

CIS 371

Computer Organization and Design

Unit 12: Reliability

Reliability of Logic and Memory

- As transistors get smaller, they are less reliable
 - Wasn't a problem a few years ago, becoming a big problem
- **Transient faults**
 - A bit "flips" randomly, temporarily
 - Cosmic rays and such (more common at higher altitudes!)
 - Memory cells (especially DRAM) vulnerable today, logic soon
- **Permanent (hard) faults**
 - A gate or memory cell wears out
 - Breaks and stays broken
- Solution for both: use **redundancy** to detect and tolerate

DRAM Error Detection

- Idea: add extra state to DRAM to detect a bit flip
- **Parity**: simplest scheme
 - One extra bit, detects any single bit flip
 - Parity bit = $\text{XOR}(\text{data}_{N-1}, \dots, \text{data}_1, \text{data}_0)$
- Example:
 - $010101 \oplus 0^1 \oplus 0^1 \oplus 0^1 \oplus 1^1 = "1"$ so parity is "odd" (versus "even")
 - So, store "010101 **1**" in memory
 - When you read the data, and re-calculate the parity, say
 - 01**1**101 **1**, if the parity bit doesn't match, error detected
- Multiple bit errors? more redundancy can detect more

DRAM Error Detection

- What to do on a parity error?
- **Crash**
 - **Dead programs tell no lies**
 - Fail-stop is better than silent data corruption
 - Avoiding writing that "\$1m check"
- For user-level data, OS can kill just the program
 - Not the whole system, unless it was OS data
- Alternative: correct the error

SEC Error Correction Code (ECC)

- **SEC**: single-error correct (a hamming code)
- Example: Four data bits, three "code" bits
 - $d_1 d_2 d_3 d_4 c_1 c_2 c_3 \rightarrow c_1 c_2 d_1 c_3 d_2 d_3 d_4$
 - $c_1 = d_1 \wedge d_2 \wedge d_4$, $c_2 = d_1 \wedge d_3 \wedge d_4$, $c_3 = d_2 \wedge d_3 \wedge d_4$
 - Syndrome: $c_i \wedge c'_i = 0$? no error : points to flipped-bit
- Working example
 - Original data = 0110 $\rightarrow c_1 = 1, c_2 = 1, c_3 = 0$
 - Flip $d_2 = 0010 \rightarrow c'_1 = 0, c'_2 = 1, c'_3 = 1$
 - Syndrome = 101 (binary 5) \rightarrow 5th bit? D_2
 - Flip $c_2 \rightarrow c'_1 = 1, c'_2 = 0, c'_3 = 0$
 - Syndrome = 010 (binary 2) \rightarrow 2nd bit? c_2

SECDED Error Correction Code (ECC)

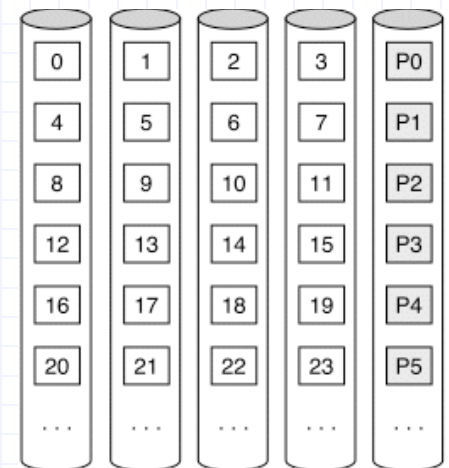
- **SECDED**: single error correct, double error detect
- Example: $D = 4 \rightarrow C = 4$
 - $d_1 d_2 d_3 d_4 c_1 c_2 c_3 \rightarrow c_1 c_2 d_1 c_3 d_2 d_3 d_4 c_4$
 - $c_4 = c_1 \wedge c_2 \wedge d_1 \wedge c_3 \wedge d_2 \wedge d_3 \wedge d_4$
 - Syndrome == 0 and $c'_4 == c_4 \rightarrow$ no error
 - Syndrome != 0 and $c'_4 != c_4 \rightarrow$ 1-bit error
 - Syndrome != 0 and $c'_4 == c_4 \rightarrow$ 2-bit error
 - Syndrome == 0 and $c'_4 != c_4 \rightarrow c_4$ error
 - **In general: $C = \log_2 D + 2$**
- Many machines today use 64-bit SECDED code
 - $C = 8$ (64bits + 8bits = 72bits, 12% overhead)
 - ChipKill - correct any aligned 4-bit error
 - If an entire DRAM chips dies, the system still works!

Disk Reliability: RAID

- **Error correction**: more important for disk than for memory
 - Error correction/detection per block (handled by disk hardware)
 - Mechanical disk failures (entire disk lost) most common failure mode
 - Many disks means high failure rates
 - Entire file system can be lost
- **RAID (redundant array of inexpensive disks)**
 - Add redundancy
 - Similar to DRAM error correction, but...
 - Major difference: which disk failed is known
 - Even parity can be used to recover from single failures
 - Parity disk can be used to reconstruct data faulty disk
 - RAID design balances bandwidth and fault-tolerance
 - Implemented in hardware (faster, more expensive) or software

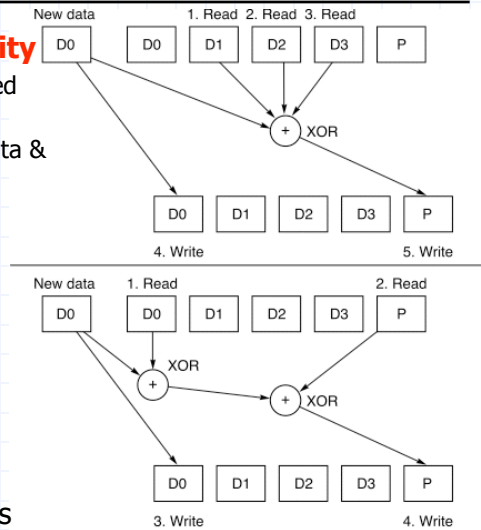
Simple Disk Array with Bit-level Parity

- **Bit-level parity**
 - dedicated parity disk
 - $N+1$ disks, calculate parity (write all, read all)
 - Good sequential read/write bandwidth, poor random accesses
 - If $N=8$, only 13% overhead



RAID with Block-level Parity

- **RAID with block parity**
 - Reads only data you need
 - Writes require reads, calculate parity, write data & parity
- Naïve approach
 1. Read all disks
 2. Calculate parity
 3. Write data & parity
- Better approach
 - Read *old* data & parity
 - Calculate parity
 - Write data & parity
- Increases # of accesses

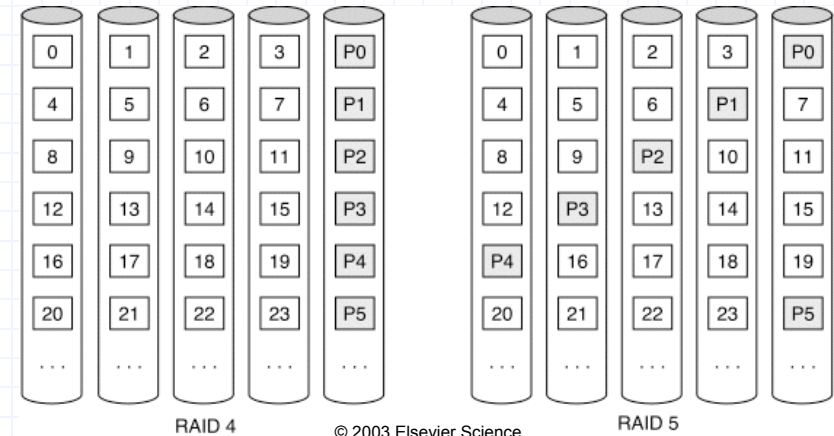


CIS 371 (Martin/Roth): Reliability

© 2003 Elsevier Science 9

RAID with Block-level Parity

- Rotates the parity disk, avoid single-disk bottleneck



CIS 371 (Martin/Roth): Reliability

© 2003 Elsevier Science

10

Aside: Storage Backup

- **Data is more valuable than hardware!**
 - Almost always true
- **Protecting data - three aspects**
 - **User error** - accidental deletion
 - Aside: ".snapshot" on enaic-l/halfdome filesystem
 - **Disk failure** - mechanical, wears out over time
 - Disk arrays (RAID) works well
 - **Disaster recovery** - An entire site is disabled
- **Two approaches:**
 - Frequent tape backups, taken off site (most common today)
 - Handle each problem distinctly
 - File system, redundant disks, network-based remote backup

CIS 371 (Martin/Roth): Reliability

11