3 Understanding RAID & Redundancy

3.1.1

Use the largest wordsize e.g. 64 bit words on our machines (or you can try 128-bit SSE2 extensions if available). Avoid computation of adresses using double index arrays – use incrementing pointers. Declare variables local to the context if possible.

3.1.2

Chunk 0 – Byte 0 + 1

3.1.4

Use checksums e.g. Adler,CRC32,MD5, SHA1 for each block! Combined with the XOR information each faulty bit can be identified if there is only one faulty block per row!
Do **getfattr -d /data/stripe-*** to read the extended attributes and calculate the sha1 checksums with **sha1sum /data/stripe-*** ... only stripe-0 is faulty!

3.1.5

You can easily compute the faulty chunk with 0 = 1^ 2 ^ 3 ^ 4

3.2.1

The larger the processor word length the less CPU cycles are needed. Bitwise computation is the slowest. 64-bit computation is the native CPU unit on the exercise machines.
In case of straigth forward loops new vectorizing compilers can optimize loops to use larger word widths, even if you wrote bytewise computations. Still the best is to understand how to write code close to the CPU design.

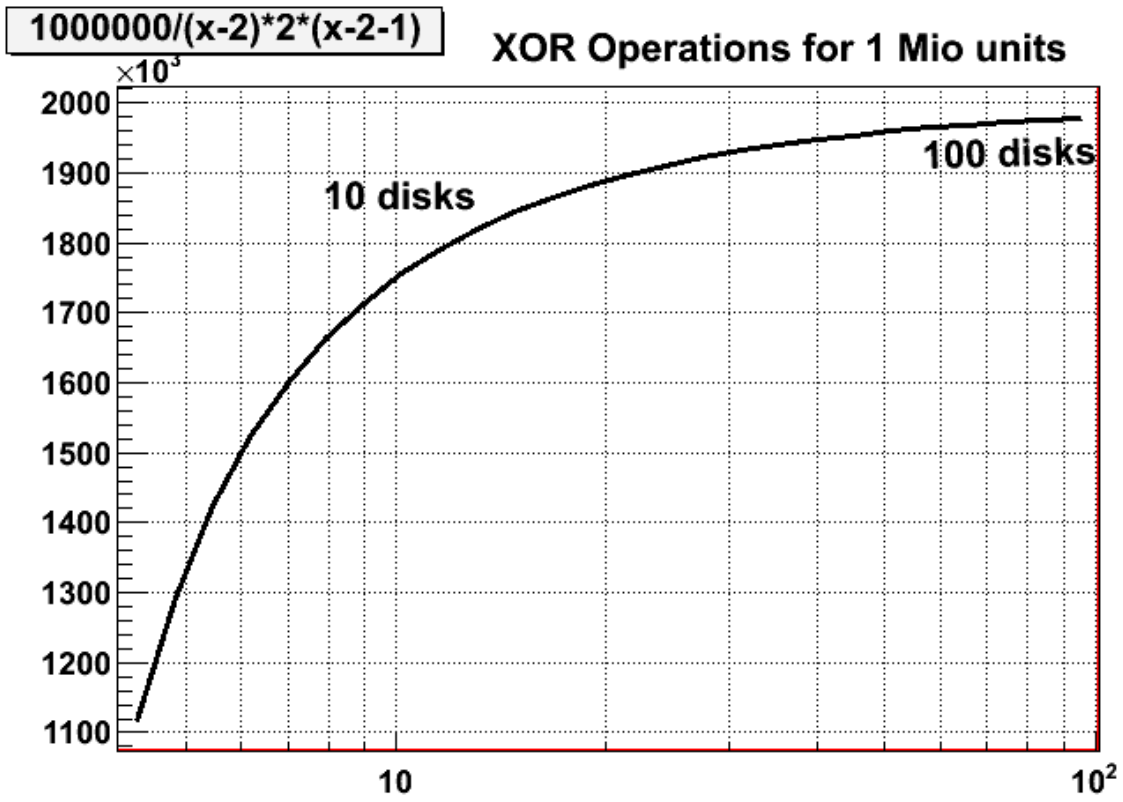
3.2.2

You need to reconstruct one block with diagonal parity, than with row parity, the next with diagonal parity aso ..... B1:A1:B2:B1:A4:B4:A3:B3 = 8 x 3 x XOR = 24 XOR
... but there are several possibilities!

3.2.3

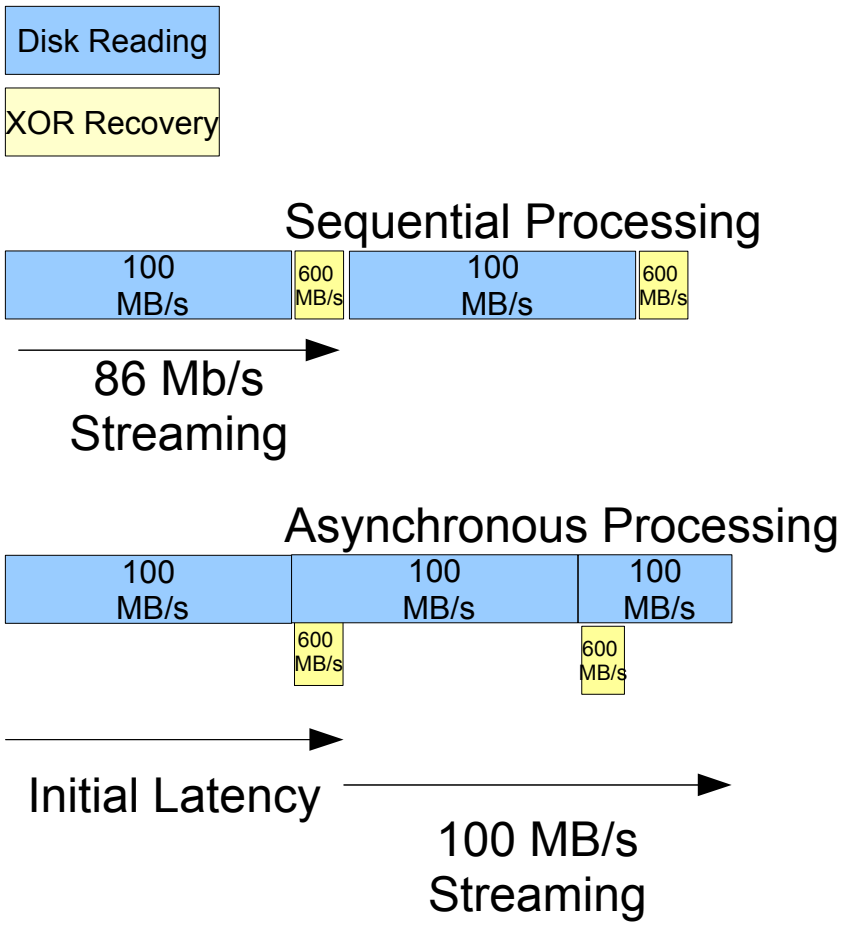
#XOR = X / (N-2) * 2 * (N-2-1)
with X = # of parity blocks (e.g. # of words or bytes)
with N = # of stripes (sum of D + P + DP stripes)

See the following function plot!



3.2.4

If two drives fail we read the same amount of information from 4 disks which are still operational. If we have to do the XOR computation sequentially after reading we will see a performance loss of the order of 16%. If we can prefetch chunks the degradation will not be visible in terms of throughput for streaming access – only in terms of CPU usage.



3.2.5

We loose 20% for the two stripe disk – plus 20 bytes per 4k Block:

100% * 4096/4116 * 8/10 ~ 79%