

# Introducción

95.57/75.03 Organización del computador

---

**Docentes:** Patricio Moreno y Adeodato Simó

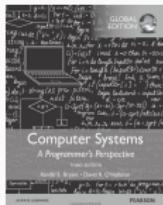
1.<sup>er</sup> cuatrimestre de 2020

Última modificación: Mon Apr 6 17:32:48 2020 -0300

Facultad de Ingeniería (UBA)

# Créditos

Para armar las presentaciones del curso nos basamos en:



R. E. Bryant and D. R. O'Hallaron, *Computer systems: a programmer's perspective*, Third edition, Global edition.

Boston Columbus Hoboken Indianapolis New York San Francisco Cape Town: Pearson, 2016.



D. A. Patterson and J. L. Hennessy, *Computer organization and design: the hardware/software interface*, RISC-V edition. Cambridge, Massachusetts: Morgan Kaufmann Publishers, an imprint of Elsevier, 2018.



J. L. Hennessy and D. A. Patterson, *Computer architecture: a quantitative approach*. 2019.

# Tabla de contenidos

1. Desde la ENIAC a las *phablets*
2. Ocho grandes ideas en arquitectura de computadoras
3. Cinco grandes realidades en computación
4. Software/Hardware interface
  - Desde tus dedos
  - Bajo tu manos
5. Rendimiento
  - Medición
  - Principio, regla, ley
6. Potencia, unicores, y multicores
7. Trampas y falacias

# Tabla de contenidos

1. Desde la ENIAC a las *phablets*
2. Ocho grandes ideas en arquitectura de computadoras
3. Cinco grandes realidades en computación
4. Software/Hardware interface
  - Desde tus dedos
  - Bajo tu manos
5. Rendimiento
  - Medición
  - Principio, regla, ley
6. Potencia, unicores, y multicores
7. Trampas y falacias

# La revolución de las computadoras / Revolución Digital

- Comienza en 1945 con la ENIAC (tubos de vacío)
  - Ocupa una habitación entera
- En 1949 se ejecuta el primer programa almacenado en la computadora
  - Anteriormente se utilizaban medios físicos para programar
- 1957, aparece FORTRAN
- 1959, primer transistor MOSFET
- 1965, Ley de Moore
- 1969, ARPANET, Empieza el desarrollo de UNIX
- 1971, Intel 4004
- 1975, Primera computadora personal (Altair 8800), UNIX
- 1983, GNU
- 1991, GNU/Linux

# La revolución de las computadoras / Revolución Digital

---

- El avance en la tecnología sigue la Ley de Moore
- Empieza a extenderse su uso
  - Computadoras en automóviles
  - Computadoras en teléfonos celulares
  - Computadoras para decodificar el genoma humano
  - Computadoras como base de la WWW
  - Computadoras como motores de búsqueda
- Las computadoras están en todos lados

# Clases de computadoras

---

## Computadoras personales

- De propósito general, ejecutan software variado
- Su desarrollo está sujeto a una relación de costo/desempeño
- De tamaño pequeño, a lo sumo como una PC de escritorio

## Servidores

- Se basan en la existencia de una red
- De gran capacidad, alto rendimiento, y confiables
- De tamaños variables, desde una PC de escritorio a un edificio

# Clases de computadoras

## Supercomputadoras

- Son las de mayores capacidades
- Uso exclusivo en ingeniería y ciencia

## Computadoras embebidas

- Normalmente no se las ve, son un componente dentro de un equipo
- Diseñadas con restricciones de potencia/rendimiento/costo
- Típicamente ejecutan siempre los mismos procesos

# Era PostPC

---

## Dispositivos Móviles Personales (PMD)

- Funcionan con baterías
- Necesitan internet
- “Económicos”
- Teléfonos inteligentes, tablets, relojes, etc.

## Cloud Computing

- Computadoras a gran escala (WCS, Warehouse Scale Computer)
- Software as a Service (SaaS)
- Parte de los programas se ejecutan en un PMD, otra parte en la computadora de una empresa, en "La Nube"
- Amazon, Google, etc.

# Tabla de contenidos

---

1. Desde la ENIAC a las *phablets*
2. Ocho grandes ideas en arquitectura de computadoras
3. Cinco grandes realidades en computación
4. Software/Hardware interface
  - Desde tus dedos
  - Bajo tu manos
5. Rendimiento
  - Medición
  - Principio, regla, ley
6. Potencia, unicores, y multicores
7. Trampas y falacias

# Ocho grandes ideas

- 1 Diseñar teniendo en cuenta la **Ley de Moore**
- 2 Usar la **abstracción** para simplificar el diseño
- 3 Hacer **rápido** el caso común
- 4 Desempeño por **paralelismo**
- 5 Desempeño por **pipelining**
- 6 Desempeño por **predicción**
- 7 **Jerarquía** de memorias
- 8 **Confiabilidad** por redundancia

# Tabla de contenidos

---

1. Desde la ENIAC a las *phablets*
2. Ocho grandes ideas en arquitectura de computadoras
3. Cinco grandes realidades en computación
4. Software/Hardware interface
  - Desde tus dedos
  - Bajo tu manos
5. Rendimiento
  - Medición
  - Principio, regla, ley
6. Potencia, unicores, y multicores
7. Trampas y falacias

# Realidad #1

los ints no son los enteros, los floats no son los reales

## Ejemplo 1: ¿es $x*x \geq 0$ ?

- floats: sí
- ints: ...
  - $40000 * 40000 \Rightarrow$   
1600000000
  - $50000 * 50000 \Rightarrow$   
???

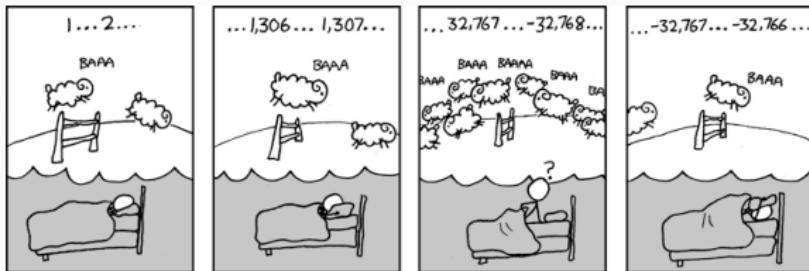


Imagen: xkcd.com/571

## Ejemplo 2: ¿es $(x+y)+z = x+(y+z)$ ?

- ints: sí
- floats: sí
  - $(1E20 + -1E20) + 3,14 \Rightarrow 3,14$
  - $1E20 + (-1E20 + 3,14) \Rightarrow ???$

# Aritmética de la computadora

- **No genera valores aleatorios**
  - Las operaciones tienen importantes propiedades matemáticas
- **No se pueden asumir válidas todas las propiedades “usuales”**
  - Por ser representaciones finitas
  - Las operaciones con enteros satisfacen las propiedades de los anillos
    - Comutatividad, asociatividad, distributividad
  - Las operaciones con flotantes satisfacen las propiedades de orden
    - Monotonidad, valores de los signos
- **Observaciones**
  - ¿Qué es válido en qué contexto?
  - Es importante para programadores de aplicaciones serias

# Realidad #2

## tenés que saber assembly

- **Seguramente nunca escribas código assembly**
  - Los compiladores son mejores que vos, y más pacientes
- **Pero, entenderlo es clave para el modelo de ejecución de la máquina**
  - ¿Cómo se comporta el programa cuando hay un bug?
    - Los modelos de lenguajes de alto nivel se rompen
  - Tunear el desempeño de un programa
    - ¿Qué optimizaciones realiza (o no) el compilador?
    - Entender las fuentes de ineficiencia de un programa
  - Van a desarrollar software que corre en otro sistema
    - El compilador tiene como objetivo el código de máquina
    - Los sistemas operativos manejan el estado de los procesos
  - Malware: crearlo o combatirlo
    - x86 / x86-64 es el lenguaje preferido

# Realidad #3: la Memoria importa la RAM es una abstracción

---

- **La memoria no es infinita**
  - Tiene que ser allocada y administrada
  - Es un limitante en muchas aplicaciones
- **Los bugs relacionados a ella son especialmente...**
  - Los efectos son distantes en tiempo y espacio..
  - A veces parecen aleatorios
- **El desempeño de la memoria no es uniforme**
  - Los efectos de la cache y la memoria virtual afectan enormemente el desempeño de un programa
  - Adaptar un programa a las características de la memoria de un sistema puede llevar a grandes mejoras en velocidad

## Ejemplo de un bug por mal manejo de la memoria

```
1 typedef struct {
2     int a[2];
3     double d;
4 } struct_t;
5
6 double fun(int i) {
7     volatile struct_t s;
8     s.d = 3.14;
9     s.a[i] = 1073741824; /* posiblemente fuera de los límites */
10    return s.d;
11 }
```

```
fun(0) -> 3.14
fun(1) -> 3.14
fun(2) -> 3.1399998664856
fun(3) -> 2.00000061035156
fun(4) -> 3.14
fun(6) -> Segmentation Fault
```

# Errores por mala administración de la memoria

- **C y C++ no dan ninguna protección**
  - Referencia a elementos por afuera de un arreglo
  - Punteros con valores inválidos
  - Abusos en `malloc()/free()`
- **Puede llevar a bugs con consecuencias terribles**
  - El efecto depende del compilador y del sistema
  - Actúan a la distancia
    - La memoria corrupta puede no guardar relación lógica con el punto de entrada del bug
    - El efecto se puede observar mucho después de haberlo producido
- **¿Cómo se lidia con esto?**
  - Salen corriendo y nunca más usan C/C++
  - Programan en Java, Ruby, Python, ...
  - Aprenden a programar y los efectos de su código
  - Usan (o desarrollan) herramientas para detectar los errores (por ejemplo, Valgrind)

# Realidad #4: hay más al desempeño que la complejidad asintótica

---

- ¡Los factores constantes también importan!
- Ni siquiera contar la cantidad exacta de instrucciones predice el desempeño
  - Depende de cómo esté escrito el código
  - Se puede optimizar un programa en muchos niveles: algorítmico, representaciones de datos, procedimientos, e incluso ciclos.
- Para poder optimizar, se tiene que entender el sistema
  - Cómo los programas se compilan y ejecutan
  - Cómo medir el desempeño de un programa y detectar cuellos de botella
  - Cómo mejorar el desempeño sin abandonar las buenas prácticas de programación (modularidad, generalidad, etc.)

# Ejemplo

```
1 void copyij(int src[2048][2048],  
2             int dst[2048][2048])  
3 {  
4     int i, j;  
5     for (i = 0; i < 2048; i++)  
6         for (j = 0; j < 2048; j++)  
7             dst[i][j] = src[i][j];  
8 }
```

```
1 void copyji(int src[2048][2048],  
2             int dst[2048][2048])  
3 {  
4     int i, j;  
5     for (j = 0; j < 2048; j++)  
6         for (i = 0; i < 2048; i++)  
7             dst[i][j] = src[i][j];  
8 }
```

2.0 GHz Intel Core i7 Haswell

# Ejemplo

```
1 void copyij(int src[2048][2048],  
2             int dst[2048][2048])  
3 {  
4     int i, j;  
5     for (i = 0; i < 2048; i++)  
6         for (j = 0; j < 2048; j++)  
7             dst[i][j] = src[i][j];  
8 }
```

4.3 ms

2.0 GHz Intel Core i7 Haswell

```
1 void copyji(int src[2048][2048],  
2             int dst[2048][2048])  
3 {  
4     int i, j;  
5     for (j = 0; j < 2048; j++)  
6         for (i = 0; i < 2048; i++)  
7             dst[i][j] = src[i][j];  
8 }
```

81.8 ms

- Organización de la jerarquía de memoria
- El desempeño depende de los patrones de acceso a la memoria

# Realidad #5:

## las computadoras hacen más que ejecutar programas

---

- **Necesitan mover datos**
  - El sistema I/O es crítico
- **Se comunican a través de redes**
  - Muchos problemas surgen en presencia de una red
    - Concurrencia
    - Copias de medios poco fiables
    - Compatibilidad entre plataformas
    - El sistema se vuelve más complejo

# Tabla de contenidos

---

1. Desde la ENIAC a las *phablets*
2. Ocho grandes ideas en arquitectura de computadoras
3. Cinco grandes realidades en computación
4. Software/Hardware interface
  - Desde tus dedos
  - Bajo tu manos
5. Rendimiento
  - Medición
  - Principio, regla, ley
6. Potencia, unicores, y multicores
7. Trampas y falacias

# Tabla de contenidos

- 
1. Desde la ENIAC a las *phablets*
  2. Ocho grandes ideas en arquitectura de computadoras
  3. Cinco grandes realidades en computación
  4. Software/Hardware interface

Desde tus dedos

Bajo tu manos

5. Rendimiento

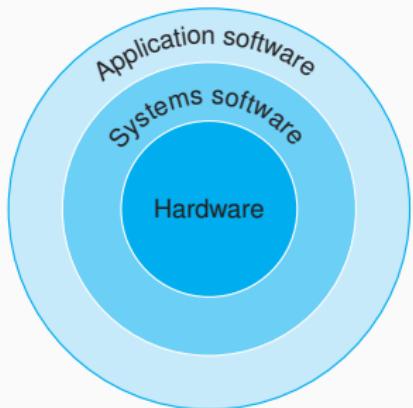
Medición

Principio, regla, ley

6. Potencia, unicores, y multicores

7. Trampas y falacias

# Por debajo del programa



- *Application software*
  - Escrito en un lenguaje de alto nivel (HLL)
- *System software*
  - Compilador: traduce código en HLL a código de máquina
  - Sistema Operativo: código de servicio
    - Maneja entrada/salida
    - Gestiona la memoria y el almacenamiento
    - Programa tareas y gestiona los recursos compartidos
- *Hardware*
  - Procesador, memoria, controladores I/O

# Variantes del lenguaje en el código de un programa

## ■ Lenguaje de alto nivel

- Nivel de abstracción cercano al dominio del problema
- Mejora la productividad y portabilidad

High-level  
language  
program  
(in C)

```
swap(int v[], int k)
{int temp;
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
}
```

Compiler

Assembly  
language  
program  
(for RISC-V)

```
swap:
slli x6, x11, 3
add x6, x10, x6
ld x5, 0(x6)
ld x7, 8(x6)
sd x7, 8(x6)
sd x5, 8(x6)
jalr x0, 0(x1)
```

## ■ Lenguaje assembly

- Representación textual de las instrucciones

## ■ Lenguaje de máquina

- Representación del hardware
- Dígitos binarios (bits)
- Instrucciones y datos codificados

Una instrucción es una sentencia de código assembly que representa únicamente una instrucción del lenguaje de máquina: ceros y unos. Están definidas en la ISA.

Por ejemplo: `jalr x0, 0(x1)`.

Binary machine  
language  
program  
(for RISC-V)

```
0000000000110101011001001100010011
00000000011001010000001100110011
0000000000000000110011001010000011
00000000010000011001100110000000011
0000000001110011001100000001000011
0000000001010011001101000001000011
0000000001010011001101000001000011
00000000000000000000000000110000000111
```

Assembler

# Tabla de contenidos

- 
- 1. Desde la ENIAC a las *phablets*
  - 2. Ocho grandes ideas en arquitectura de computadoras
  - 3. Cinco grandes realidades en computación
  - 4. Software/Hardware interface

Desde tus dedos

Bajo tu manos

- 5. Rendimiento

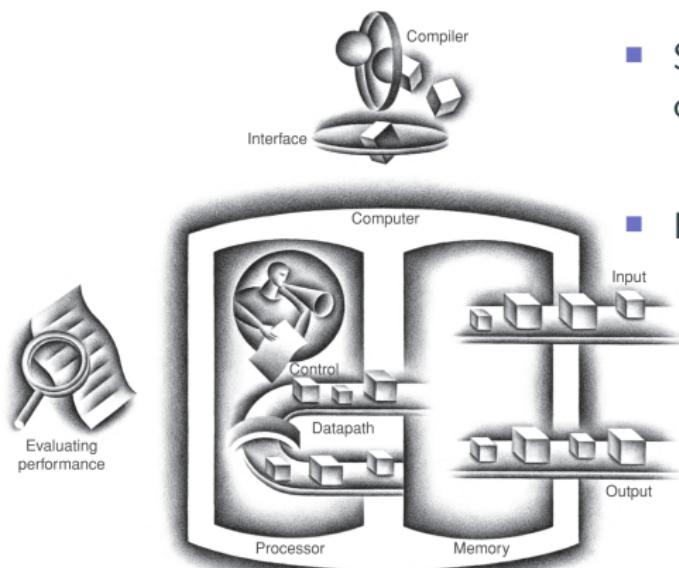
Medición

Principio, regla, ley

- 6. Potencia, unicores, y multicores

- 7. Trampas y falacias

# Componentes de una computadora



- Son los mismos componentes para cualquier computadora
  - De escritorio, servidores, embebidas
- Entrada/Salida incluye
  - Dispositivos de interfaz con el usuario
    - Display, teclado, mouse, parlantes
  - Dispositivos de almacenamiento
    - Discos rígidos, CD/DVD, flash
  - Adaptadores de red
    - Para comunicarse con otras computadoras

Imagen de 'Computer Organization and Design: The Hardware/Software Interface' de 'Patterson & Hennessy'

# Pantalla táctil

- Dispositivo PostPC
- Reemplaza al teclado Y al mouse
- Hay 2 tipos, resistivas y capacitivas
  - La mayor parte de las tablets y teléfonos inteligentes utiliza capacitivas
  - Las capacitivas permiten multi-touch

# Procesador

## Apple A5



- **Datapath**
  - lleva a cabo las operaciones sobre los datos
- **Control**
  - secuencia al *datapath*, la memoria, etc.
- **Memoria cache**
  - Pequeña memoria SRAM veloz para acceso inmediato a los datos

# Abstracción

---

- La abstracción reduce la complejidad general
  - Oculta los detalles de bajo nivel (Siempre “¿qué?”, y no “¿cómo?”)
- Instruction set architecture (ISA) — Arquitectura del Set de Instrucciones
  - La interfaz hardware/software
- Application binary interface (ABI)
  - LA ISA con el agregado de la interface que da el software del sistema
- Implementación
  - Es el hardware que obedece a la abstracción de la arquitectura

# Tabla de contenidos

---

1. Desde la ENIAC a las *phablets*
2. Ocho grandes ideas en arquitectura de computadoras
3. Cinco grandes realidades en computación
4. Software/Hardware interface
  - Desde tus dedos
  - Bajo tu manos
5. Rendimiento
  - Medición
  - Principio, regla, ley
6. Potencia, unicores, y multicores
7. Trampas y falacias

# Tabla de contenidos

---

1. Desde la ENIAC a las *phablets*
2. Ocho grandes ideas en arquitectura de computadoras
3. Cinco grandes realidades en computación
4. Software/Hardware interface

Desde tus dedos

Bajo tu manos

5. Rendimiento

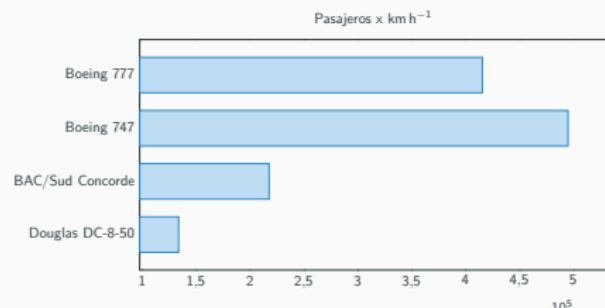
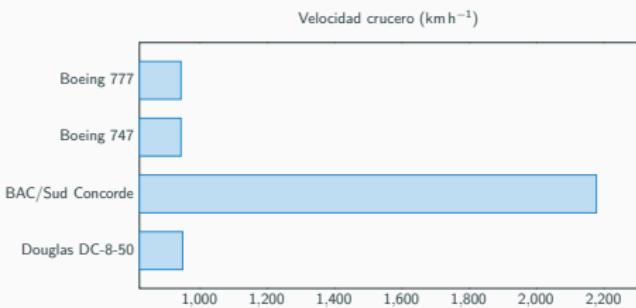
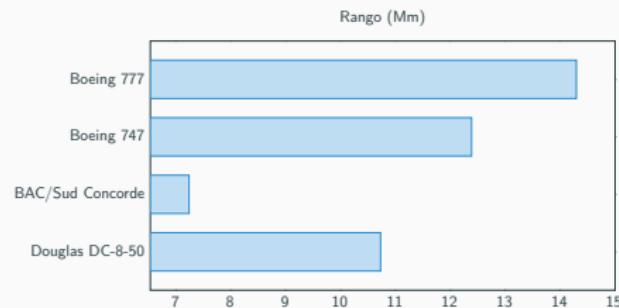
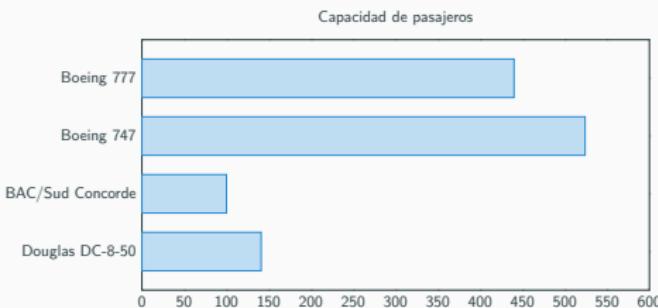
Medición

Principio, regla, ley

6. Potencia, unicores, y multicores
7. Trampas y falacias

# Definición de rendimiento

- ¿Qué avión tiene el mejor rendimiento?



# Tiempo de respuesta y throughput

- Tiempo de respuesta
  - Tiempo que toma realizar una tarea
  - También: tiempo de ejecución
- Throughput
  - Cantidad de trabajo realizado por unidad de tiempo
- ¿Cómo se ve afectado el tiempo de respuesta y el throughput
  - al cambiar el procesador por una versión más veloz?
  - al agregar más procesadores?

# Rendimiento relativo

- Definir rendimiento como  $\frac{1}{\text{Tiempo de respuesta}}$
- “X es  $n$  veces más rápido que Y”

$$\frac{\text{rendimiento}_X}{\text{rendimiento}_Y} = \frac{\frac{1}{\text{Tiempo de respuesta}_Y}}{\frac{1}{\text{Tiempo de respuesta}_X}} = n$$

## Ejemplo (Tiempo necesario para ejecutar un programa)

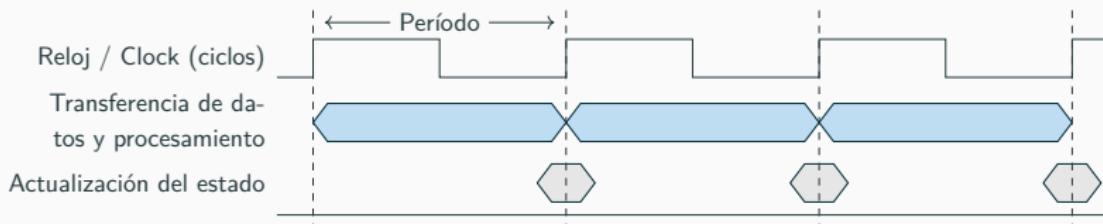
- 10 s en A, 15 s en B
- $\frac{\text{Tiempo de ejecución}_B}{\text{Tiempo de ejecución}_A} = \frac{15 \text{ s}}{10 \text{ s}} = 1,5$
- A es 1,5 veces más rápido que B

# Medición del tiempo de ejecución

- Tiempo transcurrido
  - Tiempo de respuesta *total*, teniendo en cuenta *todos* los aspectos
    - Procesamiento, I/O, Overhead del SO, tiempos de espera
  - Determina el desempeño del sistema
- Tiempo de CPU
  - Tiempo utilizado en procesar una tarea
    - Descarta el tiempo de I/O y los tiempos de otras tareas
  - Es el tiempo de CPU del usuario y el tiempo de CPU del sistema
  - El rendimiento de la CPU y del sistema afectan en modos distintos a programas distintos

# Clocking de la CPU

- La operación del hardware digital se rige por un reloj (*clock*) de frecuencia fija



- Período del reloj: duración de un ciclo del reloj
  - se mide en tiempo (segundos)
  - por ejemplo:  $250 \text{ ps} = 0.25 \text{ ns} = 250 \times 10^{-12} \text{ s}$
- Frecuencia del reloj (rate): ciclos por segundos
  - se miden ciclos por segundo (hertz)
  - por ejemplo:  $4.0 \text{ GHz} = 4000 \text{ MHz} = 4.0 \times 10^9 \text{ Hz}$

# Tiempo de CPU

$$\text{Tiempo de CPU} = (\text{Ciclos de reloj de CPU}) \times (\text{Período del reloj})$$

$$= \frac{\text{Ciclos de reloj de CPU}}{\text{Frecuencia del reloj}}$$

- El rendimiento mejora
  - al reducir los ciclos de reloj
  - al aumentar la frecuencia del reloj

es una relación de compromiso (*trade off*) entre la frecuencia del reloj, y la cantidad de ciclos.

## Ejemplo: Tiempo de CPU

- Computadora A: reloj a 2 GHz, tiempo de CPU en 10 s
- Diseño de la computadora B
  - se requiere un tiempo de CPU de 6 s
  - se puede aumentar la frecuencia del reloj, pero incrementa la cantidad de ciclos en 1,2 veces ( $1,2 \times (\text{ciclos de reloj})$ ).
- ¿Qué tan rápido tiene que ser el reloj de la computadora B para cumplir el requerimiento?

$$f_B = \frac{\text{Ciclos de reloj}_B}{\text{Tiempo de CPU}_B} = \frac{1,2 \times \text{Ciclos de reloj}_A}{6 \text{ s}}$$

$$\text{Ciclos de reloj}_A = (\text{tiempo de CPU})_A \times f_A$$

$$= 10 \text{ s} \times 2 \text{ GHz} = 20 \times 10^9$$

$$f_B = \frac{1,2 \times 20 \times 10^9}{6 \text{ s}} = \frac{24 \times 10^9}{6 \text{ s}} = 4 \text{ GHz}$$

# Recuento de instrucciones y CPI

Ciclos de reloj = (Recuento de instrucciones) × (Ciclos por instrucción)

Tiempo de CPU = (Recuento de instrucciones) × CPI × (Período del reloj)

$$= \frac{(\text{Recuento de instrucciones}) \times \text{CPI}}{\text{Frecuencia del reloj}}$$

- Recuento de instrucciones para un programa
  - Es la cantidad total de instrucciones del programa
  - Queda determinado por el programa, la ISA y el compilador
- Ciclos de reloj por instrucción (CPI) promedio
  - Determinado por el hardware del CPU
  - Si distintas instrucciones tiene distintos CPI
    - El CPI promedio varía con el *instruction mix*

## Ejemplo: CPI

- Computadora A: Período del reloj = 250 ps, CPU = 2,0
- Computadora B: Período del reloj = 500 ps, CPU = 1,2
- Misma ISA
- ¿Cuál es más rápida, y por cuánto?

$$\begin{aligned}t_{\text{CPU}_A} &= (\text{Recuento de instrucciones})_A \times \text{CPI}_A \times T_A \\&= I \times 2,0 \times 250 \text{ ps} = I \times 500 \text{ ps}\end{aligned}$$

$$\begin{aligned}t_{\text{CPU}_B} &= (\text{Recuento de instrucciones})_B \times \text{CPI}_B \times T_B \\&= I \times 1,2 \times 500 \text{ ps} = I \times 600 \text{ ps}\end{aligned}$$

$$\frac{t_{\text{CPU}_B}}{t_{\text{CPU}_A}} = \frac{I \times 600 \text{ ps}}{I \times 500 \text{ ps}} = 1,2$$

## Ejemplo: CPI

- Computadora A: Período del reloj = 250 ps, CPU = 2,0
- Computadora B: Período del reloj = 500 ps, CPU = 1,2
- Misma ISA
- ¿Cuál es más rápida, y por cuánto?

$$\begin{aligned}t_{\text{CPU}_A} &= (\text{Recuento de instrucciones})_A \times \text{CPI}_A \times T_A \\&= I \times 2,0 \times 250 \text{ ps} = I \times 500 \text{ ps}\end{aligned}$$

A es más rápida

$$\begin{aligned}t_{\text{CPU}_B} &= (\text{Recuento de instrucciones})_B \times \text{CPI}_B \times T_B \\&= I \times 1,2 \times 500 \text{ ps} = I \times 600 \text{ ps}\end{aligned}$$

$$\frac{t_{\text{CPU}_B}}{t_{\text{CPU}_A}} = \frac{I \times 600 \text{ ps}}{I \times 500 \text{ ps}} = 1,2$$

## Ejemplo: CPI

- Computadora A: Período del reloj = 250 ps, CPU = 2,0
- Computadora B: Período del reloj = 500 ps, CPU = 1,2
- Misma ISA
- ¿Cuál es más rápida, y por cuánto?

$$\begin{aligned}t_{\text{CPU}_A} &= (\text{Recuento de instrucciones})_A \times \text{CPI}_A \times T_A \\&= I \times 2,0 \times 250 \text{ ps} = I \times 500 \text{ ps}\end{aligned}$$

A es más rápida

$$\begin{aligned}t_{\text{CPU}_B} &= (\text{Recuento de instrucciones})_B \times \text{CPI}_B \times T_B \\&= I \times 1,2 \times 500 \text{ ps} = I \times 600 \text{ ps}\end{aligned}$$

$$\frac{t_{\text{CPU}_B}}{t_{\text{CPU}_A}} = \frac{I \times 600 \text{ ps}}{I \times 500 \text{ ps}} = 1,2$$

estas veces más rápida

## CPI en detalle

Si instrucciones distintas requieren una cantidad diferente de ciclos para finalizar

**CPI<sub>i</sub>**: número medio de ciclos de reloj para la instrucción *i*

**IC<sub>i</sub>**: número de veces que se ejecuta la instrucción *i* en un programa

Entonces

$$\text{Ciclos de reloj} = \sum_{i=1}^n (\text{CPI}_i \times \text{IC}_i)$$

$$\text{Tiempo de CPU} = \sum_{i=1}^n (\text{CPI}_i \times \text{IC}_i) \times \text{Duración del ciclo de reloj}$$

$$\text{CPI} = \frac{\sum_{i=1}^n (\text{CPI}_i \times \text{IC}_i)}{\#\text{Instrucciones}} = \sum_{i=1}^n \left( \text{CPI}_i \times \underbrace{\frac{\text{IC}_i}{\#\text{Instrucciones}}}_{\text{Frecuencia relativa}} \right)$$

## Ejemplo: CPI

- Datos para secuencias de código compiladas utilizando instrucciones en las clases A, B, y C

Clase	A	B	C
CPI para la clase	1	2	3
IC en secuencia 1	2	1	2
IC en secuencia 2	4	1	1

- Secuencia 1:  $IC = 5$ 
  - Ciclos de reloj  
 $= 2 \times 1 + 1 \times 2 + 2 \times 3$   
 $= 10$
  - $CPI$  promedio  $= \frac{10}{5} = 2,0$
- Secuencia 2:  $IC = 6$ 
  - Ciclos de reloj  
 $= 4 \times 1 + 1 \times 2 + 1 \times 3$   
 $= 9$
  - $CPI$  promedio  $= \frac{9}{6} = 1,5$

# MIPS

- MIPS significa **Millones de Instrucciones Por Segundo**  
*Tasa de ejecución de instrucción*
- Es otra figura de mérito, utilizada muchas veces porque a mayores valores corresponden mayores “desempeños”<sup>1</sup>.
- Su cálculo se reduce a la siguiente ecuación:

$$\begin{aligned} \text{MIPS} &= \frac{\text{IC}}{\text{tiempo} \times 10^6} \\ &= \frac{\text{IC}}{\frac{\text{IC} \times \text{CPI}}{\text{frecuencia de reloj}} \times 10^6} \\ &= \frac{\text{frecuencia del reloj}}{\text{CPI} \times 10^6} \end{aligned}$$

- No sirve para comparar distintos sets de instrucciones
- Depende del programa

<sup>1</sup>la única métrica incuestionable del desempeño es el tiempo de respuesta

# Resumen de rendimiento

$$\text{Tiempo de CPU} = \frac{\#\text{Instrucciones}}{\text{Programa}} \times \frac{\text{Ciclos de reloj}}{\text{Instrucción}} \times \frac{\text{Segundos}}{\text{Ciclo de reloj}}$$

- El rendimiento depende de
  - Algoritmo: incide en IC, posiblemente también CPI
  - Lenguaje de programación: IC, CPI
  - Compilador: IC, CPI
  - ISA: incide en IC, CPI,  $T_c$

# Tabla de contenidos

---

1. Desde la ENIAC a las *phablets*
2. Ocho grandes ideas en arquitectura de computadoras
3. Cinco grandes realidades en computación
4. Software/Hardware interface
  - Desde tus dedos
  - Bajo tu manos
5. Rendimiento
  - Medición
  - Principio, regla, ley
6. Potencia, unicores, y multicores
7. Trampas y falacias

# Principio de localidad

*Los programas tienden a usar los datos e instrucciones que usaron recientemente*

- Como regla se dice que un programa pasa el 90 % de su tiempo de ejecución en el 10 % de sus líneas de código (90/10).
- En base a las instrucciones y los datos utilizados recientemente, se puede predecir qué recursos se utilizarán próximamente.
- Hay dos tipos de localidad:

**Localidad espacial** Los elementos cuyas direcciones son cercanas suelen ser referenciados.

**Localidad temporal** Elementos utilizados recientemente es probable que sean reutilizados.

# Regla del caso común

*Siempre favorecer el caso frecuente (vs casos no frecuentes)*

- Se cumple la regla básica 90/10: optimizar esas 10 líneas.
- Funciona siempre: extrapolar a todo
  - Un aspecto importante de los teléfonos celulares es el consumo energético (energía disipada): si un caso muy extraño consume mucha energía, y muchos comunes no tanta, reducir esa *no tanta*.
- A menudo el caso frecuente es más fácil de optimizar (aunque degrade un poco el caso infrecuente).

## Ley de Amdahl

*El aumento en rendimiento debido a una mejora en un componente del sistema está limitado por la fracción de uso de ese componente.*

- Cuantifica la regla del caso común

Dado un tiempo de ejecución  $T_0$ , y de éste un componente a mejorar insume una fracción  $\alpha$ . ¿Cuál es la mejora general si se mejora el rendimiento de ese componente por un factor  $k$ ? Para el componente, el tiempo original es  $\alpha T_0$ , y el nuevo tiempo es  $\alpha T_0/k$ , entonces, el tiempo total  $T_S$  es:

$$T_S = (1 - \alpha) T_0 + \alpha T_0/k$$

entonces, la mejora  $S = T_0/T_S$  es

$$S = \frac{1}{(1 - \alpha) + \alpha/k}$$

## Ley de Amdahl: ejemplo 1

Una parte de un sistema consume el 60 % del tiempo de ejecución y se lo mejora en un factor de 3. ¿Cuál es la mejora que produce en el sistema?

Datos:

$$\alpha = 0,6$$

$$k = 3$$

Resolución:

$$S = \frac{1}{(1 - \alpha) + \alpha/k} = \frac{1}{0,4 + 0,6/3} = 1,67 \times$$

A pesar de mejorar gran parte del sistema de forma significativa la mejora total fue substancialmente menor.

## Ley de Amdahl: ejemplo 2

Van a viajar de FIUBA a Lago Verde (Chubut). Son 1812 km y demoran, según una web, 21.83 h, a  $83 \text{ km h}^{-1}$ . Como les parece mucho tiempo, investigan y descubren que pueden recorrer los 668 km que separan General Acha de Piedra del Águila, a hasta  $150 \text{ km h}^{-1}$ . Un tanto arriesgado. ¿Qué mejora se obtiene en total? ¿Cuánto es en tiempo?

Datos:

$$\alpha = \frac{668 \text{ km}}{1812 \text{ km}} = 0.37 \quad \text{más de } 1/3 \text{ del viaje}$$

$$k = \frac{150 \text{ km h}^{-1}}{83 \text{ km h}^{-1}} = \frac{8.05 \text{ h}}{4.45 \text{ h}} = 1.81 \times \quad \text{cerca del doble}$$

Resolución:

$$S = \frac{1}{0.63 + 0.37/1.81} = 1.20 \times \quad T_S = \frac{21.83 \text{ h}}{S} = 18.2 \text{ h}$$

¿Es una mejora importante? ¿Vale el esfuerzo?

## Ley de Amdahl: ejemplo 2 (cont.)

Llegan a la conclusión de que una mejora importante son 6 h. ¿Cuántas veces más rápido se debe recorrer el camino? ¿A qué velocidad, constante, deben realizar el trayecto de 668 km para ganar esas 6 horas?

Resolución:

$$S = \frac{T_0}{T_S} = \frac{21.83 \text{ h}}{21.83 \text{ h} - 6 \text{ h}} = 1.38 \times$$

sabiendo que  $S = 1/[(1 - \alpha) + \alpha/k]$ , entonces

$$k = \frac{\alpha}{1/S - (1 - \alpha)} = 3.93 \times$$

$$\nu_n = k \cdot 83 \text{ km h}^{-1} = 326.22 \text{ km h}^{-1}$$

El trayecto se debe recorrer 3.93 veces más rápido, a una velocidad de  $326.22 \text{ km h}^{-1}$ .

# Tabla de contenidos

---

1. Desde la ENIAC a las *phablets*
2. Ocho grandes ideas en arquitectura de computadoras
3. Cinco grandes realidades en computación
4. Software/Hardware interface
  - Desde tus dedos
  - Bajo tu manos
5. Rendimiento
  - Medición
  - Principio, regla, ley
6. Potencia, unicores, y multicores
7. Trampas y falacias

# ¿Qué es la potencia?

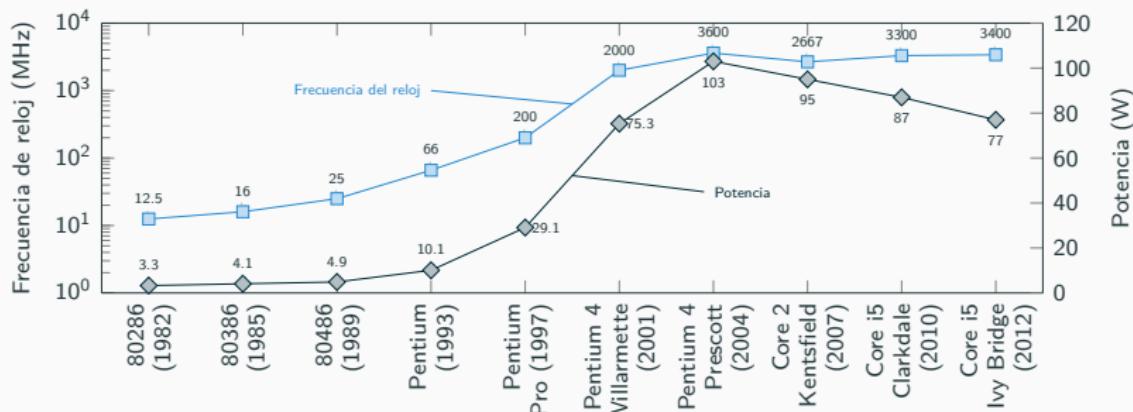
*(En electrónica) Es la cantidad de energía transferida por unidad de tiempo, y se mide en Watt (W)<sup>2</sup>.*

- La unidad de energía—eléctrica en este caso—es el Joule (J)
  - Los Watt son, entonces,  $\frac{\text{Joule}}{\text{segundos}}$
- Los procesadores necesitan energía al cambiar los estados de los bits (energía dinámica) y siempre hay una pérdida en estados ociosos (*leakage*).

---

<sup>2</sup>en realidad, la potencia es siempre energía/tiempo.

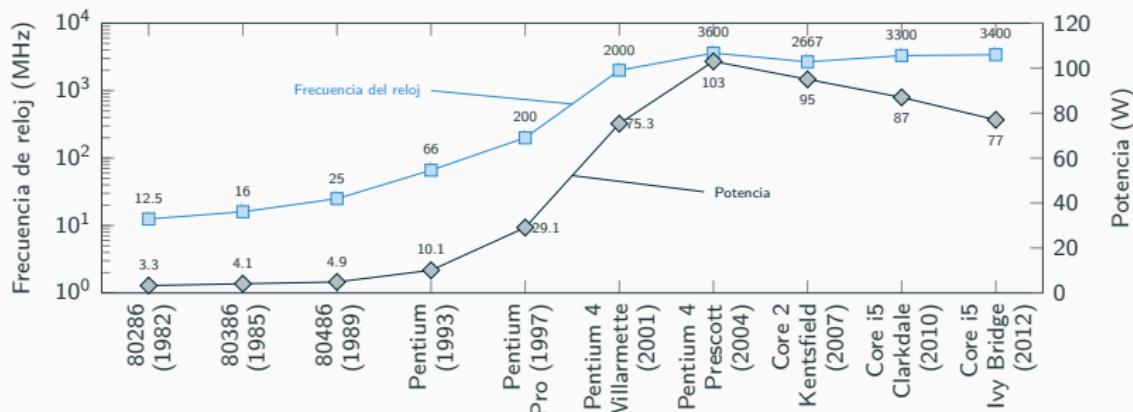
# Tendencias



- En tecnología CMOS integrada:

$$\text{Potencia (P)} \propto \text{Capacidad (C)} \times \text{Tensión (V)}^2 \times \text{Frecuencia (f)}$$

# Tendencias



- En tecnología CMOS integrada:

$$\text{Potencia (P)} \propto \text{Capacidad (C)} \times \text{Tensión (V)}^2 \times \text{Frecuencia (f)}$$

↑                      ↑                      ↑  
 ×30                  5 V → 1 V            ×1000

## Reducción del consumo (potencia)

- Supongamos que una CPU tiene
  - el 85 % de la carga capacitiva ( $C$ ) de la CPU anterior
  - una reducción de 15 % en la tensión y 15 % en la frecuencia

La relación entre la potencia nueva ( $P_n$ ) y la anterior ( $P_o$ ) es

$$\frac{P_n}{P_o} = \frac{C_o \times 0,85 \times (V_o \times 0,85)^2 \times f_o \times 0,85}{C_o \times V_o^2 \times f_o} = 0,85^4 = 0,52\dots$$

- “*The Power Wall*”
  - Es cada vez más difícil reducir la tensión
  - Es cada vez más difícil disipar el calor generado
- ¿De qué otra forma se puede mejorar el rendimiento?

# Multiprocesadores

---

- Microprocesadores multinúcleo
  - más de un procesador por chip
- Requiere programación paralela explícita
  - En comparación con paralelismo a nivel instrucción
    - El hardware ejecuta más de una instrucción a la vez
    - Es un *feature* oculto a quien programa
  - Es difícil de hacer
    - Se programa con el enfoque en el rendimiento
    - Hay que considerar el balanceo de carga
    - Se deben optimizar las comunicaciones y el sincronismo

# Tabla de contenidos

---

1. Desde la ENIAC a las *phablets*
2. Ocho grandes ideas en arquitectura de computadoras
3. Cinco grandes realidades en computación
4. Software/Hardware interface
  - Desde tus dedos
  - Bajo tu manos
5. Rendimiento
  - Medición
  - Principio, regla, ley
6. Potencia, unicores, y multicores
7. Trampas y falacias

## Trampa: Ley de Amdahl

*Esperar que al mejorar una parte de un sistema, el sistema completo mejore en igual proporción*

- Sin embargo

$$T_{\text{mejorado}} = \frac{T_{\text{modificado}}}{\text{factor de mejora}} + T_{\text{no modificado}}$$

- Ejemplo: la multiplicación ocupa 80 s/100 s
  - ¿Cuánto la tengo que mejorar para obtener un rendimiento 5 veces mayor en el sistema?

$$20 = \frac{80}{k} + 20 \Rightarrow k = \infty$$

- No se puede hacer
- Corolario: el caso común debe ser rápido

## Falacia: una CPU ociosa consume poco

*Muchas veces se cree que en modo ocioso una CPU consume poca potencia*

- Según unos *benchmarks* del Intel i7 (ver [PAT17])
  - al 100 % de la carga: 258 W
  - al 50 % de la carga: 170 W (66 %)
  - al 10 % de la carga: 121 W (47 %)

Casi sin carga consume la mitad de la potencia máxima

- Data center de Google
  - Operan mayormente entre el 10 % y el 50 % de la carga
  - Al 100 % operan menos del 1 % del tiempo
- Se empieza a considerar diseñar procesadores que consuman potencia proporcional a la carga

# Trampa: MIPS como métrica de rendimiento

- MIPS: Millones de Instrucciones Por Segundo
  - No tiene en cuenta
    - las diferencias entre las ISAs de una computadora
    - las diferencias en la complejidad entre instrucciones

$$\begin{aligned}MIPS &= \frac{\text{Recuento de instrucciones}}{\text{Tiempo de ejecución} \times 10^6} \\&= \frac{\text{Recuento de instrucciones}}{\frac{\text{Recuento de instrucciones} \times \text{CPI}}{\text{Frecuencia del reloj}} \times 10^6} = \frac{\text{Frecuencia del reloj}}{\text{CPI} \times 10^6}\end{aligned}$$

- El CPI varía entre programas en una misma CPU

# Licencia del estilo de beamer

Obtén el código de este estilo y la presentación demo en

[github.com/pamoreno/mtheme](https://github.com/pamoreno/mtheme)

El estilo *en sí* está licenciado bajo la Creative Commons Attribution-ShareAlike 4.0 International License. El estilo es una modificación del creado por Matthias Vogelgesang, disponible en

[github.com/matze/mtheme](https://github.com/matze/mtheme)

