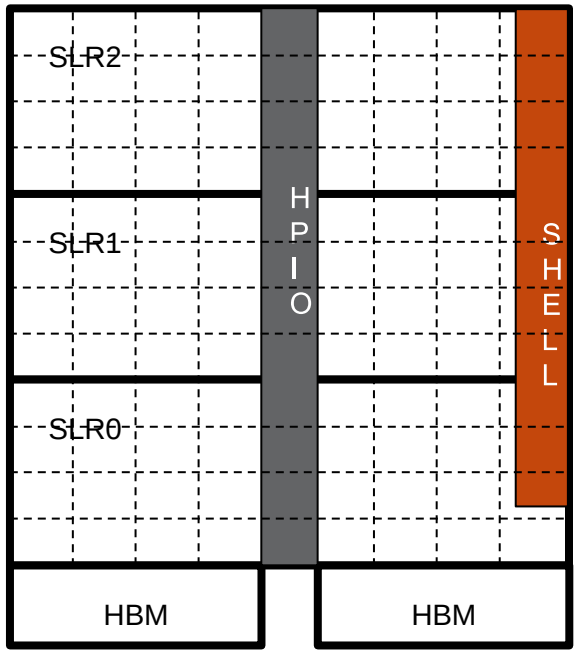
AMD

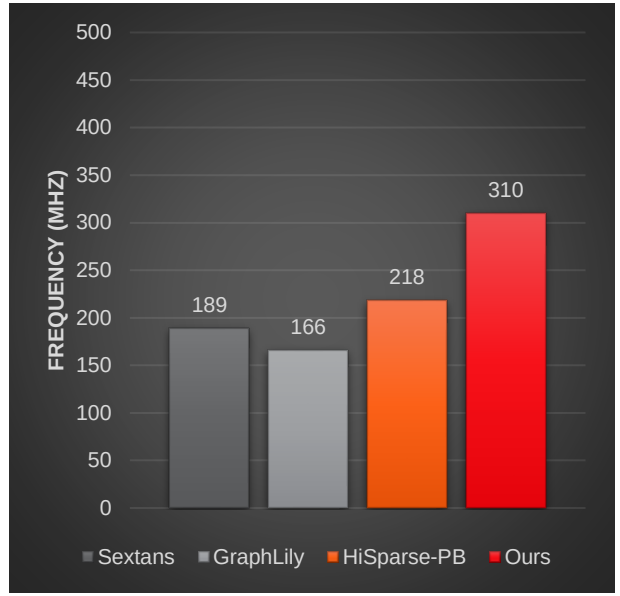
Modular and Lean Architecture with Elasticity for Sparse Matrix Vector Multiplication on FPGAs

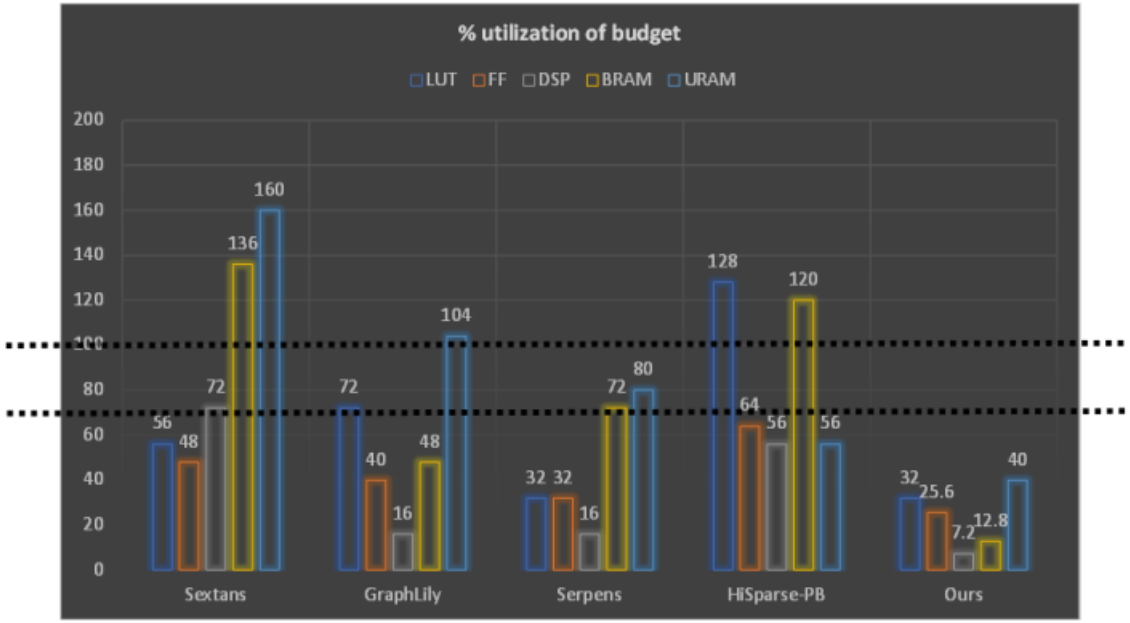
Abhishek Kumar Jain, Chirag Ravishankar, Hossein Omidian, Sharan Kumar, Maithilee Kulkarni, Aashish Tripathi, Dinesh Gaitonde
AMD AECG Architecture Group
10th May 2023

Overview

- SpMV is a suitable workload for FPGAs and HBM-enabled FPGAs can achieve good performance
  - Existing designs show a fraction of what can be achieved theoretically
  - Main issues: challenging to use all HBM bandwidth (run out of resource issue, switching is expensive, dealing with hazard is expensive) and the fmax is low due to difficult timing closure
  - Floating-point designs are challenging to scale (RAW Hazards)
- What is missing?
  - Lean designs so that we can scale to use all HBM bandwidth
  - A method to achieve very high fmax so that we can either match or exceed HBM interface frequency (450 MHz)
- Method to achieve high fmax
  - Decompose the design into smaller building blocks communicating over latency-insensitive elastic channels
  - Add elastic buffers when we see critical path in the design
  - If building blocks are hitting high frequency (>500 MHz), we ideally should be able to maintain that for the entire design

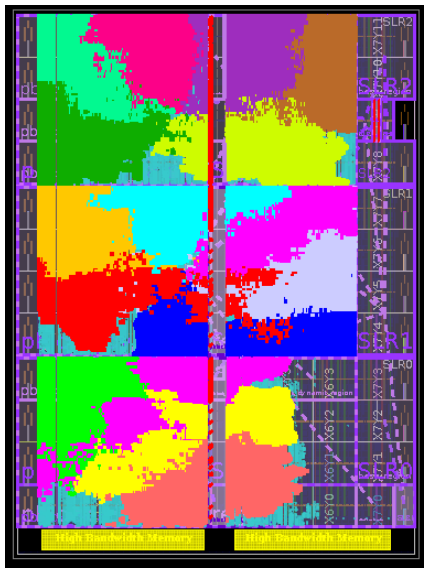




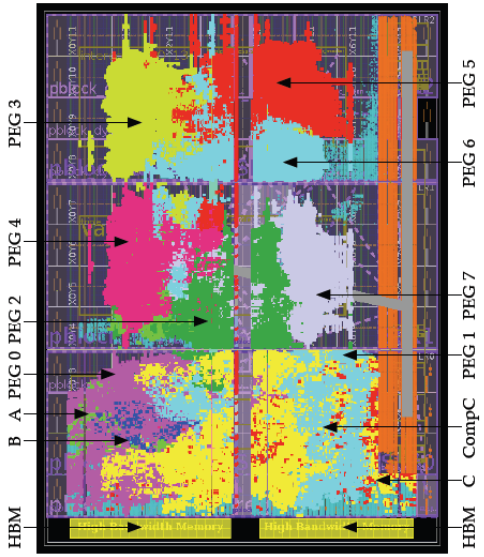


Multi-Die FPGAs with HBM Stacks → Implementation Challenges

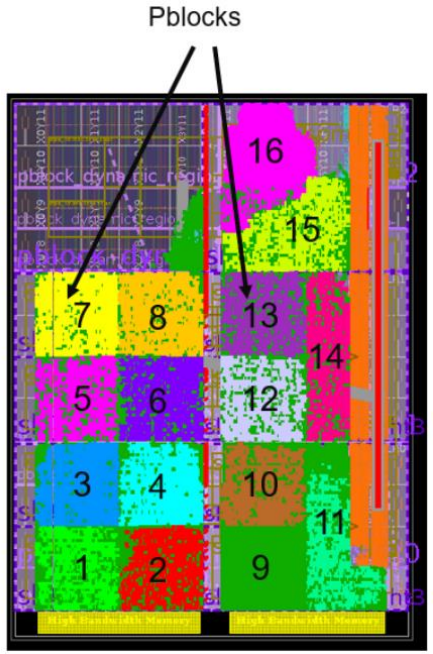
- Moving HBM bandwidth across multi-die FPGA device is challenging (HMSS IP spread all over)
- Mid Column Crossing → Long critical paths
- This work → 465 MHz Fmax for single kernel, Fmax degradation on scaling
- Up-to 2.5x better performance compared to the state-of-the-art
- Active research area: Flat Fmax on scaling



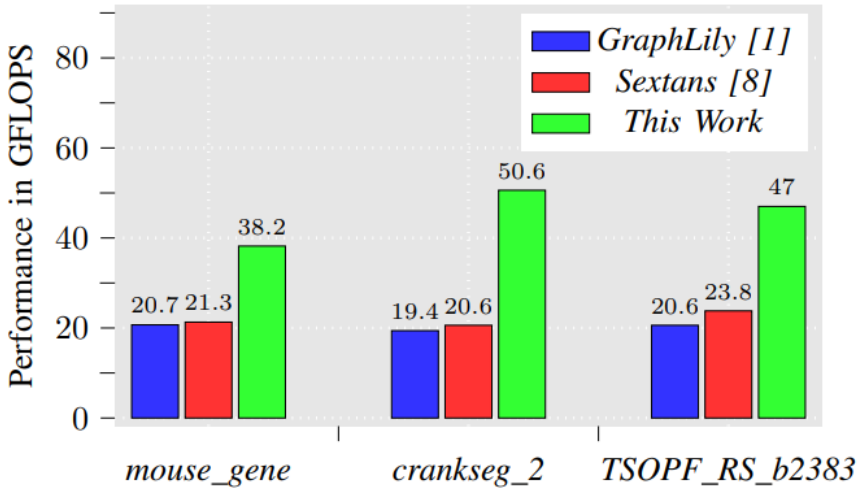
HiSparse-PB: 218 MHz



Sextans: 189 MHz



Ours: 465 MHz for 1 kernel
310 MHz for 16 kernels



Sparse Matrix Vector Multiplication (SpMV) Applications

Linear system solvers

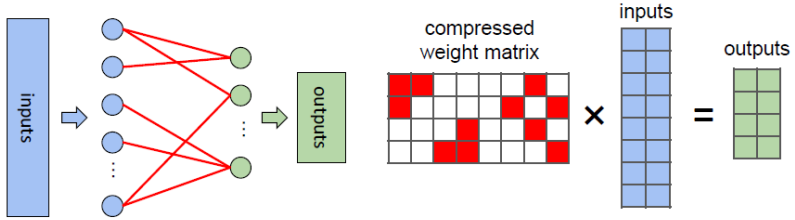
$$A\,x = b$$

$$\begin{pmatrix} 1 & 0 & 1 \\ 0 & -3 & 0 \\ 0 & 0 & 2 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$$

$$A^{-1} \times b = x$$

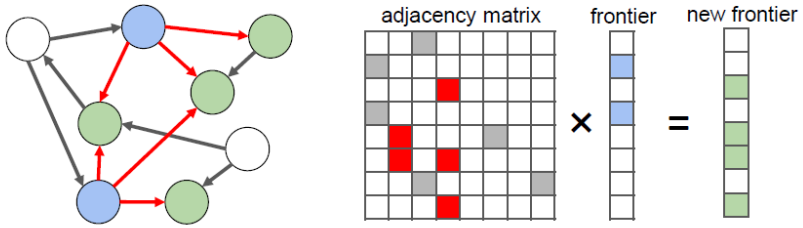
$$\begin{pmatrix} 1 & 0 & -1/2 \\ 0 & -1/3 & 0 \\ 0 & 0 & 1/2 \end{pmatrix}\begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

Compressed neural networks

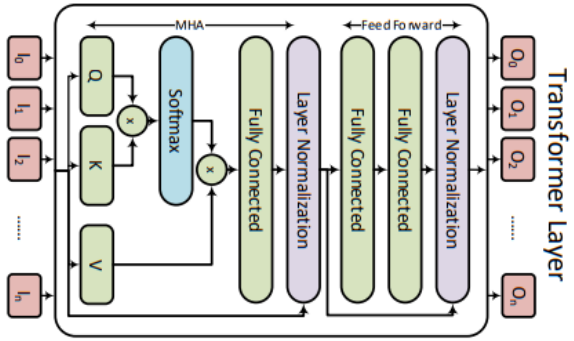


SpMV Applications

Graph traversals

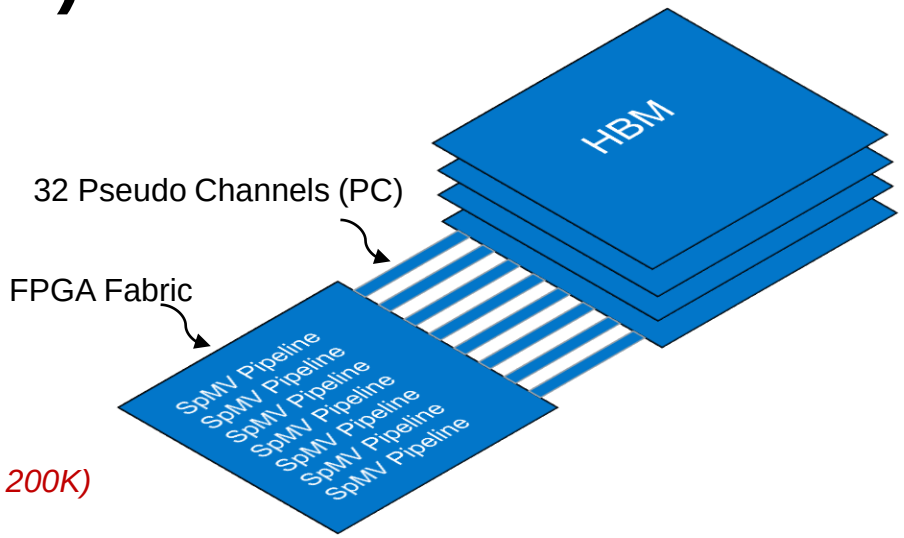


Sparse Transformers



Sparse Matrix Vector Multiplication (SpMV)

- Traditional CPU/GPU platforms do not perform well for SpMV workload
  - Due to highly irregular and random memory access pattern (very high cache miss rate)
- FPGA platforms are attractive for SpMV
  - Ability to avoid off-chip random memory access
  - Use of many block memories (BRAMs/URAMs) to hold ***x*** and ***y*** vectors on-chip
  - Streaming multiple non-zeros (NZs) in parallel from off-chip DRAM
  - *Utilizing available memory bandwidth on new HBM-FPGA platforms is challenging*
  - *Existing designs are over-provisioned and does not scale well (LUT/PC ranges between 20K to 200K)*
  - *Need to bring LUT/PC close to 10K or less*



$$\begin{bmatrix} 1.0 & -- & 1.0 & -- \\ -- & -- & -- & -- \\ -- & -- & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 & 1.0 \end{bmatrix} * \begin{bmatrix} 1.0 \\ 1.0 \\ 1.0 \\ 1.0 \end{bmatrix} = \begin{bmatrix} (1.0)(1.0) + (1.0)(1.0) \\ 0.0 \\ (1.0)(1.0) + (1.0)(1.0) \\ (1.0)(1.0) + (1.0)(1.0) + (1.0)(1.0) + (1.0)(1.0) \end{bmatrix}$$

sparse matrix **A** | dense vector *x* | dense vector *y*

data	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
row	0	0	2	2	3	3	3	3
col	0	2	2	3	0	1	2	3

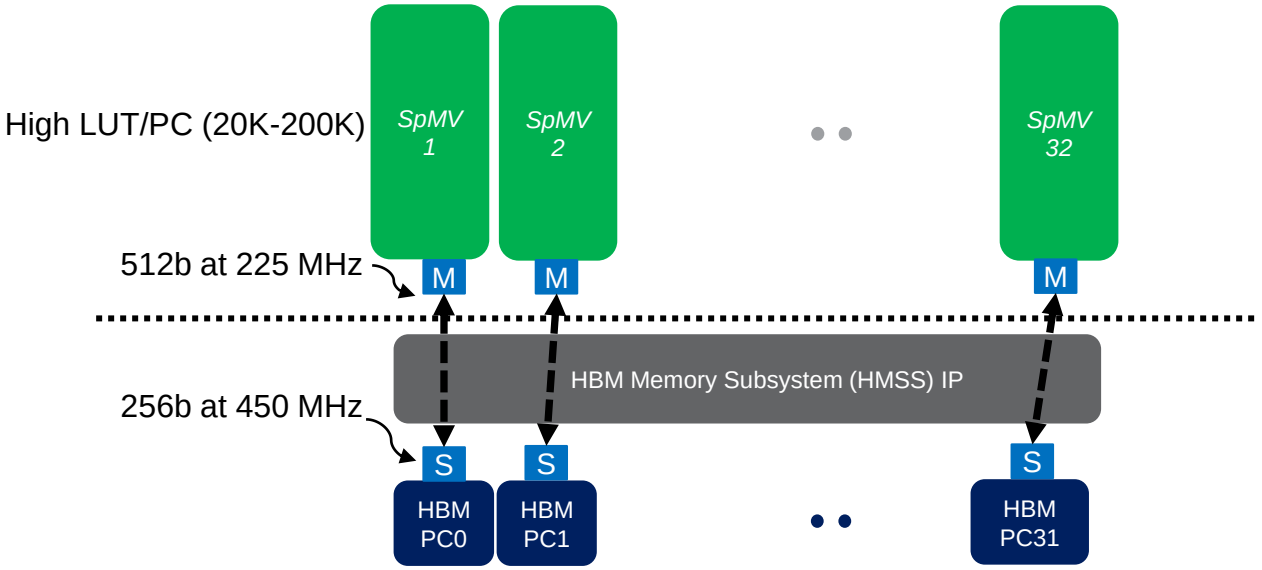
```
for(i = 0; i < NNZs; i++){
    y[row[i]] += data[i] * x[col[i]];
}
```

Note: NNZs → Number of non-zeros

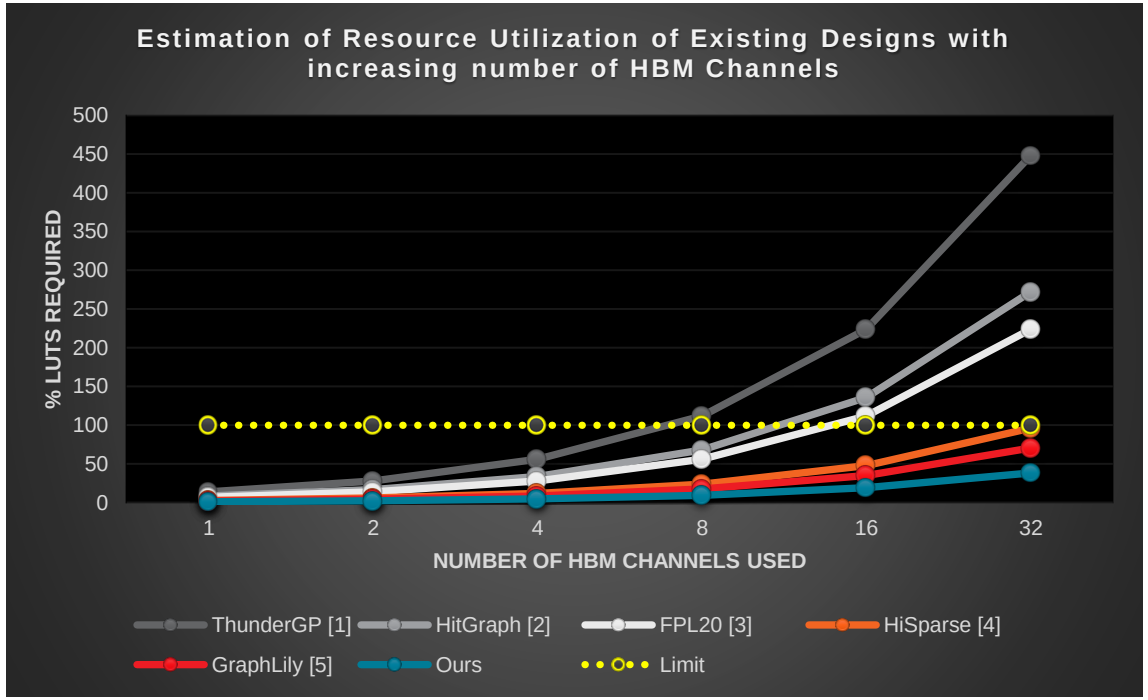
1. Chen, Xinyu, et al. "ThunderGP" *FPGA 2021*
2. Zhou, Shijie, et al. "Hitgraph" *IEEE TPDS 2019*
3. Jain, Abhishek Kumar, et al. "SpMV DSA." *FPL 2020*
4. Du, Yixiao, et al. "HiSparse" *FPGA 2022*
5. Hu, Yuwei, et al. "GraphLily" *ICCAD 2021*

Need for Lean Designs

- Existing designs run out of resources
  - ThunderGP[1] and Hitgraph[2] can use up-to ***4 HBM channels*** (Very high LUT/PC)
  - HiSparse[4] and GraphLily[5] are optimized for HBM implementation but ***16 HBM channels*** are used in the design (Moderate LUT/PC)
- Place and route issues make the timing closure difficult (for high LUT Utilization)



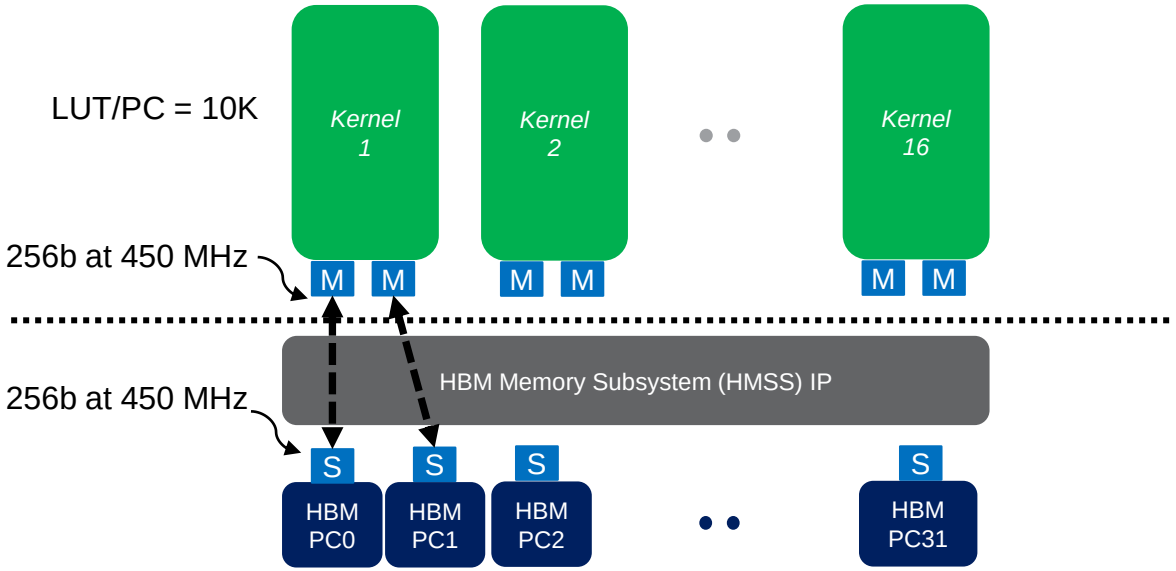
Approach used in existing designs: widen the interface (256b → 512b) to deal with lower clock frequency (raising resources count per PC)



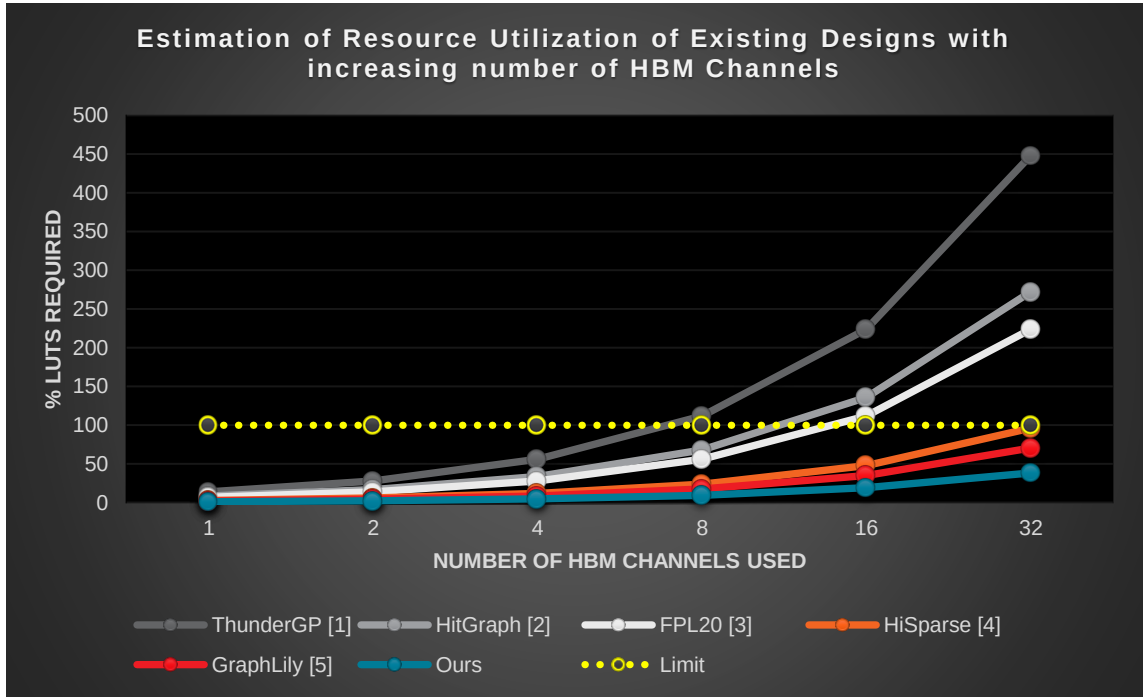
Need for Lean Designs

1. Chen, Xinyu, et al. "ThunderGP" *FPGA 2021*
2. Zhou, Shijie, et al. "Hitgraph" *IEEE TPDS 2019*
3. Jain, Abhishek Kumar, et al. "SpMV DSA." *FPL 2020*
4. Du, Yixiao, et al. "HiSparse" *FPGA 2022*
5. Hu, Yuwei, et al. "GraphLily" *ICCAD 2021*

- Existing designs run out of resources
  - ThunderGP[1] and Hitgraph[2] can use up-to ***4 HBM channels*** (Very high LUT/PC)
  - HiSparse[4] and GraphLily[5] are optimized for HBM implementation but ***16 HBM channels*** are used in the design (Moderate LUT/PC)
- Place and route issues make the timing closure difficult (for high LUT Utilization)
- Our design uses 10K LUTs per PC (1% of U280 LUTs) and utilizes all ***32 HBM channels***



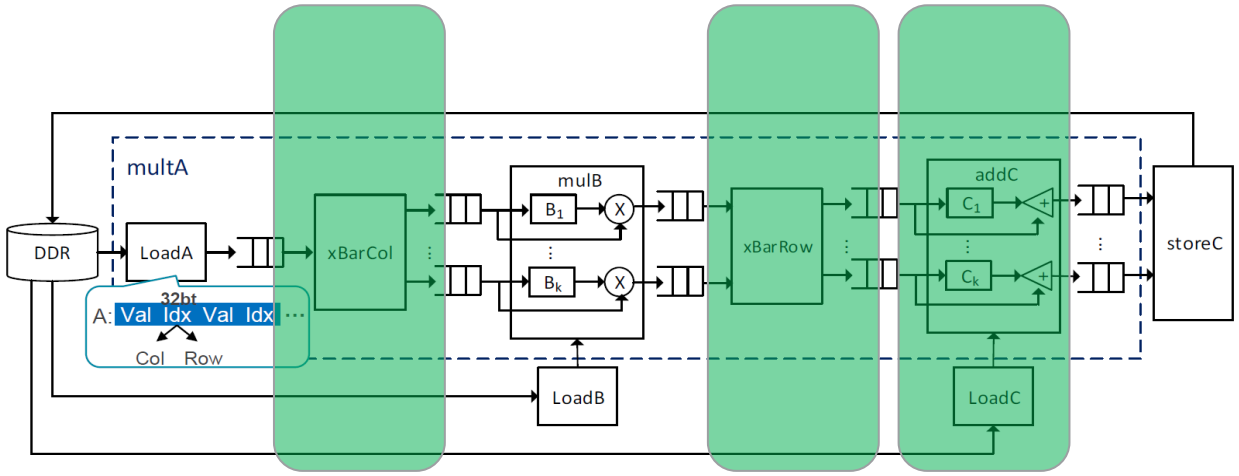
Our approach: design high frequency kernel and avoid widening the interface (reducing resources count per PC)



AMD

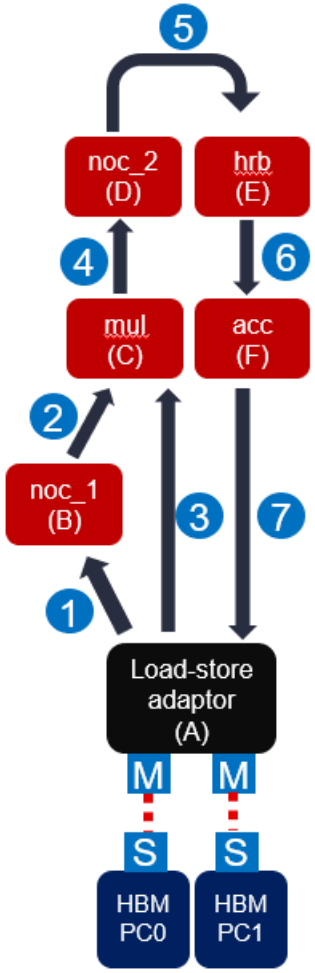
Generic SpMV Pipeline: Key components

- Switching Network
  - Need steering mechanism to route the non-zero to correct vector banks
  - Network is usually the bottleneck → requires large area and runs at low frequencies
- Floating-point Accumulators
  - Need lean mechanism for handling Read-After-Write (RAW) hazards



Example SpMV dataflow: GEMX

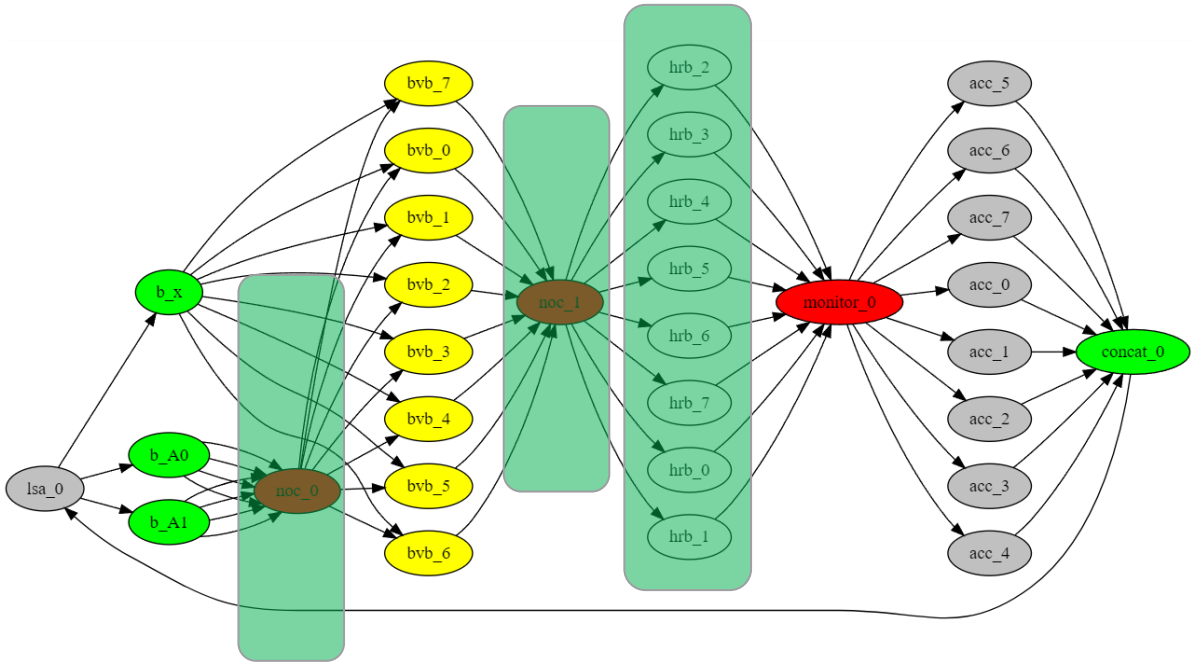
- AXI master 32B interface: M
- AXI slave 32B interface: S
- *AXI stream interfaces connecting six sub-blocks (A,B,C,D,E and F) via AXI stream channels (1,2,3,4,5,6 and 7)*
- 1 and 2 *are 8B wide 8 parallel channels*
- 3 *4B wide 8 parallel channels*
- 4, 5 and 6 *are 6B wide 8 parallel channels*
- 7 *32B wide single channel*



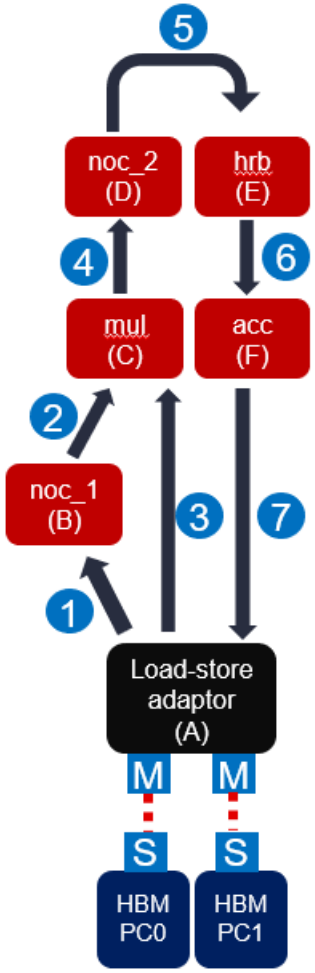
Proposed SpMV dataflow → Elastic channel between each building block

Generic SpMV Pipeline: Key components

- Switching Network
  - Need steering mechanism to route the non-zero to correct vector banks
  - Network is usually the bottleneck → requires large area and runs at low frequencies
- Floating-point Accumulators
  - Need lean mechanism for handling Read-After-Write (RAW) hazards



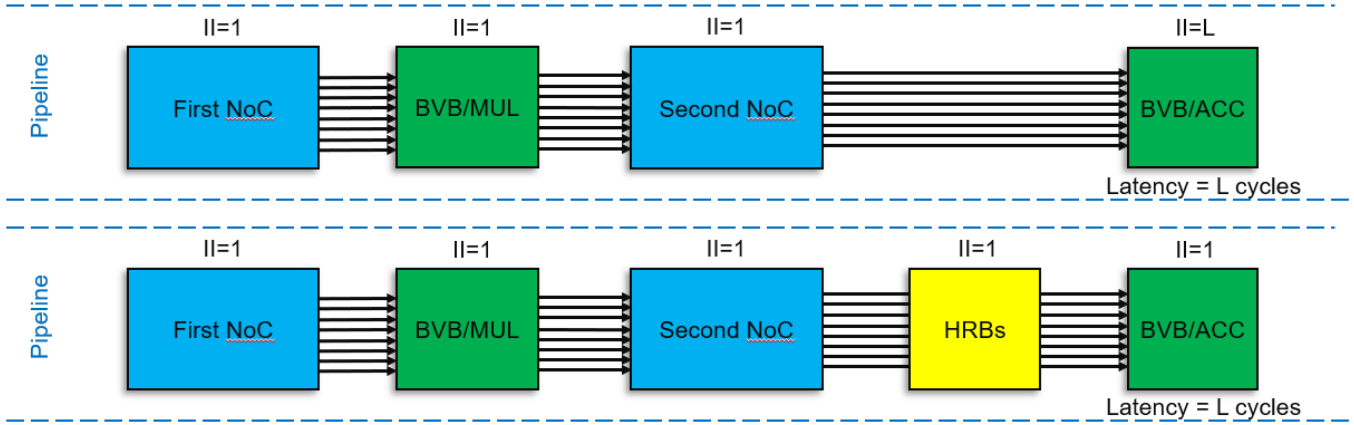
- AXI master 32B interface: M
- AXI slave 32B interface: S
- *AXI stream interfaces connecting six sub-blocks (A,B,C,D,E and F) via AXI stream channels (1,2,3,4,5,6 and 7)*
- 1 and 2 *are 8B wide 8 parallel channels*
- 3 *4B wide 8 parallel channels*
- 4, 5 and 6 *are 6B wide 8 parallel channels*
- 7 *32B wide single channel*



Proposed SpMV dataflow → Elastic channel between each building block

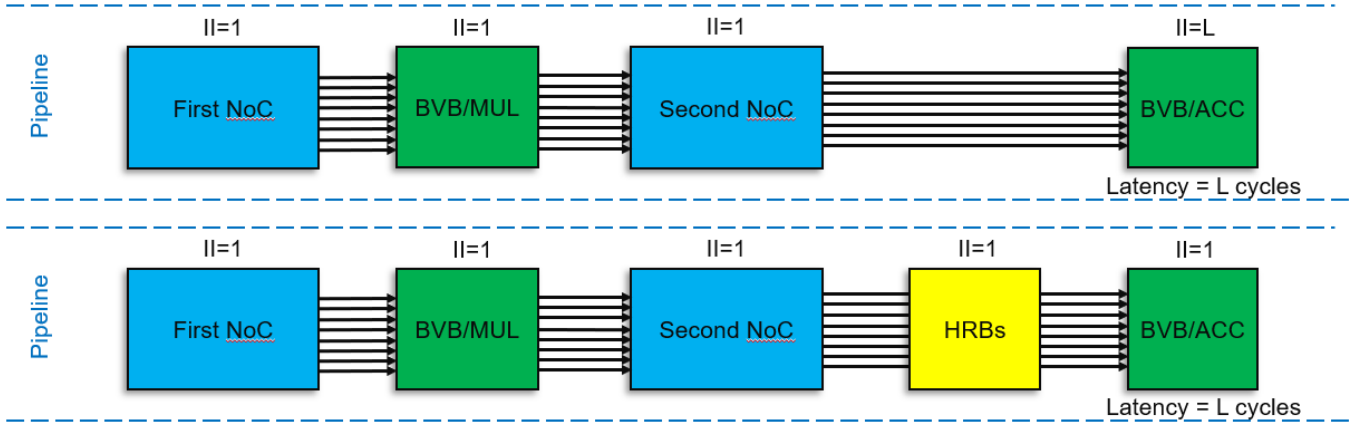
Issue of RAW Hazards during Accumulation

- RAW Hazard
  - Common issue in many linear algebra FPGA designs
  - FP32 Read, Accumulate, Write → L cycles latency, II = L cycles
  - Overall pipeline II = L, Throughput reduced L times
  - Insert Hazard Reducing Back-pressure (HRB) unit
  - Ensure no hazard during accumulation
  - New effective II for ACC = 1, New II for overall pipeline = 1

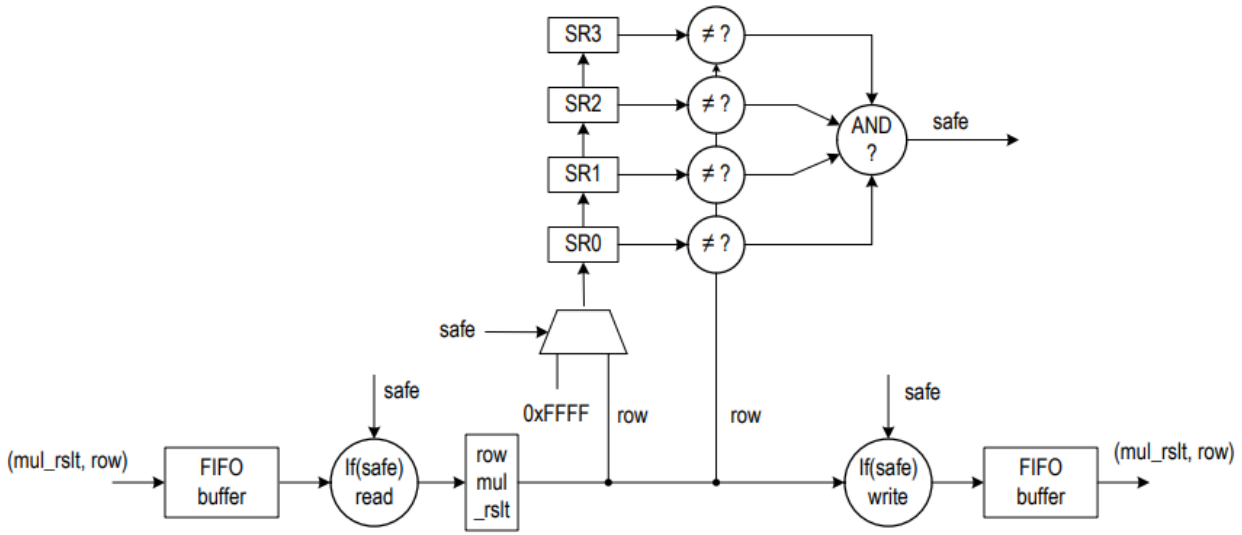


Lean Block for handling RAW Hazards during Accumulation

- RAW Hazard
  - Common issue in many linear algebra FPGA designs
  - FP32 Read, Accumulate, Write → L cycles latency, II = L cycles
  - Overall pipeline II = L, Throughput reduced L times
  - Insert Hazard Reducing Back-pressure (HRB) unit
  - Ensure no hazard during accumulation
  - New effective II for ACC = 1, New II for overall pipeline = 1

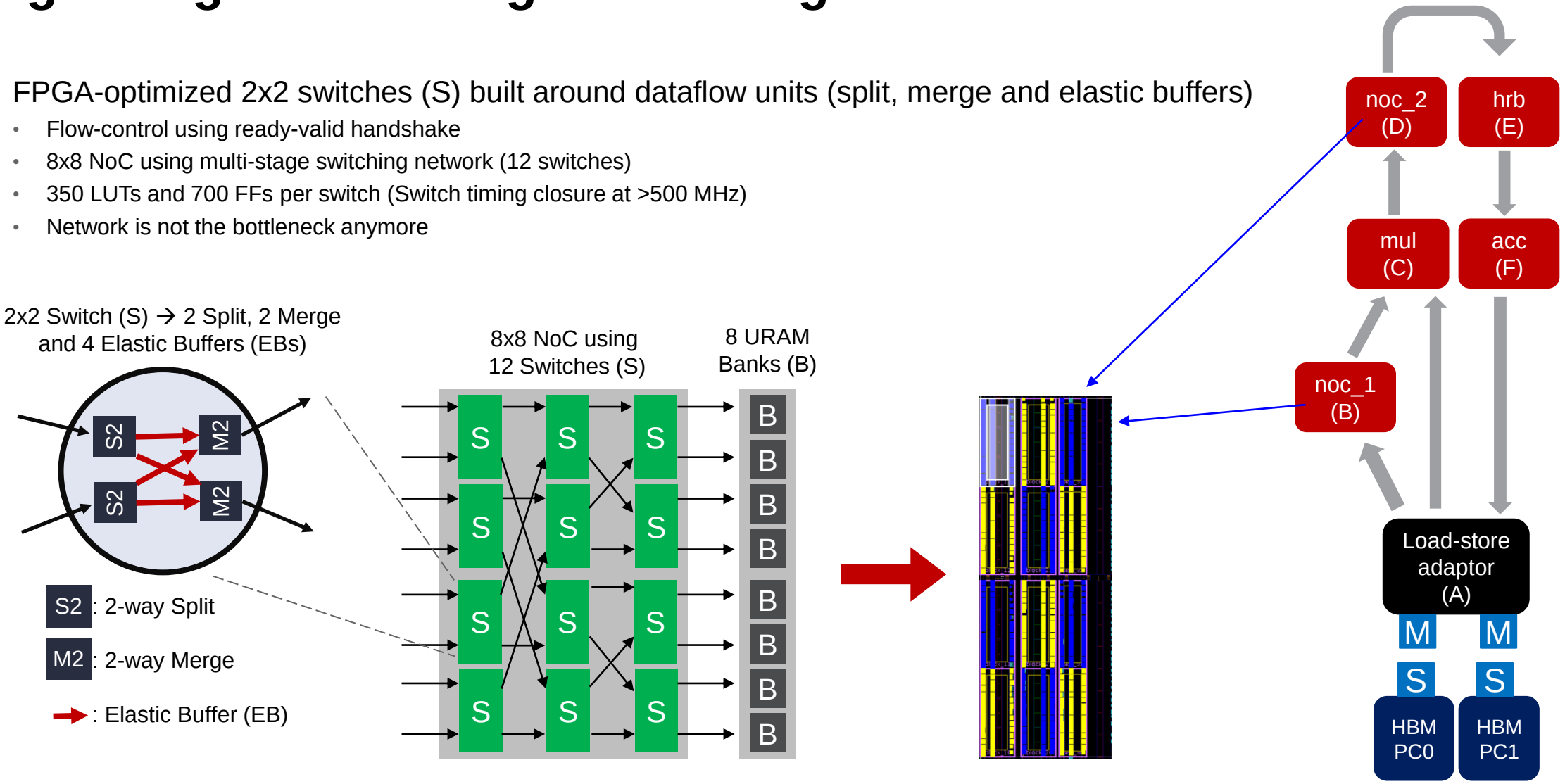


- **H**azard **R**educing **B**ack-pressure (HRB) unit
  - Low area, high frequency, latency insensitive interface
  - Using shift registers to keep history of in-flight indexes
  - New index-value pair (IVP) comes in → compare incoming index with all the indexes in shift registers
  - If no match, safe to proceed further
  - If match, stall the pipeline and wait



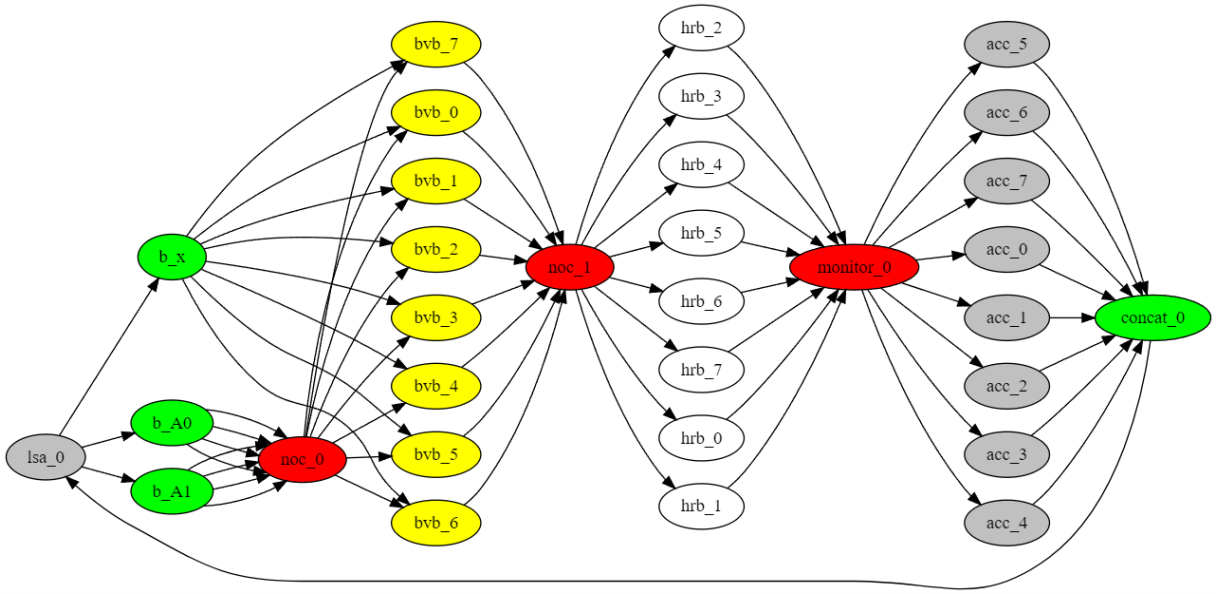
Lightweight Multi-stage Switching Network

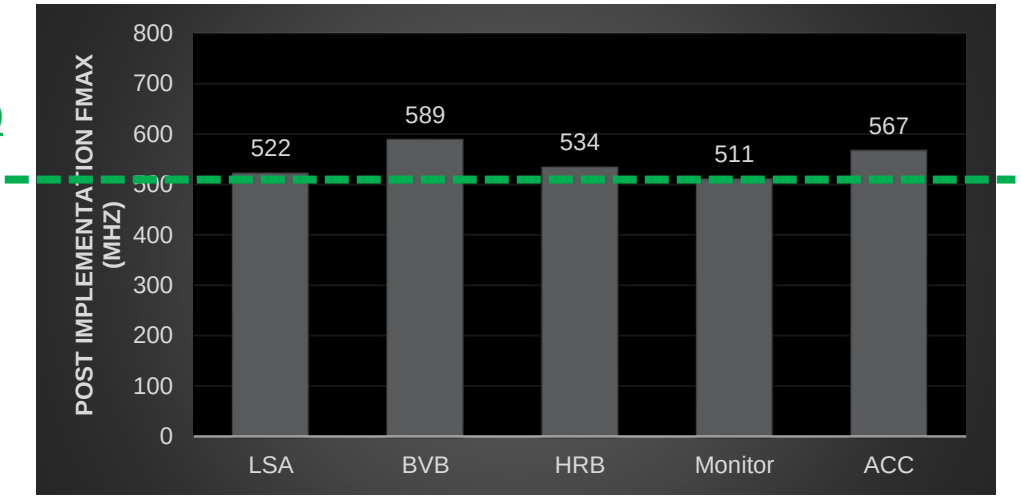
- FPGA-optimized 2x2 switches (S) built around dataflow units (split, merge and elastic buffers)
  - Flow-control using ready-valid handshake
  - 8x8 NoC using multi-stage switching network (12 switches)
  - 350 LUTs and 700 FFs per switch (Switch timing closure at >500 MHz)
  - Network is not the bottleneck anymore

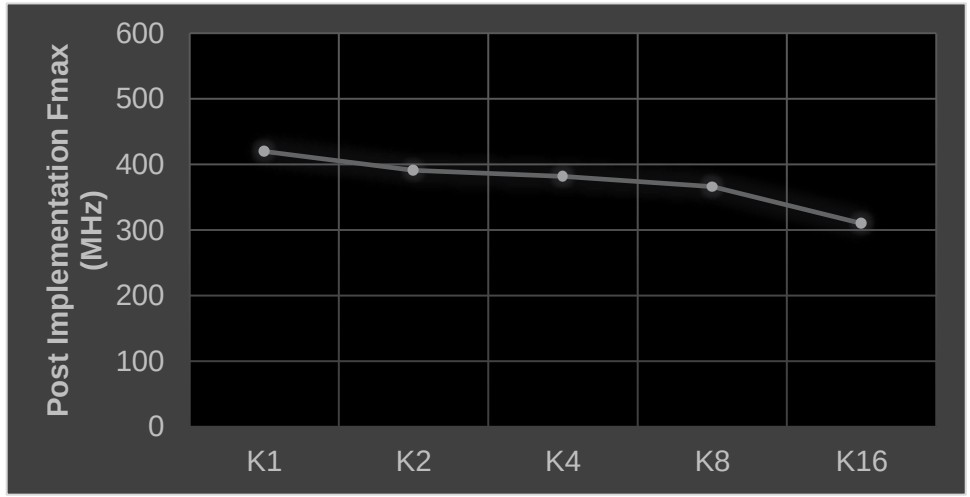


Maintain Design Semantics for Achieving Good Performance

- Monolithic versus Modular & Elastic Design
  - Each module within the kernel can close timing >500 MHz
  - Naïve stitching of the modules → 340 MHz clock frequency for the kernel below
  - Adding EBs and optimizing the modules → ***465 MHz clock frequency (37% improvement)***
  - Full design (16 kernels) shows degradation in fmax
  - Active area of research → Flat Fmax while scaling the design size



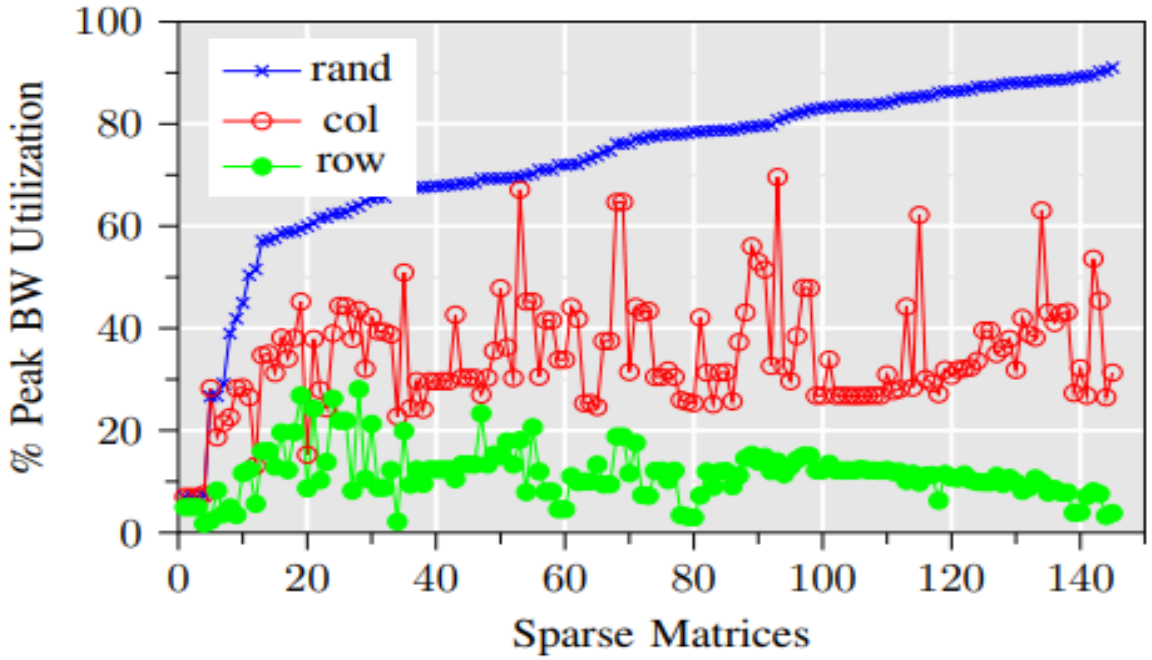


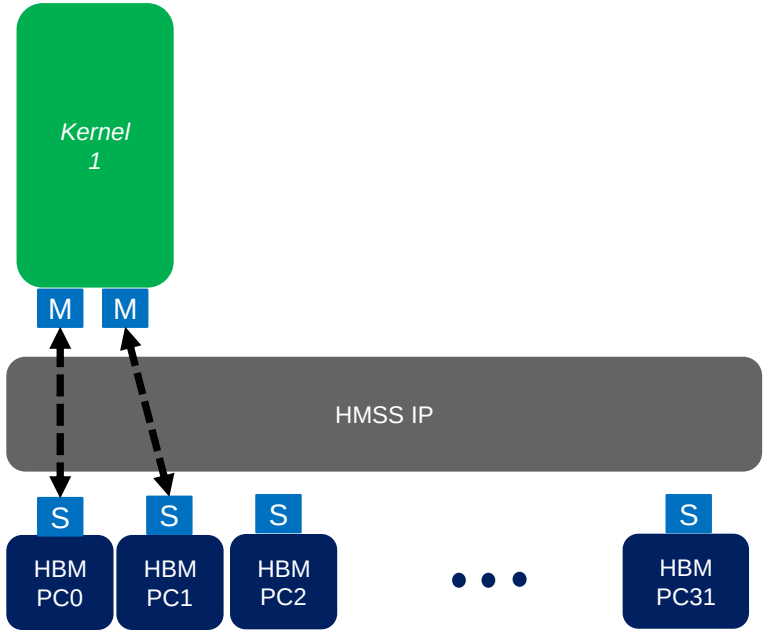


Experiments: Single Kernel

Resources	FFs	LUTs	DSPs	BRAMs	URAMs
Kernel (2 PC)	50K (1.6%)	20K (2%)	40 (0.44%)	16 (0.8%)	24 (2.5%)

- Goal: observe how much bandwidth can be utilized using our kernel connected to two HBM PCs (peak 14.4 GB/s)
- 150 Matrices from SuiteSparse Matrix Collection
- Design runs at 465 MHz → up-to 90% utilization of available bandwidth
- Random traversal outperforms column-major and row-major

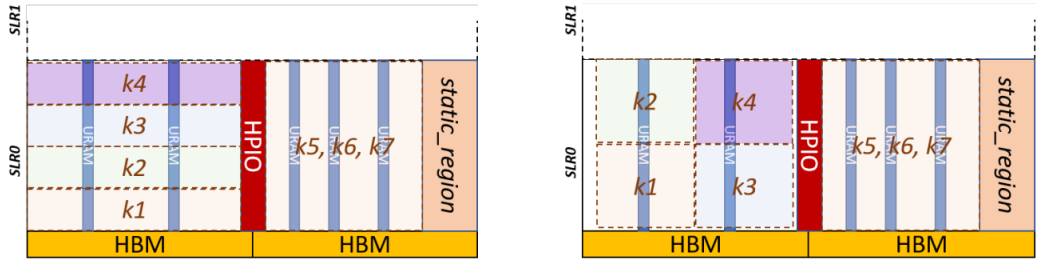




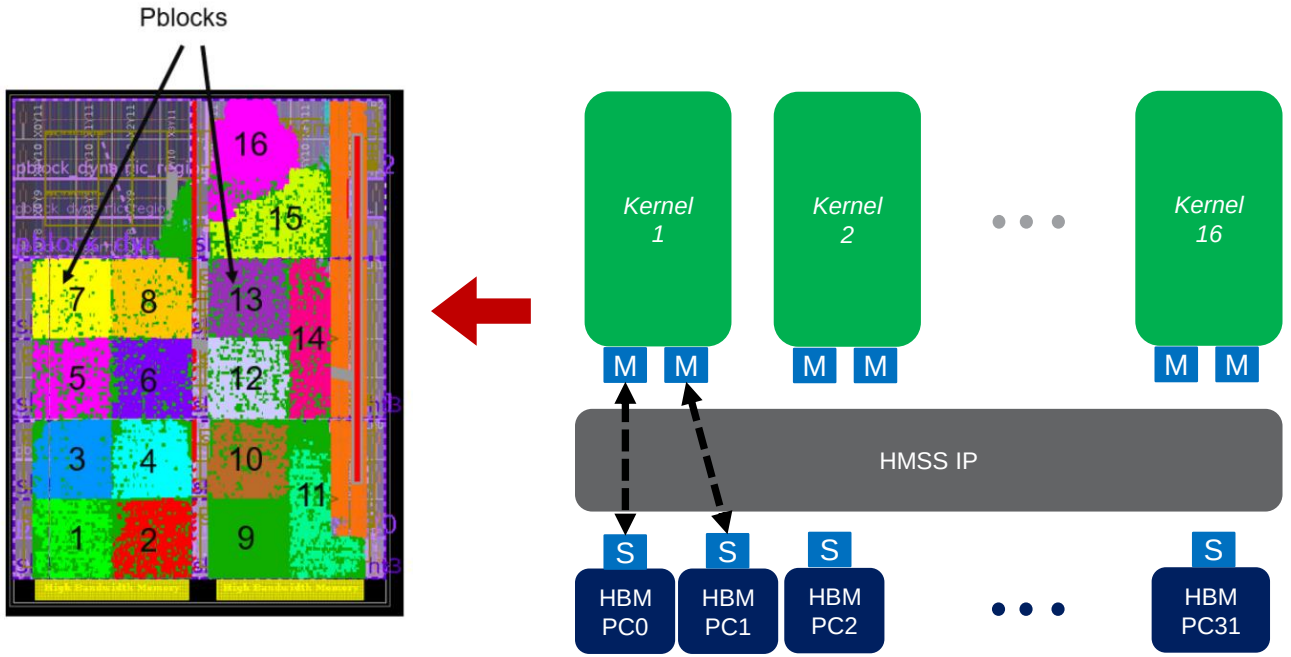
AMD

Scaling the Design for utilizing all HBM Channels

- 16 Kernels implemented on Alveo U280
  - Considered two floorplan options as shown here
  - Location of URAM columns on the device → challenging to floorplan effectively
- Timing closed at 310 MHz for 16 kernel design
  - Floorplan each kernel, each kernel uses < 2.5% of device resources

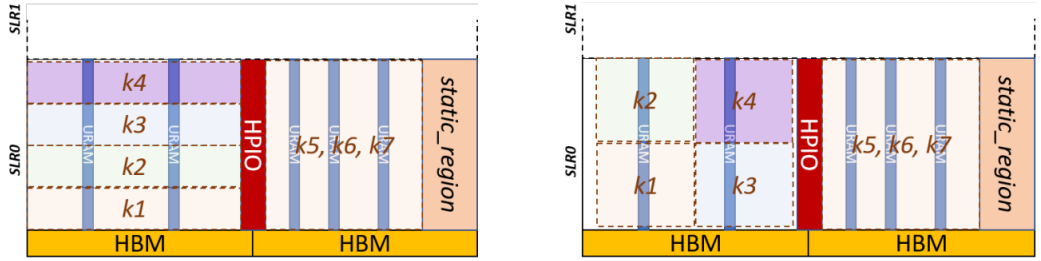


(a) Option 1 (b) Option 2

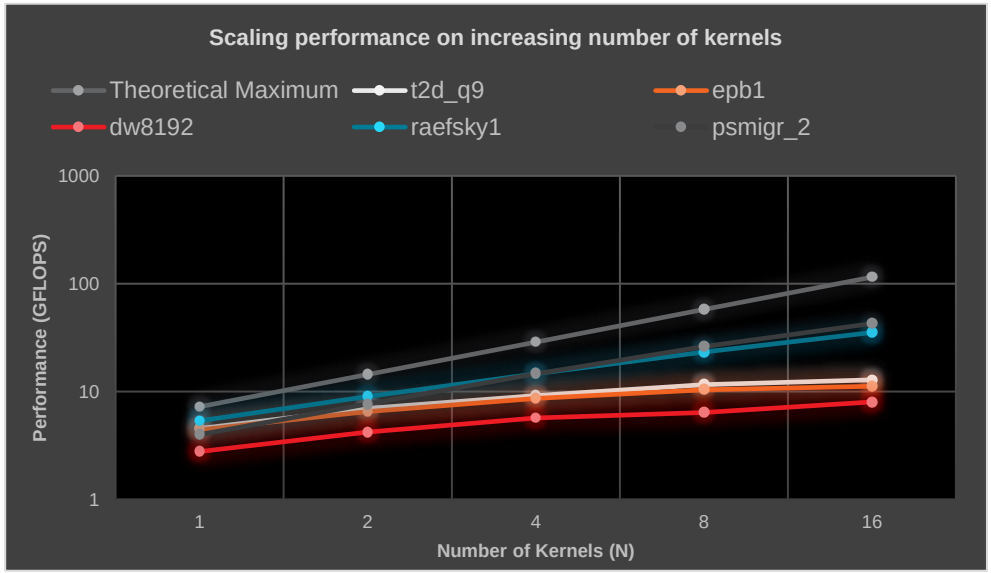


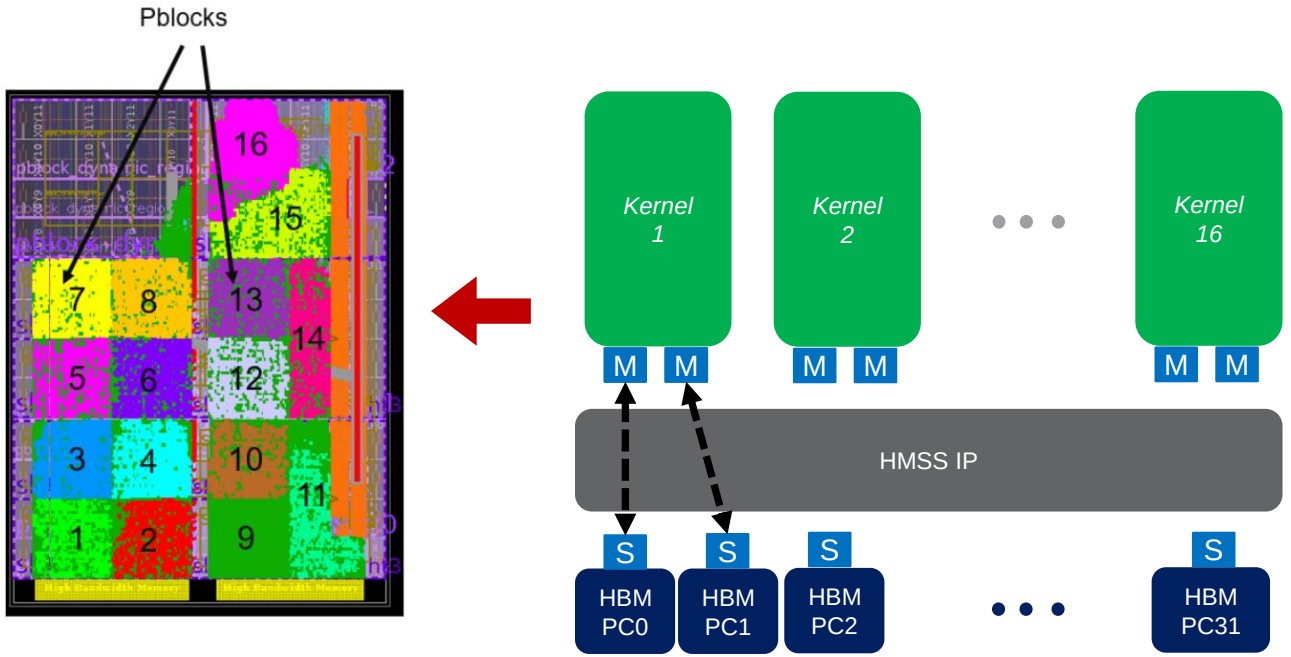
Scaling the Design for utilizing all HBM Channels

- 16 Kernels implemented on Alveo U280
  - Considered two floorplan options as shown here
  - Location of URAM columns on the device → challenging to floorplan effectively
- Timing closed at 310 MHz for 16 kernel design
  - Floorplan each kernel, each kernel uses < 2.5% of device resources
- More kernels allow SpMV problem to scale better
  - Matrix is divided equally among kernel by creating horizontal cuts
  - Each subproblem becomes smaller SpMV problem
  - Enabling more kernels allow performance scaling as shown here



(a) Option 1 (b) Option 2

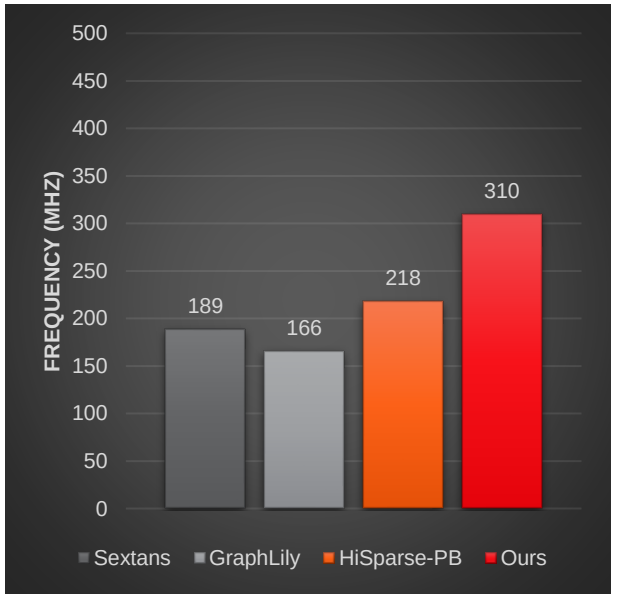


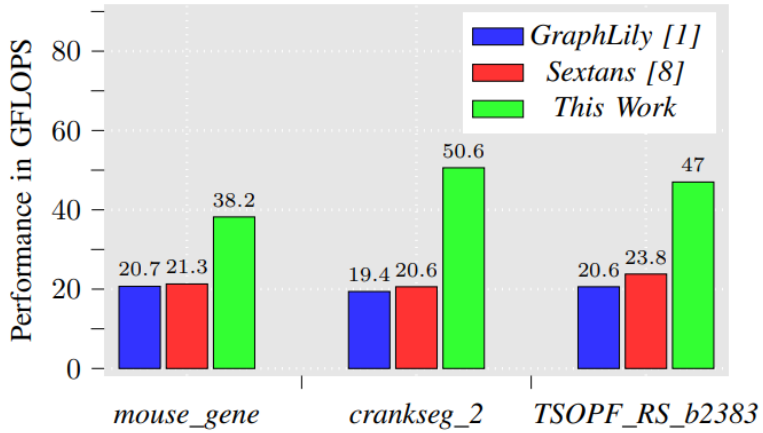


AMD

Comparison with State-of-the-art SpMV Accelerators

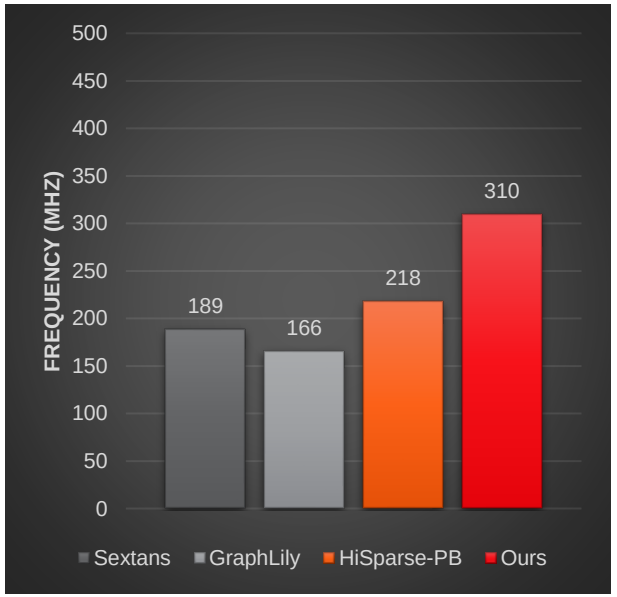
- Comparing our design with state-of-the-art (all on Alveo U280)
  - Our design Fmax: 310 MHz
  - Delivering a performance of up-to 50.6 GFLOPS on Alveo U280
  - Up-to 2.5x higher performance compared to the state-of-the-art

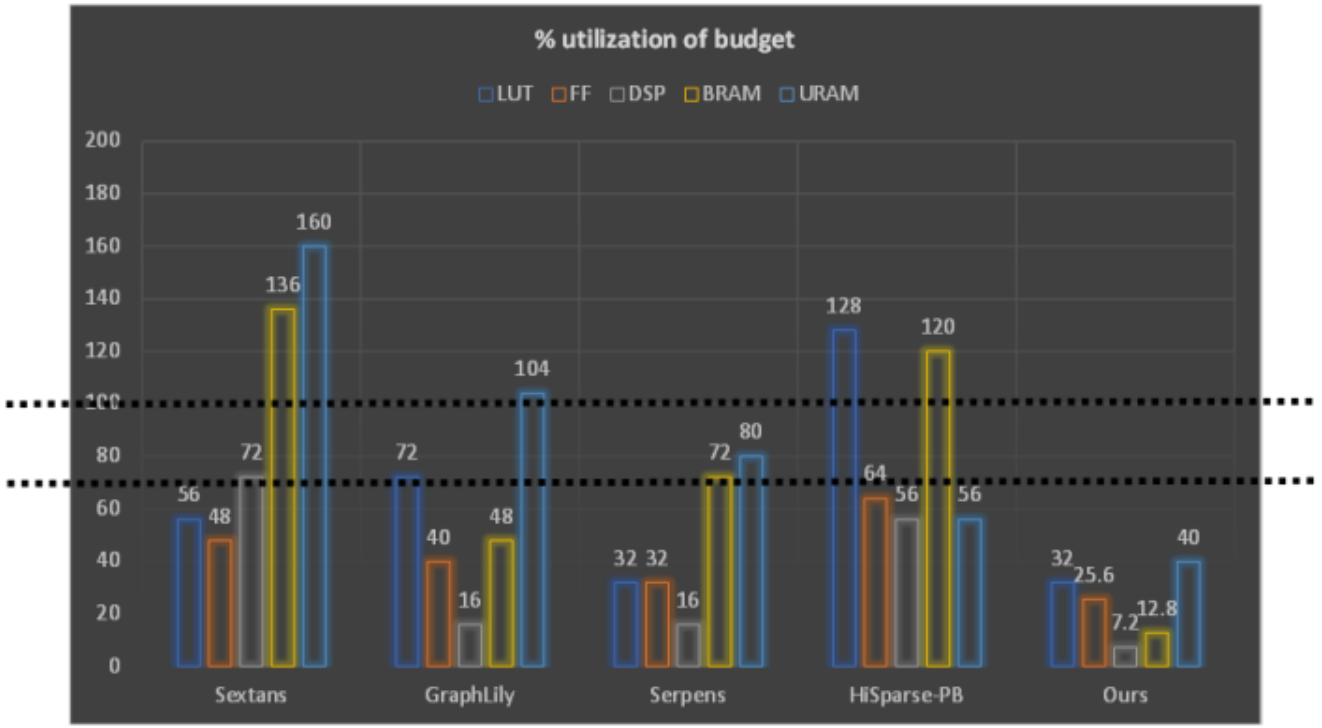


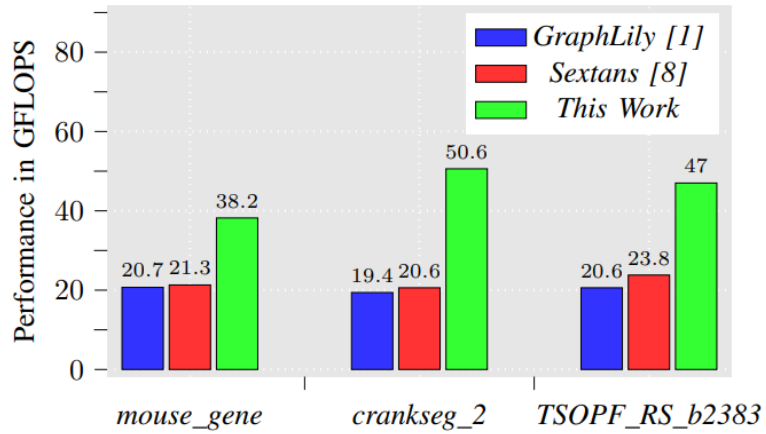


Comparison with State-of-the-art SpMV Accelerators

- Comparing our design with state-of-the-art (all on Alveo U280)
  - Our design Fmax: 310 MHz
  - Delivering a performance of up-to 50.6 GFLOPS on Alveo U280
  - Up-to 2.5x higher performance compared to the state-of-the-art
- Comparing resource requirements (%) with other SpMV Accelerators

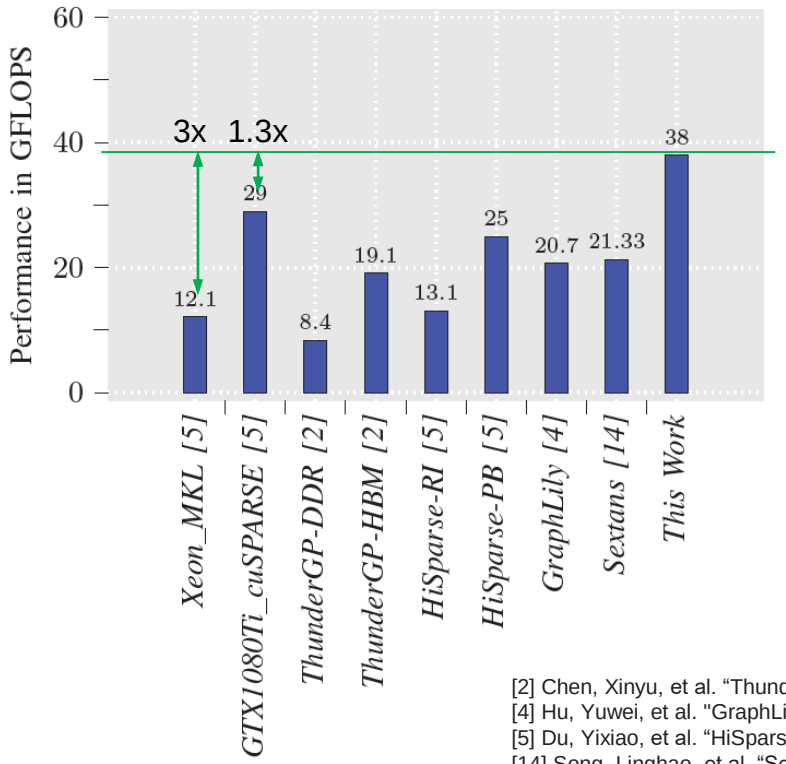


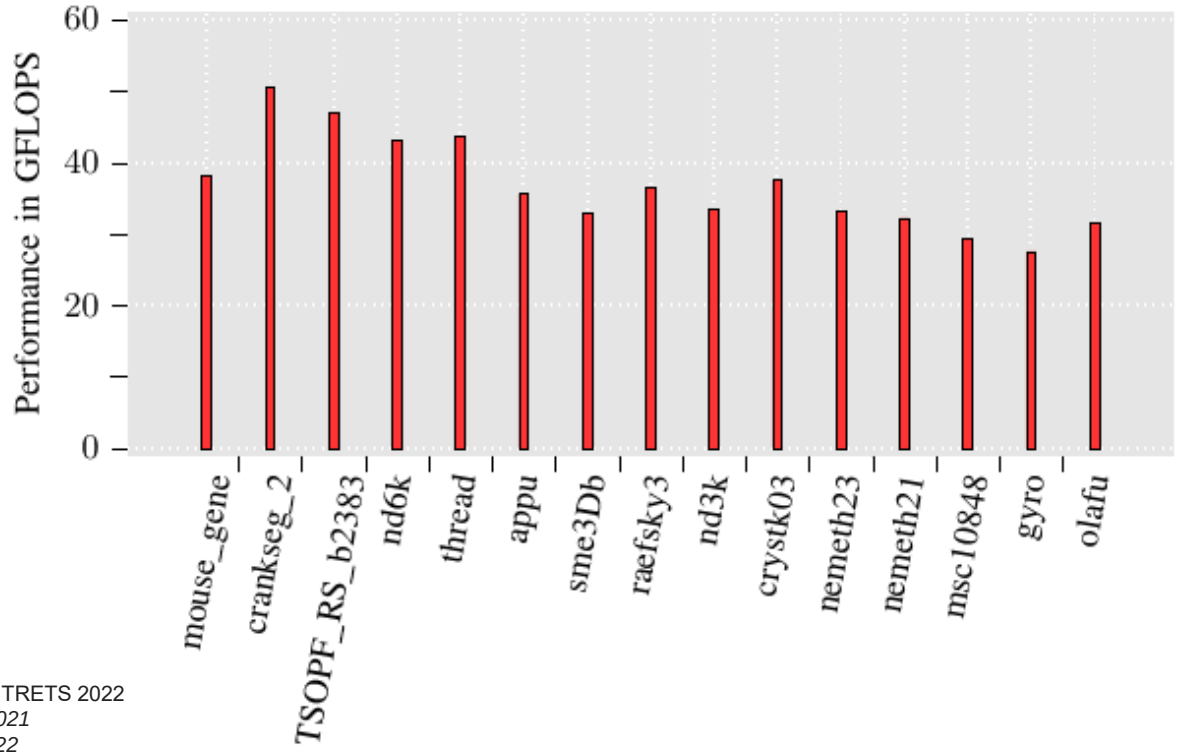




Comparison with State-of-the-art SpMV Accelerators

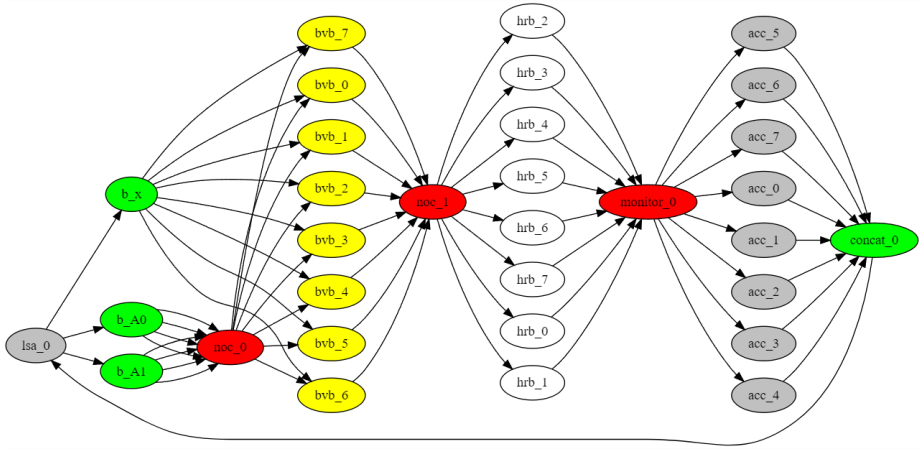
- Comparing with CPU, GPU and other implementations
  - Matrix used: *mouse_gene* (29M non-zero, 98.6% Sparsity)
  - Our design outperforms CPU by 3x and GPU by 1.3x
  - Our design consistently outperforms other implementations
- Up-to 50 GFLOPS performance for a wide range of sparse matrices



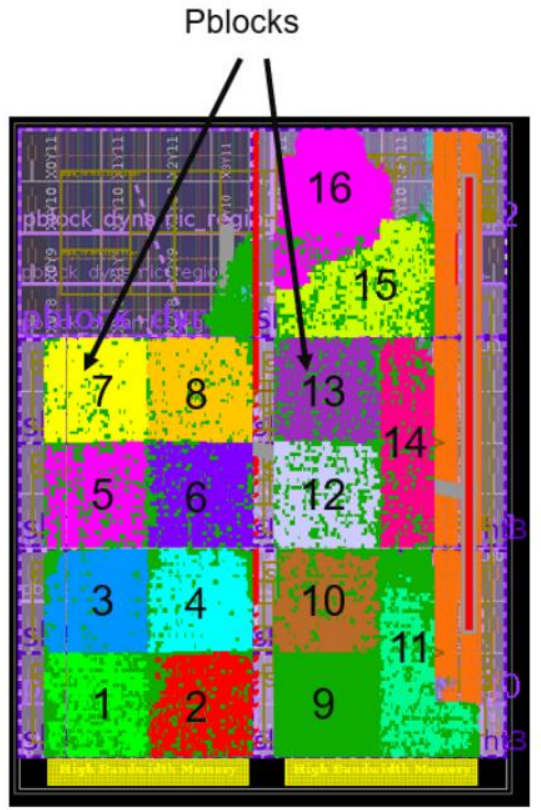


[2] Chen, Xinyu, et al. “ThunderGP” ACM TRETS 2022
[4] Hu, Yuwei, et al. "GraphLily" *ICCAD 2021*
[5] Du, Yixiao, et al. “HiSparse" *FPGA 2022*
[14] Song, Linghao, et al. “Sextans" *FPGA 2022*

Conclusions and Future Work



- FPGA devices remain attractive for sparse computations
  - Higher utilization of available bandwidth is possible through lean and modular SpMV design
  - First order concern → design should be lean and fast
- Presented a modular and lean design for high performance SpMV implementation
  - All building blocks running at high clock frequencies (> 500 MHz)
  - Single-kernel design implemented on Alveo U280 (running at 465 MHz and utilizing up-to 90% Bandwidth)
- Benchmarked various sparse matrices from SuiteSparse matrix collection
  - On the Alveo U280 implementation of our 16-kernel SpMV accelerator (using all HBM channels)
  - Runs at 310 MHz and achieves up-to 50 GFLOPS performance
- High performance can be achieved when the implementation is
  - Aware of device floorplan
  - Aware of communication and compute semantics
- Planning to extend this work
  - For achieving flat fmax on scaling the design size
  - For exploiting hardened floating-point DSP in Versal-HBM platforms




AMD