The Memory *Controller* Wall: Benchmarking the Intel FPGA SDK for OpenCL Memory Interface

Hamid Reza Zohouri*

H2RC'19 @ Denver, CO, November 2019

†Edgecortix is a deep-tech startup based in Tokyo working on automated hardware & software co-design for AI accelerators

*This work was conducted as a Research Staff at Tokyo Institute of Technology

Introduction

Motivation

- FPGAs are becoming more popular in HPC/Cloud
  - Largely due to advancement of HLS/OpenCL and high power efficiency
- Large body of work has been dedicated to application optimizations
- Optimizations mostly focused on kernel-level optimizations
  - Minimizing loop initiation interval, reducing area usage, and maximizing parallelism
- External memory and controller largely overlooked
  - Very crucial on FPGAs due to low byte/FLOP ratio

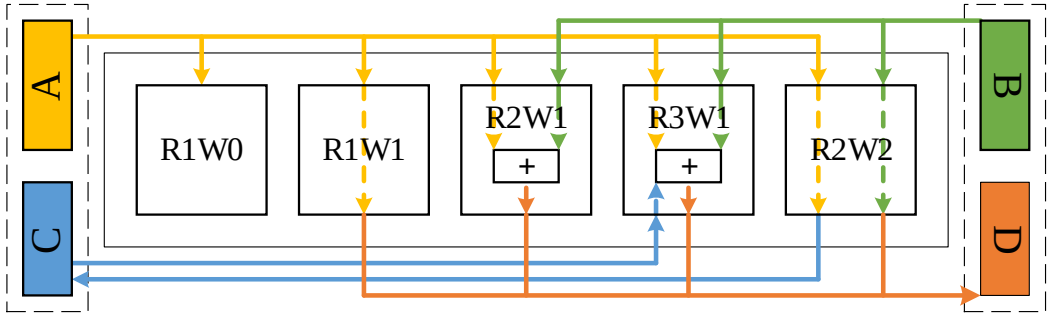
Contributions

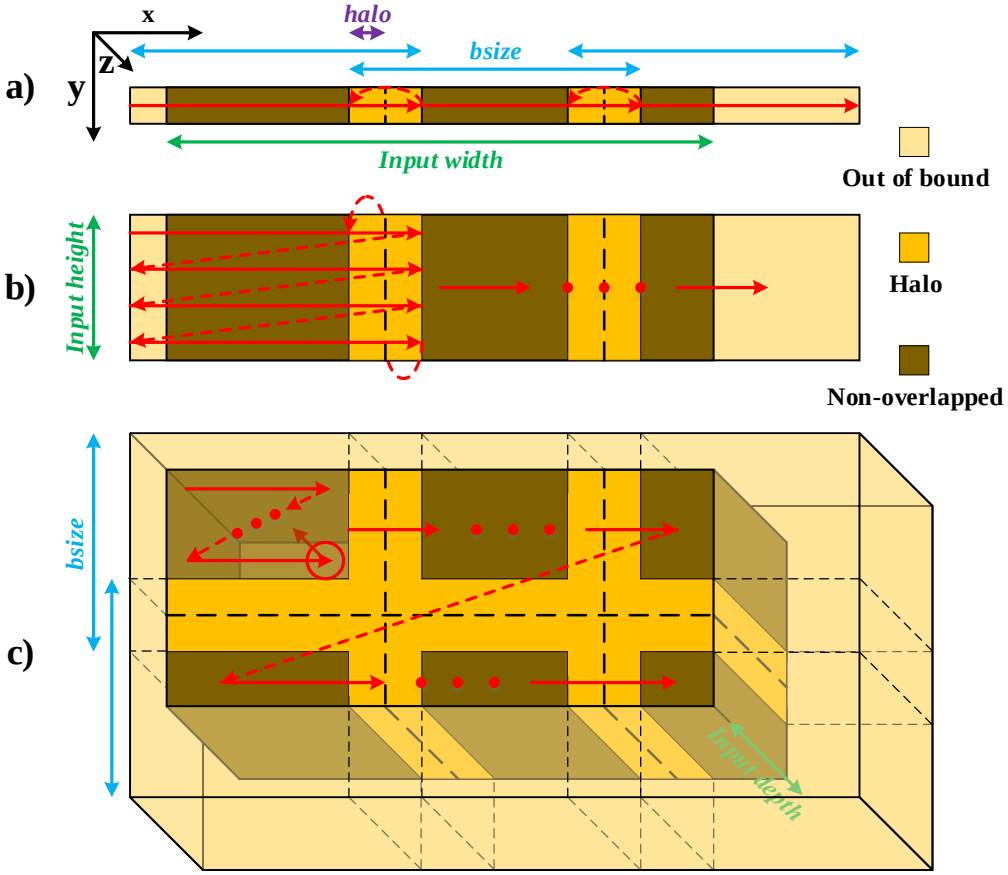
- Open-source highly-configurable memory bandwidth benchmark
  - Sequential access + 1D blocking + 1.5D/2.5D overlapped blocking
- Analysis of effect of multiple parameters on memory bandwidth
  - Blocking type, number of Input/Output arrays, vector size, interleaving, etc.
- Reference points to show what can be expected
- Outline deficiencies in compiler and memory controller/interface
  - Provide work-arounds in some cases

Methodology

Benchmark Suite

- OpenCL-based
- 3 blocking classes
- 5 array configurations
  - R1W0, R1W1, R2W1, R3W1, R2W2
- NDRange and Single Work-item
  - Also versions with channel between memory read and write





Hardware and Software

- Nallatech 385A
  - Arria 10 GX 1150
  - 2x DDR4 @2133 MHz => 34.128 GB/s
  - Memory controller @266 MHz
  - AOC 18.1.2 + Quartus 17.1.2 + BSP 17.1

Source: http://www.nallatech.com/wp-content/uploads/385A-FPGA_Side-Shot-DS2.png

- Tesla K20X and V100 SXM2
  - GDDR5 @249.6 GB/s and HBM2 @ 897.0 GB/s
  - CUDA 10.0

Source: https://images-na.ssl-images-amazon.com/images/I/71lqEOY2jjL._SL1500_.jpg

Benchmark Settings

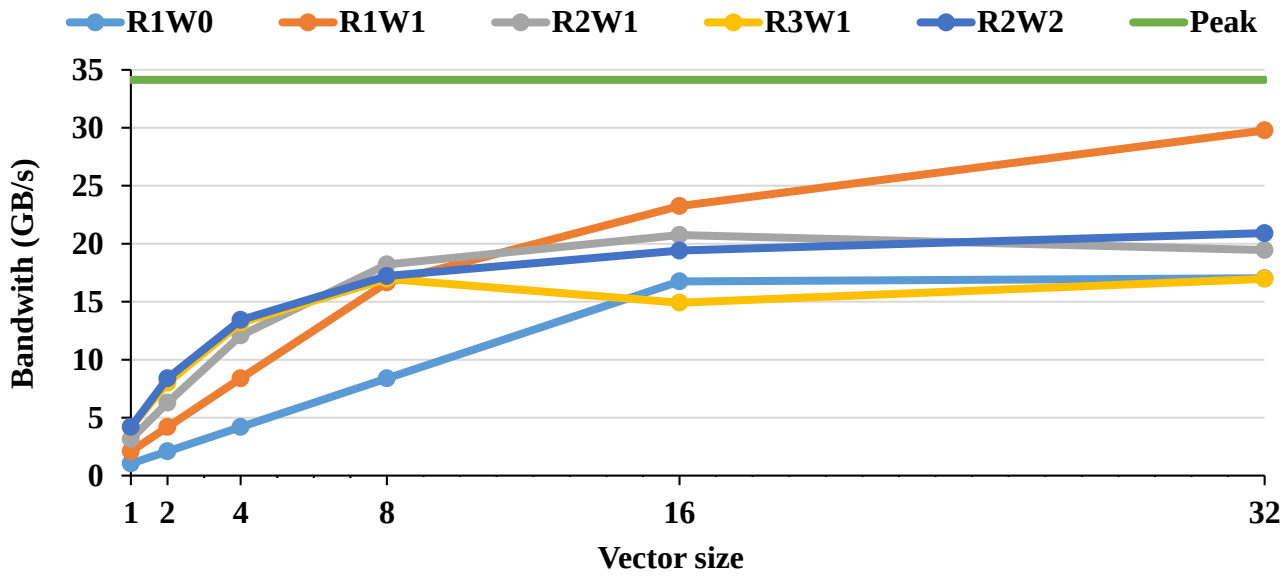
- Array sizes of 1 GiB
  - Total memory usage of 4 GiB
- Block size of 1024 for 1D/1.5D and 256×256 for 2.5D on FPGA
  - No direct effect on performance
- Block size is individually tuned for GPUs
- “float” datatype
- Quartus hack to allow control over kernel frequency
  - Use fixed kernel frequency (same value as memory controller)

Scope

- Not limited to the FPGA/compiler version used
  - Many experiments repeated on other versions of compiler
  - Many experiments repeated on Stratix V
- Performance trends were the same
  - Only some differences in measured performance
- Performance trends likely apply to all existing versions of the compiler
- Performance trends likely also apply to Stratix 10 with DDR memory

Results

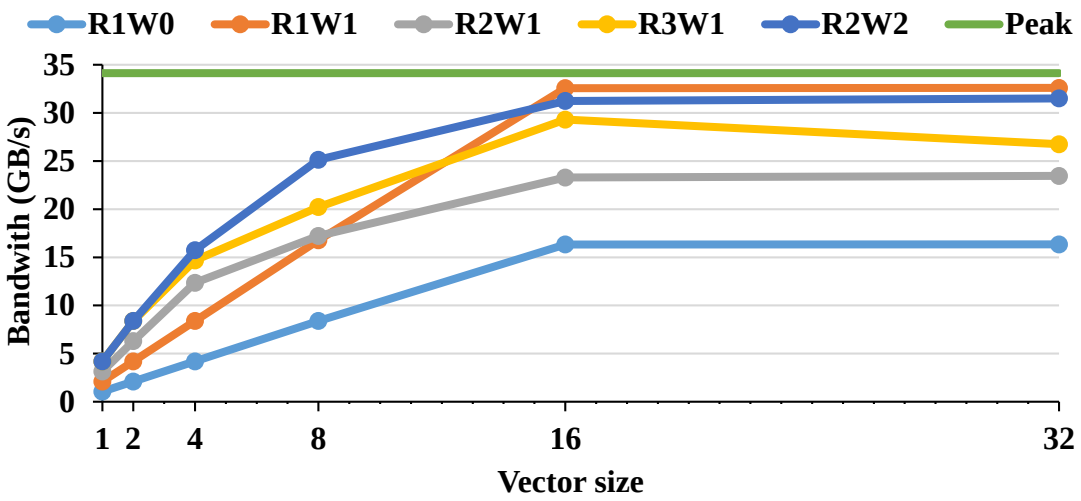
No Blocking – Vector Size



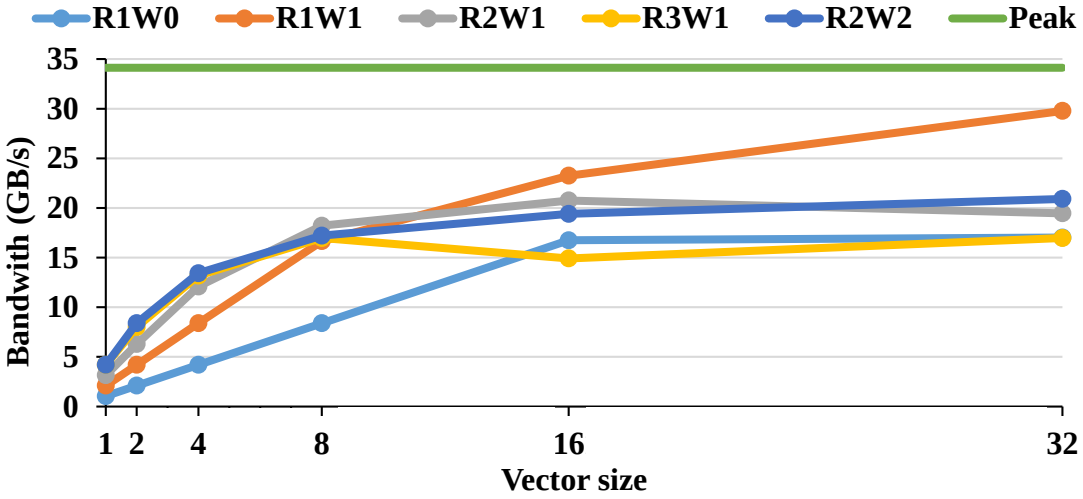
- Performance should saturate at $vector\_size = 32/num\_buff$
  - R1W0 stops at 16 => performance does not scale with interleaving
  - R1W1 does not saturate at either 16 or 32
  - Other cases do not scale beyond 16 without saturating memory bandwidth

No Blocking – Interleaving disabled

Interleaving Disabled



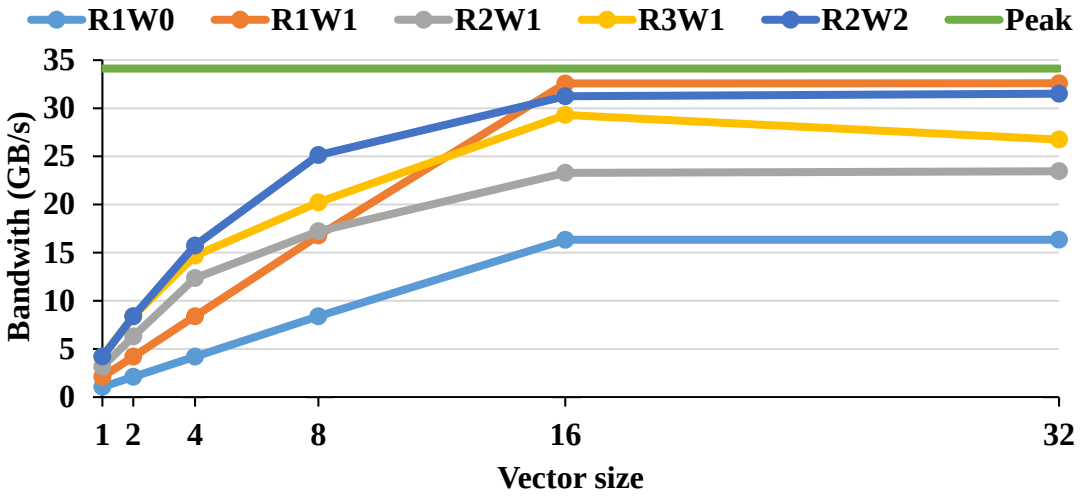
Interleaving Enabled



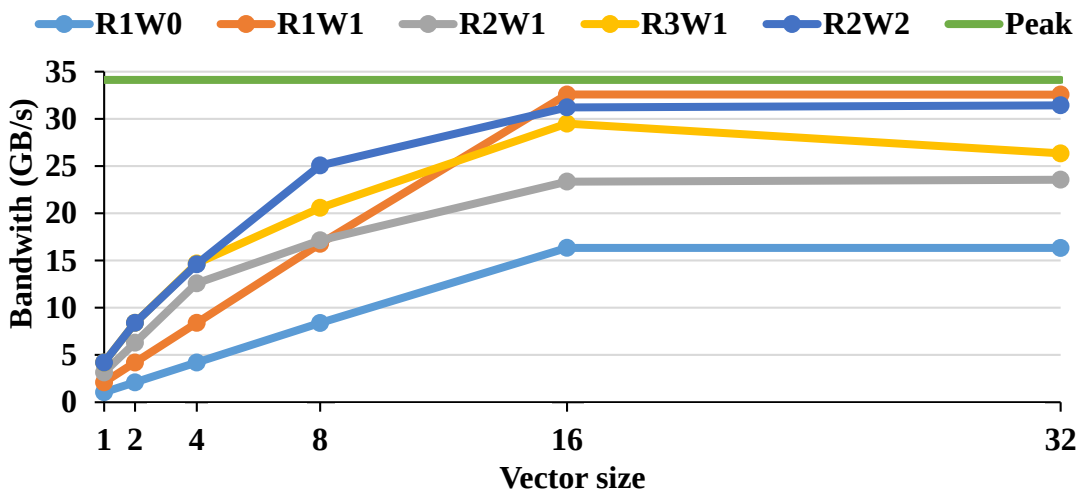
- Performance is always the same or higher with interleaving disabled
- R1W1 @16 nearly saturates the memory bandwidth (32.6 vs. 34.1 GB/s)
- R2W2 does not saturate @8 but nearly saturates @16
  - Double required vector size => higher area and lower power efficiency
- Other configs do not achieve over 85% of bandwidth
  - Work-around: Use array of structs to merge buffers with same memory access pattern

No Blocking – Single Work-item

NDRange



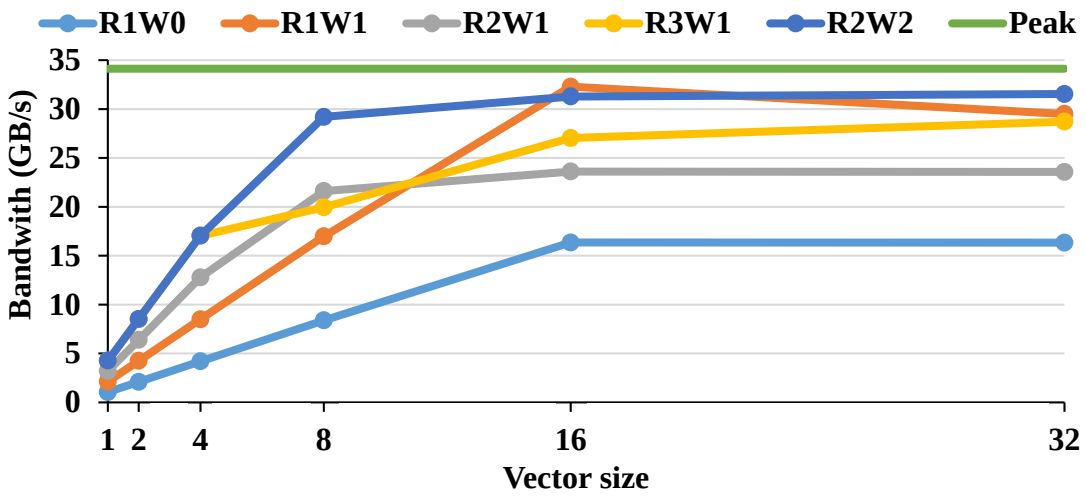
Single Work-item



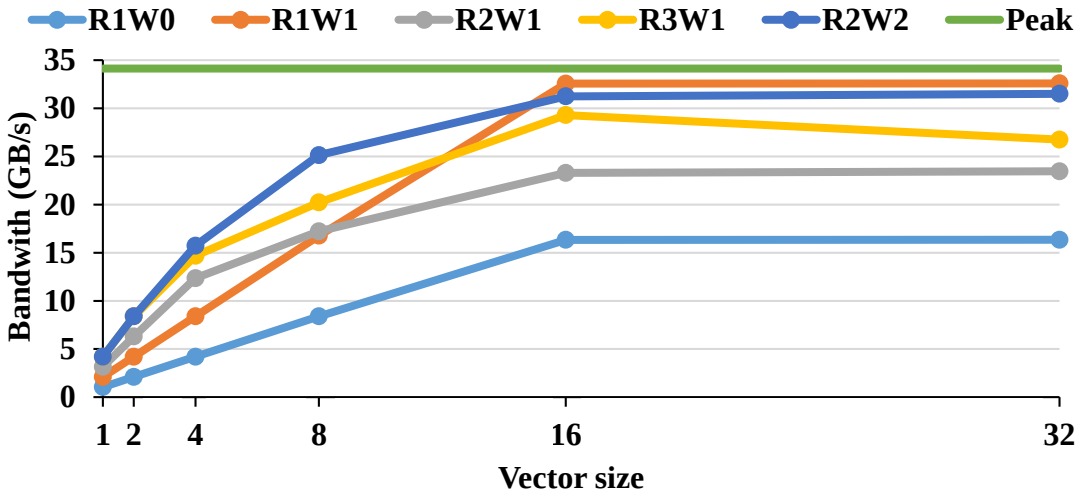
- No noticeable performance difference
  - Neither of programming models are preferred with respect to memory performance

No Blocking – Channelized NDRange

Channelized



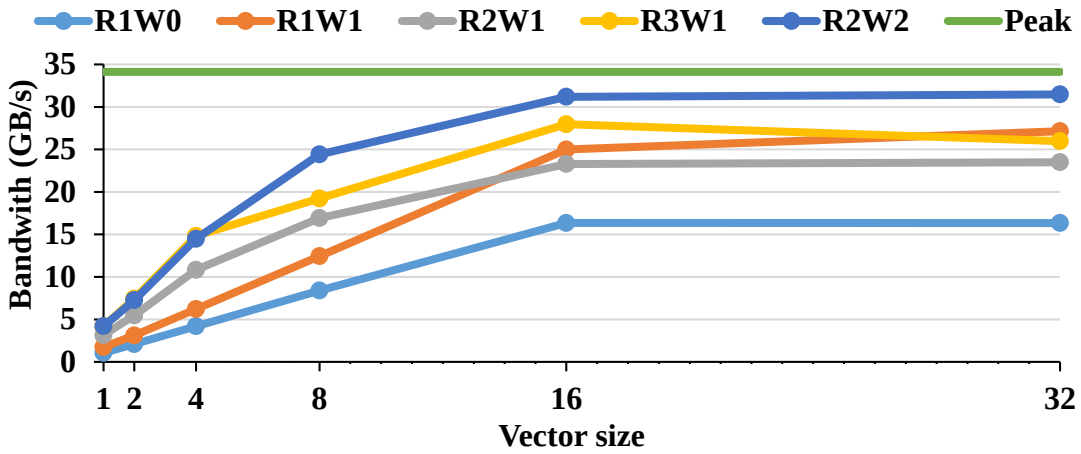
Standard



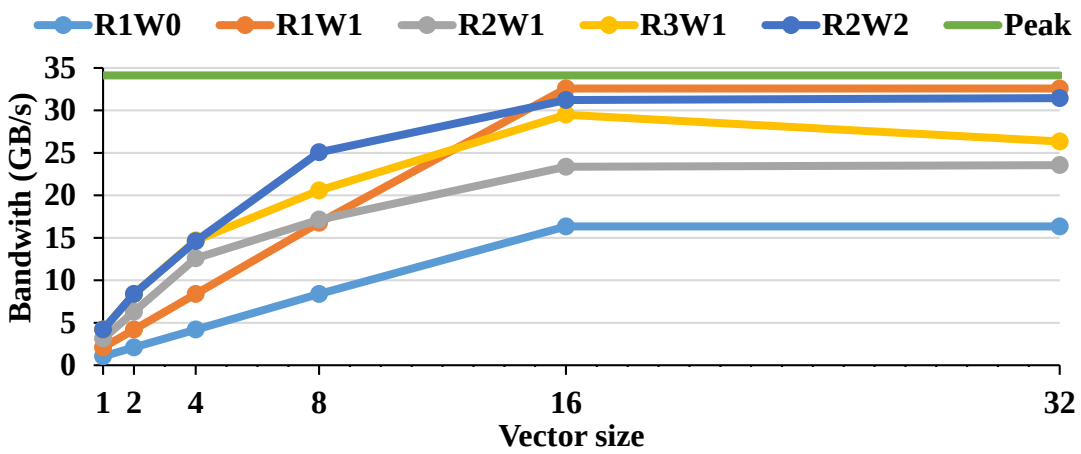
- No performance difference is expected
  - Channel only increases pipeline depth
- Small improvements are observed for more than two array
  - The channel probably improves memory stall absorption

No Blocking – Channelized Single Work-item

Channelized

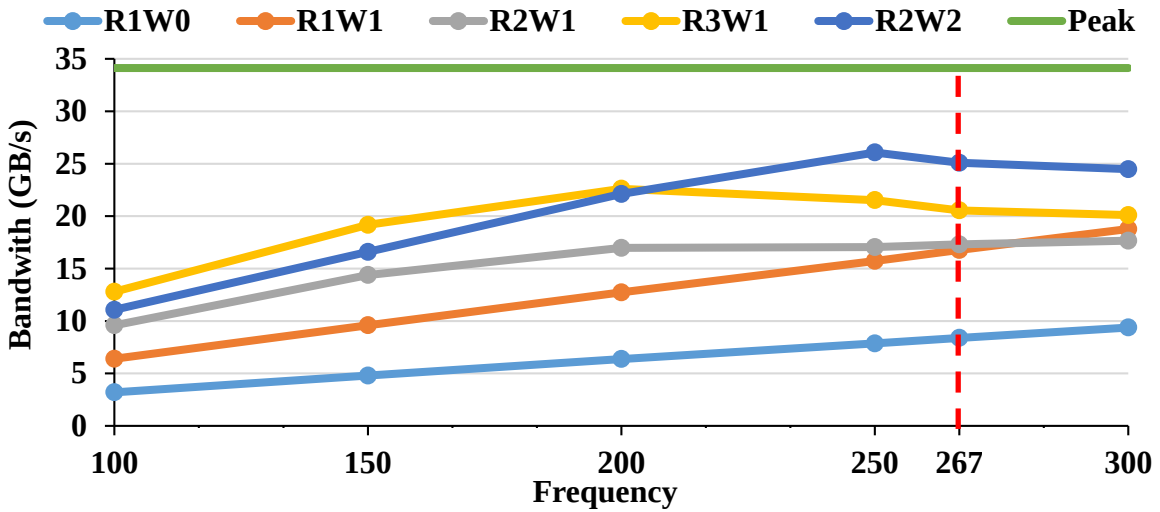


Standard



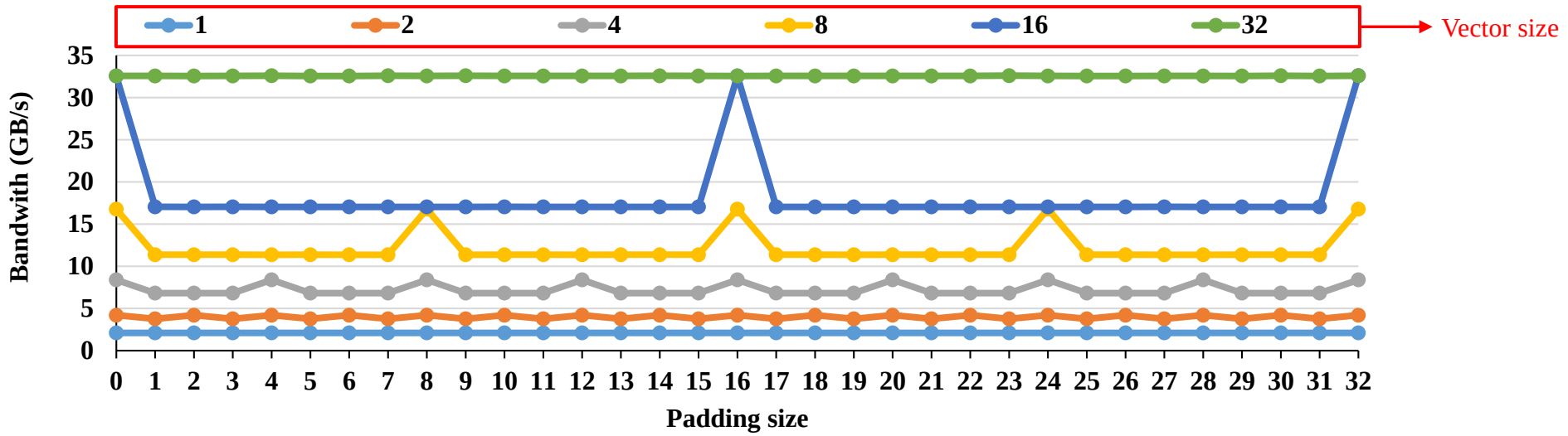
- Performance loss in multiple cases
  - Up to 20% loss for R1W1
- Happens with or without interleaving, other compiler versions, Stratix V
  - This is a performance bug in the compiler => replicated by intel
  - Affects multi-kernel designs with separate memory access and compute, incl. systolic array
  - Work-around: Use NDRange for memory access kernel in multi-kernel designs

No Blocking – Operating Frequency @Vector 8



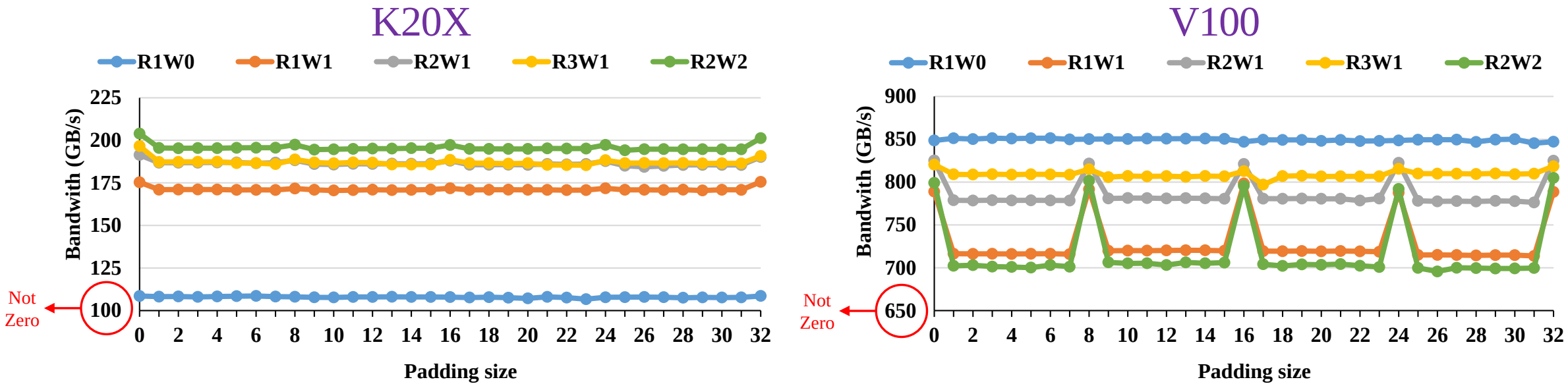
- Performance increases linearly until peak but decreases after that
- Seed and F_{max} sweeping can be used to *increase* frequency and go up the *positive* slope
- Our F_{max} hack can be used to *decrease* frequency and go up the *negative* slope

FPGA: No Blocking – Padding for R1W1



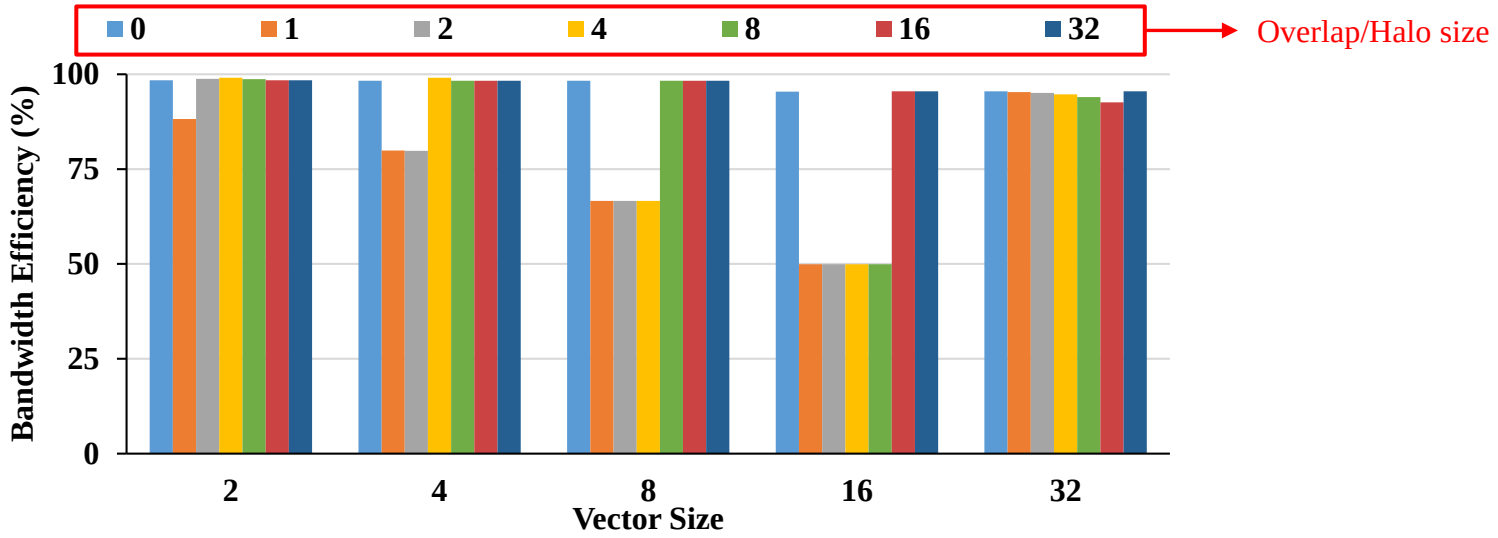
- Memory controller is incapable of access realignment
  - Up to 50% performance can be lost with unaligned memory accesses
  - Performance loss gets smaller with oversubscription; e.g. @32 is unaffected by alignment
- Required alignment is equal vector length; e.g. @16 => 512-bit alignment

GPU: No Blocking – Padding



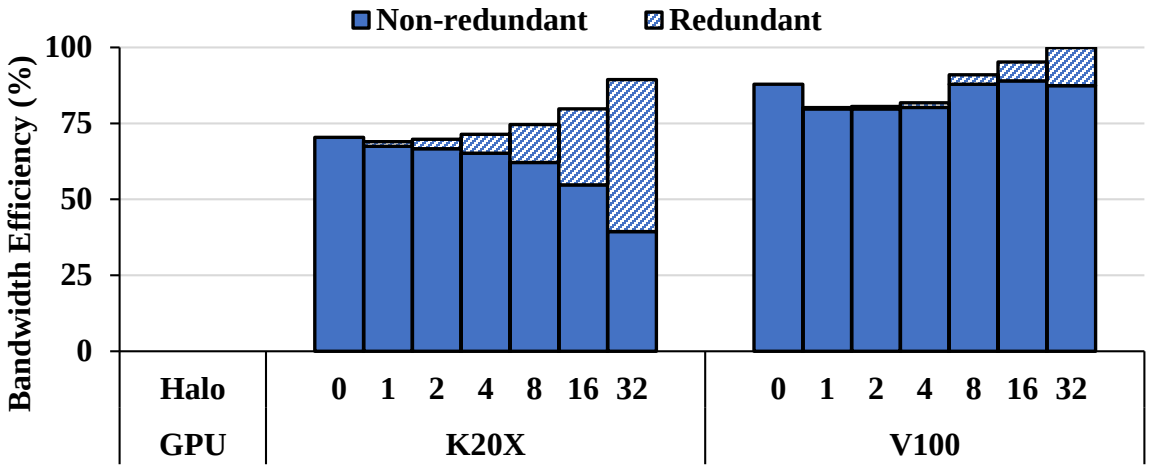
- Small variations are also observed on GPUs with alignment
- Alignment size is fixed and 256 bits
- GDDR seems to be less susceptible compared to HBM
- Maximum performance loss is 13% << 50% on Intel FPGAs

FPGA: 1D Blocking – Overlapping for R1W1



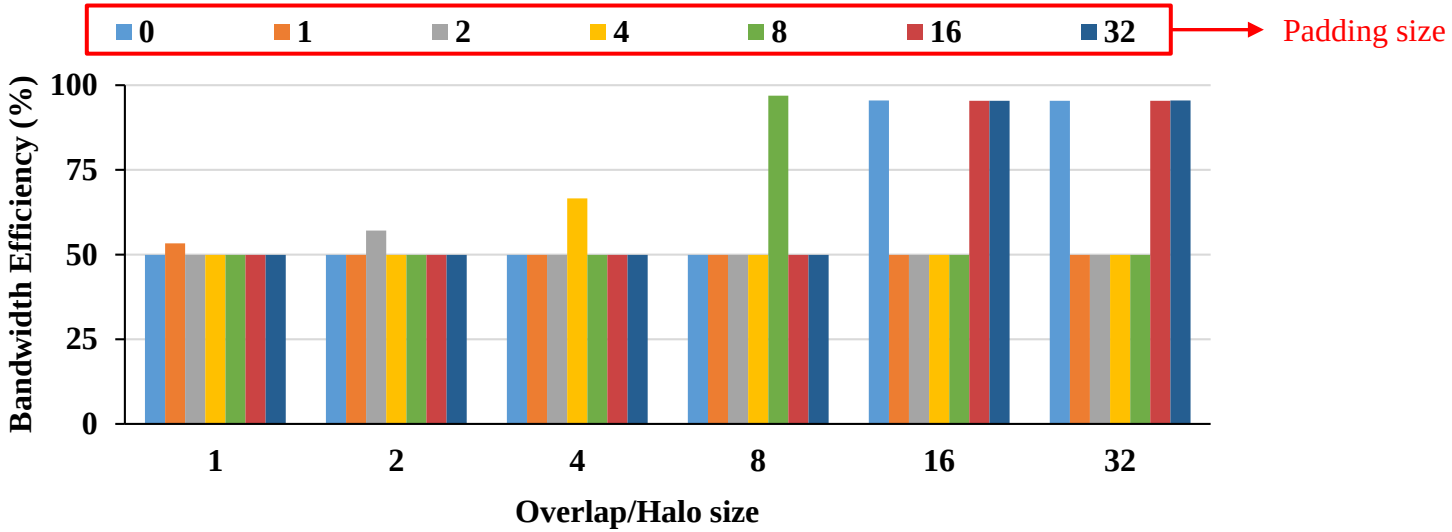
- Overlap size % vector size = 0 => full alignment and maximum efficiency
- Overlap size % vector size ≠ 0 => unaligned accesses and loss of performance

GPU: 1D Blocking – Overlapping for R1W1



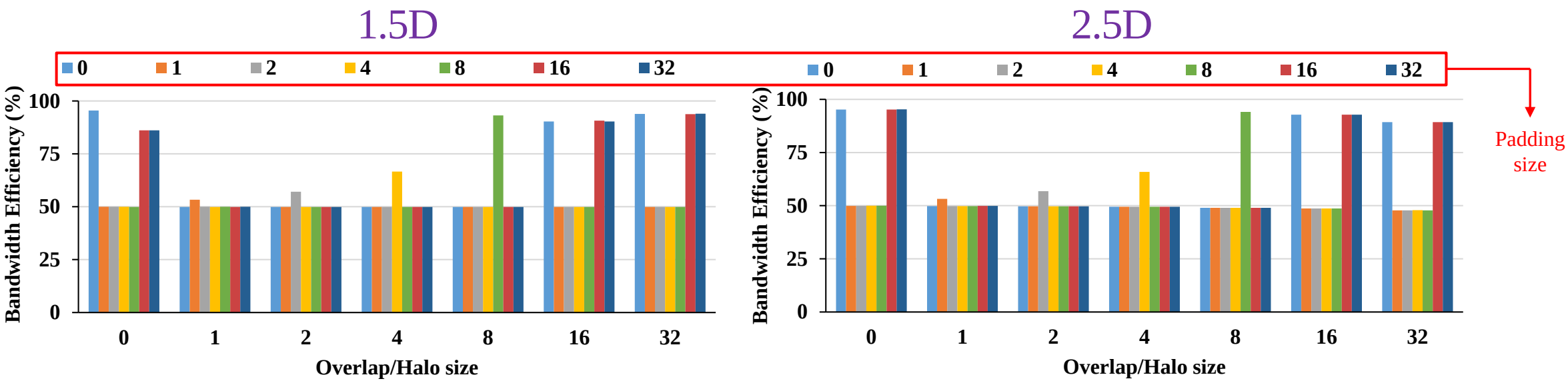
- Performances *increases* with bigger overlapping
  - GPU cache absorbs redundant accesses
- If redundant accesses are ignored
  - V100: Similar pattern as FPGA
  - K20X: Loses cache efficiency as overlap size gets larger

FPGA: 1D Blocking – Overlapping and Padding R1W1 with vector size 16



- Pad arrays to improve alignment
  - Proposed in our previous work at FPGA'18
- Full alignment with padding if overlap size % (vector size / 2) = 0
- Improvements also in other cases if padding size = overlap size % vector size
  - Still a far cry from bandwidth efficiency on GPUs without padding

FPGA: 1.5D and 2.5D Blocking R1W1 with vector size 16



- Performance pattern is the same as 1D blocking
  - Blocking type has little effect on memory performance
  - Main factor is access alignment
- Overlap size might not be configurable and padding might not be enough to fix alignment
  - Only solution would be to improve the memory controller

Conclusion

- For maximum bandwidth efficiency very strict array requirements should be met
  - In typical cases we might not achieve over 70% of the peak memory bandwidth
- The memory controller is very sensitive to memory access alignment
  - Up to 50% of memory performance can be lost due to unaligned accesses
- Padding is useful to improve memory alignment in specific cases
  - The only universal solution is modifying the memory controller

Benchmark source code:
https://github.com/zohourih/FPGAMemBench