ALN - HPC

In. Co.
Facultad de Ingeniería
Universidad de la República

- Ordenes
- Evaluación de desempeño
- Máquina de Von Neumann (simplificada)
- Pipeline
- Computadores vectoriales
- Jerarquías de memorias
- Optimización al compilar
- Multiplicación de matrices

Ordenes

- Tiempo de ejecución de un programa en función de N, lo que denominaremos T(N)

```
for (int i= 0; i < N; i++)
        a = a + t[i]*v[i];
```

- Requiere T(N)= N (suma-multiplicación)

Ordenes

```
for (int i= 0; i < N; i++)

      for (int j= 0; j < N; j++)

                a = a + t[i]*v[j];
```

- Requiere T(N)= N^2

Ordenes

```
for (int i= 0; i < N; i++)
      for (int j= 0; j < N; j++) {
                  a = a + t[i]*v[j];
                        b = b + a*b;
      }
```

- Requiere T(N)=2N^2

Ordenes

- Si definimos (nada formal) el Orden (O()) de un algoritmo como una función tal que c*f(N) > T(N) para una constante.
- En los casos anteriores:

T(N)	O(N)
N	N
N	N
2N^2	N^2

Ordenes

- ¿Cuán útil es medir el Orden de un algoritmo ?
  - Solo mide operaciones a realizar !!!!
  - No tiene en cuenta otros aspectos …
    - Patrones de acceso a memoria
    - Localidad espacial / temporal
    - Etc…

- Ordenes
- Evaluación de desempeño
- Máquina de Von Neumann (simplificada)
- Pipeline
- Computadores vectoriales
- Jerarquías de memorias
- Optimización al compilar
- Multiplicación de matrices

Evaluación de desempeño

- CPU time

  CPU = NI * CPI * CCT

NI: número de instrucciones

CPI: número de ciclos de reloj por instrucción

CCT: tiempo del ciclo de reloj

Evaluación de desempeño

- MIPS (Millons of Instructions Per Seconds)
- Depende del conjunto de instrucciones del equipo.
- En un equipo depende del problema a resolver.

$$MIPS = \frac{\text{frecuencia del reloj}}{\text{CPI} * 10^6}$$

Evaluación de desempeño

- FLOPS (Floating Point Operations per Second)

$$FLOPS = NFPU * FPC * CPS$$

NFPU: número de unidades de PF

FPC: flops por ciclo

CPS: frecuencia del reloj (ciclos por seg Hz)

Evaluación de desempeño

- El tiempo del proceso se divide en:
  - tiempo de usuario: Tiempo que se dedica a operaciones a nivel de usuario.
  - tiempo del sistema: Llamadas a sistema (ej: operaciones de E/S)

Evaluación de desempeño

- Tiempo de CPU: tiempo en que la cpu está ocupada en ejecutar instrucciones del proceso.
- Walltime: es el tiempo real que transcurre entre un evento y otro.
- Ej: Comando time (linux)
  - **Real**: tiempo “reloj” entre el inicio y el fin del programa.
  - **User**: tiempo de cpu en el espacio de usuario
  - **System**: tiempo de cpu en el espacio de sistema
  - Real <= User + System

Evaluación de desempeño

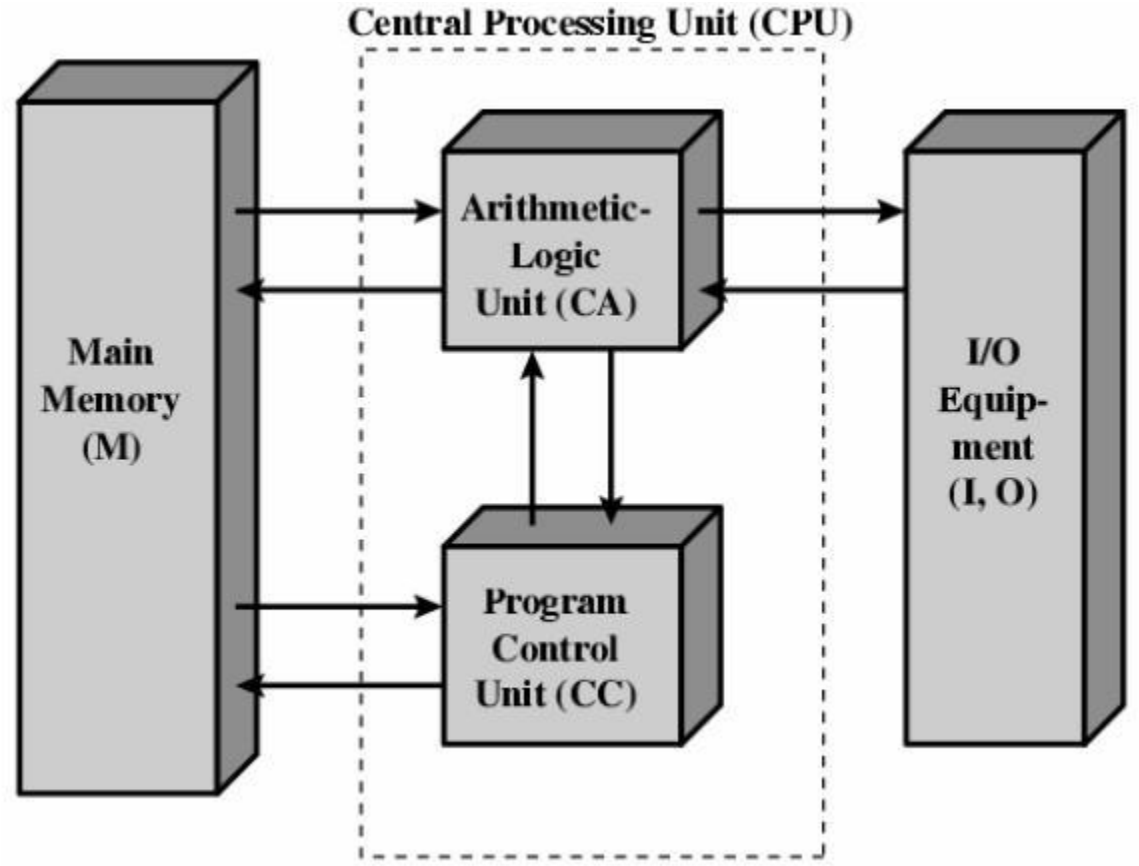
- Micro benchmarks
  - Diseñados para medir el desempeño de un código pequeño y específico o cierta feature del hardware.
  - Velocidad del procesador
  - Acceso a memoria
- Kernel benchmarks
  - Conjuntos de rutinas simples
  - Evalúan el desempeño de hardware y compiladores
  - Ej: Livermore Loops, LINPACK…
- Aplicaciones completas
  - Ej: procesadores de textos, aplicaciones de usuario

Evaluación de desempeño

- Resumen
  - No evaluar desde un solo punto de vista (velocidad del procesador, conjunto de instrucciones, cantidad de ciclos de reloj por instrucción, otros).
  - Pueden considerarse otros factores, como por ejemplo el consumo energético del programa.

- Ordenes
- Evaluación de desempeño
- Máquina de Von Neumann (simplificada)
- Pipeline
- Computadores vectoriales
- Jerarquías de memorias
- Optimización al compilar
- Multiplicación de matrices

Máquina de Von Neumann



Máquina de Von Neumann

- Obtiene la siguiente instrucción desde la memoria según el contador de programa.
- Aumenta el contador de programa.
- Decodifica la instrucción mediante la unidad de control.
- Se ejecuta la instrucción.

Pipeline

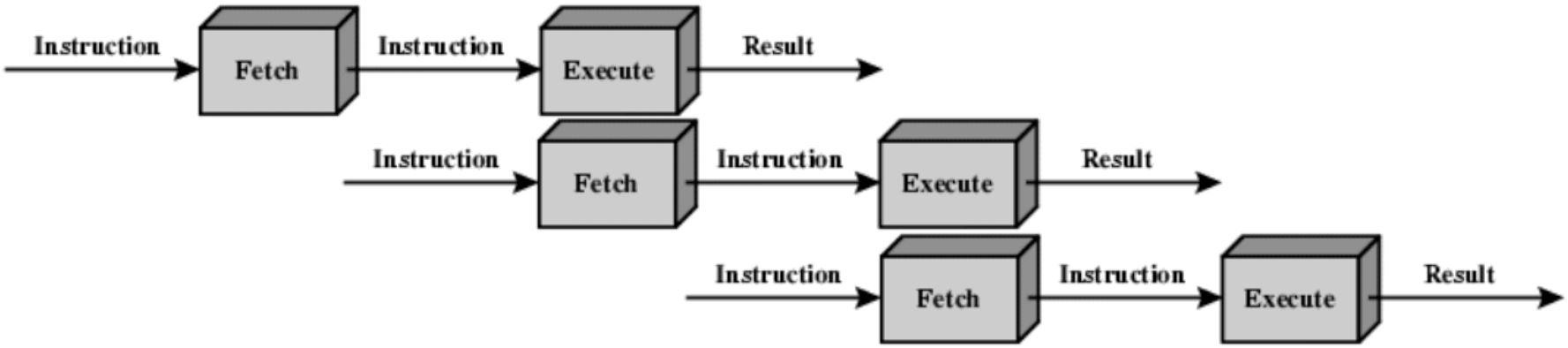
- Una tarea se divide en sub-tareas, la unidad funcional se divide en sub-segmentos.
- Motivación se encuentra en varias realidades
  - Línea de montaje
  - Lavandería
  - Desarrollo de software

Pipeline

- Pipeline básico:
  - Ejecución de una instrucción dividida en dos etapas:
    - **Fetch:** traer la siguiente instrucción desde la memoria
    - **Execute:** ejecutar la instrucción
  - Cada etapa es realizada por una unidad funcional independiente

Pipeline

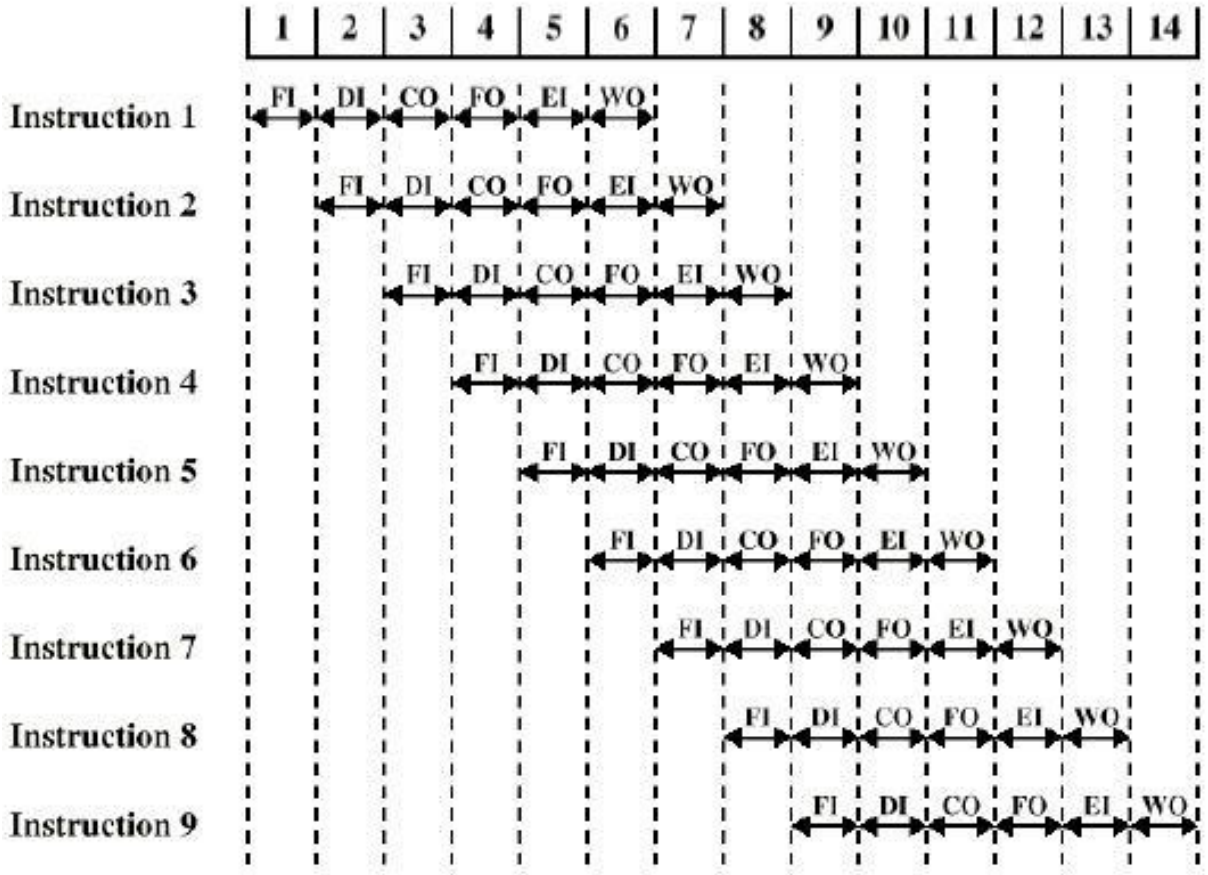
- Pipeline básico (prefetch):
  - Ejecución de una instrucción dividida en dos etapas:
    - **Fetch:** traer la siguiente instrucción desde la memoria
    - **Execute:** ejecutar la instrucción
  - Cada etapa es realizada por una unidad funcional independiente



Pipeline

- Fetch de una instrucción. (FI)
- Decodificar instrucción. (DI)
- Calcular los operandos. (CO)
- Fetch de los datos. (FO)
- Ejecutar instrucciones. (EI)
- Escribir en la memoria. (WO)

Pipeline


1 2 3 4 5 6 7 8 9 10 11 12 13 14
Instruction 1
Instruction 2
Instruction 3
Instruction 4
Instruction 5
Instruction 6
Instruction 7
Instruction 8
Instruction 9
FI DI CO FO EI WO

Pipeline

- Problemas:
  - Llenar el pipeline
  - Hazards (obstáculos)
    - Pueden impedir que la próxima instrucción se ejecute en el ciclo correspondiente
    - Ej:
      - Bifurcaciones en el código
      - Dependencias de datos
      - Conflictos de hardware
    - Es necesario detener el pipeline y a veces vaciarlo (saltos)
    - Existen estrategias para minimizar las penalizaciones: (Ej: estrategias predictivas)

Pipeline

- Latencia
  - Tiempo en procesar una tarea
- Throughput
  - Rendimiento medido, en unidades producidas por tiempo

Pipeline

- Pipelining no mejora la latencia de cada tarea, sino el throughput de toda la carga de trabajo
- Limitado por el estado más lento
- Largo desbalanceado de los pasos del pipeline reduce la aceleración
- Aumentar la aceleración, aumentando la cantidad de pasos del pipeline

Otras ideas

- Pueden trabajar en forma paralela varias sub-unidades de proceso
  - Unidad de suma de PF
  - Unidad de multiplicación de PF
  - Unidad lógica

Máquinas Vectoriales

- Las instrucciones operan sobre arreglos de largo variable (vectores)
- Utilizan registros vectoriales.
  - Se carga los datos en forma vectorial
  - Se realizan las operaciones en forma vectorial
- Importantes en los 70s
  - Cray (Cray 1, Cray X-MP)
  - Fujitsu
  - NEC

Máquinas Vectoriales

- Caído en desuso, máquinas generalmente muy caras.
- Idea utilizada en otro tipo de hardware (GPUs).
- Los procesadores escalares actuales poseen extensiones SIMD (de largo fijo)
  - P.ej: AVX, AVX-512

Máquinas Vectoriales

- Generalmente los equipos disponen de compiladores que transforman automáticamente el código.
- Utilizar notación matricial al programar (por ej. en FORTRAN).

Encadenamiento

- Combinar pipelines para que el resultado de un proceso sea directamente la entrada de otro.
- Ejemplo típico (Multiply-Accumulate):
  - a= a+bc

- Ordenes
- Evaluación de desempeño
- Máquina de Von Neumann (simplificada)
- Pipeline
- Computadores vectoriales
- Jerarquías de memorias
- Optimización al compilar
- Multiplicación de matrices

Estrategias de compilación

- Cada compilador dispone de diversas opciones.
  - La mayoría permite aplicar optimizaciones básicas o compilar para una arquitectura en específico
- Algunas opciones del Intel Fortran para Linux...

Estrategias de compilación

- Optimización:
  - -O0,-O1,-O2,-O3,-Ofast
- Generación de código:
  - −x{code}: permite generar código específico para ejecutar en los procesadores indicados por “code”
    - SSE2, SSE3, SSSE3,SSE4.2,…,AVX, etc.
  - -xHost: generar código con el set de instrucciones más alto disponible en el host

Estrategias de compilación

- Generación de código:
  - −mtune=cpu: Optimizar para una cpu específica
    - `skylake`, `haswell`, `broadwell`, `skylake-avx512`, etc.
    - Ver: https://gcc.gnu.org/onlinedocs/gcc/x86-Options.html

Estrategias de compilación

- Optimización Interprocedural:
  - −ip o −ipo: −ip busca en funciones del mismo archivo, con −ipo se realiza la búsqueda en todos los archivos del sistema.
    - Expansión de funciones inline.
    - Propagación de constantes.
    - Eliminación de código muerto.
    - Pasaje de argumentos por registros.
    - Sacar los códigos invariantes de los loops.

Estrategias de compilación

- Punto Flotante:
  - -fp-model {nombre}:
    - Fast, Precise, Source, Strict
  - -pc{32|64|80(default)}
    - Precisión interna de la unidad de punto flotante (FPU).
  - -rcd y –fp-port
    - Controlan la utilización de redondeos y truncamientos.

Estrategias de compilación

- Otras
  - -openmp: API (Application Program Interface) OpenMP para programar “paralelismo” de memoria compartida

Estrategias de compilación

- Algunas prácticas que pueden permitir al optimizador realizar su tarea de una mejor forma:
  - Minimizar la utilización de variables globales.
  - Evitar el uso de controles de flujo complejos.
  - No usar la instrucción cast.
  - Evitar las referencias indirectas.

Estrategias de compilación

- Algunas prácticas que pueden permitir al optimizador realizar su tarea de una mejor forma:
  - Fetch Scheduling – Software Pipelining

```
for (i=4; i < N; i += 4){
      dot0 += x0 * y0; x0 = x[0]; y0 = y[0];
      dot1 += x1 * y1; x1 = x[1]; y1 = y[1];
      dot2 += x2 * y2; x2 = x[2]; y2 = y[2];
      dot3 += y3 * y3; x3 = x[3]; y3 = y[3];
}
```

Estrategias de compilación

- Loop unrolling

```
for (i=0; i < N; i++)
    dot += X[i] * Y[i];

for (i=0; i < N; i += 4) {
    dot += X[i] * Y[i];
    dot += X[i+1] * Y[i+1];
    dot += X[i+2] * Y[i+2];
    dot += X[i+3] * Y[i+3];
}
```

Estrategias de compilación

- Scalar expansion

```
sum = 0.0;
Do {
      sum += *X;
      sum += X[1];
      sum += X[2];
      sum += X[3];
      X += 4;
}
while (X != stX);
```

```
sum1 = sum2 = 0.0;
sum3 = sum = 0.0;
do
{
      sum += *X;
      sum1 += X[1];
      sum2 += X[2];
      sum3 += X[3];
      X += 4;
}
while (X != stX);
sum += sum1 + sum2 +
sum3;
```

- Ordenes
- Evaluación de desempeño
- Máquina de Von Neumann (simplificada)
- Pipeline
- Computadores vectoriales
- Jerarquías de memorias
- Optimización al compilar
- Multiplicación de matrices

Memoria

- La velocidad de acceso a memoria aumenta menos que la velocidad de cómputo de la CPU !!!
- Hay distintos tipos de memorias:
  - Diferentes costos, tamaño y velocidad
  - Cuanto más grande y más lejana al procesador: más lenta
- Por qué? – Física…
  - c = 3.0x10^8 m/s, clock = 3GHz
  - c/clock = 10cm/ciclo
  - Otros: Propiedades como capacitancia dependen de la longitud de los cables
- Existe una jerarquía de memorias

Memoria

- Registros
- Cache (2 niveles)
- RAM
- Disco
- Almacenamiento remoto

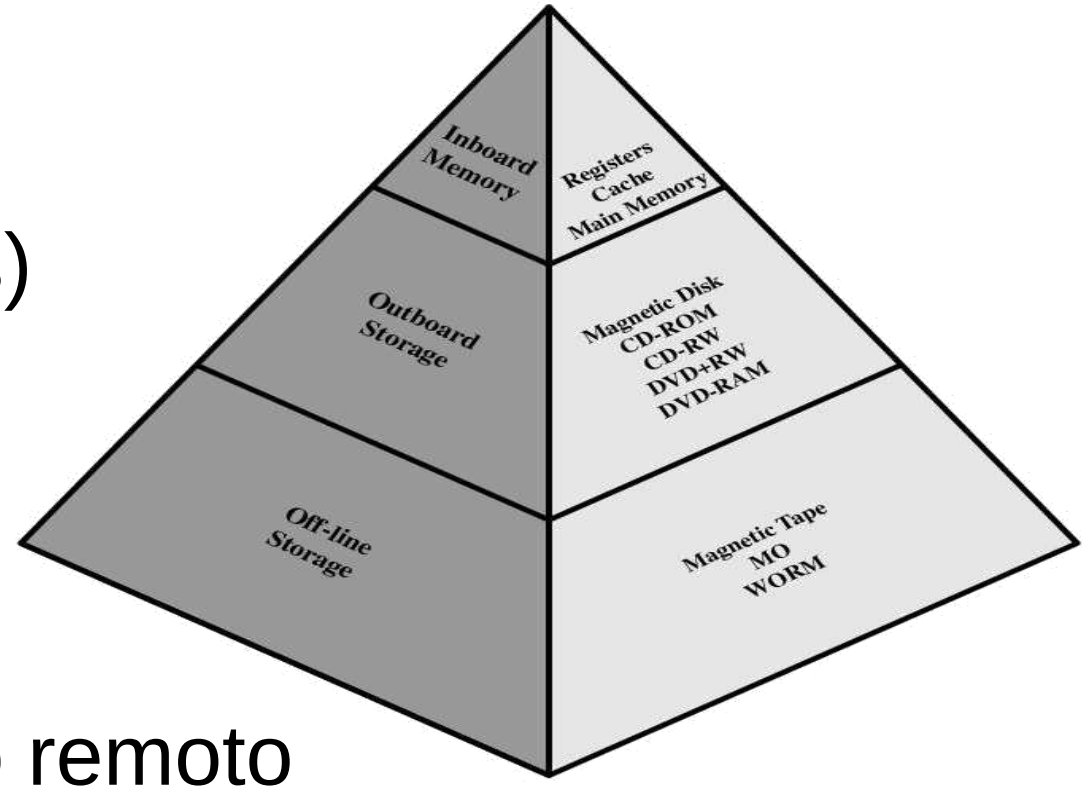
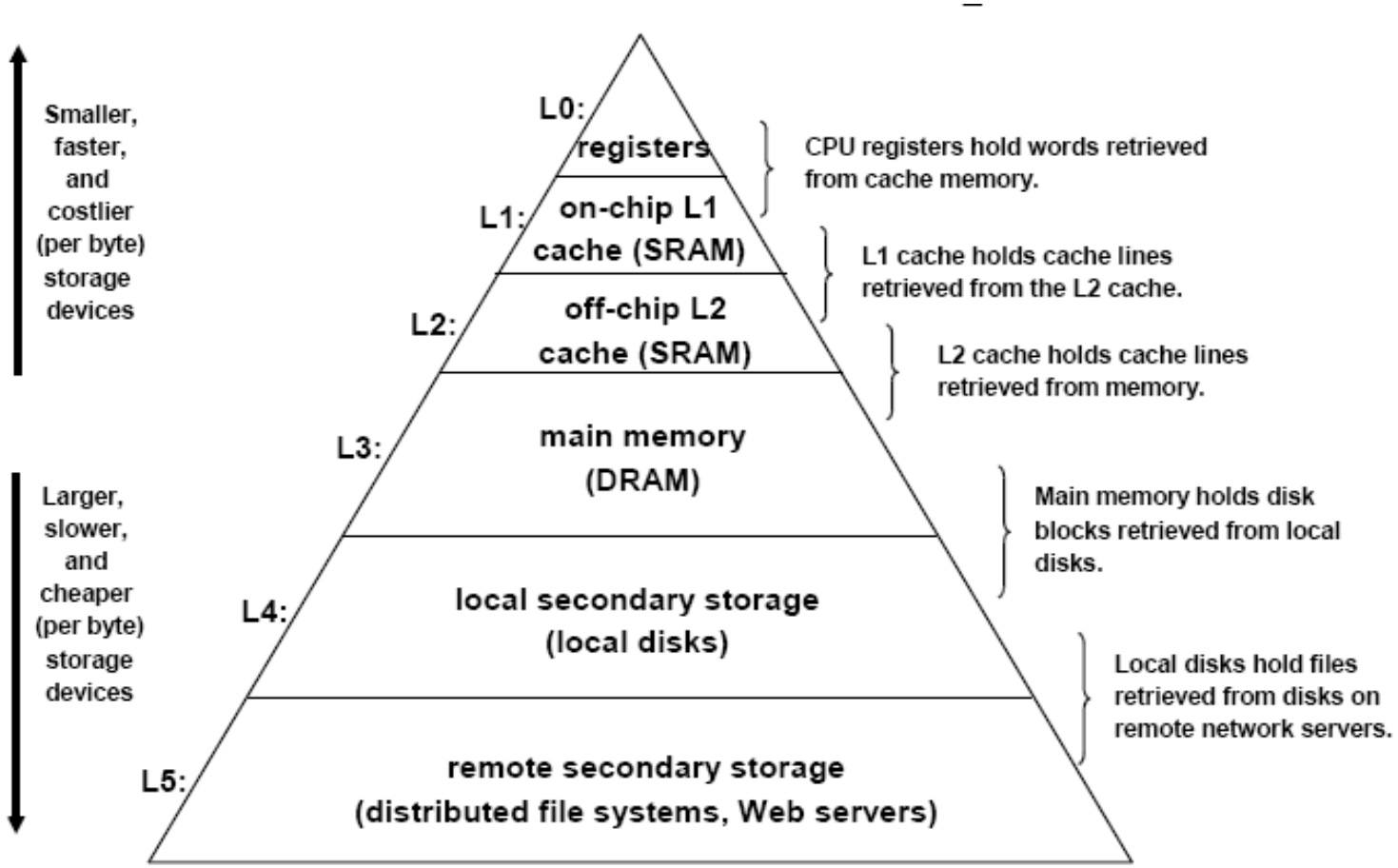


Figure 4.1 The Memory Hierarchy

Memoria

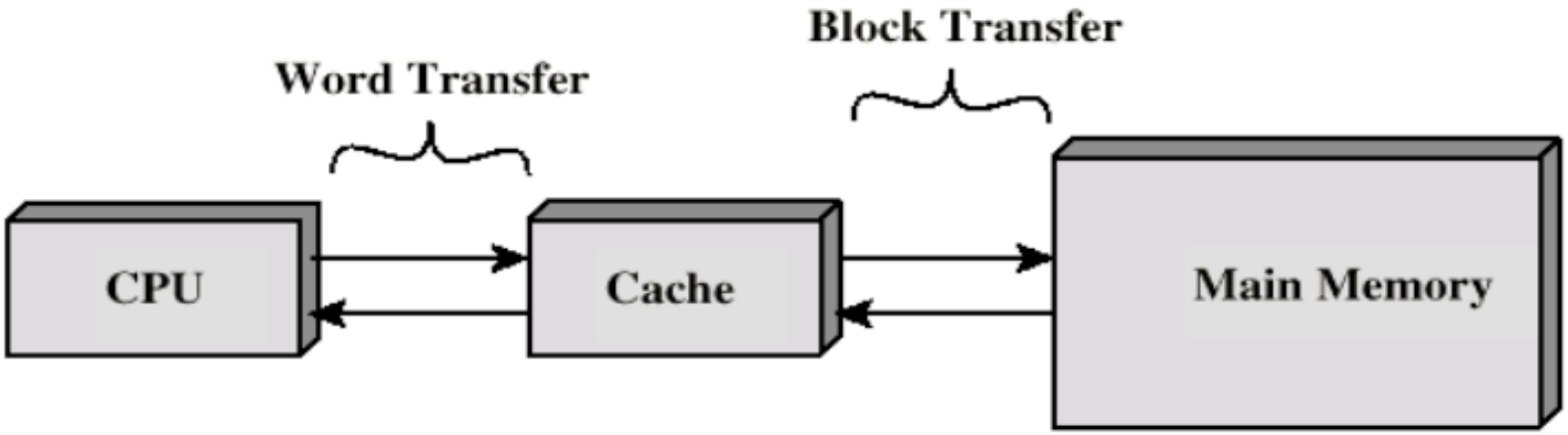

Smaller, faster, and costlier (per byte) storage devices
Larger, slower, and cheaper (per byte) storage devices
L0:
registers
CPU registers hold words retrieved from cache memory.
L1:
on-chip L1 cache (SRAM)
L1 cache holds cache lines retrieved from the L2 cache.
L2:
off-chip L2 cache (SRAM)
L2 cache holds cache lines retrieved from memory.
L3:
main memory (DRAM)
Main memory holds disk blocks retrieved from local disks.
L4:
local secondary storage (local disks)
Local disks hold files retrieved from disks on remote network servers.
L5:
remote secondary storage (distributed file systems, Web servers)

Memoria

- **Principio de Localidad:** Los programas tienden a acceder a datos e instrucciones cercanas a las accedidas recientemente.
- Localidad temporal
  - Si un ítem es referenciado, tenderá a ser referenciado nuevamente a la brevedad
- Localidad espacial
  - Si un ítem es referenciado, los items con direcciones “cercanas” tenderán a ser referenciados a la brevedad

Cache

- Memoria pequeña y rápida, entre el procesador y la memoria principal, diseñada para aprovechar el principio de localidad.



Cache

- CPU requiere el contenido de una posición de memoria:
  - Se busca en el cache
    - Si está (caché hit), se lee del cache (rápido)
    - Si no está (caché miss), se copia el bloque requerido desde la memoria principal al cache
  - Luego se entrega desde el cache a la CPU
- El cache incluye “tags” (etiquetas) para identificar que bloque de memoria principal está en cada línea

Cache

- Memoria Caché - Mejorar desempeño
  - Reducir el miss rate
  - Reducir la penalización por miss
  - Reducir el tiempo de hit en la caché

Organización de la memoria

- Memoria virtual
  - La memoria física es limitada, se utiliza el disco para tener un espacio de memoria mayor.
  - Page Fault – acceso a un dato (página)

    que no está cargado en memoria física.

Memoria

- Algunos aspectos a tener en cuenta al efectuar cálculos matriciales
  - Matrices completas vs dispersas
  - Lenguaje de programación en el que se está trabajando
    - En C se almacena por filas.
    - En Fortran se almacena por columnas.

Memoria

```
for (int i= 0; i < N; i++)
    for (int j= 0; j < N; j++)
        a = a + A(i,j);
```

```
for (int j= 0; i < N; i++)
    for (int i= 0; j < N; j++)
        a = a + A(i,j);
```