# Memory Hierarchy and Cache

Notice: This document is not complete…..

Dheeraj Bhardwaj
Department of Computer Science and Engineering
Indian Institute of Technology, Delhi – 110 016

1

# Memory Hierarchy and Cache

**Cache:** A safe place for hiding and storing things.

(Webster's New World Dictionary (1976))

**Tools for Performance Evaluation**

- **Timing and performance evaluation has been an art**
  - **Resolution of the clock**
  - **Issues about cache effects**
  - **Different systems**

**Situation about to change**

- **Today's processors have counters**

2

## Performance Counters

- Almost all high performance processors include hardware performance counters.
- On most platforms the APIs, if they exist, are not appropriate for a common user, functional or well documented.
- Existing performance counter APIs
  - **Intel Pentium**
  - **SGI MIPS R10000**
  - **IBM Power series**
  - **DEC Alpha pfm pseudo-device interface**
  - **Via Windows 95, NT and Linux on these systems**

3

## Performance Data

- Cycle count
- Floating point
- instruction count
- Integer instruction count
- Instruction count
- Load/store count
- Branch taken / not taken count
- Branch mispredictions
- Pipeline stalls due to memory subsystem
- Pipeline stalls due to resource conflicts
- I/D cache misses for different levels
- Cache invalidations
- TLB misses
- TLB invalidations

4

## Cache and Its Importance in Performance

- **Motivation:**
  - **Time to run code = clock cycles running code + clock cycles waiting for memory**
  - **For many years, CPU's have sped up an average of 50% per year over memory chip speed ups.**
- **Hence, memory access is the bottleneck to computing fast.**
- **Definition of a cache:**

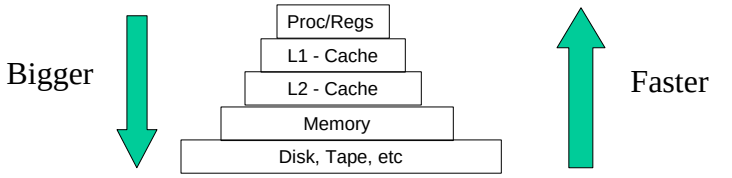
  **Dictionary: a safe place to hide or store things.**

  **Computer: a level in a memory hierarchy.**

5

## What is a cache?

- Small, fast storage used to improve average access time to slow memory.
- Exploits spacial and temporal locality
- In computer architecture, almost everything is a cache!
  - Registers "a cache" on variables – software managed
  - First-level cache a cache on second-level cache
  - Second-level cache a cache on memory
  - Memory a cache on disk (virtual memory)
  - TLB a cache on page table
  - Branch-prediction a cache on prediction information?




6

## Cache Sporting Terms

- Cache Hit: The CPU requests data that is already in the cache. We want to maximize this. The hit rate is the percentage of cache hits.
- Cache Miss: The CPU requests data that is not in cache. We want to minimize this. The miss time is how long it takes to get data, which can be variable and is highly architecture dependent.
- Two level caches are common. The L1 cache is on the CPU chip and the L2 cache is separate. The L1 misses are handled faster than the L2 misses in most designs.
- Upstream caches are closer to the CPU than downstream caches. A typical Alpha CPU has L1-L3 caches. Some MIPS CPU's do, too.

7

## Cache Benefits

- Data cache was designed with two key concepts in mind
  - **Spatial Locality**
    - When an element is referenced its neighbors will be referenced too
    - Cache lines are fetched together
    - Work on consecutive data elements in the same cache line
  - **Temporal Locality**
    - When an element is referenced, it might be referenced again soon
    - Arrange code so that data in cache is reused often

**Cache-Related Terms**

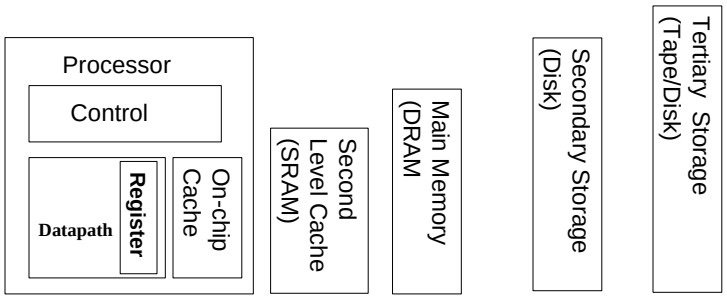
*Least Recently Used (LRU):* Cache replacement strategy for set associative caches. The cache block that is least recently used is replaced with a new block.

*Random Replace*: Cache replacement strategy for set associative caches. A cache block is randomly replaced.

8

## A Modern Memory Hierarchy

- By taking advantage of the principle of locality:
  - Present the user with as much memory as is available in the cheapest technology.
  - Provide access at the speed offered by the fastest technology.




| | | | | |
|---|---|---|---|---|
| Speed (ns): 1s | 10s | 100s | 1000000s | 10,000,000,000s |
| Size (bytes): 100s | Ks | Ms | Gs | Ts |

9

## Uniprocessor Reality

- Modern processors use a variety of techniques for performance
  - **caches**
    - small amount of fast memory where values are “cached” in hope of reusing recently used or nearby data
    - different memory ops can have very different costs
  - **parallelism**
    - superscalar processors have multiple “functional units” that can run in parallel
    - different orders, instruction mixes have different costs
  - **pipelining**
    - a form of parallelism, like an assembly line in a factory
- Why is this your problem?
  - In theory, compilers understand all of this and can optimize your program; in practice they don’t.

10

## Traditional Four Questions for Memory Hierarchy Designers

- Q1: Where can a block be placed in the upper level? (Block placement)
  - Fully Associative, Set Associative, Direct Mapped
- Q2: How is a block found if it is in the upper level? (Block identification)
  - Tag/Block
- Q3: Which block should be replaced on a miss? (Block replacement)
  - Random, LRU
- Q4: What happens on a write?

  (Write strategy)
  - Write Back or Write Through (with Write Buffer)

11

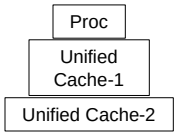
## Cache-Related Terms

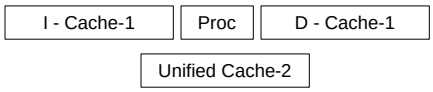
- ICACHE : Instruction cache
- DCACHE (L1) : Data cache closest to registers
- SCACHE (L2) : Secondary data cache
  - Data from SCACHE has to go through DCACHE to registers
  - SCACHE is larger than DCACHE
  - Not all processors have SCACHE
- **Unified versus Split Caches**
  - This refers to having a single or separate caches for data and machine instructions.
  - Split is obviously superior. It reduces thrashing, which we will come to shortly..

12

## Unified Vs Split Caches

- Unified vs Separate I&D




- Example:
  - 16KB I&D: Inst miss rate=0.64%, Data miss rate=6.47%
  - 32KB unified: Aggregate miss rate=1.99%
- Which is better (ignore L2 cache)?
  - Assume 33% data ops . 75% accesses from instructions (1.0/1.33)
  - hit time=1, miss time=50
  - Note that data hit has 1 stall for unified cache (only one port)

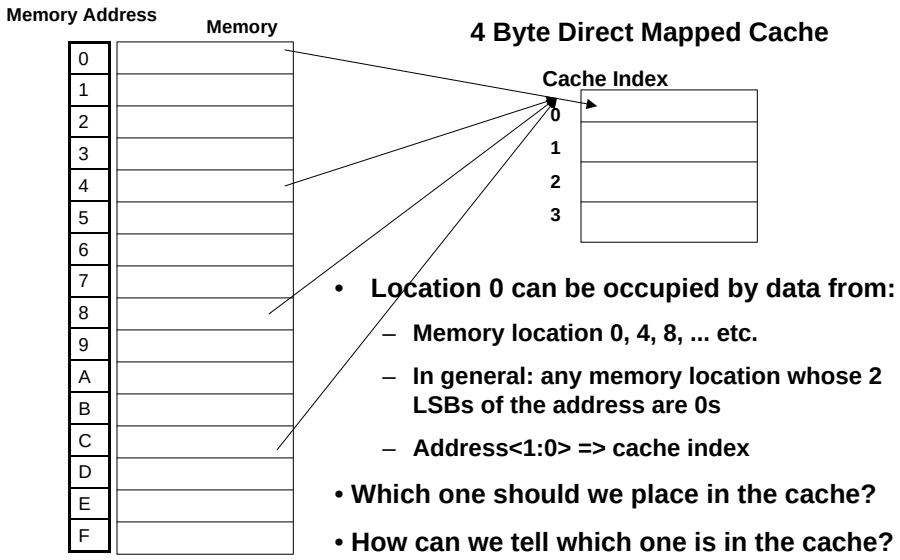
13

## Where to misses come from?

- Classifying Misses: 3 Cs
  - ***Compulsory***—The first access to a block is not in the cache, so the block must be brought into the cache. Also called cold start misses or first reference misses. (Misses in even an Infinite Cache)
  - ***Capacity***—If the cache cannot contain all the blocks needed during execution of a program, capacity misses will occur due to blocks being discarded and later retrieved. (Misses in Fully Associative Size X Cache)
  - ***Conflict***—If block-placement strategy is set associative or direct mapped, conflict misses (in addition to compulsory & capacity misses) will occur because a block can be discarded and later retrieved if too many blocks map to its set. Also called collision misses or interference misses. (Misses in N-way Associative, Size X Cache)
- 4th “C”: (for parallel)
  - Coherence- Misses caused by cache coherence.

14

## Simples Cache: Direct Mapped




- **Location 0 can be occupied by data from:**
  - **Memory location 0, 4, 8, ... etc.**
  - **In general: any memory location whose 2 LSBs of the address are 0s**
  - **Address<1:0> => cache index**
- **Which one should we place in the cache?**
- **How can we tell which one is in the cache?**

15

## Cache Mapping Strategies

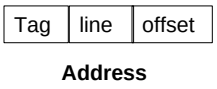
- There are two common sets of methods in use for determining which cache lines are used to hold copies of memory lines.
- Direct: Cache address = memory address MODULO cache size.
- Set associative: There are N cache banks and memory is assigned to just one of the banks. There are three algorithmic choices for which line to replace:
  - Random: Choose any line using an analog random number generator. This is cheap and simple to make.
  - LRU (least recently used): Preserves temporal locality, but is expensive. This is not much better than random according to (biased) studies.
  - FIFO (first in, first out): Random is far superior.

16

## Cache Basics

- Cache hit: a memory access that is found in the cache
  - cheap
- Cache miss: a memory access that is not in the cache
  - expensive, because we need to get the data from elsewhere
- Consider a tiny cache (for illustration only)

| | |
|---|---|
| X10010 | X001 |
| X010 | X011 |
| X100 | X101 |
| X110 | X111 |




**Address**

- Cache line length: number of bytes loaded together in one entry
- Direct mapped: only one address (line) in a given range in cache
- Associative: 2 or more lines with different addresses exist

17

## Direct-Mapped Cache

- Direct mapped cache: A block from main memory can go in exactly one place in the cache. This is called direct mapped because there is direct mapping from any block address in memory to a single location in the cache.

18

## Fully Associative Cache

- Fully Associative Cache : A block from main memory can be placed in any location in the cache. This is called fully associative because a block in main memory may be associated with any entry in the cache.

19

## Diagrams

20

## Tuning for Caches

1. Preserve locality.
2. Reduce cache thrashing.
3. Loop blocking when out of cache.
4. Software pipelining.

21

## Registers

- Registers are the source and destination of most CPU data operations.
- They hold one element each.
- They are made of static RAM (SRAM), which is very expensive.
- The access time is usually 1-1.5 CPU clock cycles.
- Registers are at the top of the memory subsystem.

22

## Memory Banking

- This started in the 1960's with both 2 and 4 way interleaved memory banks. Each bank can produce one unit of memory per bank cycle. Multiple reads and writes are possible in parallel.
  - Memory chips must internally recover from an access before it is reaccessed
- The bank cycle time is currently 4-8 times the CPU clock time and getting worse every year.
- Very fast memory (e.g., SRAM) is unaffordable in large quantities.
- This is not perfect. Consider a 4 way interleaved memory and a stride 4 algorithm. This is equivalent to non-interleaved memory systems.

23

## The Principle of Locality

- Principal of Locality
  - Program access a relatively small portion of the address space at any instant of time.
- Two Different Types of Locality:
  - **Temporal Locality (Locality in Time):** If an item is referenced, it will tend to be referenced again soon (e.g., loops, reuse)
  - **Spatial Locality (Locality in Space):** If an item is referenced, items whose addresses are close by tend to be referenced soon (e.g., straightline code, array access)
- Last 15 years, HW relied on locality for speed

24

## Principals of Locality

- Temporal: an item referenced now will be again soon.
- Spatial: an item referenced now causes neighbors to be referenced soon.
- Lines, not words, are moved between memory levels. Both principals are satisfied. There is an optimal line size based on the properties of the data bus and the memory subsystem designs.
- Cache lines are typically 32-128 bytes with 1024 being the longest currently.

25

## What happens on a write?

- Write through—The information is written to both the block in the cache and to the block in the lower-level memory.
- Write back—The information is written only to the block in the cache. The modified cache block is written to main memory only when it is replaced in cache.
  - is block clean or dirty?
- Pros and Cons of each?
  - WT: read misses cannot result in writes
  - WB: no repeated writes to same location
- WT always combined with write buffers so that don't wait for lower level memory

26

## Cache Thrashing

- Thrashing occurs when frequently used cache lines replace each other. There are three primary causes for thrashing:
  - Instructions and data can conflict, particularly in unified caches.
  - Too many variables or too large of arrays are accessed that do not fit into cache.
  - Indirect addressing, e.g., sparse matrices.
- Machine architects can add sets to the associativity. Users can buy another vendor's machine. However, neither solution is realistic.

27

## Cache Coherence for Multiprocessors

- All data must be coherent between memory levels. Multiple processors with separate caches must inform the other processors quickly about data modifications (by the cache line). Only hardware is fast enough to do this.
- Standard protocols on multiprocessors:
  - Snoopy: all processors monitor the memory bus.
  - Directory based: Cache lines maintain an extra 2 bits per processor to maintain clean/dirty status bits.
- False sharing occurs when two different shared variables are located in the in the same cache block, causing the block to be exchanged between the processors even though the processors are accessing different variables. Size of block (line) important.

28

## Processor Stall

- Processor stall is the condition where a cache miss occurs and the processor waits on the data.
- A better design allows any instruction in the instruction queue to execute that is ready. You see this in the design of some RISC CPU's, e.g., the RS6000 line.
- Memory subsystems with hardware data prefetch allow scheduling of data movement to cache.
- Software pipelining can be done when loops are unrolled. In this case, the data movement overlaps with computing, usually with reuse of the data.
- out of order execution, software pipelining, and **prefetch.**

29

## Indirect Addressing

```
d = 0
    do i = 1,n
    j = ind(i)
        d = d + sqrt( x(j)*x(j) + y(j)*y(j) + z(j)*z(j) )
    end do
```

- Change loop statement to

```
d = d + sqrt( r(1,j)*r(1,j) + r(2,j)*r(2,j) + r(3,j)*r(3,j) )
```

- Note that r(1,j)-r(3,j) are in contiguous memory and probably are in the same cache line (d is probably in a register and is irrelevant). The original form uses 3 cache lines at every instance of the loop and can cause cache thrashing.

30

## Cache Thrashing by Memory Allocation

```
parameter ( m = 1024*1024 )
real a(m), b(m)
```

- For a 4 Mb direct mapped cache, a(i) and b(i) are always mapped to the same cache line.This is trivially avoided using padding.

```
real a(m), extra(32), b(m)
```

- extra is at least 128 bytes in length, which is longer than a cache line on all but one memory subsystem that is available today.

31

## Cache Blocking

- We want blocks to fit into cache. On parallel computers we have p x cache so that data may fit into cache on p processors, but not one. This leads to superlinear speed up! Consider matrix-matrix multiply.

```
do k = 1,n
    do j = 1,n
        do i = 1,n
            c(i, j) = c(i, j) + a(i,k)*b(k,j)
        end do
    end do
end do
```

- An alternate form is ...

32

## Cache Blocking

```
do kk = 1,n,nblk
    do jj = 1,n,nblk
        do ii = 1,n,nblk
            do k = kk,kk+nblk-1
                do j = jj,jj+nblk-1
                    do i = ii,ii+nblk-1
                c(i,j) = c(i, j) + a(i,k) * b(k,j)
            end do
        . . .
            end do
```

33

## Summary: The Cache Design Space

- **Several Interacting dimensions**
  - **cache size**
  - **block size**
  - **associativity**
  - **replacement policy**
  - **write-through vs write-back**
  - **write allocation**
- **The optimal choice is a compromise**
  - **depends on access characteristics**
    - **workload**
    - **use (I-cache, D-cache, TLB)**
  - **depends on technology / cost**
- **• Simplicity often wins**

34

## Lessons

- The actual performance of a simple program can be a complicated function of the architecture
- Slight changes in the architecture or program change the performance significantly
- Since we want to write fast programs, we must take the architecture into account, even on uniprocessors
- Since the actual performance is so complicated, we need simple models to help us design efficient algorithms
- We will illustrate with a common technique for improving cache performance, called blocking

35

## Optimizing Matrix Addition for Caches

- Dimension A(n,n), B(n,n), C(n,n)
- A, B, C stored by column (as in Fortran)
- Algorithm 1:
  - for i=1:n, for j=1:n, A(i,j) = B(i,j) + C(i,j)
- Algorithm 2:
  - for j=1:n, for i=1:n, A(i,j) = B(i,j) + C(i,j)
- What is “memory access pattern” for Algs 1 and 2?
- Which is faster?
- What if A, B, C stored by row (as in C)?

36