe de ser secuencial**



## Ejemplo Ley de Amdahl

- Suponiendo que queremos realizar simultáneamente 2 sumas:
  - Suma de 10 variables escalares
  - Suma de una matriz 2D con dimensión 10x10

¿Qué speedup sería esperable para 10 y 100 procs?

- Suponiendo que **t** es el tiempo empleado en realizar una suma y que la **suma de la matriz puede escalar** mientras que **las escalares no**

1  $timeSeq = 100t + 10t = 110t$



## Ejemplo Ley de Amdahl

¿Qué speedup sería esperable para 10 procs?

- 1  $time_{10\,procs} = \frac{100t}{10} + 10t = 20t$
- 2  $Speedup_{10\,procs} = 110t/20t = 5,5(10\ max)$
- 3  $Efficiency_{10\,procs} = \frac{5,5}{10} = 55\%$

¿Qué speedup sería esperable para 100 procs?

- 1  $time_{100\,procs} = \frac{100t}{100} + 10t = 11t$
- 2  $Speedup_{100\,procs} = 110t/11t = 10(100\ max)$
- 3  $Efficiency_{100\,procs} = \frac{10}{100} = 10\%$



## Ejemplo Ley de Amdahl

- ...y para un problema mayor?
- Suponiendo que queremos realizar simultáneamente 2 sumas:
  - Suma de 10 variables escalares
  - Suma de una **matriz 2D con dimensión 100x100**

¿Qué speedup sería esperable para 10 y 100 procs?

1  $timeSeq = 10000t + 10t = 10010t$

2  $time_{10\,procs} = \frac{10000t}{10} + 10t = 1010t$

3  $Speedup_{10\,procs} = 10010t / 1010t = 9,9(10\ max) \rightarrow 99\%$

4  $time_{100\,procs} = \frac{10000t}{100} + 10t = 110t$

5  $Speedup_{100\,procs} = 10010t / 110t = 91(100\ max) \rightarrow 91\%$



## Ejemplo Ley de Amdahl

- ...y que ocurre si no hay un balanceo de carga ideal?
- Suponiendo que queremos realizar simultáneamente 2 sumas:
  - Suma de 10 variables escalares
  - Suma de una **matriz 2D con dimensión 100x100**

### Balanceo perfecto

- Para el caso de 100 procs, y distribución de carga equitativa
- ...cada procesador tendría el **1% de la carga total**
- ¿Que pasa si 1 de los 100 procs. tiene un poco más que el resto?
  - Si un proc. tiene un 2% en lugar el 1%
  - y con un 5% en lugar el 1%



## Ejemplo Ley de Amdahl

- Si un proc. tiene el 2%, el tendría que procesar  $2\% * 10000 = 200$  sumas
  - Los otros 99 procs. tendrían que procesar el resto:  $10000 - 200 = 9800$  sumas
- 1  $time_{100procs} = \max\left(\frac{9800t}{99}, \frac{200t}{1}\right) + 10t = 210t$
  - 2  $Speedup_{100procs} = 10010t / 210t = 48(100 \text{ max}) \rightarrow 48\%$



¿Qué impacto tiene el desbalanceo?

- El speedup cae al  $48\times$  en lugar del  $91\times$  observado
- Para un 5% de carga en un proc. tendría que procesar 500 sumas: **speedup=19×**



# ¿Por qué no se puede lograr el máximo speedup?

## Overheads

- **Gestión paralela:** el coste de crear y gestionar el paralelismo tiende a escalar exponencialmente con el número de hilos.
- **Comunicación:** depende principalmente del hardware (red de interconexión)
- **Sincronización:** depende modelo de programación y la eficiencia en la implementación
- **Desbalanceo de carga:** no existe suficiente trabajo para mantener todos nuestros recursos ocupados



# ¿Por qué no se puede lograr el máximo speedup?

- Todos estos sobrecostos tienden a **serializar** nuestro programa
  - pérdida correspondiente en escalabilidad
- El impacto exacto de cada aspecto depende de la arquitectura, modelo de programación y aplicación



# Elementos de los computadores paralelos

## ■ Hardware

- Multiples niveles de paralelismo: ILP, DLP, TLP
- Jerarquía de memoria: registros (~0.1ns), Caches L1/L2/L3 (1-10ns), DDR DRAM (80-100ns), memorias persistentes (<1 $\mu$ s), SSD (10-100 $\mu$ s), discos duros mecánicos (~10ms), cintas (~100ms)
- Red interconexión

## ■ Software

- Lenguajes de programación paralelo
- Programación concurrente

## ■ Aplicaciones software

- Algoritmos paralelos



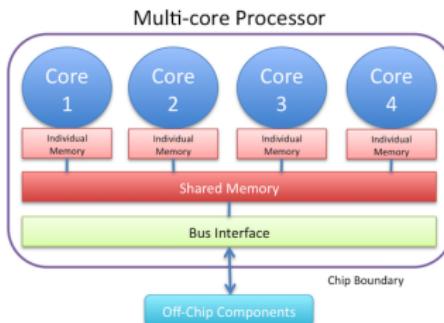
# Clusters

- Conjunto de computadores conectados sobre una red local (LAN) que funciona como un gran multiprocesador
- El rendimiento proviene de la suma de los procesadores de un chip



# Multicores

- Varios cores o CPUs por chip
- Cada procesador se denomina core



# Multicore, Multiprocesador y Clusters

- Nodo: un computador *standalone* que habitualmente contiene varias CPUs/procesadores/cores
- CPU/socket/procesador/core: varía dependiendo del contexto:
  - CPU: Central Processing Unit o componente de cómputo
  - *Socket* o zócalo, chip multicore

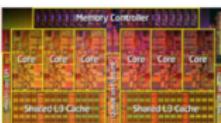


Supercomputer - each blue light is a node

Node - standalone

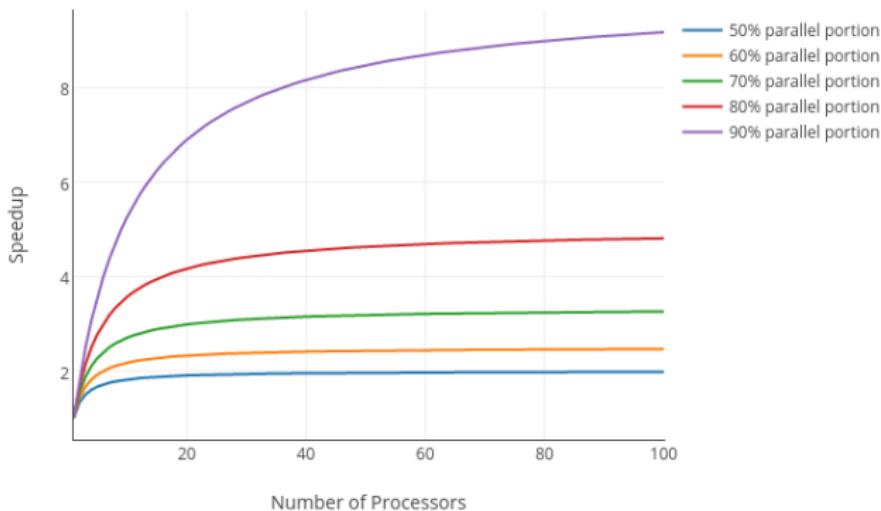
Von Neumann computer

CPU / Processor / Socket - each has multiple cores / processors.



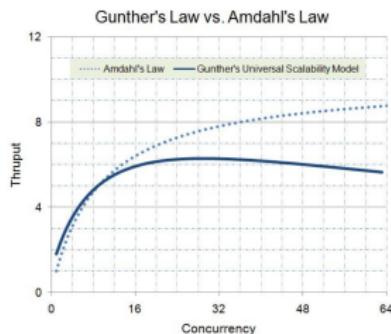
# Ley de Amdahl

Amdahl's Law



# Ley de Gunther

- Un extensión de la ley de Amdahl...
- ... que añade el coste de la comunicación



# Ley de Gustafson

- Establece que cualquier problema suficientemente grande puede ser paralelizable

$$S_{(n)} = n - f(n - 1)$$

- Donde
  - $p$  son el número de procesadores
  - $f$  la fracción paralelizable



# Ley de Gustafson

- Supongamos la fracción paralelizable definida como  $f = \frac{s}{s+p}$ 
  - $s$  es la parte secuencial
  - $p$  es la parte paralela
- La ley de Gustafson: más  $n$  permite ejec. problemas mayores
  - $t_{1proc} = s + n * p$
  - $t_{nprocs} = s + p$
  - $Speedup = \frac{s+n*p}{s+p} = f + n(1 - f) = n + f(1 - n)$



## Ley Gustafson vs Amdahl: ¿Contradicción?

- Ley Amdahl: speedup con volumen fijo de datos
- Ley Gustafson: problemas más grandes con más procesadores:  
**escalado**

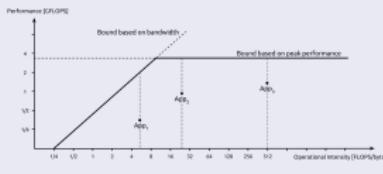


## Modelo Roofline

- Modelo intuitivo: **idea del rendimiento de una aplicación**
    - Para sistemas paralelos como un multicore, manicore u otras arquitecturas

## Modelo básico<sup>2</sup>

- Evaluación del rendimiento de apps en **fp**
  - Con métrica sencillas: intensidad aritmética, ancho-banda pico



<sup>2</sup>Roofline: an insightful visual performance model for multicore architectures, Communication to ACM, 2009



# Modelo Roofline

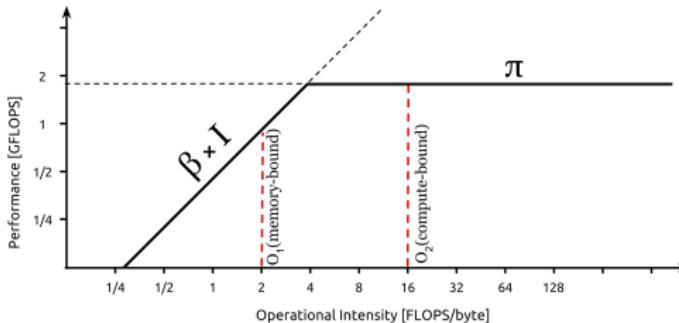
## Métricas básicas

- Trabajo  $W$  como número de operaciones de un kernel
  - op aritméticas *int, fp*
- Tráfico con memoria o  $Q$  que expresa el número de bytes que se transfieren con memoria del kernel
  - $Q$  es muy dependiente de la arquitectura y plataforma (niveles de cache)
- Intensidad aritmética o  $I$  definido como  $I = \frac{W}{Q}$



# Modelo Roofline

- El modelo se obtiene aplicando un análisis sencillo de cuellos de botella (GFLOPS vs ancho-banda)
- Donde el punto P se define como  $P = \min \left\{ \frac{\pi}{\beta * I} \right\}$ 
  - $\pi$  es rendimiento pico
  - $\beta$  es el ancho de banda pico
  - $I$  es la intensidad aritmética



# Modelo Roofline

- **¿Cómo poder estimar los valores  $\beta$  y  $\pi$ ?**
  - Evidentemente depende del sistema en cuestión

## En Linux

### Terminal #1

```
carlos@7picos:~ $ type cat /proc/cpuinfo
carlos@7picos:~ $ sudo dmidecode --type memory
```



## Modelo Roofline (cálculo de $\pi$ )

## Terminal #1

```
carlos@7picos:~ $ more /proc/cpuinfo
processor : 0
vendor_id : GenuineIntel
cpu family : 6
model : 60
model name : Intel(R) Xeon(R) CPU E3-1225 v3 @ 3.20GHz
stepping : 3
microcode : 0x1d
cpu MHz : 1466.632
cache size : 8192 KB
physical id : 0
siblings : 4
core id : 0
cpu cores : 4
apicid : 0
initial apicid : 0
fpu : yes
fpu_exception : yes
cpuid level : 13
wp : yes
flags : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dta
e sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm constant_tsc arch_perfmon pebs bts rep_good nopl
c cpuid aperf mperf pni pclmulqdq dtes64 monitor ds_cpl vmx smx est tm2 ssse3 sdbg fma cx16 xtrp
t2xapic move popcnt aes xsave avx f16c rdrandlahf_lm abm cpuid_fault epb invpcid_single pt1 tm
ority ept vpid fsgsbase tsc_adjust bm1l avx2 smpem bm12 erms invpcid xsaveopt dtherm ida arat pln p
bugs : cpu_meltdown spectre_v1 spectre_v2 spec_store_bypass l1tf
bogomips : 6399.95
clflush size : 64
cache_alignment : 64
address sizes : 39 bits physical, 48 bits virtual
power management:
```



# Modelo Roofline (cálculo de $\pi$ )

- $\pi = 3,2[\text{GHz}] * 8[\text{SIMD} - \text{width}] * 1[\text{issueFLOP/clock}] * 4[\text{cores}] = 102,40\text{GFlops}$ 
  - Número de cores=4
  - Frecuencia=3.2GHz
  - SIMD=8fp(avx2 y una unidad FMA)

Terminal #1

```
carlos@7picos:~ $ more /proc/cpuinfo
...
processor    : 3
...
model name   : Intel(R) Xeon(R) CPU E3-1225 v3 @ 3.20GHz
...
flags        : ... avx ... avx2 ...
...
```



## Modelo Roofline (cálculo de $\beta$ )

Terminal #1

```
carlos@tpicos: ~ $ sudo dmidecode --type memory
# dmidecode 3.1
Getting SMBIOS data from sysfs.
SMBIOS 2.7 present.

Handle 0x004C, DMI type 16, 23 bytes
Physical Memory Array
  Location: System Board Or Motherboard
  Use: System Memory
  Error Correction Type: Single-bit ECC
  Maximum Capacity: 32 GB
  Error Information Handle: Not Provided
  Number Of Devices: 4

Handle 0x004D, DMI type 17, 34 bytes
Memory Device
  Array Handle: 0x004C
  Error Information Handle: Not Provided
  Total Width: 72 bits
  Data Width: 64 bits
  Size: 8192 MB
  Form Factor: DIMM
  Set: None
  Locator: DIMMA1
  Bank Locator: P0_Node0_Channel0_Dimm0
  Type: DDR3
  Type Detail: Synchronous
  Speed: 1600 MT/s
  Manufacturer: Kingston
  Serial Number: D23C077
  Asset Tag: 9876543210
  Part Number: 9965525-024.A0OLF
  Rank: 2
  Configured Clock Speed: 1600 MT/s
```



# Modelo Roofline (cálculo de $\beta$ )

- $4 \times 8\text{GB} = 32\text{GB}$
- Speed: 1600 MT/s y Data Width: 64 bits  $\Rightarrow$   
$$\beta = 1600 * 64/8 = 12,800\text{GB/s}$$

Terminal #1

```
carlos@7picos:~ $ sudo dmidecode --type memory
# dmidecode 3.1

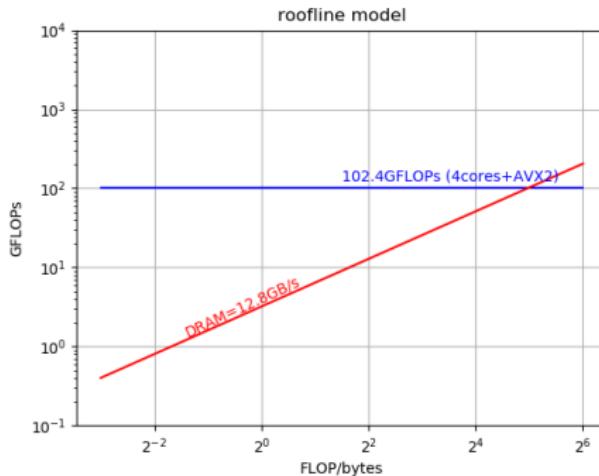
Handle 0x004C, DMI type 16, 23 bytes
Physical Memory Array
  Location: System Board Or Motherboard
  Use: System Memory
  Error Correction Type: Single-bit ECC
  Maximum Capacity: 32 GB
  Error Information Handle: Not Provided
  Number Of Devices: 4

Handle 0x004D, DMI type 17, 34 bytes
Memory Device
...
  Data Width: 64 bits
  Size: 8192 MB
...
  Speed: 1600 MT/s
...
```



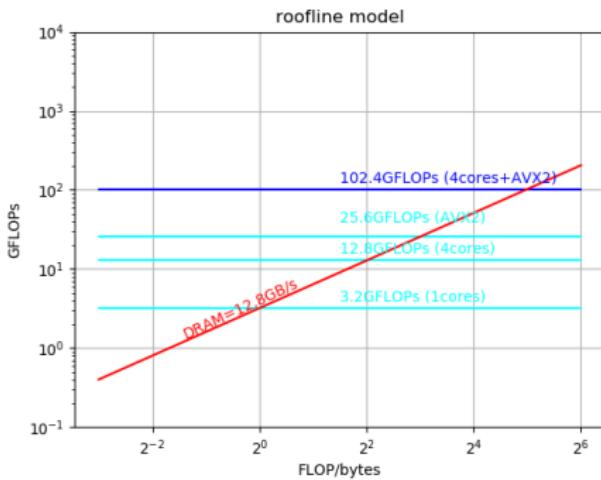
# Modelo Roofline (cálculo de $\pi$ y $\beta$ )

- ¿Que rendimiento sería esperable si tengo un código con **intensidad aritmética de 1?**
- ¿Y con  $I = 4$ ?



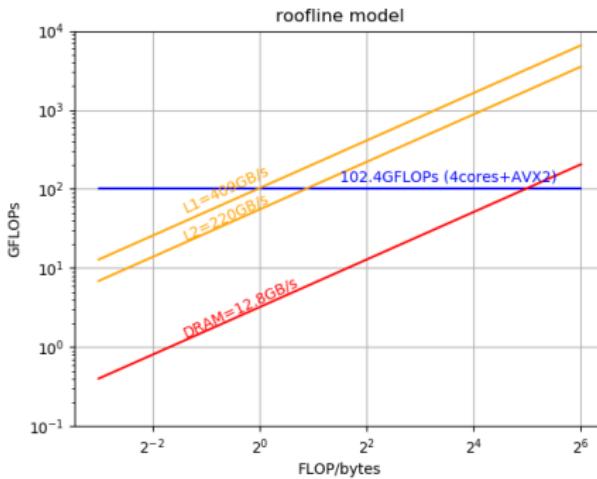
# Modelo Roofline (paralelización)

- ¿Merece la pena paralelizar en este equipo para  $I = 1$  o  $I = 4$ ?



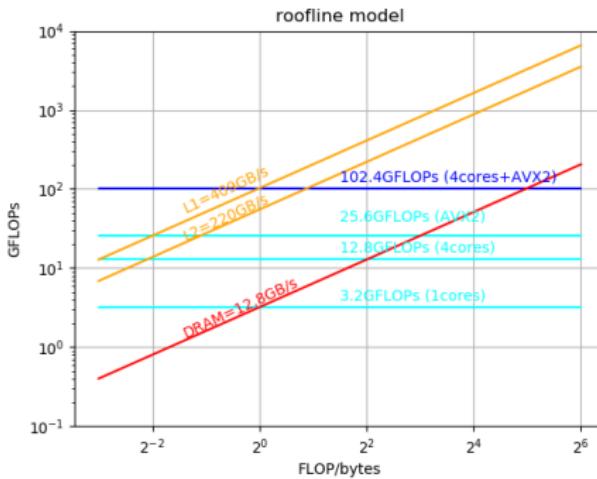
## Modelo Roofline (localidad)

- ¿Merece la pena paralelizar en este equipo para  $I = 1$  o  $I = 4$ ?
- **¿Y si mejoramos y explotamos la jerarquía de memoria?**



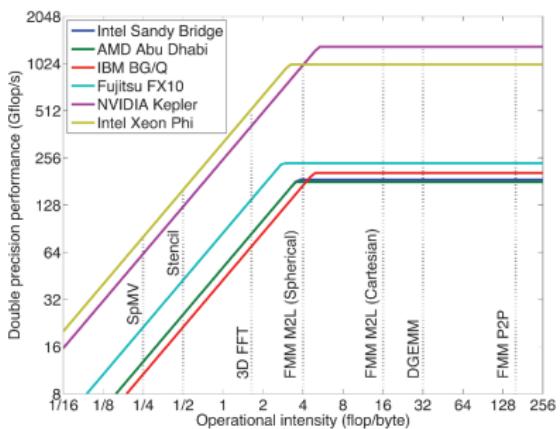
## Modelo Roofline

- ¿Merece la pena paralelizar en este equipo para  $I = 1$  o  $I = 4$ ?
  - **¿Y si mejoramos y explotamos la jerarquía de memoria?**



# Modelo Roofline

- Nos permite también evaluar el comportamiento de nuestro algoritmo y su rendimiento en diferentes arquitecturas <sup>3</sup>



<sup>3</sup>How Will the Fast Multipole Method Fare in the Exascale Era. SIAM News 2013

