Clase 02: Cómputo Monoprocesador

IIC3533 Computación de Alto Rendimiento

Ignacio Labarca – ijlabarca@uc.cl

Departamento de Ciencia de la Computación
Escuela de Ingeniería
Pontificia Universidad Católica de Chile

Jueves 7 de Agosto del 2025

Repaso

Cómputo Monoprocesador

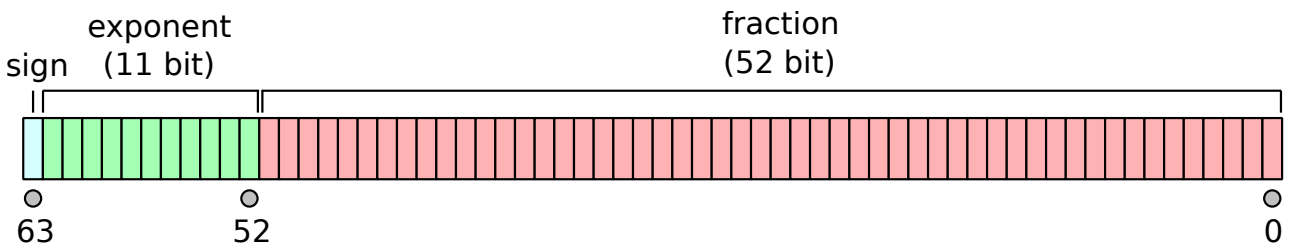
Repaso

Cómputo Monoprocesador

Sistemas de Cómputo Intensivo

Métricas en supercomputadores:

- **FLOP**. Floating Point Operation en precisión doble (8 byte).
- **FLOPS**, o también **FLOP/s**: FLOP por segundo.
- **Byte**. Tamaño de los datos. Precisión doble: 8 byte. Simple: 4 byte.
- **Bit**. Unidad básica. 1 Byte = 8 Bit.



¿Cómo ser más rápido?

Para mayor rapidez, se puede tener

- Más cores en una máquina – *multicore*
  - Dual, Quad, Octa, ...
- Más máquinas con múltiples cores
  - Clusters
  - Memoria compartida o distribuida
- Chips con más procesadores – *manycore* – GPU

¿Paralelismo, distribución, o HPC?

- **Paralelismo**. Cómputo que involucra múltiples tareas ejecutándose simultaneamente como parte de un mismo trabajo.
- **Multiprocesamiento**. Computador ejecutando múltiples trabajos simultáneamente.
- **HPC**. Cómputo de una tarea muy larga, pero que se ejecuta a la mayor velocidad posible.
- **Cómputo distribuido**. Cómputo que involucra múltiples computadores colaborando en una tarea.

¿Por qué es importante?

Ejecutar código eficientemente depende de

- Algoritmo utilizado.
- Aprovechar la arquitectura.

¿Por qué es importante?

Ejecutar código eficientemente depende de

- Algoritmo utilizado.
- Aprovechar la arquitectura.

Tiempo de ejecución puede variar incluso órdenes de magnitud.

Para algunos problems, **un único procesador** es suficiente.

Repaso

Cómputo Monoprocesador

Arquitectura Von Neumann

El modelo plantea

- Memoria única, no dividida. Almacena programa y datos.
- Unidad de procesamiento (CPU) que ejecuta instrucciones:
  - Ciclo: *Fetch*, *Execute*, *Store*.
  - Modelo *Control flow*, flujo dirigido por instrucciones (vs *data flow*).
- Código fuente puede ser modificado.
- Programas pueden modificar otros programas.
- Programas también pueden ser "datos".

Arquitectura Von Neumann

Procesador ejecuta instrucciones "de bajo nivel":

Operación	Dirección Operandos	Dirección Resultado

- Instrucciones referencian direcciones de memoria.
- Estas direcciones suelen ser **registros**.
- Memoria tradicional es mucho más lenta que la velocidad del procesador.

Arquitectura Von Neumann

```
1       void store(double *a, double *b, double *c){
2          *c = *a + *b;
3       }
```

Compilar: `gcc -O2 -S -o - ejemplo.c`

```
1          .file "ejemplo.c"
2          .text
3          .p2align 4
4          .globl  store
5          .type store, @function
6       store:
7       .LFB0:
8       .cfi_startproc
9       movsd (%rdi), %xmm0     # Load *a to %xmm0
10      addsd (%rsi), %xmm0     # Load *b and add to %xmm0
11      movsd %xmm0, (%rdx)     # Store result to *c
12      ret
13      .cfi_endproc
```

Arquitectura Von Neumann

Ciclo:

- **Fetch**.
  Cargar instrucción en el procesador de acuerdo a *Program Counter*.
- **Decode**.
  Determinar qué operación es y qué operandos necesita.
- **Memory fetch**.
  Traer operandos desde memoria a registros (si es necesario).
- **Execute**.
  Ejecuta la operación, leyendo y escribiendo registros.
- **Write-back**.
  Guarda el resultado en memoria (si es necesario).

Arquitectura Von Neumann

- Dos principios
  - instrucciones se manejan como datos
  - secuencias de *fetch–execute–store*

Arquitectura Von Neumann

El funcionamiento básico es

$$\text{Fetch} \longrightarrow \text{Decode} \longrightarrow \text{Memory fetch} \longrightarrow \text{Execute} \longrightarrow \text{Write-back}$$

pero los computadores modernos introducen varios elementos:

ILP: Instruction Level Parallelism

- **Localidad**. Luego de leer una dirección, la siguiente es más rápida.
- **Caché**. Datos quedan en jerarquías intermedias de memoria.
- **Pipelining**. Varias instrucciones se procesan al mismo tiempo en distintas etapas.

En general se ejecuta una instrucción por ciclo.

Arquitectura Von Neumann

La velocidad del procesador parece ser la métrica principal, pero

- Algoritmos *cpu-bound*: limitados por velocidad del procesamiento.
- Algoritmos *memory-bound*: limitados por velocidad de acceso a memoria.

Procesadores Modernos

El modelo Von Neumann supone solo una unidad de ejecución (*core*), pero ya no es así.

- *Out-of-order execution*: Manejo de instrucciones.
- *Floating Point Units* (FPU): Múltiples unidades para suma y multiplicaciones de punto flotante.
- *Fused Multiply-Add* (FMA). Ejecuta $x \leftarrow ax + y$ (DAXPY)
- *Pipelining*.

Procesadores modernos

Processor	year	add/mult/fma units (count $\times$ width)	daxpy cycles (arith vs load/store)
MIPS R10000	1996	$1 \times 1 + 1 \times 1 + 0$	8/24
Alpha EV5	1996	$1 \times 1 + 1 \times 1 + 0$	8/12
IBM Power5	2004	$0 + 0 + 2 \times 1$	4/12
AMD Bulldozer	2011	$2 \times 2 + 2 \times 2 + 0$	2/4
Intel Sandy Bridge	2012	$1 \times 4 + 1 \times 4 + 0$	2/4
Intel Haswell	2014	$0 + 0 + 2 \times 4$	1/2

Cuadro: Capacidad de cómputo para operaciones de punto flotante DAXPY con 8 operandos en distintos procesadores.

¿Cuánto gano con un *pipelining*?

FPU Tradicional: Obtener n resultados toma $T(n) = n \times \ell \times \tau$

- ℓ : número de etapas
- τ : *clock cycle time*.
- Tasa de resultados es $n/T(n)$, $r_{\text{serial}} = \frac{n}{T(n)} = \frac{1}{\ell \times \tau}$.

Pipelined FPU: $T(n) = (s + \ell + n - 1) \times \tau$.

- s : costo de inicialización (*setup*).
- La primera operación recorre todo el *pipeline*.
- Después, las siguientes terminan una a la vez.

Operación

Las instrucciones en el *pipeline* deberían ser independientes

- $a_i \leftarrow b_i + c_i$

Operación

Las instrucciones en el *pipeline* deberían ser independientes

- $a_i \leftarrow b_i + c_i$
- $a_{i+1} \leftarrow a_i b_i + c_i$

¿Cómo ejecutar esto en un pipeline?

```
1       for(i = 0; i < 1000; i++){
2          a[i+1] = a[i] * b[i] + c[i];
3       }
```

Peak Performance

En computadores con *pipelining*, cada FPU produce un resultado por ciclo (asintóticamente)

$$\text{PeakPerformance} = \text{clockspeed} \times \#\text{FPUs}$$

Instruction Level Parallelism

Mecanismos:

- Instrucciones independientes pueden iniciarse al mismo tiempo.
- *Pipelining*.
- *Branch prediction*, *Speculative Execution*.
- *Out-of-order execution*.
- *Prefetching*.

Movimiento de datos

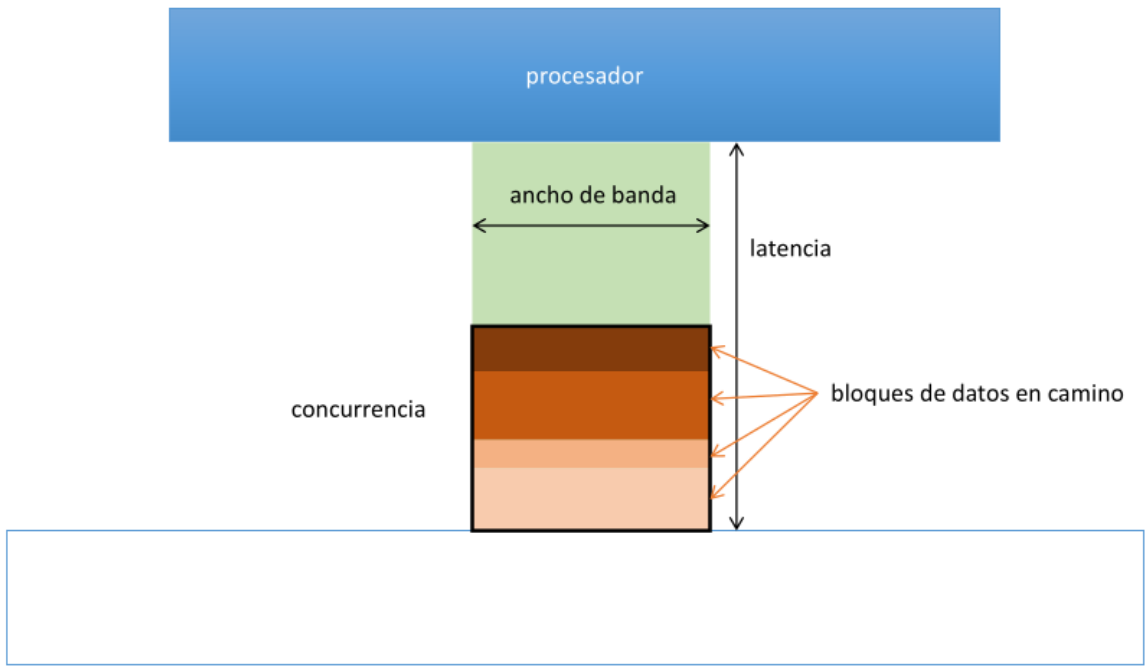
Acceso a memoria es la principal limitante.

- **Jerarquías de memoria**.
- **Latencia**: tiempo desde la solicitud del dato, hasta que el dato llega.
  - Transferencia entre memorias. Medida en ciclos o en *ns*.
  - *Memory stall*: procesador está bloqueado esperando datos.
  - *Latency hiding*.
- **Ancho de banda**
  - Tasa de llegada de datos.
  - Posterior a periodo de latencia inicial.
  - Medido en (kilo, mega, giga) Byte/s o en (kilo, mega, giga) Byte/ciclo.
  - Incluye: velocidad del canal (*bus speed*), ancho del bus, canales por *socket*, cantidad de *sockets*.
- **Tiempo de transferencia**: $T(n) = \alpha + \beta n$
  - α: latencia, β: inverso del ancho de banda (tiempo por byte).

Concurrencia

El máximo nivel de concurrencia se obtiene cuando todo el volumen de ancho de banda está en uso

$$\text{Concurrencia} = \text{Ancho de Banda} \times \text{Latencia}$$



Caché

- Registros se acceden mediante instrucciones de *assembler* .
- Movimientos caché-memoria se hacen por hardware .
- Menor margen para intervenir en el uso de caché vs uso de registros.

Caché

- El caché es el punto intermedio entre registros y memoria principal.
- Existen varios **niveles** de memoria caché.
- Se caracterizan por baja latencia y alto ancho de banda.
- Los datos se mantienen en la memoria caché de manera temporal.

Memoria Principal $\longrightarrow$ Memoria caché $\longrightarrow$ Registros

Caché

- L1 tiene **32 bytes de ancho de banda**, 2 operandos para 2 operaciones, 4 operaciones por ciclo.
- Existe L1d para datos y L1i para instrucciones.
- L1 y L2 suelen estar en el chip del procesador.