COMPUTER ARCHITECTURE

Code: CT173

Part X: CacheOptimization

MSc. NGUYEN Huu Van LONG

Department of Computer Networking and Communication,

College of Information & Communication Technology,

CanTho University

Agenda

- **Reduce the Miss Rate**
  - Reduce Misses Rates via *Larger Cache Sizes*
  - Reduce Misses Rates via *Larger Block Size*
  - Reduce Misses Rates via *Higher Associativity*
  - Reducing Misses Rates via *Victim Cache*
  - Reducing Misses Rates via *Pseudo-Associativity & Way Prediction*
  - Reducing Misses Rates by *HW Prefetching Instructions / Data*
  - Reducing Misses Rates by *SW Prefetching Data*
  - Reducing Misses Rates by *Compiler Optimizations*
- **Reduce the Miss Penalty**
  - Priority to Read Misses
  - Sub-block placement
  - Early Restart and Critical Word first
  - Non-blocking caches
  - Second (or Multilevel) caches
- **Reduce the Hit Time**

Improving Cache Performance

- **Average Memory Access Time**
  - **AMAT = Hit Time + Miss Rate * Miss Penalty**
- **How to improve cache performance**
  - Reduce the miss rate
  - Reduce the miss penalty
  - Reduce the hit time
- **Overall goal: Balancing fast hits and few misses**

Classifying Cache Misses

- **Three major categories** of cache misses
  - **Compulsory Misses**: cold start misses or *first reference misses*
    - The first access to a block is not in the cache, so the block must be loaded in the cache. Also called cold start misses or first reference misses.
    - Misses in even an infinite cache: *Compulsory misses are independent of cache size*
  - **Capacity Misses**: can be *reduced by increasing cache size*
    - 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*
    - Capacity misses decrease as Capacity increases
  - **Conflict Misses**: can be *reduced by increasing cache size, associativity*
    - 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 the same location in the cache*
    - Also called **collision misses** or **interference misses**.
    - *Conflict misses decrease as associativity increases*: Fully associative placement avoids all conflict misses but full associativity is expensive in area

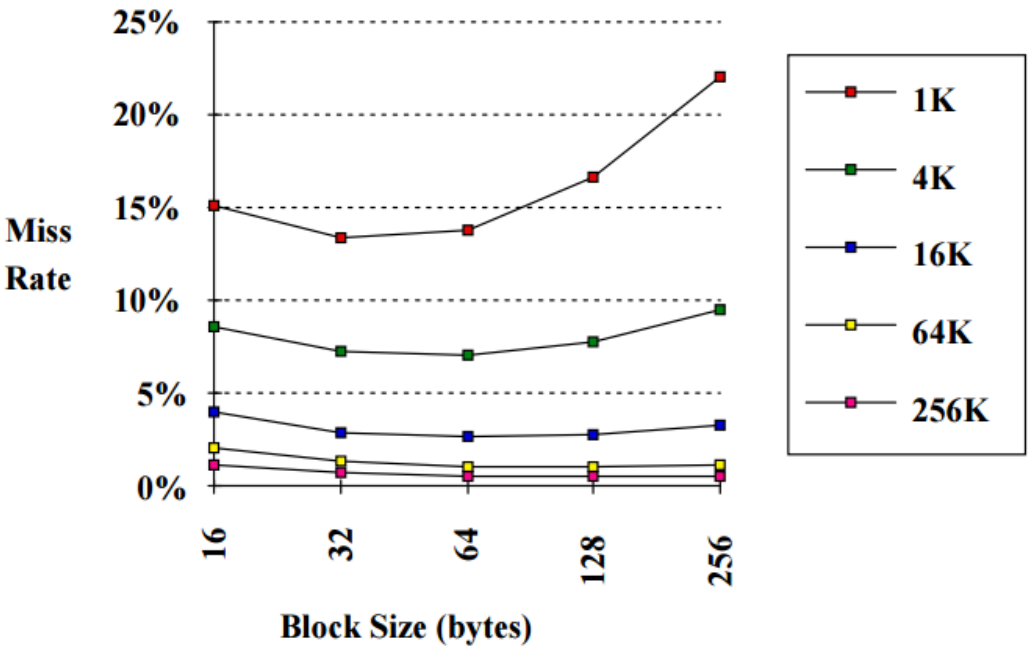
How to reduce Misses Rates

- Reduce Misses Rates via Larger Cache Sizes
- Reduce Misses Rates via Larger Block Size
- Reduce Misses Rates via Higher Associativity
- Reducing Misses Rates via Victim Cache
- Reducing Misses Rates via Pseudo-Associativity & Way Prediction
- Reducing Misses Rates by HW Prefetching Instructions / Data
- Reducing Misses Rates by SW Prefetching Data
- Reducing Misses Rates by Compiler Optimizations

Larger Block Size vs Larger Cache Size

- **Reducing Misses Rates via Larger Block Size**
  - *Miss rate goes up* if the block size is too large with respect to cache size
  - Larger block size will reduce compulsory misses *taking advantage of spatial locality*
  - Main drawbacks:
    - Larger blocks *increase miss penalty*
    - Larger blocks reduce the number of blocks so *increase conflict misses* (and *even capacity misses*) if the cache is small

- **Reducing Misses Rates via Larger Cache Size**
  - Obvious way to reduce capacity misses: to increase cache capacity
  - Drawback: Increases hit time, area, power consumption and cost



Reduce Misses Rates: Higher Associativity

- **Reducing Misses via Higher Associativity**
  - Higher associativity *decreases the conflict misses*
  - Main drawbacks
    - It increases hit time due to the complexity
    - It *increases area, power consumption and cost*
- **2:1 Cache Rule**:
  - Miss Rate Cache Size N = Miss Rate 2 ways Cache Size N/2
- **Multiple banked caches** to *increase cache bandwidth*
  - Organize cache as independent banks to support simultaneous access
    - ARM Cortex-A8 supports 1-4 banks for L2
    - Intel i7 supports 4 banks for L1 and 8 banks for L2
  - Interleave banks according to block address (sequential interleaving)

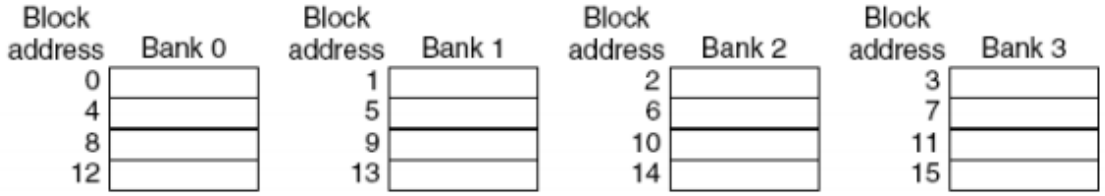
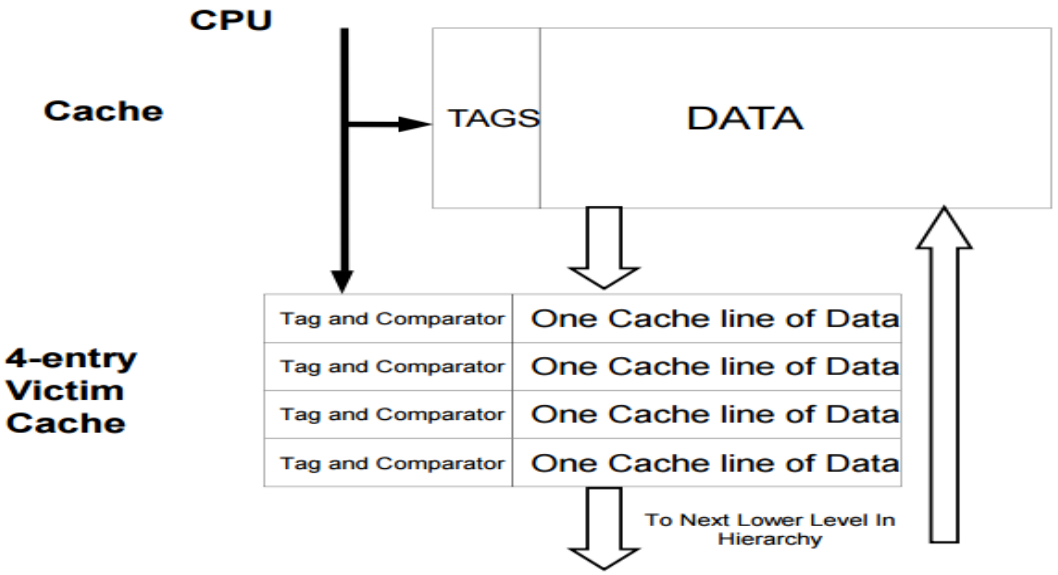


Figure 2.6 Four-way interleaved cache banks using block addressing. Assuming 64 bytes per blocks, each of these addresses would be multiplied by 64 to get byte addressing.

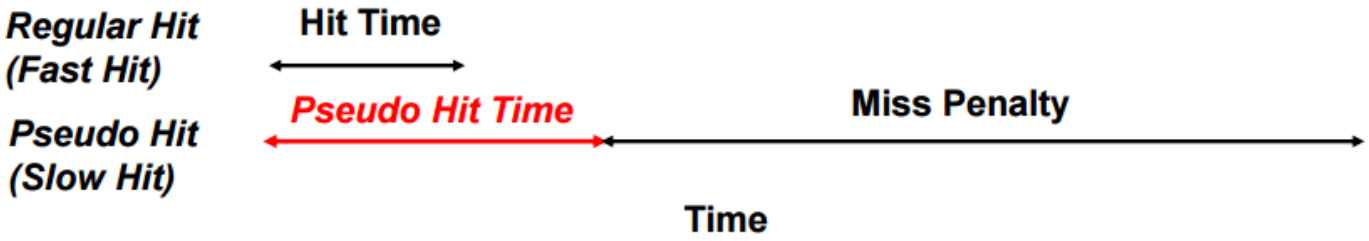
Reduce Misses Rates: Victim cache

- Add **buffer** to place *data discarded from cache* to better *exploit temporal locality*
- **Victim cache** is a *small fully associative cache* containing data discarded from cache
- **Victim cache** *placed between cache and its refilling path*
- **Victim cache** *is checked on a miss* to see if it has the required data *before going to lower-level memory*
- If the block is found in victim cache, the victim block and the cache block are swapped

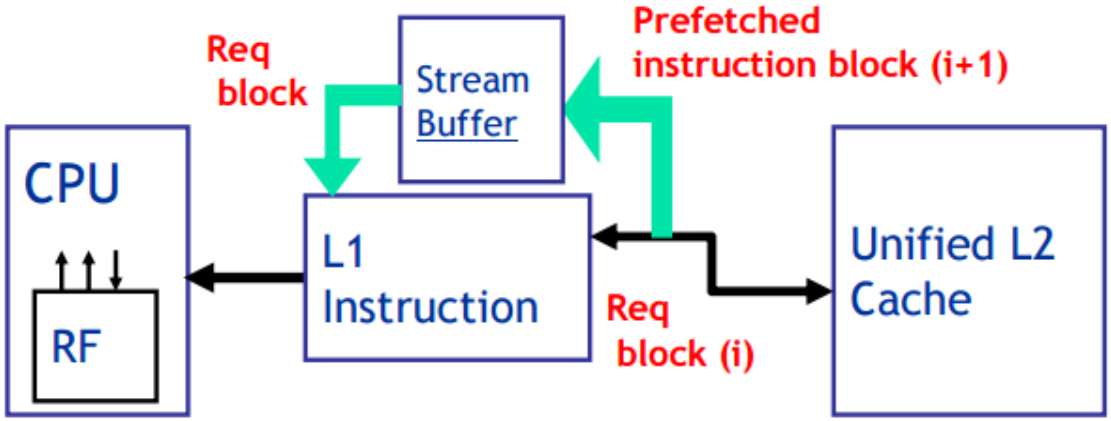


Pseudo Associativity vs Hardware Prefetching

- **Reducing Misses via "Pseudo - Associativity" and Way Prediction**
  - Basic idea: To improve hit time, *predict the way to pre-set the mux*: way misprediction takes longer hit time (pseudo hit time)



- **Reducing Misses by Hardware Prefetching of Instructions & Data**
  - Basic idea: *Pre-fetching instructions or data* before they are requested by the processor
  - Prefetching can be done in cache or in an external stream buffer

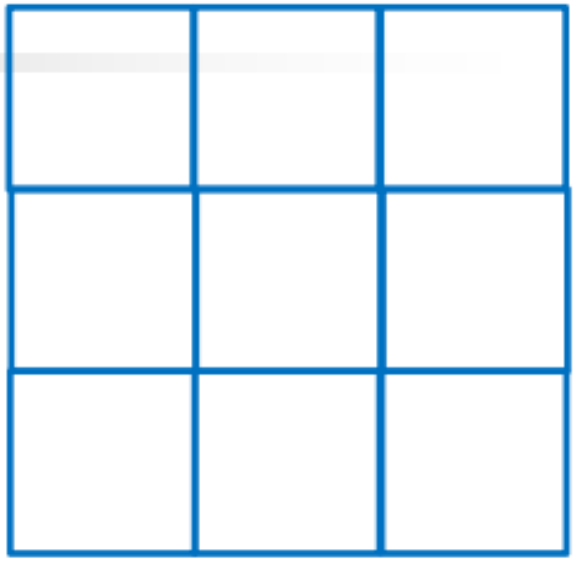


Software Prefetching vs Compiler Optimization

- **Reducing Misses by Software Prefetching Data**
  - Compiler-controlled pre-fetching data or instructions (*the compiler can help in reducing useless pre-fetching*)
- **Reducing Misses by Compiler Optimizations**
  - **Managing instructions**:
    - Reorder instructions in memory so as to reduce conflict misses
    - Profiling to look at instruction conflicts
  - **Managing Data**
    - *Merging Arrays*: improve spatial locality by single array of compound elements vs. 2 arrays (to operate on data in the same cache block)
    - *Loop Interchange*: improve spatial locality by changing loops nesting to access data in the order stored in memory (re-ordering maximizes re-use of data in a cache block)
    - *Loop Fusion*: improve spatial locality by combining 2 independent loops that have same looping and some variables overlap
    - *Loop Blocking*: Improve temporal locality by accessing "sub-blocks" of data repeatedly vs. accessing by columns or rows

Reduce Misses Rates: Compiler Optimization

```
/* Before */
for (k = 0; k < 100; k = k+1)
      for (j = 0; j < 100; j = j+1)
       for (i = 0; i < 5000; i = i+1)
                    x[i][j] = 2 * x[i][j];
/* After */
for (k = 0; k < 100; k = k+1)
      for (i = 0; i < 5000; i = i+1)
       for (j = 0; j < 100; j = j+1)
                    x[i][j] = 2 * x[i][j];
```



```
/* Before: 2 sequential arrays */
int val[SIZE];
int key[SIZE];
/* After: 1 array of stuctures */
struct merge {
 int val;
 int key;
};
struct merge merged_array[SIZE];
```

```
/* Before */
for (i = 0; i < N; i = i+1)
 for (j = 0; j < N; j = j+1)
       a[i][j] = 1/b[i][j] * c[i][j];
for (i = 0; i < N; i = i+1)
 for (j = 0; j < N; j = j+1)
       d[i][j] = a[i][j] + c[i][j];
/* After */
for (i = 0; i < N; i = i+1)
 for (j = 0; j < N; j = j+1)
 {     a[i][j] = 1/b[i][j] * c[i][j];
       d[i][j] = a[i][j] + c[i][j];}
```

How to reduce Misses Penalties

- Priority to Read Misses
- Sub-block placement
- Early Restart and Critical Word first
- Non-blocking caches
- Second (or Multilevel) caches

Reduce Miss Penalty

- **Read Priority over Write on Miss**
  - Basic idea: *Giving higher priority to read misses over writes*
  - Write buffer must be properly sized
  - The approach can *complicate the memory access* because the *write buffer might hold the updated value* if a memory location needed on a read miss
- **Sub block placement**
  - *Don't have to load full block on a miss*: move sub blocks
  - Have valid bits per sub-block to indicate validity
- **Early restart and Critical word first**
  - *Usually CPU needs just one word of the block on a miss*
  - Basic idea: *Don't wait for full block to be loaded before restarting CPU* (by sending the requested missed word)
    - **Early restart**: *Request the words in normal order* from memory, but *as soon as the requested word of the block arrives, send it to the CPU and let the CPU continue execution* while filling in the rest of the words in the cache block
    - **Critical Word First**: *Request the missed word first from memory* and send it to the CPU as soon as it arrives; let the CPU continue execution while filling the rest of the words in the cache block. Also called requested word first

Reduce Miss Penalty

- **Non blocking cache (Hit under Miss)**
  - Allows data cache to continue to *supply cache hits during a previous miss* but requires
    - **Out-of-order execution CPU**: the CPU needs to do not stall on a cache miss (CPU can continue fetching instructions from I-cache while waiting for D-cache to return the missing data)
  - *Reduces the effective miss penalty* by working during miss instead of stalling CPUs on misses
- **Hit under Multiple Miss or Miss under Miss**
  - May further lower the effective miss penalty by *overlapping multiple misses*
  - *Significantly increases the complexity of the cache controller* as there can be multiple outstanding memory accesses
  - *Requires multiple memory banks* (otherwise cannot support) to serve multiple misses
  - Pentium Pro allows 4 outstanding memory misses

Reduce Miss Penalty

- **Second Level or Multilevel Cache**
  - *L1 cache small enough* to match the fast CPU clock cycle
  - *L2 cache large enough* to capture many accesses that would go to main memory reducing the effective miss penalty
  - *L2 cache not tied to CPU clock cycle*
    - Speed of L1 affects the CPU clock rate
    - Speed of L2 only affects the miss penalty of L1
- AMAT = Hit Time_{L1} + Miss Rate_{L1} * (Hit Time_{L2} + Miss Rate_{L2} * Miss Penalty_{L2})
  - *Global Miss Rate = Miss Rate_{L1} * Miss Rate_{L2}* indicates what fraction of the memory accesses from CPU go all the way to main memory

How to reduce Hit Time

- Very important: The speed of L1 (Hit Time) affects the CPU clock rate
- Fast Hit Time via Small and Simple L1 caches → using *Direct Mapped cache*:
  - Fast access
  - Tag comparing and data transmission could be overlap
  - Reduce power
- Fast Hit Time by Avoiding Address Translation → *Virtual cache (Translation Look aside Buffer - TLB) vs Physical Cache*
- Fast Hit Time via Pipeline Writes
- Fast Writes on Misses Via Small Sub-blocks for Write Through

Cache Optimization Summary

	Technique	MR	MP	HT	Complexity
miss rate	Larger Block Size	+	–		0
	Higher Associativity	+		–	1
	Victim Caches	+			2
	Pseudo-Associative Caches	+			2
	HW Prefetching of Instr/Data	+			2
	Compiler Controlled Prefetching	+			3
	Compiler Reduce Misses	+			0
miss penalty	Priority to Read Misses		+		1
	Subblock Placement		+	+	1
	Early Restart & Critical Word 1st		+		2
	Non-Blocking Caches		+		3
	Second Level Caches		+		2
hit time	Small & Simple Caches	–		+	0
	Avoiding Address Translation			+	2
	Pipelining Writes			+	1

Cache Optimization Summary

Technique	Hit time	Band-width	Miss penalty	Miss rate	Power consumption	Hardware cost/ complexity	Comment
Small and simple caches	+			–	+	0	Trivial; widely used
Way-predicting caches	+				+	1	Used in Pentium 4
Pipelined cache access	–	+				1	Widely used
Nonblocking caches		+	+			3	Widely used
Banked caches		+			+	1	Used in L2 of both i7 and Cortex-A8
Critical word first and early restart			+			2	Widely used
Merging write buffer			+			1	Widely used with write through
Compiler techniques to reduce cache misses				+		0	Software is a challenge, but many compilers handle common linear algebra calculations
Hardware prefetching of instructions and data			+	+	–	2 instr., 3 data	Most provide prefetch instructions; modern high-end processors also automatically prefetch in hardware.
Compiler-controlled prefetching			+	+		3	Needs nonblocking cache; possible instruction overhead; in many CPUs

Figure 2.11 Summary of 10 advanced cache optimizations showing impact on cache performance, power consumption, and complexity. Although generally a technique helps only one factor, prefetching can reduce misses if done sufficiently early; if not, it can reduce miss penalty. + means that the technique improves the factor, – means it hurts that factor, and blank means it has no impact. The complexity measure is subjective, with 0 being the easiest and 3 being a challenge.

Question?