

## IMT2112 - Algoritmos Paralelos en Computación Científica

### Jerarquía de memoria

Elwin van 't Wout

20 de agosto de 2019



PONTIFICIA  
UNIVERSIDAD  
CATÓLICA  
DE CHILE

Facultad de Matemáticas • Escuela de Ingeniería

imc.uc.cl

# Clases previas

- Arquitectura de Von Nuemann
  - instrucciones se manejan como datos
  - secuencias de *fetch - execute - store*
- Paralelismo de nivel de instrucción
  - canalización de instrucciones en una tubería (*pipelining*)

# Agenda

- Arquitecturas de memoria en computadores
  - ¿Cómo se maneja el flujo de datos en una computador?

# Motivación

- El muro de la memoria (*memory wall*)
- Leer y escribir datos lleva más tiempo que ejecutar instrucciones
  - la carga de datos desde la memoria principal puede tomar 1000 ciclos de reloj
- Una solución inteligente: jerarquías de memoria

# Jerarquía de memoria

Sección 1.3 del libro de Eijkhout

# Flujo de datos en una computador

- Los cables que mueven datos a través de los niveles de memoria y hacia la CPU se denominan "*buses*"
- Frecuencia: el número de ciclos de *bus* por segundo
  - alrededor de 1 GHz
  - recuerda que la velocidad de reloj de la CPU es 3 GHz
- Capacidad: número de bits por ciclo de *bus*
  - el ancho del *bus* es típicamente de 64 o 128 bits
- Ancho de banda (*bandwidth*): número de bytes por segundo

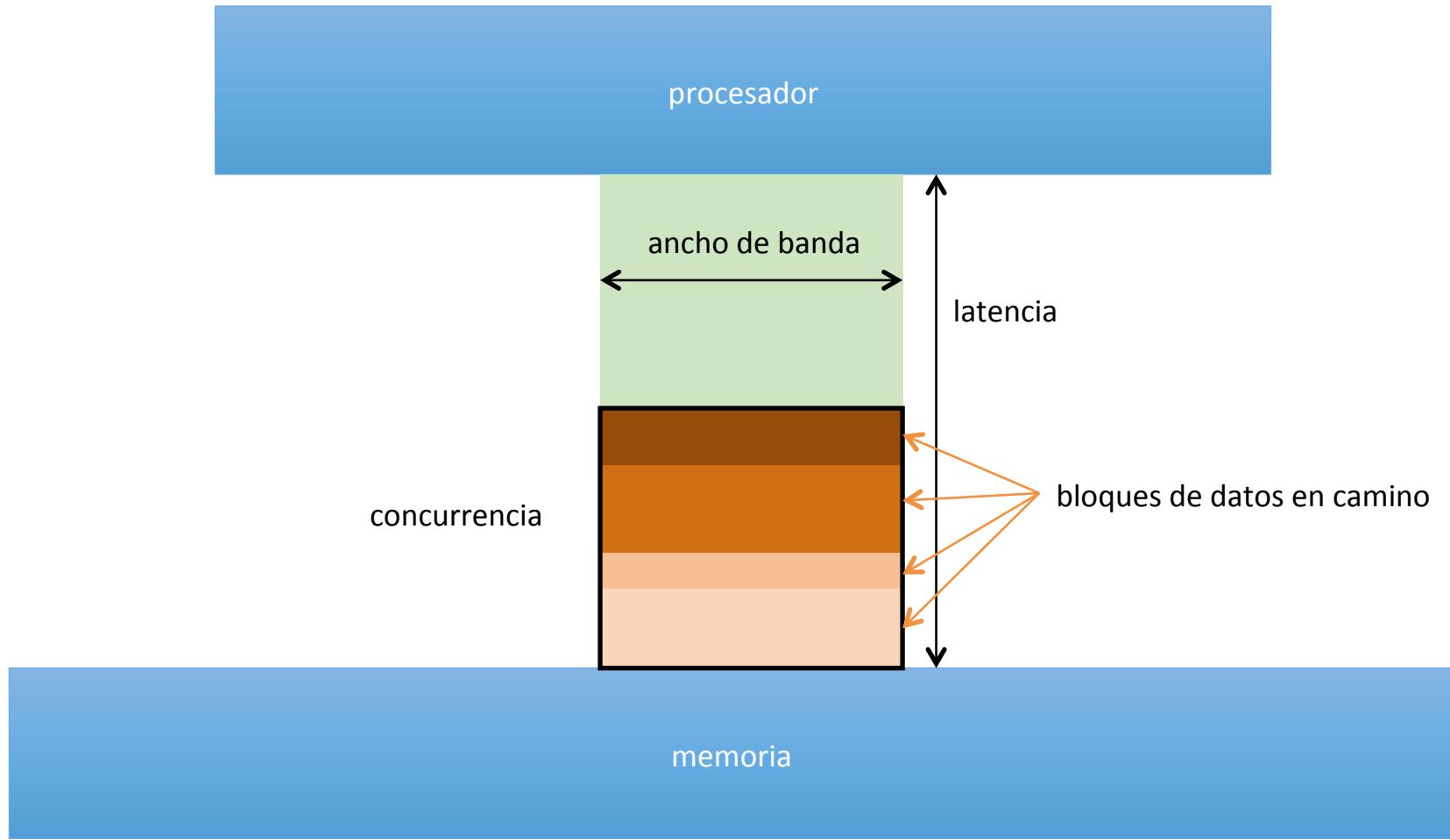
# Latencia

- Latencia (*latency*): el retraso entre la solicitud y la llegada de los datos
  - medido en segundos o ciclos
  - entre 1 y 1000 ciclos dependiendo de la arquitectura
- Los procesadores pueden detenerse: no están ejecutando instrucciones, sino que están esperando datos
  - llamado “bloqueo de memoria” (*memory stall*)

# Concurrencia

- Ancho de banda (*bandwidth*): la velocidad de transferencia de datos
  - ancho de *bus* por velocidad de *bus*, medido en bits por segundo
  - una tasa máxima alcanzada después de la latencia inicial
- Concurrencia: la cantidad de datos que estan viajando en el *bus*
  - medido en bits o bytes
  - máximo: acho de banda por latencia
- Se puede lograr una alta concurrencia cuando ya se llaman y traen (*fetch*) datos para instrucciones futuras

# Flujo de datos



# El muro de la memoria

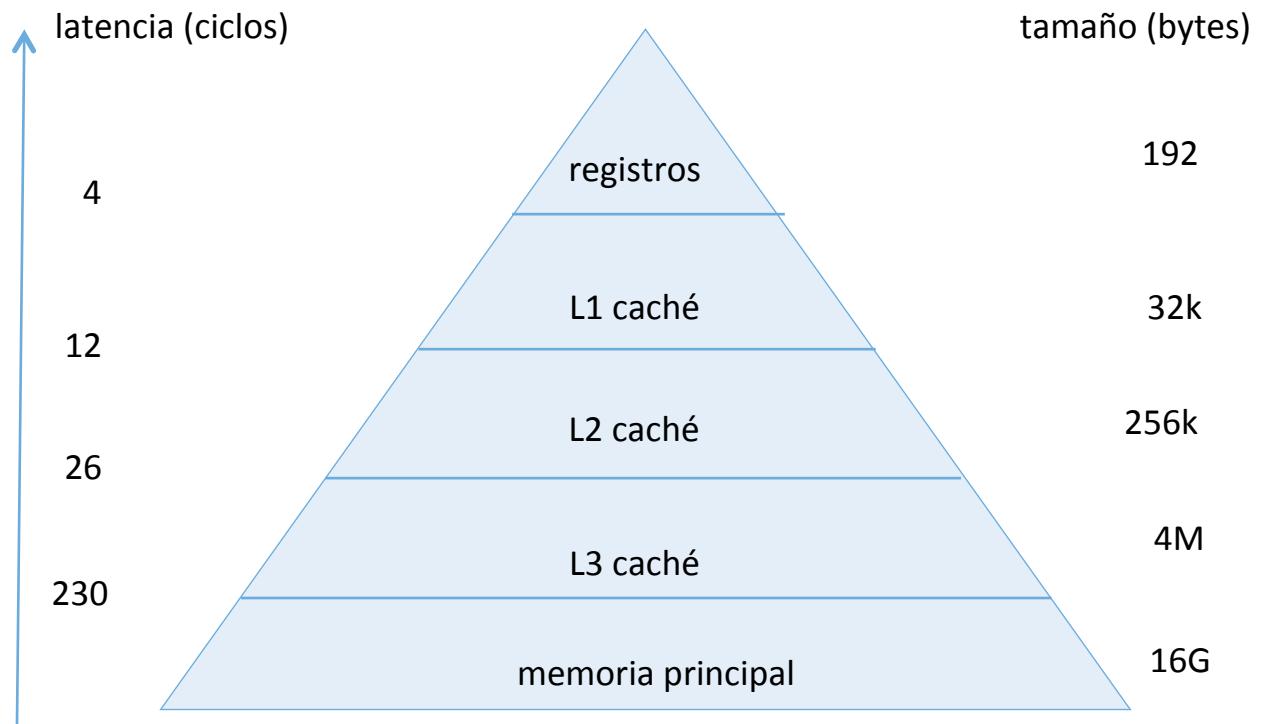
- La transferencia de datos es a menudo (mucho) más lenta que la ejecución de operaciones de punto flotante
- ¿Cómo acelerar el proceso del flujo de datos entre memoria y CPU?
  - dadas las limitaciones del *hardware* del procesador

# Jerarquía de memoria

- Existen diferentes tipos de dispositivos de memoria
  - hay de gran capacidad de almacenamiento pero lento para leer y escribir
  - hay de lectura y escritura rápidas pero pequeña capacidad de almacenamiento
- Existen diferentes *buses* (latencia, ancho de banda)
- Idea: usar una jerarquía de niveles de memoria y almacenar datos de uso frecuente en dispositivos de lectura rápida
- Niveles de memoria principal (RAM), cachés y registros

# Jerarquía de memoria

Los niveles de caché se encuentran entre la memoria principal y los registros; típicamente 2 o 3 capas



# Memoria principal

- El más alto nivel de memoria
  - con ‘memoria’ normalmente se entiende la RAM, no el almacenamiento en el disco duro
- Gran cantidad de almacenamiento, p.ej. 16 GB
- Relativamente barato de producir
- Volátil: necesita fuente de alimentación
- A veces se utiliza “*swapping*”, donde los datos se transfieren al disco duro
  - esto es muy lento

# Registros

- Memoria de nivel más bajo
- Es parte de la CPU, por lo que la latencia es despreciable
- Consejo: intente mantener los datos en los registros
  - el compilador normalmente lo hará
- Hay espacio para 16 a 32 números de puntos flotantes solamente

# Mapeo de caché

Sección 1.3 del libro de Eijkhout

# El caché

- Se produce una “pérdida de caché” (*cache miss*) cuando la unidad de procesamiento necesita datos que no están en el caché
  - *obligatorio*: primera vez de referencia
  - *capacidad*: el registro se borró del caché debido al tamaño limitado del caché
  - *conflicto*: uso múltiple de la misma ubicación del caché
- Un “éxito de caché” (*cache hit*) ocurre cuando los datos solicitados se pueden encontrar en el caché

# Mapeo del caché

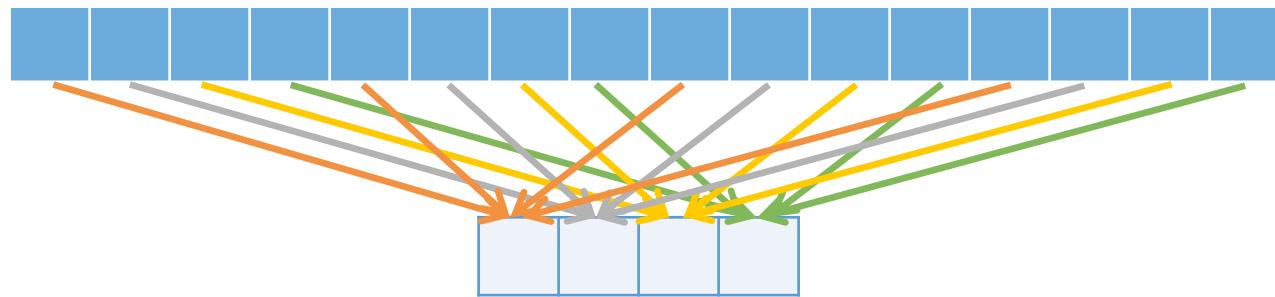
- Pregunta central: ¿como hacer uso eficiente del caché?
- Específicamente, ¿como almacenar los datos que vienen del RAM en el caché?
  - Recuerda que el RAM es ~16 GB y el L3 caché ~4 MB

# Mapeo del caché

- En la memoria, cada unidad de datos - normalmente un byte - tiene una dirección asignada (*memory address*)
  - Esta dirección es un número
  - Se puede modelar la memoria como una secuencia unidimensional de datos
  - Como programador puedes usar esta dirección
    - se llama *pointers* en C++
    - la función `id()` en Python es similar pero es distinto que la dirección de memoria

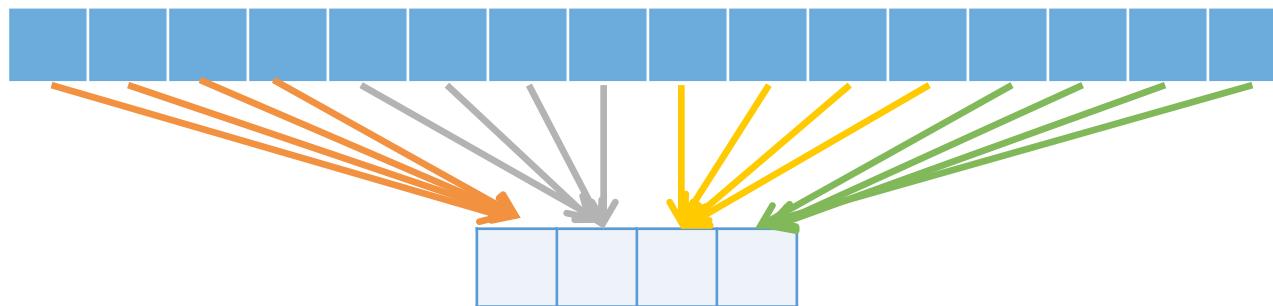
# Mapeo del caché

- Mapeo directo del caché: la ubicación está determinada por la ubicación en el nivel superior, módulo el tamaño de memoria
  - identificador de ubicación único, sin necesidad de tablas de búsqueda
  - riesgo de muchos pérdidas de caché del tipo conflicto

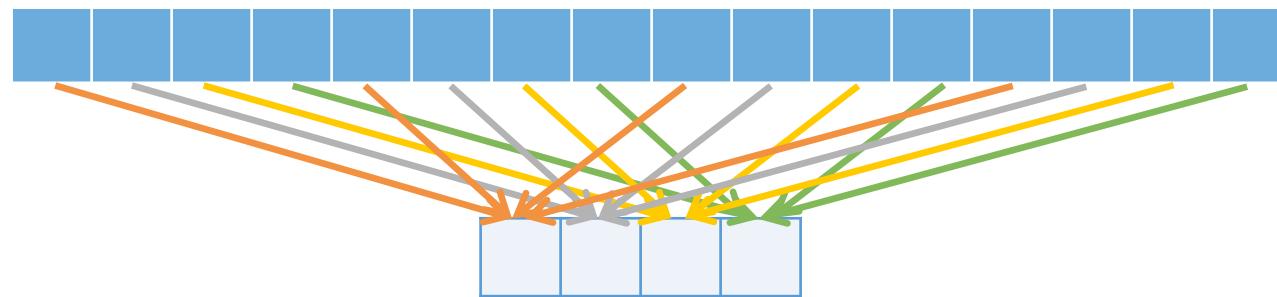


# Mapeo del caché

¿Por qué no usar el mapeo



en vez del siguiente?

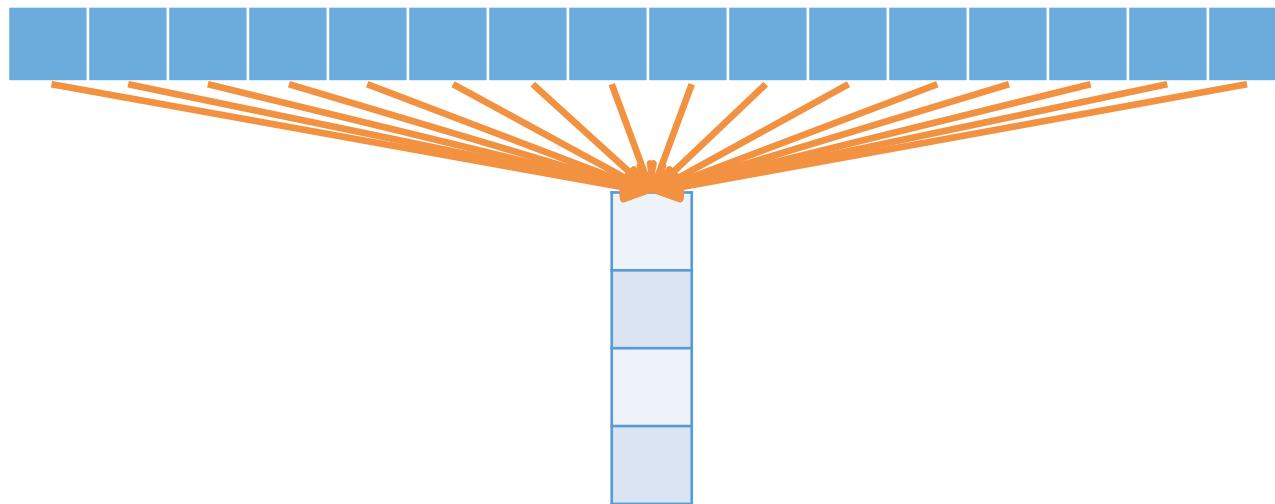


# Mapeo del caché

- Los bucles con paso 1 son mucho más comunes y fáciles de usar que un paso del tamaño de caché
- Consejo del programador: mantenga todo ‘local’ para mejorar el rendimiento
  - formulación del modelo
  - código de software
  - asignación de memoria
  - mapeo de caché

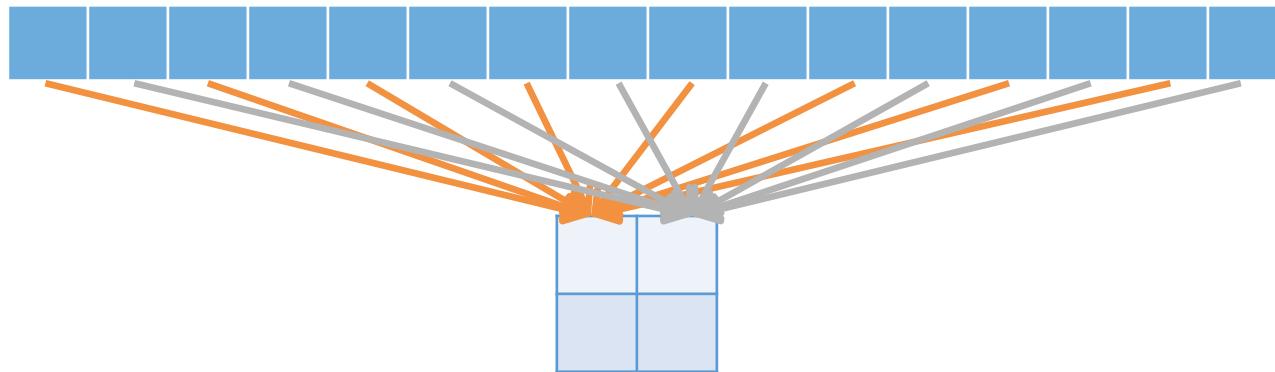
# Mapeo del caché

- Mapeo de caché asociativo: los datos se pueden transferir a cada ubicación
  - flexible con pocos pérdidas de caché
  - requiere tablas de búsqueda y, por lo tanto, *overhead*



# Mapeo del caché

- Mapeo de caché asociativo  $n$ -vías: combinación de mapa de caché directo y asociativo
  - los datos se pueden transferir a  $n$  ubicaciones
  - tiene flexibilidad y velocidad



# Mapeo del caché

- Los mapeos de caché asociativos requieren una política de reemplazo
  - FIFO (first in first out): primero en entrar, primero en salir
  - LRU (least recently used): menos utilizado recientemente
- Típicamente asociatividad de hasta 8 vías
- Esta asociatividad es pequeña en comparación con el tamaño de caché
  - los mapeos de caché asociativos son muy ‘horizontales’
- La localidad de datos sigue siendo importante

# Mapeo del caché: el control del programador

- No se puede controlar el flujo de datos en la memoria caché explícitamente
- Se puede programar de manera que se mejore la reutilización de la memoria caché
  - Localizar datos en tus algoritmos
  - Crear fragmentos que caben en el caché
  - Evitar pasos grandes en los bucles
- Tenga cuidado con *benchmarking*: pequeños problemas encajan en la memoria caché y, por lo tanto, darán una imagen optimista del rendimiento

# Asignación de memoria

- Los compiladores colocan elementos de un vector consecutivamente en la memoria
- Los datos que son ‘locales’ en la memoria pueden permanecer en el mismo nivel de caché, gracias al mapeo de caché
- El flujo de datos entre niveles de caché es siempre con una línea de caché (*cache line*)
  - una línea de caché tiene normalmente 64 o 128 bytes (son 8 o 16 floats)

# Unidades de tamaño de los datos

- Un *bit* tiene el valor 0 o 1
- Un *byte* es una colección de 8 bits.
- Una *palabra* es la unidad de datos en el procesador, hoy en día 64 bits = 8 bytes
- Una *línea de caché* es la unidad de datos en el caché, generalmente 64 bytes = 8 palabras
- Sin embargo, la memoria principal suele ser direccionable en bytes (*byte-addressable*)

# Dirección de memoria

- Los datos en memoria tienen una dirección
  - *offset*: ubicación dentro de la línea de caché
  - *index*: ubicación de la línea de caché en el conjunto de cachés
  - *tag*: ubicación de la línea de caché en la memoria principal



# Mejoras

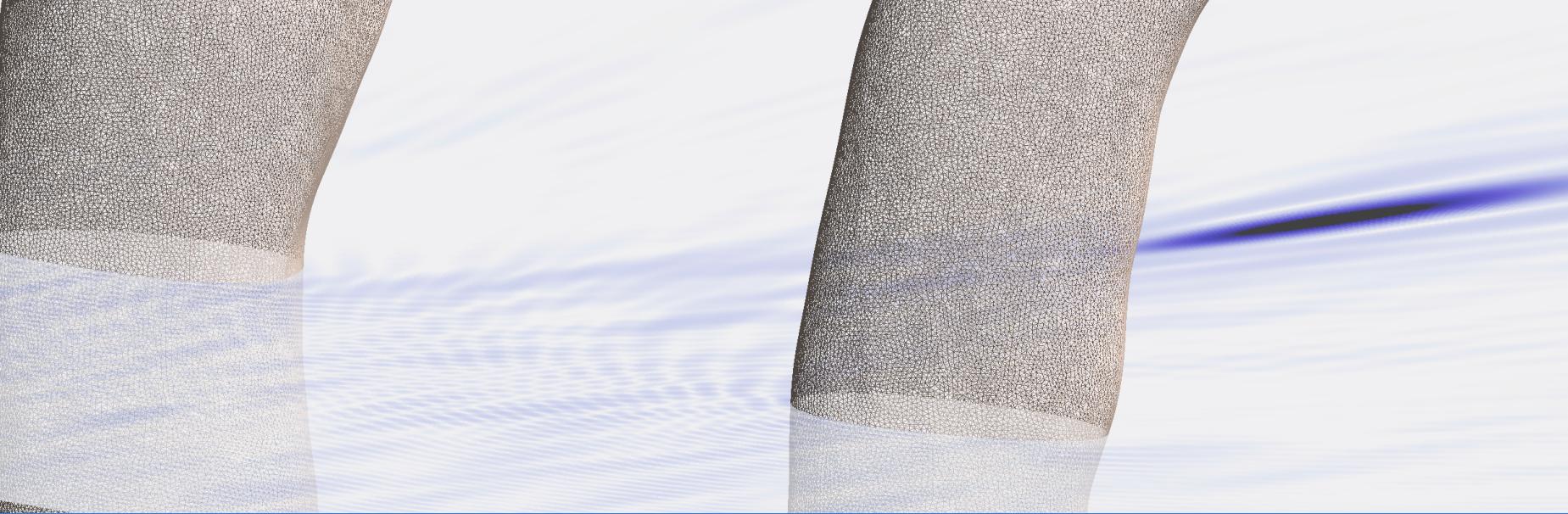
- Los procesadores pueden mejorar el flujo de datos
  - *prefetch streams*: buscar datos antes que se los require
  - *out-of-order executions*: cambiar orden de líneas de código
- Los programadores pueden mejorar el flujo de datos
  - explotar la ‘localidad’ de los datos
- Diseño de arquitectura de caché
  - mapeo de caché asociativo

# Resumen

- Jerarquía de memoria
- Niveles de caché
- Mapeo de caché

# Clase siguiente

- Arquitectura de múltiples núcleos
- Localidad de los datos



## IMT2112 - Algoritmos Paralelos en Computación Científica

### Jerarquía de memoria

Elwin van 't Wout

20 de agosto de 2019



PONTIFICIA  
UNIVERSIDAD  
CATÓLICA  
DE CHILE

Facultad de Matemáticas • Escuela de Ingeniería

imc.uc.cl