NAND Chip Driver Optimization and Tuning

Vitaly Wool
Embedded Alley Solutions Inc.

NAND chip driver

Background and structure

Basic MTD/NAND chip driver

- Provides
  - I/O base address
  - ALE/CLE/nCE control function
- Uses
  - Default I/O functions (PIO)
  - Default OOB layout
  - Default week software ECC (Hamming)
  - Poll-based wait for operation completion

Advanced NAND chip driver

- May provide
  - ready/busy indication function
  - Chip parameters (delay etc)
  - Timings (re)initialization
  - HW ECC functions
  - Non-standard I/O functions
    - DMA-based I/O
    - Interrupt-based wait

Modern NAND chips features

- Large page size
  - 2K/4K
- MLC everywhere
  - Cheaper
  - More compact
  - Faster
  - Less robust
    - Needs strong ECC algorithms deployment

Consequencies

- Capacity increase
  - Mostly due to MLC deployment
  - 8+ GB chips
- Speed increase
  - 100+ MB/s
- NAND controller hardware ECC support
  - Should be applicable for different chip page sizes

NAND driver requirements

- Functional
  - Strong error correction
  - No writes w/o ECC
  - Ability to handle more than 4GB
    - Actually not a chip driver level requirement
- Performance
  - Lose no more than 50% of chip I/O performance capabilities
    - What capabilities?
    - How to calculate “best achievable” rate

Consequences

- Hardware ECC necessary
  - Can't meet performance requirements otherwise
- DMA is desired
  - Hard to meet performance requirements otherwise
  - Lightens CPU load
- Spare OOB area should be either covered with ECC or kept unused
  - Can't use some flash filesystems if OOB is not covered
    - «stock» yaffs2 is out then

embedded alley

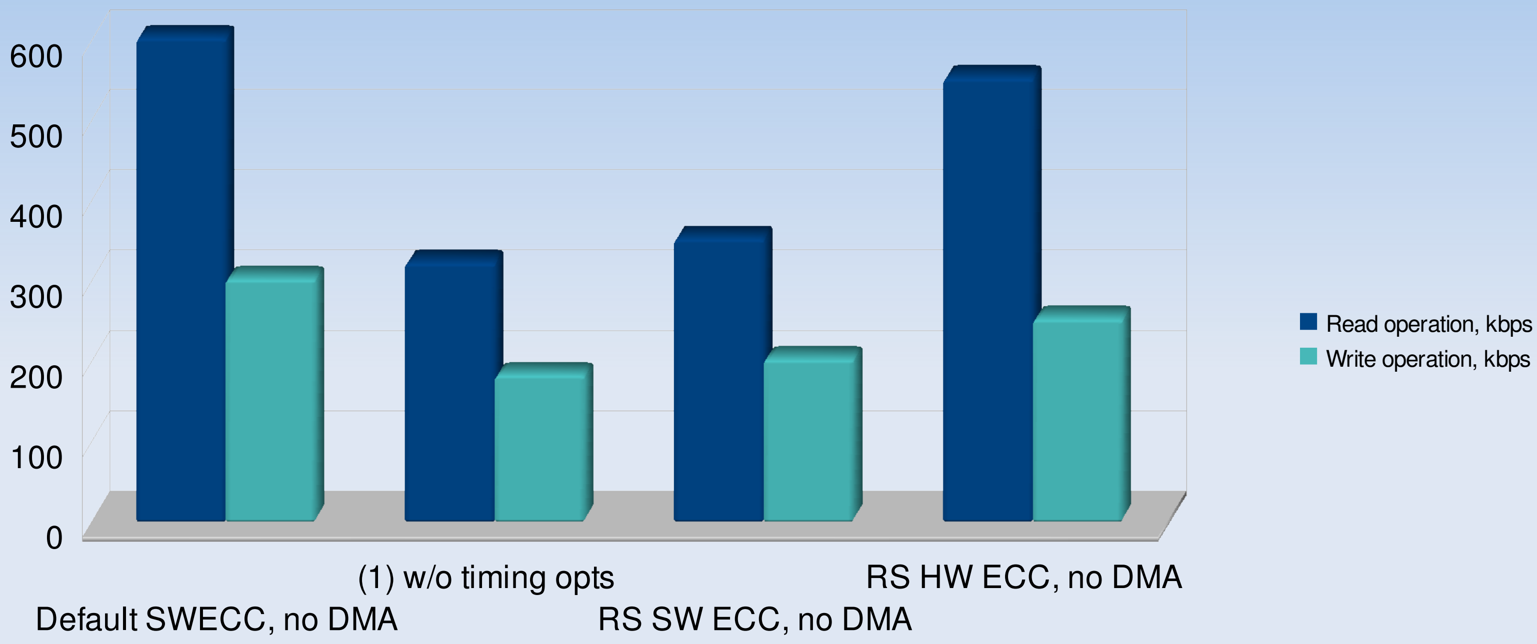
NAND chip driver

Optimization and profiling step-by-step

Adding HW ECC support

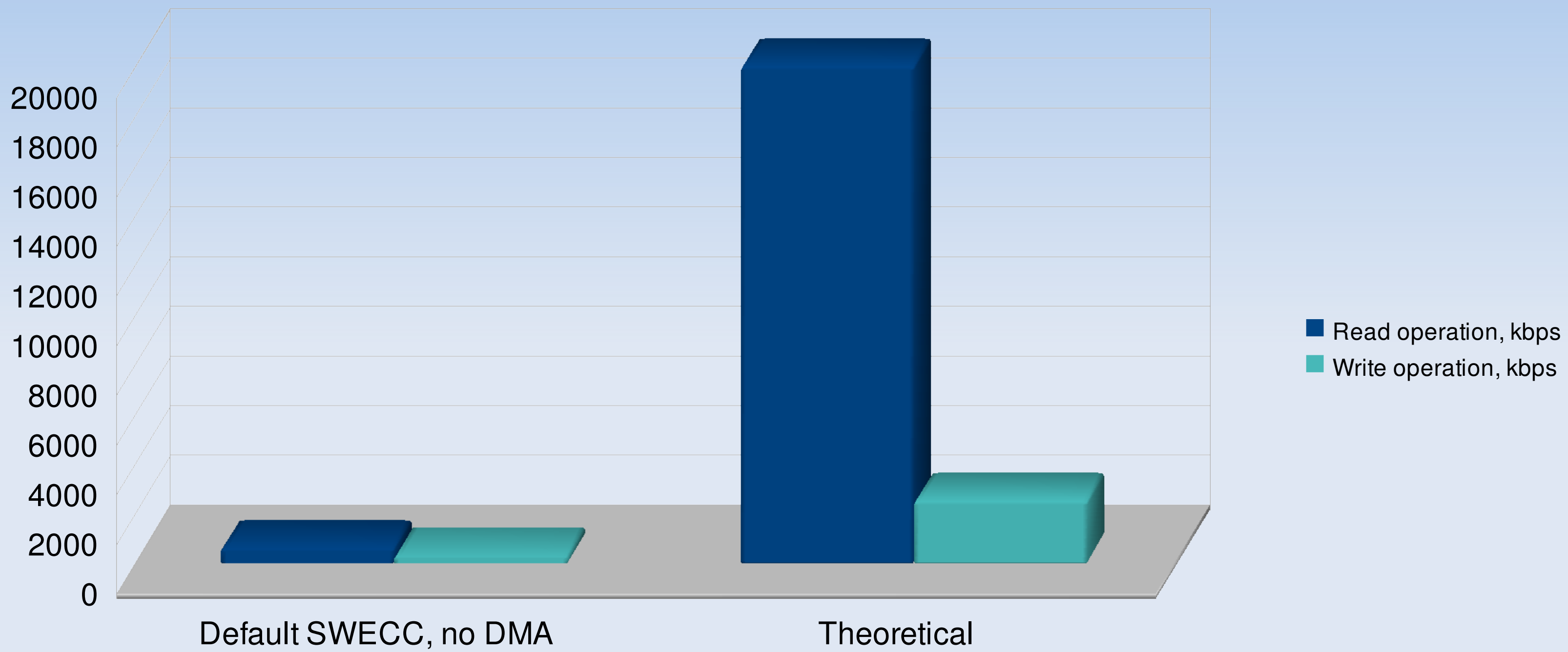
- HW ECC controller
  - May just be calculating syndromes over provided data
  - But may as well be doing NAND I/O itself
- HW ECC is not a performance issue cure
  - HW ECC might be calculated over 512-byte blocks

MLC I/O performance chart



- Comments
  - Timing optimization is important
  - RS HW ECC runs over 512b blocks
    - Slower than stock SW ECC

MLC I/O performance chart



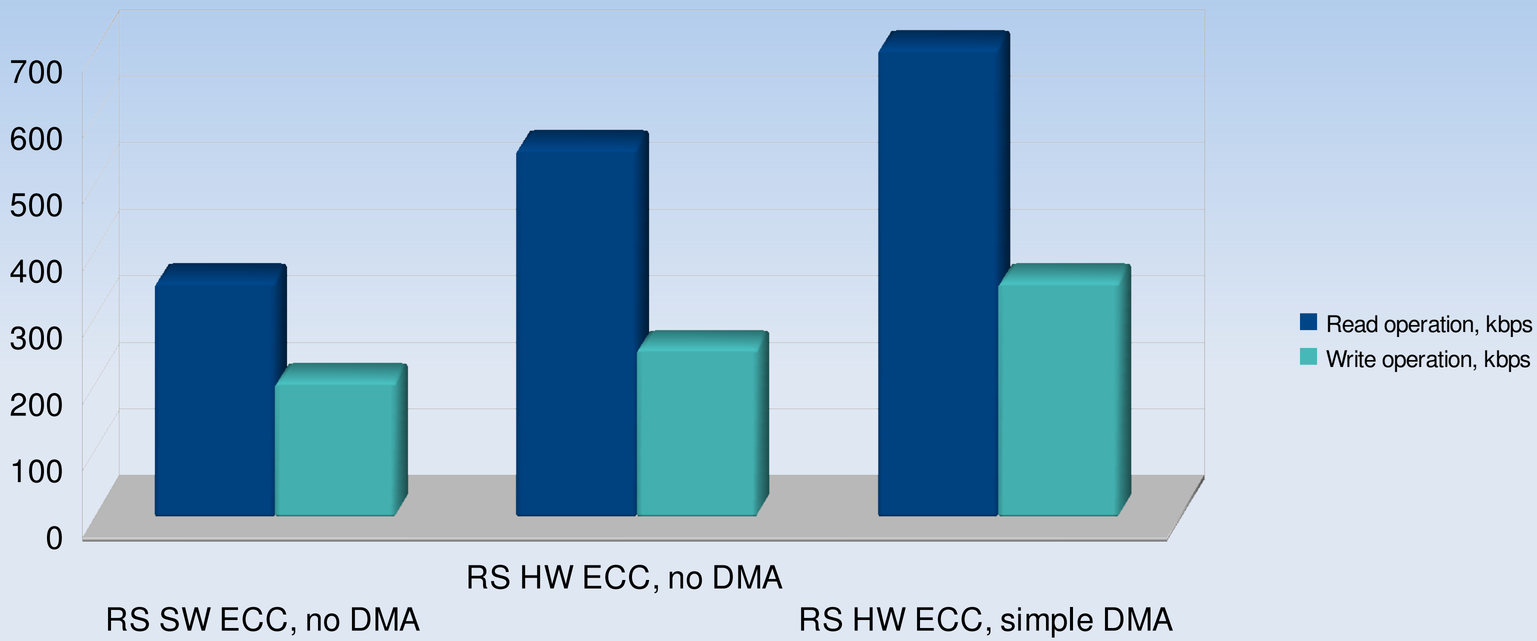
- Comments
  - Still a lot slower than the chip allows

Adding DMA support

- NAND I/O methods should use DMA
- Problem: making friends with HW ECC
  - HW ECC might be calculated over 512-byte blocks
  - ECC bytes might be spread across the page
  - HW ECC engine does ECC NAND I/O automatically
    - I/O is not quite consequent

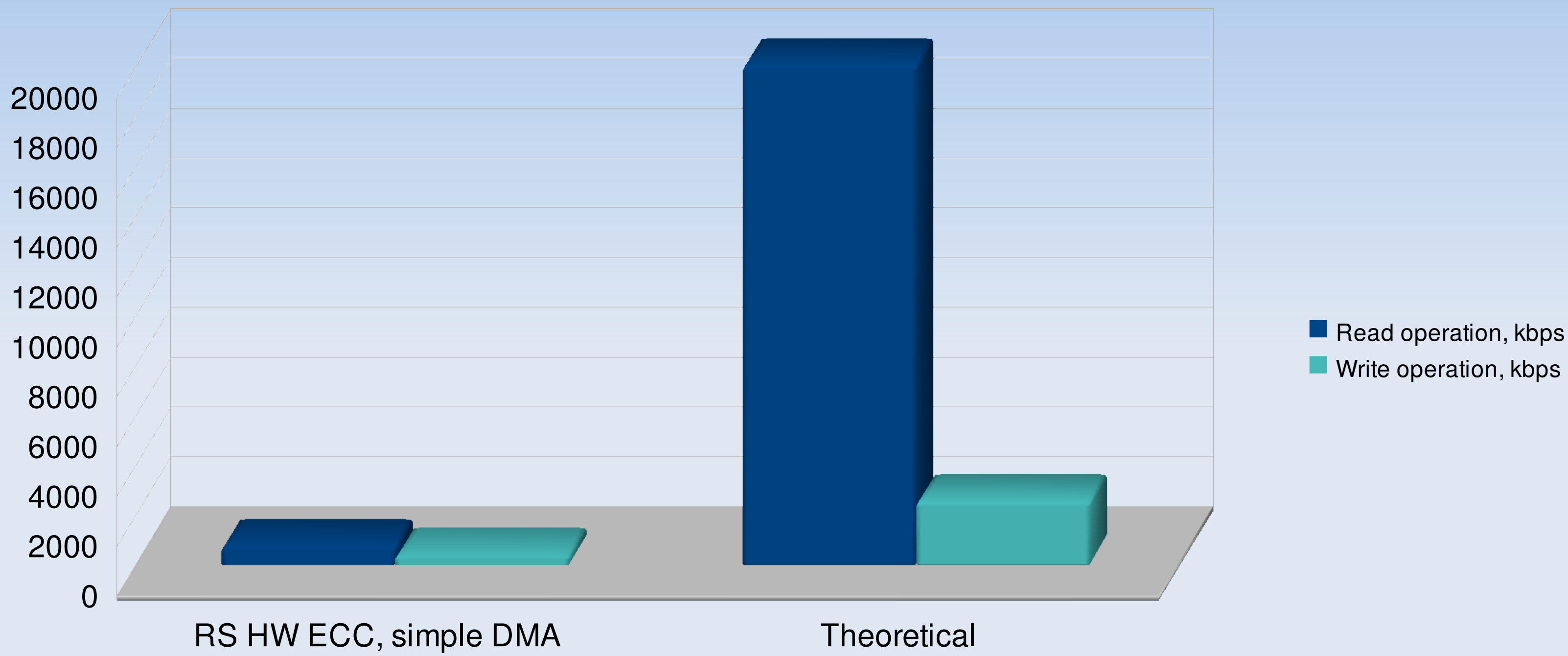
embedded alley

MLC I/O performance chart



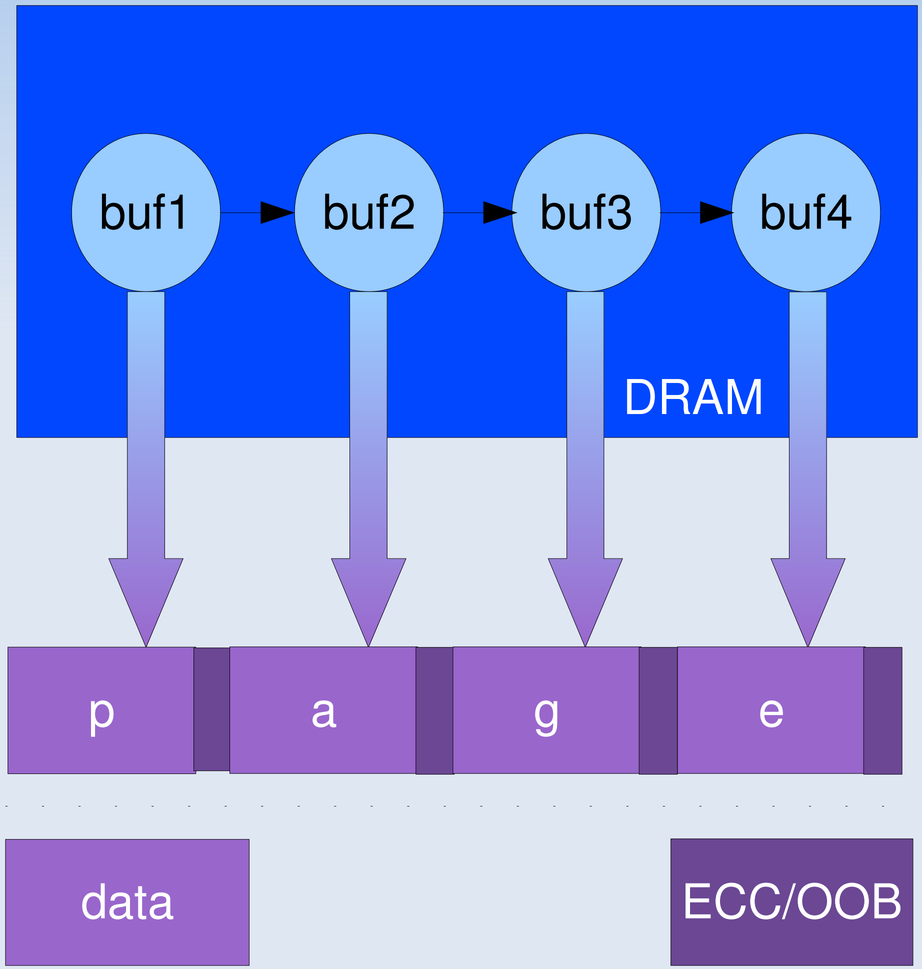
- Comments
  - Hamming SW ECC dropped from the chart
    - Not strong enough anyway
  - Straightfoward DMA didn't help much

MLC I/O performance chart



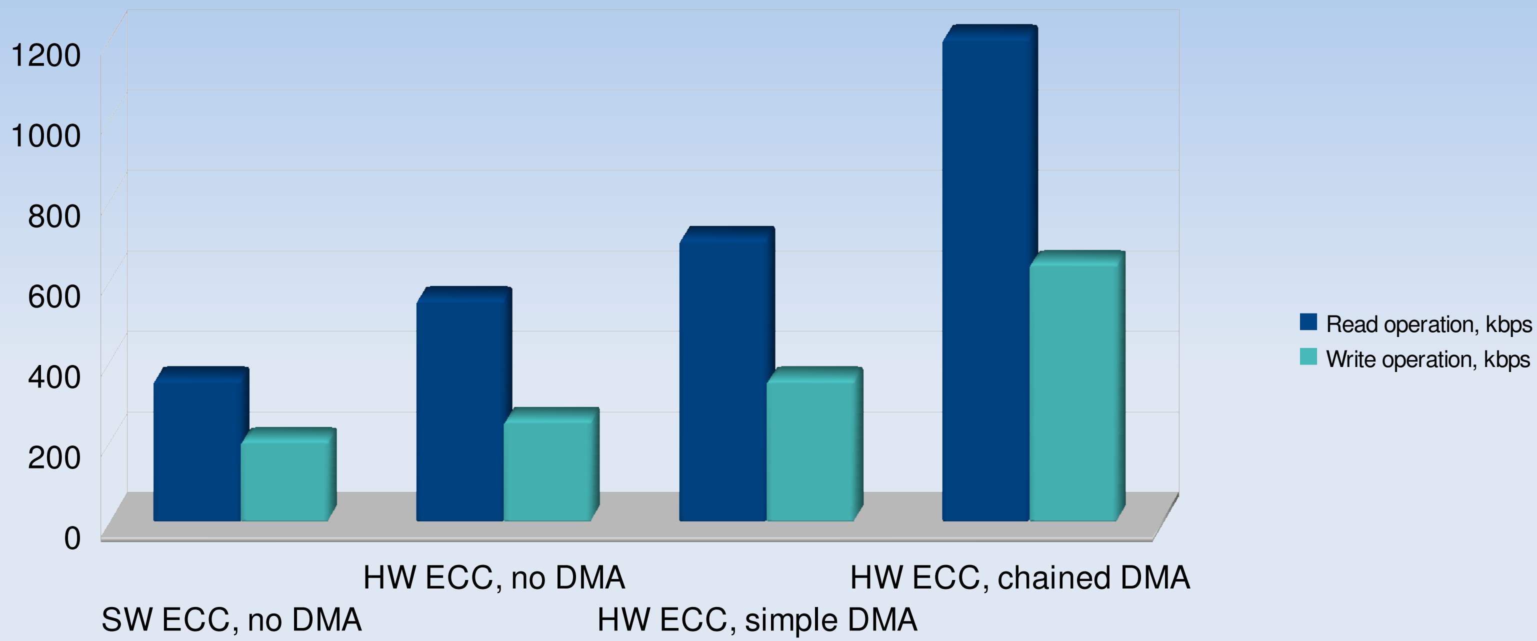
- Comments
  - Again a lot slower than the chip allows

DMA usage pattern



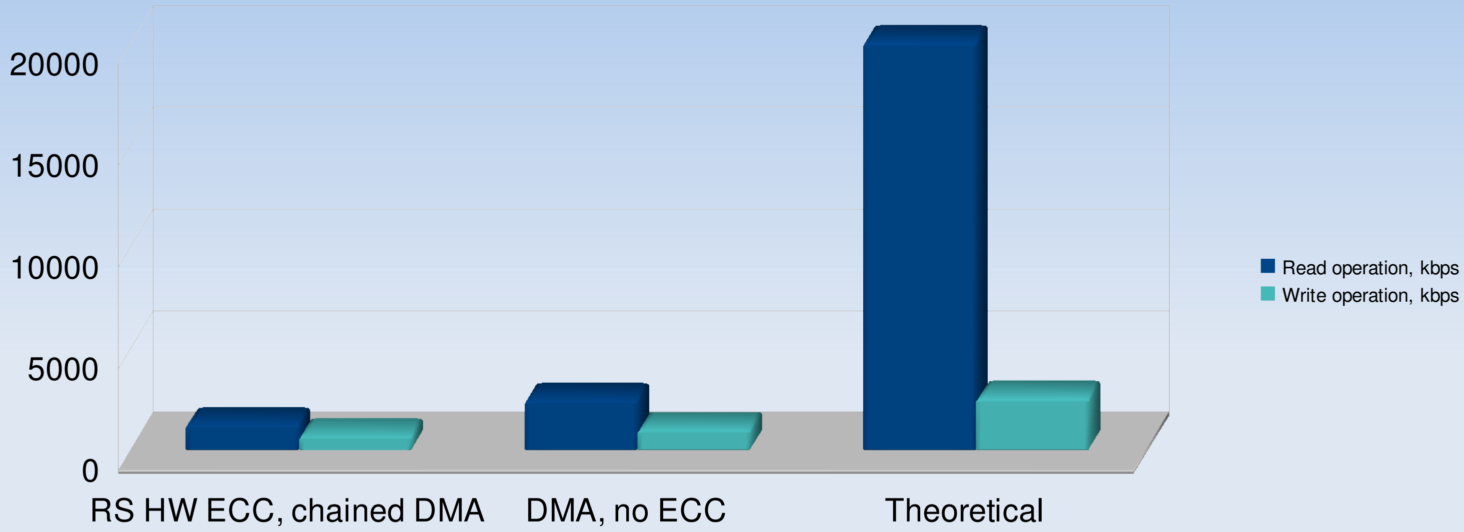
- NAND chips better do sequential I/O operations
  - Goes well with DMA w/ chaining
  - Can do HW ECC page read in a signe DMA chain

MLC I/O performance chart (RS ECC)



- Comments
  - Getting better... :-)

MLC I/O performance chart



- Comments
  - How to calculate the best achievable rate?
    - DMA with no ECC gives the idea
  - We're not that far from it (about 50%)

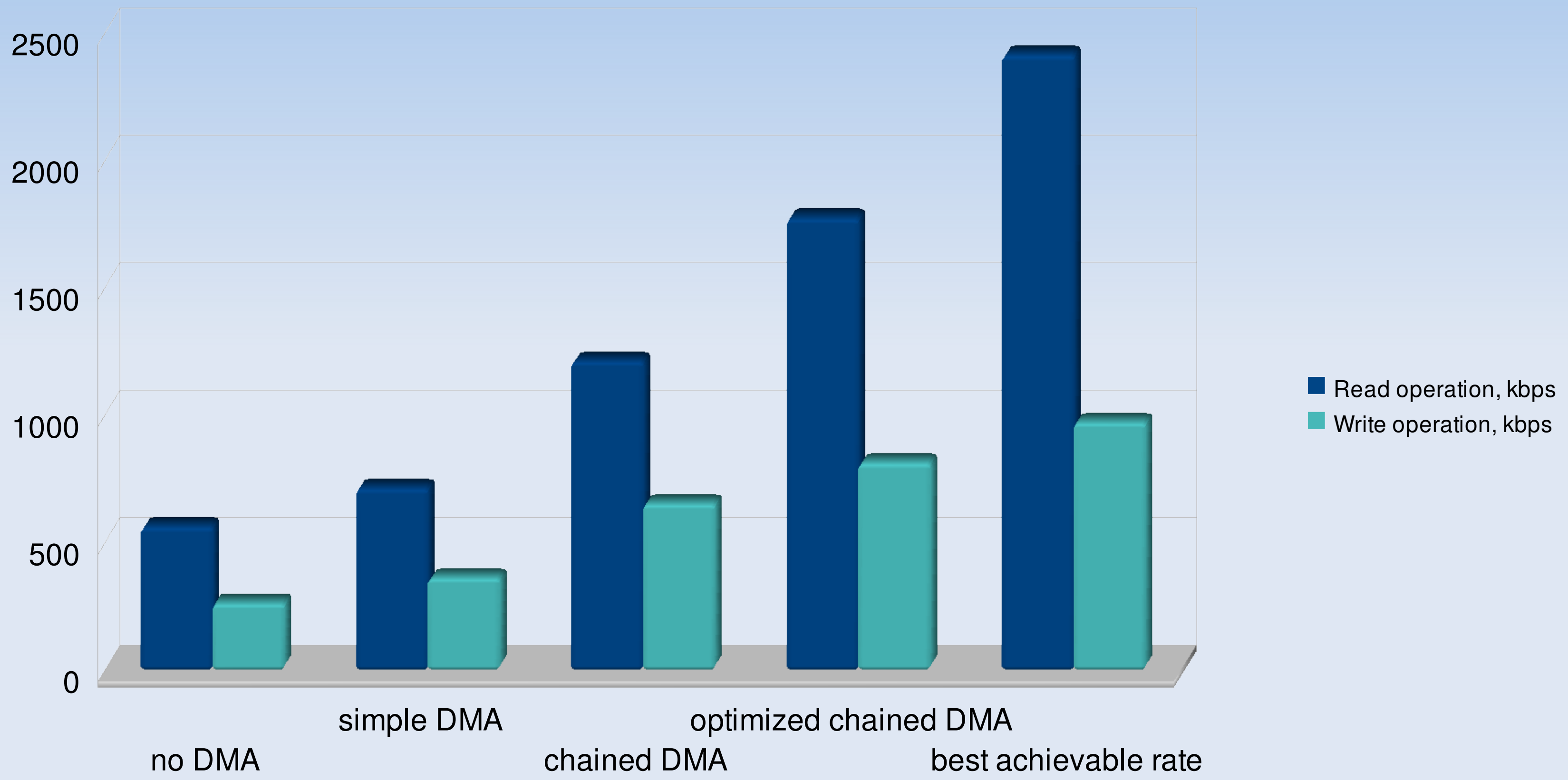
Further optimization

- No redundant data copys in driver
- Data from a buffer supplied is copied to the local buffer
  - Redundant: why not use the supplied buffer directly?
    - That's UNSAFE
    - e.g. vmalloc()'s not kmalloc()'s in jffs2 and ubi code

Further optimization

- The solution is to avoid redundant copys
- Also, preallocate DMAable buffers for the other case
  - kmalloc'ing won't stand in the critical path
- A simple own memory management thing
  - Very simple one — buffers are of the same size
  - Either linked list or stack of buffers
  - Use kmem_cache_XXX for that

MLC HW ECC performance chart



Comments/Artifacts

- The former results are all for filesystem-less data transfers
- The performance results for filesystems might deviate from the former quite a bit
  - YAFFS2 is faster on single big file I/O than JFFS2
    - As soon as we don't hack JFFS2 to not use vmalloc ;-)

embedded alley

Summary

- Modern NAND chips offer performance level that can't be easily achieved within an SoC
  - One has to consider the «best achievable» rate for a particular SoC/NAND chip combination
  - No exact techniques
- Optimized NAND driver may work some 5x faster than a non-optimized one
  - worth messing around!
  - Get closer to the best achievable rate
  - But... farther from community acceptance?

Questions?

mailto:vital@embeddedalley.com