UNIT 3
Memory Hierarchy Design

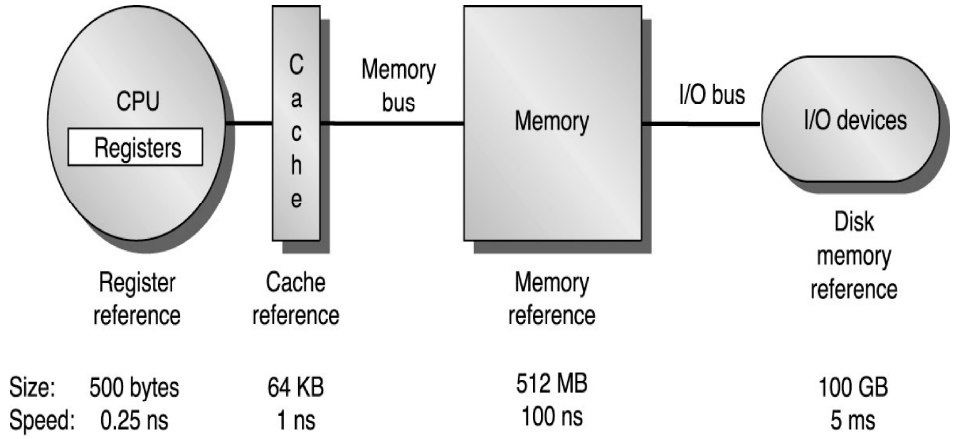
Outline

- Introduction- Review of concepts
- Six Basic Cache Optimizations
- Ten Advanced Cache Optimizations
- Memory Technology and Optimizations

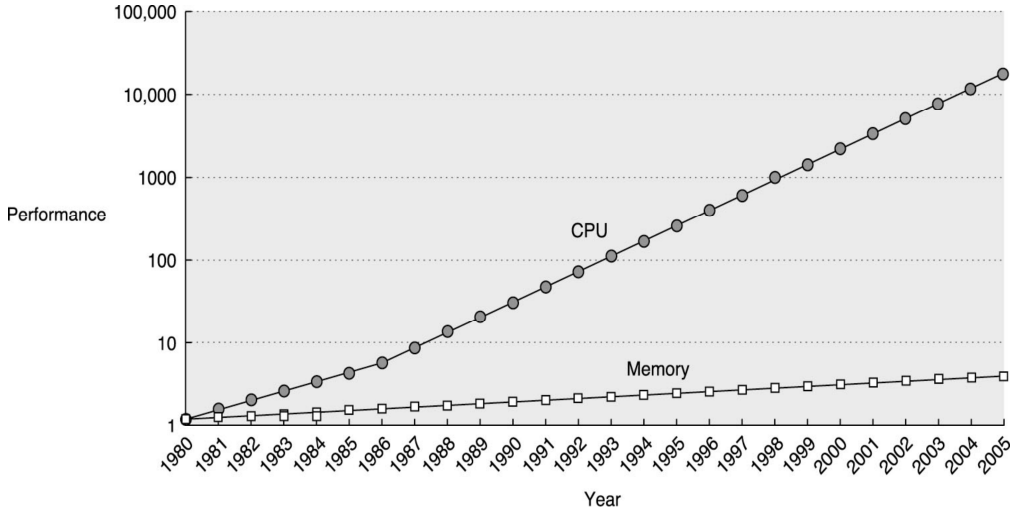
Introduction

– The necessity of memory-hierarchy in a computer system design is enabled by the following two factors:
  - **Locality of reference**: The nature of program behavior
  - **Large gap in speed** between CPU and mass storage devices such as DRAM.
– Level of memory hierarchy
  - High level <--- --> Low level
  - CPU-Register, Cache, Main-memory, Disk
  - The levels of the hierarchy subset one another: all data in one level is also found in the level below.

Memory Hierarchy



Speed Gap between CPU and DRAM



ABCs of Caches

- Recalling some terms
  - ***Cache***: The name given to the first level of the memory hierarchy encountered once the address leaves the CPU.
  - **Miss rate**: The fraction of accesses not in the cache.
  - **Miss penalty**: The additional time to service the miss.
  - **Block**: The minimum unit of information that can be present in the cache.

Cache Performance

– Formula for performance evaluation

- **CPU execution time = (CPU clock cycles + Memory stall cycles) * Clock cycle time**
- **Memory stall cycles = IC * Memory reference per instruction *miss rate *miss penalty**
- Measure of memory-hierarchy performance

  Average memory access time = Hit time + Miss rate * Miss penalty

Example Assume we have a computer where the clocks per instruction (CPI) is 1.0 when all memory accesses hit in the cache. The only data accesses are loads and stores, and these total 50% of the instructions. If the miss penalty is 25 clock cycles and the miss rate is 2%, how much faster would the computer be if all instructions were cache hits?

Answer First compute the performance for the computer that always hits:

$$\begin{aligned}\text{CPU execution time} &= (\text{CPU clock cycles} + \text{Memory stall cycles}) \times \text{Clock cycle}\\ &= (\text{IC} \times \text{CPI} + 0) \times \text{Clock cycle}\\ &= \text{IC} \times 1.0 \times \text{Clock cycle}\end{aligned}$$

Now for the computer with the real cache, first we compute memory stall cycles:

$$\begin{aligned}\text{Memory stall cycles} &= \text{IC} \times \frac{\text{Memory accesses}}{\text{Instruction}} \times \text{Miss rate} \times \text{Miss penalty}\\ &= \text{IC} \times (1 + 0.5) \times 0.02 \times 25\\ &= \text{IC} \times 0.75\end{aligned}$$

where the middle term (1 + 0.5) represents one instruction access and 0.5 data accesses per instruction. The total performance is thus

$$\text{CPU execution time}_{\text{cache}} = (\text{IC} \times 1.0 + \text{IC} \times 0.75) \times \text{Clock cycle}$$
$$= 1.75 \times \text{IC} \times \text{Clock cycle}$$

The performance ratio is the inverse of the execution times:

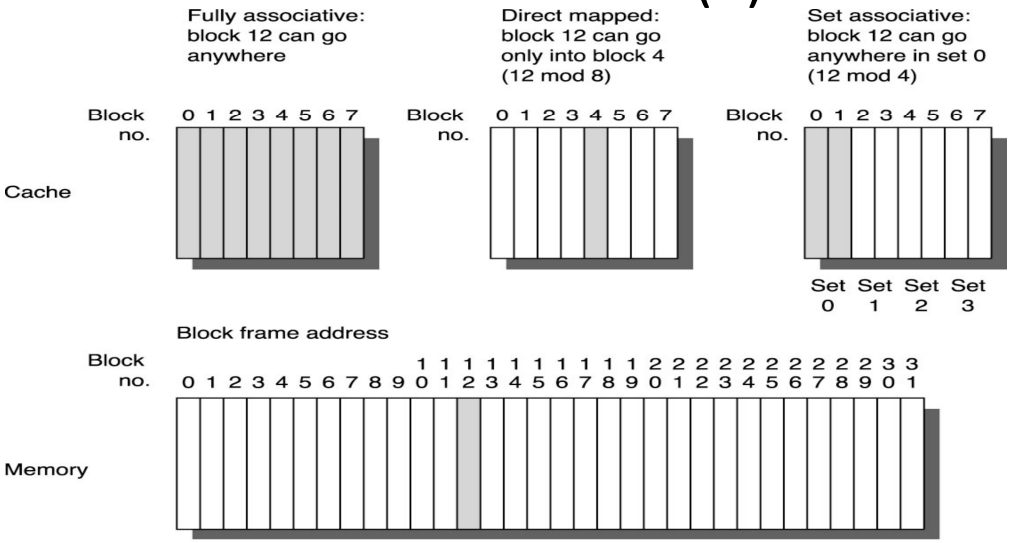
$$\frac{\text{CPU execution time}_{\text{cache}}}{\text{CPU execution time}} = \frac{1.75 \times \text{IC} \times \text{Clock cycle}}{1.0 \times \text{IC} \times \text{Clock cycle}}$$
$$= 1.75$$

The computer with no cache misses is 1.75 times faster.

Block Placement (1)

- Q1: Where can a block be placed in a cache?
  - Direct mapped: Each block has only one place it can appear in the cache. The mapping is usually
    (*Block address*) **MOD** (*Number of blocks in cache*)
  - Fully associative: A block can be placed anywhere in the cache.
  - Set associative: A block can be placed in a restricted set of places in the cache. A set is a group of blocks in the cache. A block is first mapped onto a set, and then the block can be placed anywhere within that set. The set is usually obtained by
    (*block address*) **MOD** (*Number of sets in a cache*)
    - If there are *n* blocks in a set, the cache is called *n*-way set associative.

Block Placement (2)



Block Replacement

- Q3: Which block should be replaced on a cache miss?
  - For direct mapped cache, the answer is obvious.
  - For set associative or fully associative cache, the following two strategies can be used:
    - Random:- to spread allocation automatically, candidate blocks are randomly selected.
    - Least-recently used (LRU):- to reduce the chance of throwing out information that will be needed soon, accesses to blocks are recorded. LRU relies on a corollary of locality
    - First in, first out (FIFO):- Because LRU can be complicated to calculate, this approximate LRU by determining the oldest block rather than the LRU.

Write Strategy

- Q4: What happens on a write?
  - Traffic patterns
    - "Writes" take about 7% of the overall memory traffic and take about 28% of the data cache traffic.
  - "Read" can be done faster than "write"
    - In reading, the block data can be read at the same time that the tag is read and compared.
      -If the read is hit then requested data passed to the processor
      -If the read is a miss then there is no benefit and also no harm except the power consumption.
    - In writing, modifying a block cannot begin until the tag is checked to see if the address is a hit.
    - Since tag checking cannot be done in parallel, write normally takes longer than reads

Write Policies

- Write policies
  - Write through (or store through)
    - Write to both the block in the cache and the block in the lower-level memory.
  - Write back
    - Write only to the block in the cache. A dirty bit, attached to each block in the cache, is set when the block is modified. When a block is being replaced and the dirty bit is set, the block is copied back to main memory. Clean bocks are not copied back to MM. This can reduce bus traffic.

•**A write-through cache uses no-write allocate.**
•Here, subsequent writes have no advantage, since they still need to be written directly to the backing store, read after write is hit

•**write-back cache uses write allocate**,
•hoping for subsequent writes (or even reads) to the same location, which is now cached.

Example

- Assume a fully associative write-back cache with many cache entries that starts empty. Below is a sequence of five memory operations (the address is in square brackets):
  Write Mem[100];
  WriteMem[100];
  Read Mem[200];
  WriteMem[200];
  WriteMem[100].
- What are the number of hits and misses when using no-write allocate versus write allocate?

Answer

- *For no-write allocate, the address 100 is not in the cache, and there is no allocation* on write, so the first two writes will result in misses. Address 200 is also not in the cache, so the read is also a miss. The subsequent write to address 200 is a hit. The last write to 100 is still a miss. The result for no-write allocate is four misses and one hit.
- For write allocate, the first accesses to 100 and 200 are misses, and the rest are hits since 100 and 200 are both found in the cache. Thus, the result for write allocate is two misses and three hits.

Answer

No allocate: (1 hit , 4 misses)
Write Mem[100] miss
WriteMem[100] miss
Read Mem[200] miss
WriteMem[200] hit
WriteMem[100] miss

- Allocate: (3 hits , 2 misses)
  Write Mem[100] miss
  WriteMem[100] hit
  Read Mem[200] miss
  WriteMem[200] hit
  WriteMem[100] hit

Miss Categories

– Compulsory miss
  - The first access to a block is not in the cache.

– Capacity miss
  - Occur because of blocks being discarded and later retrieved if the cache cannot contain all the blocks needed during execution of a program.

– Conflict miss
  - Occur because a block can be discarded and later retrieved if too many blocks map to its set for direct mapped or set associative caches.

Improving Cache Performance

Six Basic Cache Optimizations

– **Average memory access time = Hit time + Miss rate x Miss penalty**

– Reduce the miss rate
  - larger block size
  - larger cache size
  - and higher associativity

– Reduce the miss penalty
  - multilevel caches
  - giving reads priority over writes

– Reduce the hit time
  - avoiding address translation when indexing the cache

First:**Larger block size to reduce miss rate:**

- To reduce miss rate through spatial locality.
- Increase block size.
- Larger block size reduce compulsory misses.
- But they increase the miss penalty.

Second :**Bigger caches to reduce miss rate:**

- capacity misses can be reduced by increasing the cache capacity.
- - Increases larger hit time for larger cache memory and higher cost and power

Second: Larger Caches to reduce miss rate

- Drawbacks
  - Longer hit time
  - Higher cost, power
  - This technique has been especially popular in Off chip caches.

Third: Higher Associativity

  - **Increase in associativity reduces conflict misses.**
- Greater associativity can come at the cost of increased hit time

Fourth: **Multilevel caches to reduce penalty:**

- - Introduces additional level cache
- - Between original cache and memory.
- - L1- original cache
- L2- added cache.
- L1 cache: - small enough
- - speed matches with clock cycle time.
- L2 cache: - large enough
- - capture many access that would go to main memory.
- Average access time can be redefined as
- Hit timeL1+ Miss rate L1 X (Hit time L2 + Miss rate L2 X Miss penalty L2)

Fifth: Give Priority to Read Miss over Writes to reduce miss penalty

- write buffer is a good place to implement this optimization.
- - write buffer creates hazards: read after write hazard.

Sixth. Avoiding address translation during indexing of the cache to reduce hit time:

- Caches must cope with the translation of a virtual address from the processor to a physical address to access memory.
- - common optimization is to use the page offset.
- - part that is identical in both virtual and physical addresses- to index the cache.

Six basic cache optimizations

1. **Larger block size to reduce miss rate:**
   - To reduce miss rate through spatial locality.
   - Increase block size.
   - Larger block size reduce compulsory misses.
   - But they increase the miss penalty.
2. **Bigger caches to reduce miss rate:**
   - capacity misses can be reduced by increasing the cache capacity.
   - Increases larger hit time for larger cache memory and higher cost and power.
3. **Higher associativity to reduce miss rate:**
   - Increase in associativity reduces conflict misses.
4. **Multilevel caches to reduce penalty:**
   - Introduces additional level cache
   - Between original cache and memory.
   - L1- original cache
   L2- added cache.
   L1 cache: - small enough
   - speed matches with clock cycle time.
   L2 cache: - large enough
   - capture many access that would go to main memory.
   Average access time can be redefined as
   Hit timeL1+ Miss rate L1 X (Hit time L2 + Miss rate L2 X Miss penalty L2)
5. **Giving priority to read misses over writes to reduce miss penalty:**
   - write buffer is a good place to implement this optimization.
   - write buffer creates hazards: read after write hazard.
6. **Avoiding address translation during indexing of the cache to reduce hit time:**
   - Caches must cope with the translation of a virtual address from the processor to a physical address to access memory.
   - common optimization is to use the page offset.
   - part that is identical in both virtual and physical addresses- to index the cache.

Eleven Advanced cache optimizations

- Categories :
  - Reducing the hit time: small and simple caches, way prediction, and trace caches
  - Increasing cache bandwidth: pipelined caches, multibanked caches, and nonblocking caches
  - Reducing the miss penalty: critical word first and merging write buffers
  - Reducing the miss rate: compiler optimizations
  - Reducing the miss penalty or miss rate via parallelism: hardware prefetching and compiler prefetching

1st: Hit Time Reduction Technique: Small and Simple Caches

- Smaller hardware is faster => small cache helps the hit time
- Keep the cache small enough to fit on the same chip as the processor (avoid the time penalty of going off-chip)
- Keep the cache simple
  - Use Direct Mapped cache: it overlaps the tag check with the transmission of data

Example : Compare AMAT of 2-way and 4-way

Assume that the hit time of a two-way set-associative first-level data cache is 1.1 times faster than a four-way set-associative cache of the same size. The miss rate falls from 0.049 to 0.044 for an 8 KB data cache, according to Figure C.8 in Appendix C. Assume a hit is 1 clock cycle and that the cache is the critical path for the clock. Assume the miss penalty is 10 clock cycles to the L2 cache for the two-way set-associative cache, and that the L2 cache does not miss. Which has the faster average memory access time?

For the two-way cache:

$$\text{Average memory access time}_{2\_way} = \text{Hit time} + \text{Miss rate} \times \text{Miss penalty}$$
$$= 1 + 0.049 \times 10 = 1.49$$

For the four-way cache, the clock time is 1.1 times longer. The elapsed time of the miss penalty should be the same since it's not affected by the processor clock rate, so assume it takes 9 of the longer clock cycles:

$$\text{Average memory access time}_4 = \text{Hit time} \times 1.1 + \text{Miss rate} \times \text{Miss penalty}$$
$$= 1.1 + 0.044 \times 9 = 1.50$$

If it really stretched the clock cycle time by a factor of 1.1, the performance impact would be even worse than indicated by the average memory access time, as the clock would be slower even when the processor is not accessing the cache.

2nd: Hit Time Reduction Technique: Way Prediction

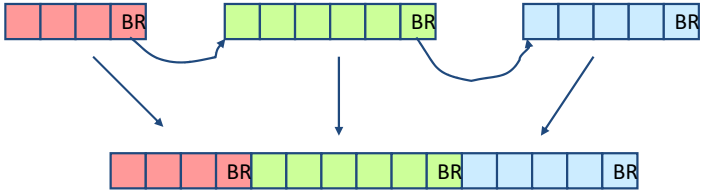
- How to combine fast hit time of Direct Mapped and have the lower conflict misses of 2-way SA cache?
- Way Prediction:
  - block predictor bits are added each block of the cache
  - These bits select which of the blocks to try on the *next cache access.*
  - Mux is set early to select the desired block
  - Only a single tag comparison is performed in parallel with reading the cache data
  - If tag match succeeds (hit) predicted block returned in one cycle
  - If tag match fails (miss) rest of the blocks checked in second cycle
    - Two hit times: fast hit (way predicted), slow hit (way mispredicted)
  - Prediction accuracy is 85%
  - Used in Pentium 4

3rd: Hit Time Reduction Technique: Trace Caches

- Higher ILP needs enough instructions every cycle.
- Solution – Trace caches:
  - Contains dynamic trace of executed intructions
  - Branch prediction is now implemented (built in) by the cache
    - Real target address must be validated with trace cache addresses
  - Better utilization of long blocks in instr. trace cache
    - Conventional caches may be entered in the middle and exited before the end by branches.
    - So, poor space utilization
  - drawback:
    - instructions may appear multiple times in multiple dynamic traces due to different branch outcomes
  - Pentium 4 uses decoded micro operations in trace cache (saves decode time)

Trace Cache

Key Idea: Pack multiple non-contiguous basic blocks into one contiguous trace cache line



- Single fetch brings in multiple basic blocks
- Trace cache indexed by start address *and* next *n* branch predictions

4th: Pipelined Cache Access to Increase Cache Bandwidth

- Pipeline cache access to maintain bandwidth (but higher latency)
- Instruction-cache pipeline stages:

  Pentium: 1 stage

  Pentium Pro through Pentium III: 2 stages

  Pentium 4: 4 stages

⇒Greater penalty on mispredicted branches

⇒More clock cycles between issue of load and use of data

5th : Nonblocking Caches to Increase Cache Bandwidth

- Pipelined computers that allow out-of-order completion the processor need not stall on a data cache miss.
- Processor could continue fetching instructions from the instruction cache while waiting for the data cache to return the missing data
- Thus nonblocking cache allows data cache to continue to supply cache hits during a miss
  - Known as “hit under miss” optimization
  - Lowers miss penalty
- cache may further lower the effective miss penalty if it can overlap multiple misses
  - Known as "hit under multiple miss"
- Requires suitable multi-bank, pipelined memory system
- Hard to measure miss penalty due to overlap of hits and misses

6th: Multibanked Caches to Increase Cache Bandwidth

- divide cache into independent banks that can support simultaneous accesses
- banking works best when the accesses naturally spread themselves across the banks
- mapping of addresses to banks:
  - spread the addresses of the block sequentially across the banks
  - called as *sequential interleaving*

Example : Four-way interleaved cache banks using block addressing

- Assuming 64 bytes per blocks.
- assume cache has four banks,
  - bank 0 has all blocks whose address modulo 4 is 0;
  - bank 1 has all blocks whose address modulo 4 is 1; and so on.
- each of these addresses would be multiplied by 64 to get byte addressing

Block address	Bank 0
0	0-63
4	256-319
8	
12	

Block address	Bank 1
1	64-127
5	
9	
13	

Block address	Bank 2
2	128-191
6	
10	
14	

Block address	Bank 3
3	192-255
7	
11	
15	

7th : Critical Word First and Early Restart to Reduce Miss Penalty

- Cache block size tends to increase to exploit spatial locality
- Any given reference needs only one word from a multi-word Block
- CWF fetches requested word first and sends it to processor
- Processor continues execution while rest of the block is fetched

7th : Critical Word First and Early Restart to Reduce Miss Penalty

- **Early restart**
  - Fetches words in the order stored in the block
  - As soon as critical word arrives, sends to processor and processor restarts

8th : Merging Write Buffer to Reduce Miss Penalty

- Processor blocks on write if write buffer is full
- Processor checks write address with address in write buffer
- Processor merges writes to same address if address is present in write buffer
- Assume write buffer with 4 entries, with 4 words of 64-bit each

Example for write merging

4 Writes to same cache block in different cycles, without write merging

Write address	V		V		V		V	
100	1	Mem[100]	0	Addr 108	0	116	0	124
108	1	Mem[108]	0		0		0	
116	1	Mem[116]	0		0		0	
124	1	Mem[124]	0		0		0	

4 Writes to same cache block in different cycles, with write merging

Write address	V		V		V		V	
100	1	Mem[100]	1	Mem[108]	1	Mem[116]	1	Mem[124]
	0		0		0		0	
	0		0		0		0	
	0		0		0		0	

Example…

- Without write merging, all 4 buffer entries are used
- If writes are merged only one buffer entry is used
- Multiword writes are faster than writes one word at a time
- Write misses can be served faster
- Reduces stalls due to write buffer being full

Victim cache

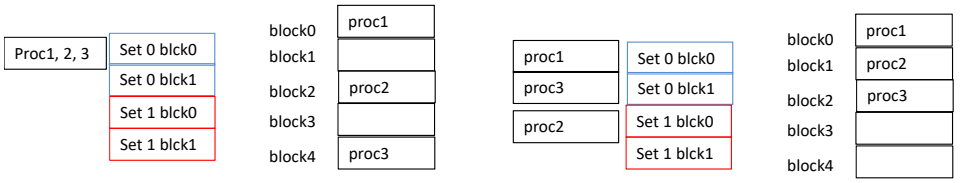
- Tiny cache holds evicted cache blocks
- On a subsequent cache miss, check the victim cache for the desired data before going to lower level memory
- Reduces the impact of conflict miss
- Used in AMD Opteron

9th : Compiler Optimizations to Reduce Miss Rate

- Code transformations to improve:
  - Spatial locality, through higher utilization of fetched cache blocks
  - Temporal locality, through reduction of the reuse distance of cache blocks
  - Examples: loop interchange, loop fusion, blocking …
- Data layout and data structure transformations to improve:
  - Spatial locality, through higher utilization of fetched cache blocks
  - Examples: array merging, structure/object class member reordering in memory, block array layouts

Reordering procedures to reduce instruction miss rate

- Assume that cache has 2 sets of 2way set associativity
- Proc1 through proc3 map to set 0 – hence conflicts
- Reorder procedures in MM such that they occupy contiguous blocks



Before reordering

After reordering

Array merging

- Data structure reorganization for spatial locality

```
/* Before */
int val[SIZE];
int key[SIZE];
```

- Assume code accessing val[i], key[i], for every i
- Accesses to *val* and *key* may conflict in direct-mapped caches
- Solution, merge arrays, accesses to val[i], key[i] do not conflict in the cache, spatial locality exploited

```
/* After */
struct merge {
int val;
int key;
}
struct merge merged_array[SIZE];
```


Loop interchange

- Code transformation for spatial locality

```
/* Before */
for (j = 0; j < 500; j++)
for (i = 0; i < 100; i++)
   x[i][j] = 2 * x[i][j];
```

- Arrays in C stored in row-major order
- Innermost loop over column of x
- Long strides lead to poor spatial locality
  - assume that size of cache block = size of 1 row (ie 100 words)
  - At each iteration of inner loop accesses go to different blocks
  - Hence more cache misses

```
/* After */
for (i = 0; i < 100; i = i+1)
for (j = 0; j < 500; j = j+1)
x[i][j] = 2 * x[i][j];
```

  - Here, once a block is brought in there wont be any cache miss for all iteration of the inner loop

Data blocking

A Code transformation for temporal locality

- Reduce reuse distance for same data
- Organize code so that data is accessed in blocks
- Best performance if block accessed many times and few accesses to data outside block
- **Example: Matrix multiplication without blocking**

```
/* Before */
for (i = 0; i < N; i = i+1)
for (j = 0; j < N; j = j+1)
{r = 0;
for (k = 0; k < N; k = k + 1)
r = r + y[i][k]*z[k][j];
x[i][j] = r;
};
```


Prefetching of instructions and data to reduce miss penalty or miss rate

- Processor requests data in advance
  - Processor speculates that requested data will be accessed in the future
- Need a guess for what data will be needed in the future
- Guess is easy in linear memory access pattern
  - Fixed stride between data accesses
  - Strided prefetching
- Need prefetch triggers, e.g. two consecutive misses

Prefetching -Performance implications

- Prefetched data may displace other useful data from cache
- Prefetched data may come too early and be evicted before used
- Prefetched data may come too late and not be there when needed
- Prefetched data may be useless, i.e. not accessed at all
- Prefetched data wastes memory bandwidth, if useless

10^{th} : Hardware Prefetching of Instructions and Data

- A processor fetches two (consecutive) blocks on a miss.
  - The requested block is placed in the instruction (data) cache when it returns.
  - The prefetched block is placed into instruction (data) stream buffer.
  - When the requested block can be found and read from the stream buffer, the next prefetch request is issued.
- With four instruction (data) stream buffers, the hit rate improves to 50% (43%).

11th : Software Prefetching Data

- Compiler inserts prefetching instructions to request data before the processor needs it
- 2 types
  - Register prefetch : Load data into register (HP PA-RISC loads)
  - Cache Prefetch: load into cache (MIPS IV, PowerPC, SPARC v. 9)
- Issuing Prefetch Instructions takes time
  - cost of prefetch issues should be < savings in reduced misses

Summary of Cache Optimizations

- Fig. 5.26

Technique	Miss penalty	Miss rate	Hit time	Hardware complexity	Comment
Multilevel caches	+			2	Costly hardware; harder if block size L1 ≠ L2; widely used
Critical word first and early restart	+			2	Widely used
Giving priority to read misses over writes	+			1	Trivial for uniprocessor, and widely used
Merging write buffer	+			1	Used with write through; in 21164, UltraSPARC III; widely used
Victim caches	+	+		2	AMD Athlon has eight entries
Larger block size	–	+		0	Trivial; Pentium 4 L2 uses 128 bytes
Larger cache size		+	–	1	Widely used, especially for L2 caches
Higher associativity		+	–	1	Widely used
Way-predicting caches		+		2	Used in I-cache of UltraSPARC III; D-cache of MIPS R4300 series
Pseudoassociative		+		2	Used in L2 of MIPS R10000
Compiler techniques to reduce cache misses		+		0	Software is a challenge; some computers have compiler option
Nonblocking caches	+			3	Used with all out-of-order CPUs
Hardware prefetching of instructions and data	+	+		2 instr., 3 data	Many prefetch instructions; UltraSPARC III prefetches data
Compiler-controlled prefetching	+	+		3	Needs nonblocking cache too; several processors support it
Small and simple caches		–	+	0	Trivial; widely used
Avoiding address translation during indexing of the cache			+	2	Trivial if small cache; used in Alpha 21164, UltraSPARC III
Pipelined cache access			+	1	Widely used
Trace cache			+	3	Used in Pentium 4

Figure 5.26 Summary of cache optimizations showing impact on cache performance and complexity for the techniques in Sections 5.4–5.7. 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.

Memory Technology and Optimizations

Main Memory

- SRAM for cache
- DRAM for main memory

SRAM

- Static Random Access Memory
- Six transistors per bit to prevent the information from being disturbed when read
- Don't need to refresh, so access time is very close to cycle time

DRAM

- Dynamic Random Access Memory
- Single transistor per bit
- Reading destroys the information
- Refresh periodically
- cycle time > access time

DRAM

- Dynamic Random Access Memory
- Single transistor per bit
- Reading destroys the information
- Refresh periodically
- cycle time > access time

- *DRAMs are commonly sold on small boards called **DIMM** (dual inline memory modules), typically containing 4 ~ 16 DRAMs*

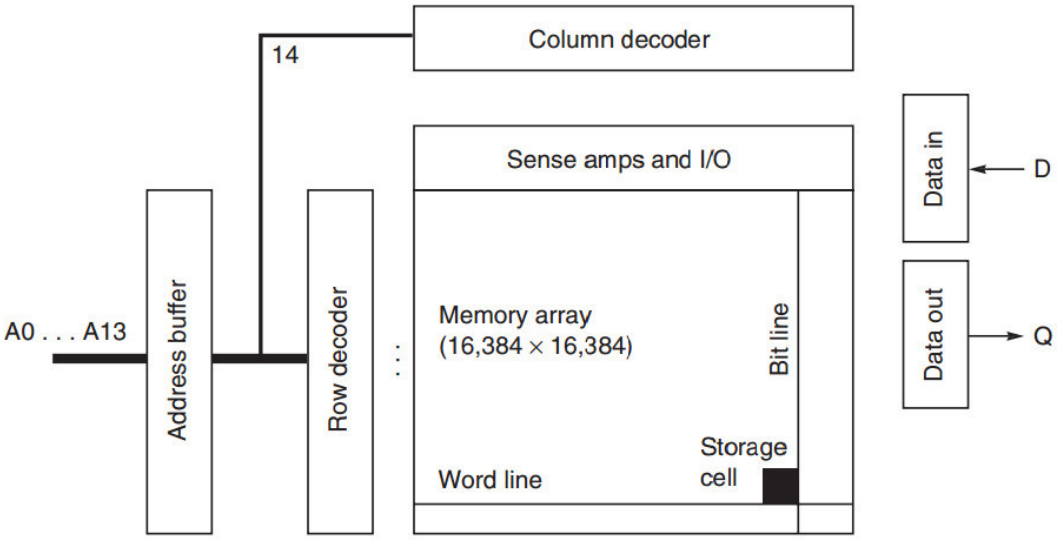
As DRAMs grew in capacity, the cost of a package with all the necessary address lines was an issue. 2. The solution was to multiplex the address lines, thereby cutting the number of address pins in half.

- One-half of the address is sent first, called the row access strobe (RAS).
- The other half of the address, sent during the column access strobe (CAS), follows it.

These names come from the internal chip organization, since the memory is organized as a rectangular matrix addressed by rows and columns.

- To pack more bits per chip, DRAMs use only a single transistor to store a bit.
- Reading that bit destroys the information, so it must be restored. This is one reason the DRAM cycle time is much longer than the access time.
- To prevent loss of information when a bit is not read or written, the bit must be “refreshed” periodically, all the bits in a row can be refreshed simultaneously just by reading that row.
- Hence, every DRAM in the memory system must access every row within a certain time window, such as 8 ms.
- Memory controllers include hardware to refresh the DRAMs periodically.
- This requirement means that the memory system is occasionally unavailable because it is sending a signal telling every chip to refresh.
- The time for a refresh is typically a full memory access (RAS and CAS) for each row of the DRAM. Since the memory matrix in a DRAM is conceptually square, the number of steps in a refresh is usually the square root of the DRAM capacity

DRAM Organization



DRAM Improvement

- **Timing signals**

  allow repeated accesses to the row buffer w/o another row access time;
- Leverage spatial locality

  each array will buffer 1024 to 4096 bits for each access;

DRAM Improvement

- **Clock signal**

  added to the DRAM interface,

  so that repeated transfers will not involve overhead to synchronize with memory controller;
- **SDRAM:** synchronous DRAM

DRAM Improvement

- **Wider DRAM**

  to overcome the problem of getting a wide stream of bits from memory without having to make the memory system too large as memory system density increased;

  widening the cache and memory widens memory bandwidth;

  e.g., *4-bit transfer mode up to 16-bit buses*

DRAM Improvement

- **DDR: double data rate**

  to increase bandwidth,

  transfer data on both the rising edge and falling edge of the DRAM clock signal,

  thereby doubling the peak data rate;

DRAM Improvement

- **Multiple Banks**

  break a single SDRAM into 2 to 8 blocks;

  they can operate independently;
- Provide some of the advantages of interleaving
- Help with power management

DRAM Improvement

- **Reducing power consumption in SDRAMs**

  dynamic power: used in a read or write

  static/standby power
- Depend on the operating voltage
- Power down mode: entered by telling the DRAM to ignore the clock

  disables the SDRAM except for internal automatic refresh;

DRAM Improvement

- 1. A DRAM access is divided into row access and column access. DRAMs must buffer a row of bits inside the DRAM for the column access,
- 2. Although presented logically as a single monolithic array of memory bits, the internal organization of DRAM actually consists of many memory modules. For a variety of manufacturing reasons, these modules are usually 1–4M bits.
- 3. To improve bandwidth, there has been a variety of evolutionary innovations over time.

4.The first was timing signals that allow repeated accesses to the row buffer without another row access time, typically called fast page mode. Conventional DRAMs had an asynchronous interface to the memory controller, and hence every transfer involved overhead to synchronize with the controller.

•5. The second major change was to add a clock signal to the DRAM interface, so that the repeated transfers would not bear that overhead. Synchronous DRAM (SDRAM) is the name of this optimization. SDRAMs typically also had a programmable register to hold the number of bytes requested, and hence can send many bytes over several cycles per request.

• 6.The third major DRAM innovation to increase bandwidth is to transfer data on both the rising edge and falling edge of the DRAM clock signal, thereby doubling the peak data rate. This optimization is called double data rate (DDR).

END
of
UNIT IV

Thank you