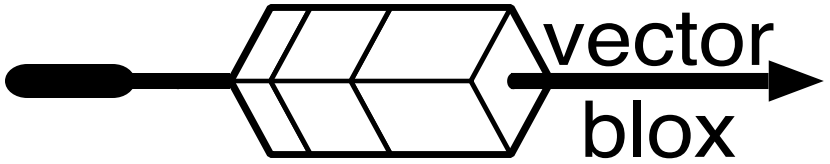
THE UNIVERSITY OF BRITISH COLUMBIA

UBC

System on a Chip research lab



VENICE: A Soft Vector Processor

Aaron Severance

Advised by Prof. Guy Lemieux

Zhiduo Liu, Chris Chou, Jason Yu,
Alex Brant, Maxime Perreault, Chris Eagleston

Motivation

- FPGAs for embedded systems
  - In use for glue logic, interfaces, etc.
  - Would like to do data processing on-chip
    - Fixed performance requirements

Motivation

- FPGAs for embedded systems
  - In use for glue logic, interfaces, etc.
  - Would like to do data processing on-chip
    - Fixed performance requirements
- Options
  - Custom hardware accelerator
  - Soft processor
  - Multiprocessor-on-FPGA
  - Synthesized accelerator

Problems…

- Custom hardware accelerator cost
  - Need hardware engineer
  - Time-consuming to design and debug
  - 1 hardware accelerator per function
- Soft processor limited performance
  - Single issue, in-order
  - 2 or 4-way superscalar/VLIW register file maps inefficiently to FPGA
  - Expensive to implement CAMs for OoOE
- Multiprocessor-on-FPGA complexity
  - Parallel programming and debugging
  - Area overhead for interconnect
  - Cache coherence, memory consistency

Problems…

- Automatically synthesized hardware accelerators
  - Change algorithm → regenerate FPGA bitstream
    - Altera C2H
    - Xilinx AutoESL
    - Mentor Graphics Catapult Synthesis
    - Forte Design Cynthesizer
  - Parallelism obscured by starting from sequential code

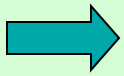
Soft Vector Processor

- Change algorithm → same RTL, just recompile software
- Simple programming model
  - Data-level parallelism, exposed to the programmer
- One hardware accelerator supports many applications
- Scalable performance and area
  - Write once, run anywhere…
  - Small FPGA: 1 ALU (smaller than Nios II/f)
  - Large FPGA: 10's to 100's of parallel ALUs
  - Parameterizable; remove unused functions to save area

Vector Processing

- Organize data as long vectors
  - Replace inner loop with vector instruction

```
for( i=0; i<N; i++ )
  a[i] = b[i] * c[i]
```

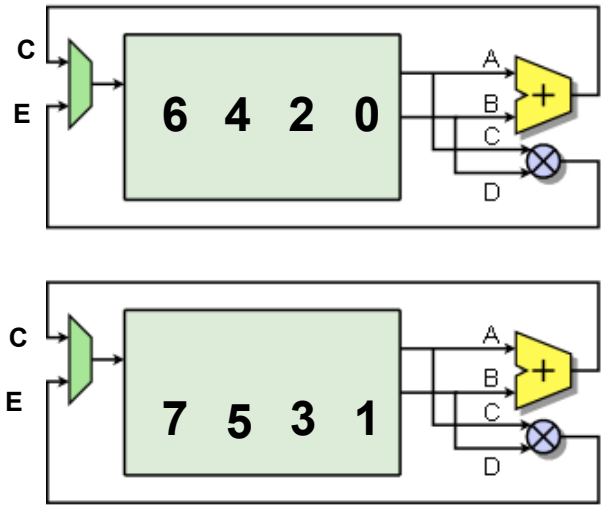


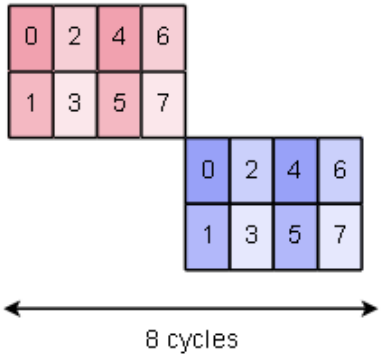
```
set vl, N
vmult  a, b, c
```

- Hardware loops until all elements are processed
  - May execute repeatedly (**sequentially**)
  - May process multiple at once (**in parallel**)

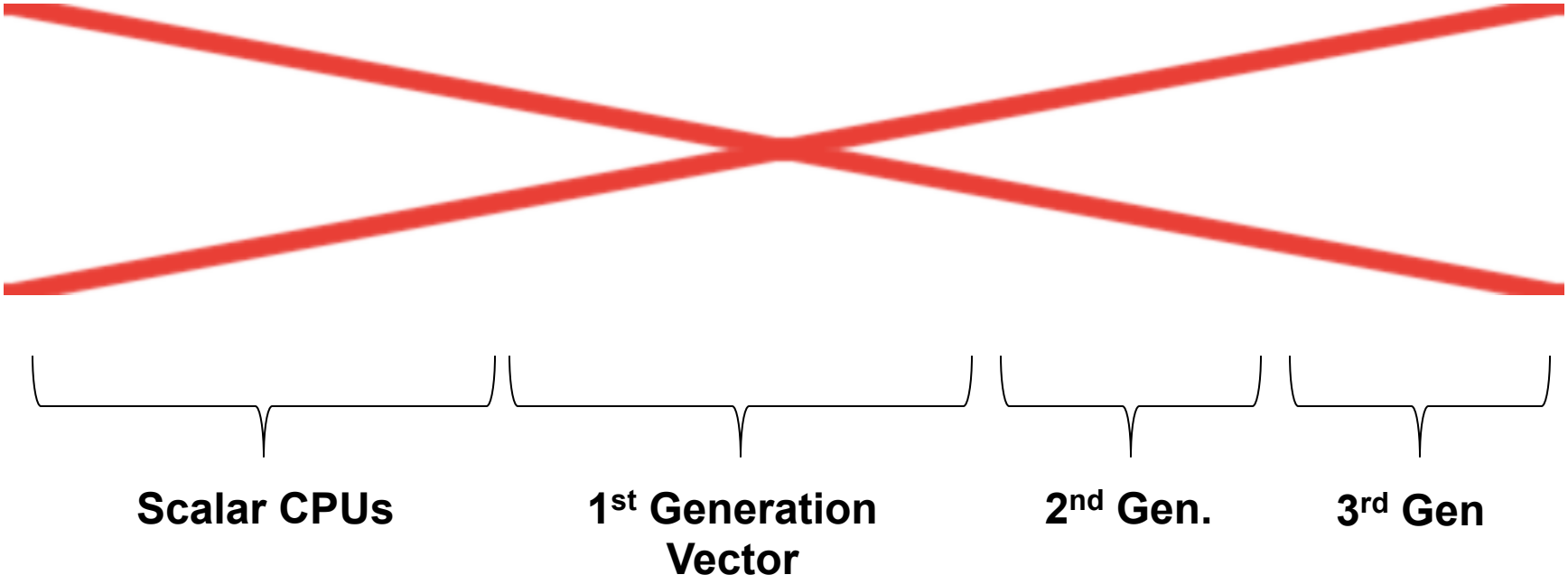
Hybrid Vector-SIMD

```
for( i=0; i<8; i++ ) {
    C[i] = A[i] + B[i]
    E[i] = C[i] * D[i]
}
```





Vector Processing on FPGAs



1st Generation SVPs

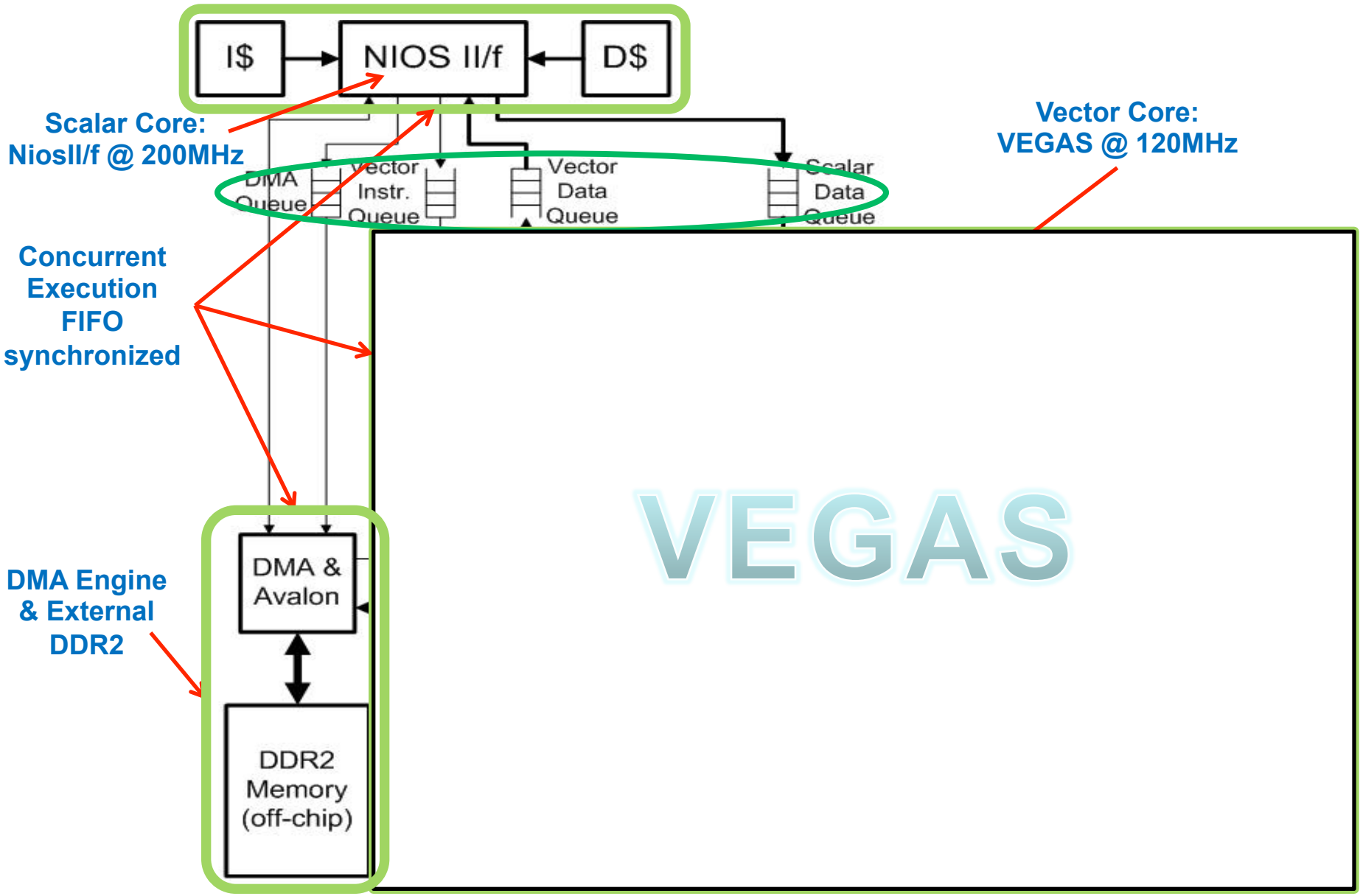
- VESPA (UofT)/VIPERS (UBC)
- Based on VIRAM (2002)
  - Vector > VLIW/Superscalar for embedded multimedia
  - Traditional load/store architecture
  - Fixed # of vector data registers
    - Maximum vector length
- Demonstrated feasibility of SVPs
  - But not specifically architected for FPGAs

VEGAS Architecture

2nd Generation

Better Utilizing On-Chip Memory

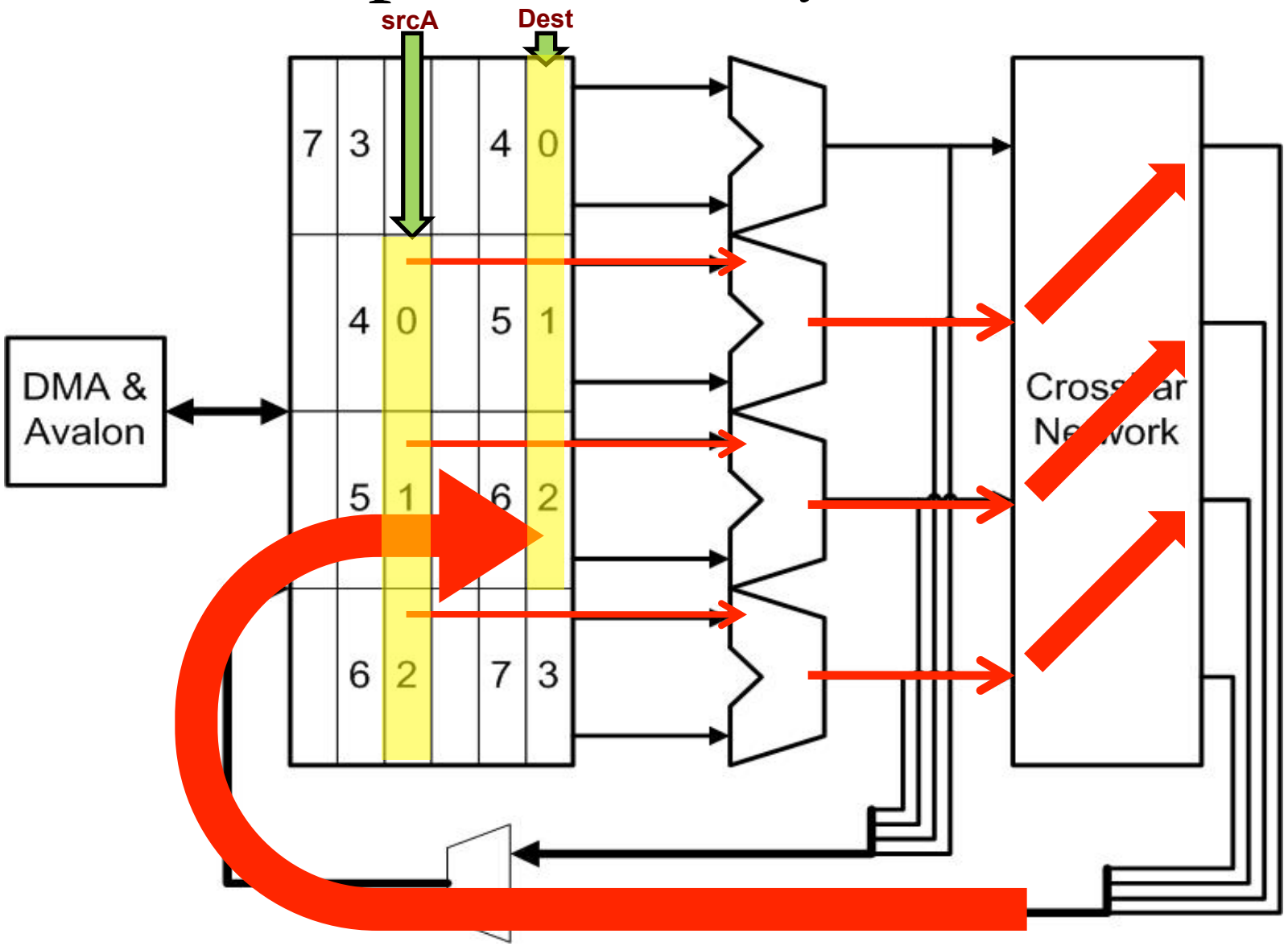
VEGAS Architecture



VEGAS: Better Utilizing On-Chip Memory

- Scratchpad-based "register file" (2kB..2MB)
  - Vector address register file stores vector locations
  - Very long vectors (no maximum length)
  - Any number of vectors
    (no set number of "vector data registers")
  - Double-pumped to achieve 4 ports
    - 2 Read, 1 Write, 1 DMA Read/Write
- Lanes support sub-word SIMD
  - 32-bit ALU configurable as 1x32, 2x16, 4x8
  - Keeps data packed in scratchpad for larger working set

Scratchpad Memory in Action



VENICE Architecture

3rd Generation
High Frequency, Low Area

VENICE Overview

Vector Extensions to NIOS Implemented Compactly and Elegantly

- 3 Key ideas borrowed from VEGAS
  - Scratchpad memory
  - Asynchronous DMA transactions
  - Sub-word SIMD (1x32, 2x16, 4x8 bit operations)

- Optimized for smaller implementations
  - VEGAS achieves best performance/area at 4-8 lanes
    - Vector programs don't scale indefinitely
    - Communications networks scale > O(N)
  - VENICE targets 1-4 lanes
    - About 50% .. 75% of the size of VEGAS
  - Vector Multiprocessor
    - N small VENICE processors > 1 big VEGAS processor ?

VENICE Overview

Vector Extensions to NIOS Implemented Compactly and Elegantly

- Removed vector address register file
  - Address stored in scalar processor's registers
  - Reduces area/control overhead
  - Now 2 cycle instruction dispatch though
- In pipeline vector alignment network
  - Much faster for convolutions
  - Low overhead for small number of lanes
- Single clock domain with master CPU
  - Deeper pipelining (registers cheap on FPGAs)
  - Reaches 200MHz, ~50% faster than previous SVPs

VENICE Overview

Vector Extensions to NIOS Implemented Compactly and Elegantly

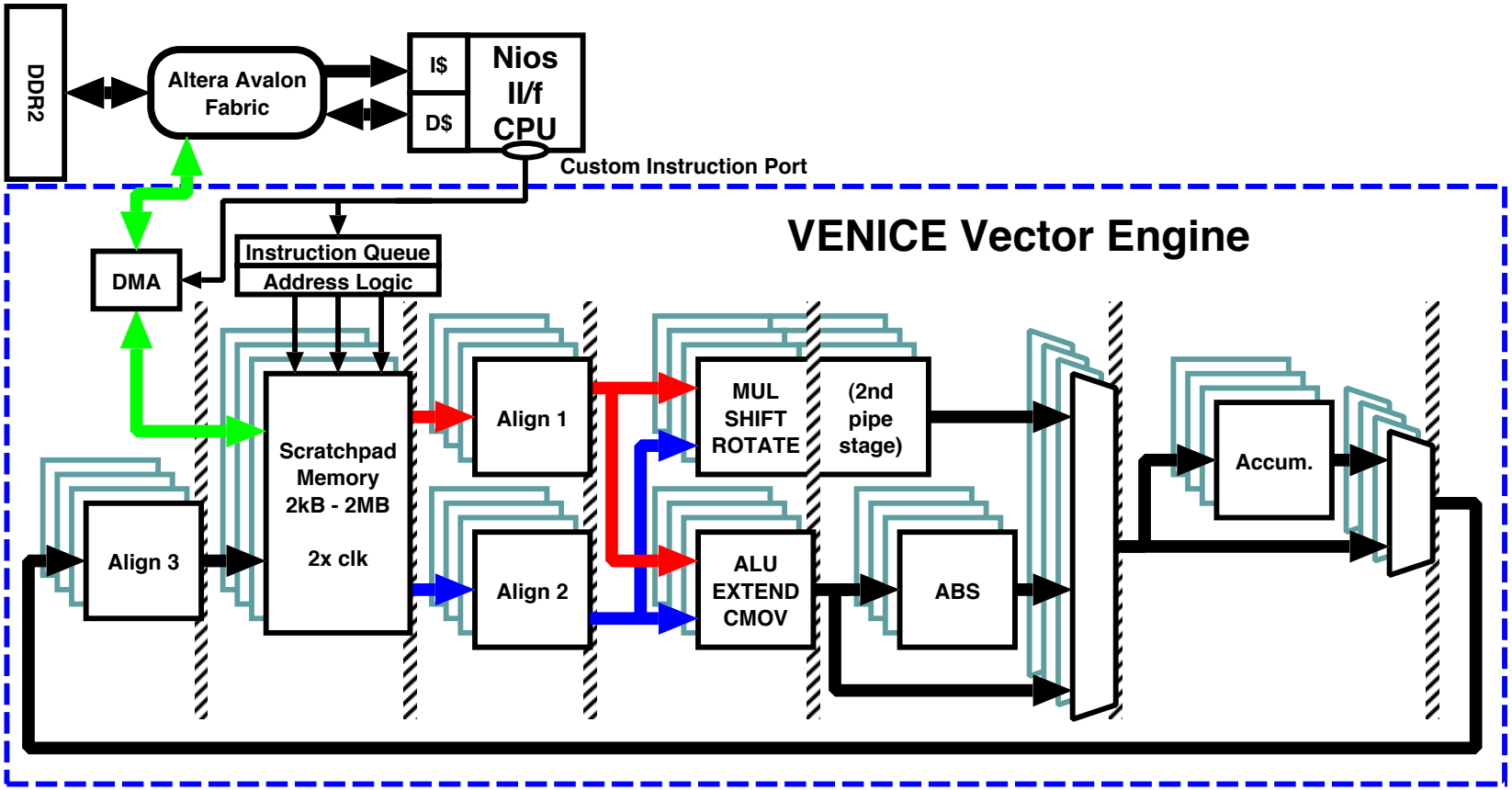
- Programming changes
  - 2D/3D vectors for multimedia/linear algebra/etc.
    - Repeated vector instruction with separate strides for srcA, srcB, and Dest
    - Reduces instruction dispatch bandwidth requirements
  - C pointers used to index into scratchpad
    - VEGAS (old):

```
vegas_set( VADDR, V1, pBufSrc1 );
vegas_set( VADDR, V2, pBufSrc2 );
vegas_set( VADDR, V4, pBufDest );
vegas_vvw( VADD, V4, V1, V2 );      // V4 = V1 + V2
```

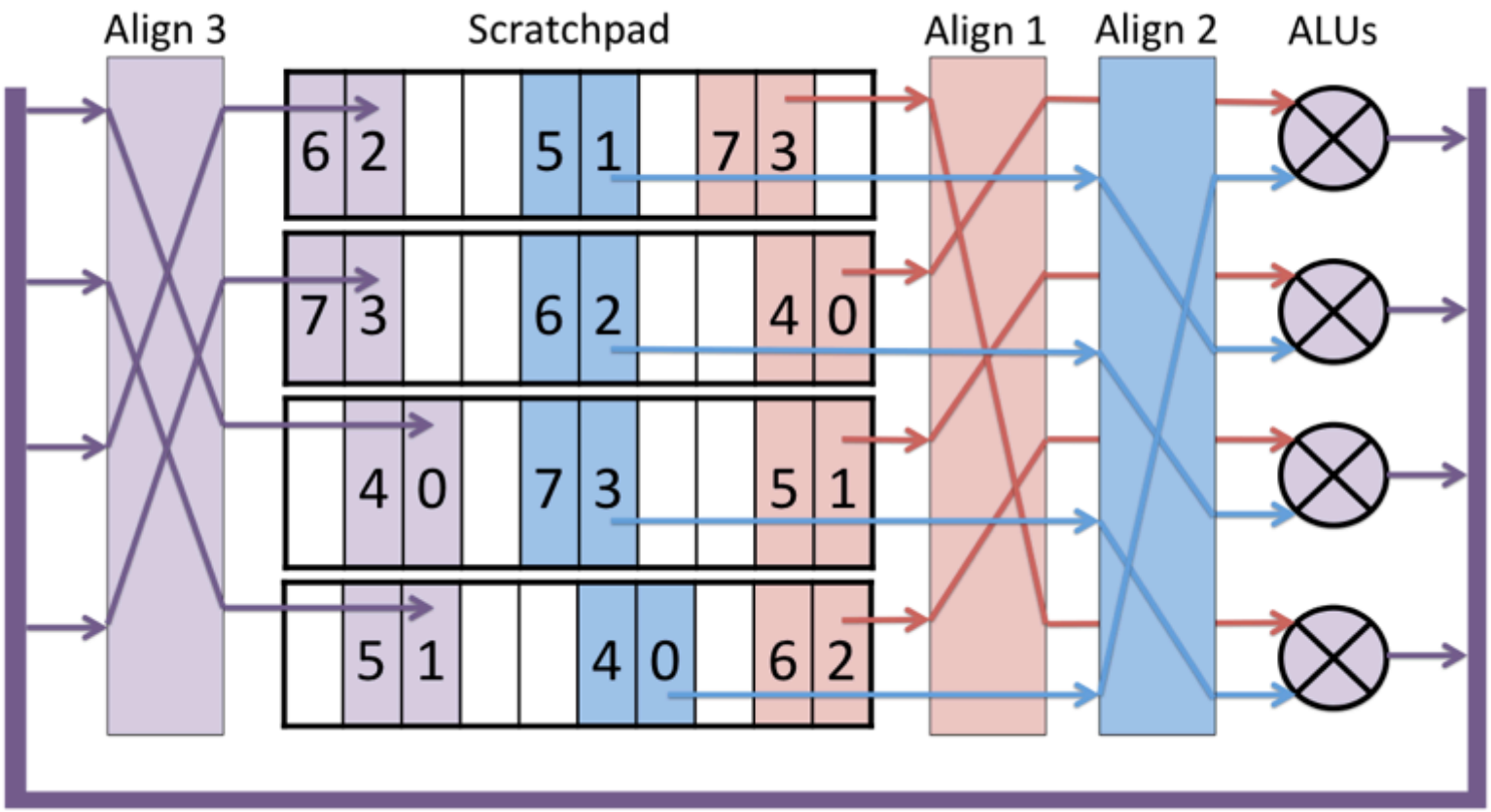
    - VENICE (new):

```
vector( VVW, VADD, pBufDest, pBufSrc1, pBufSrc2 );
```

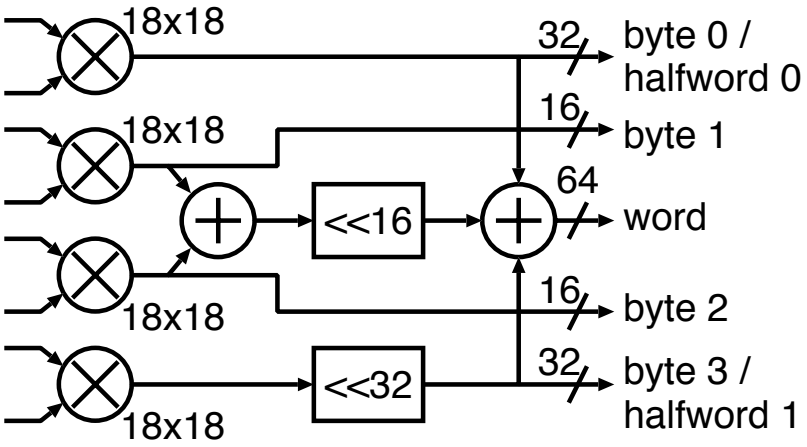
VENICE Architecture



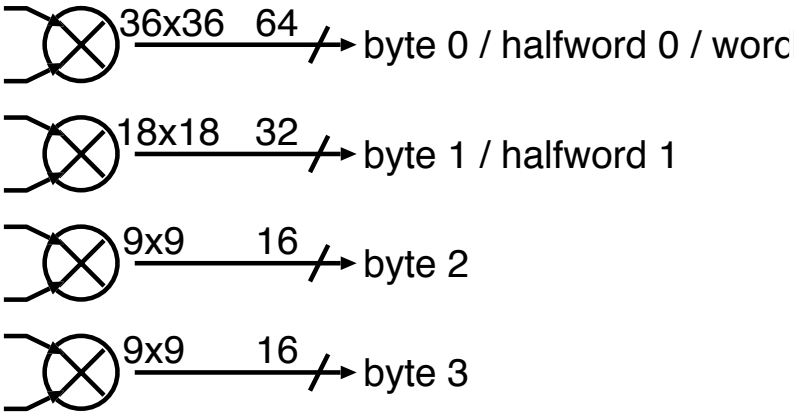
VENICE Scratchpad Alignment



VENICE Fracturable Multiplier



VEGAS Fracturable Multiplier



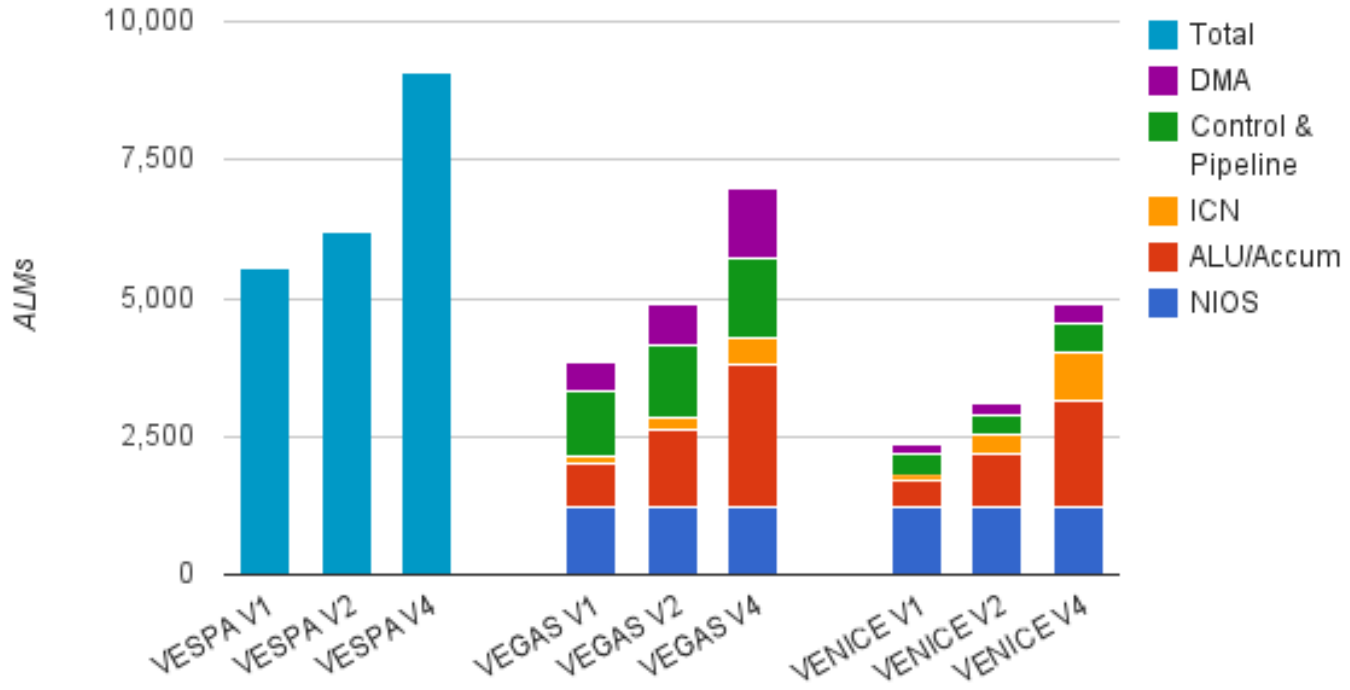
VENICE Fracturable Multiplier

2 DSP Blocks + extra logic

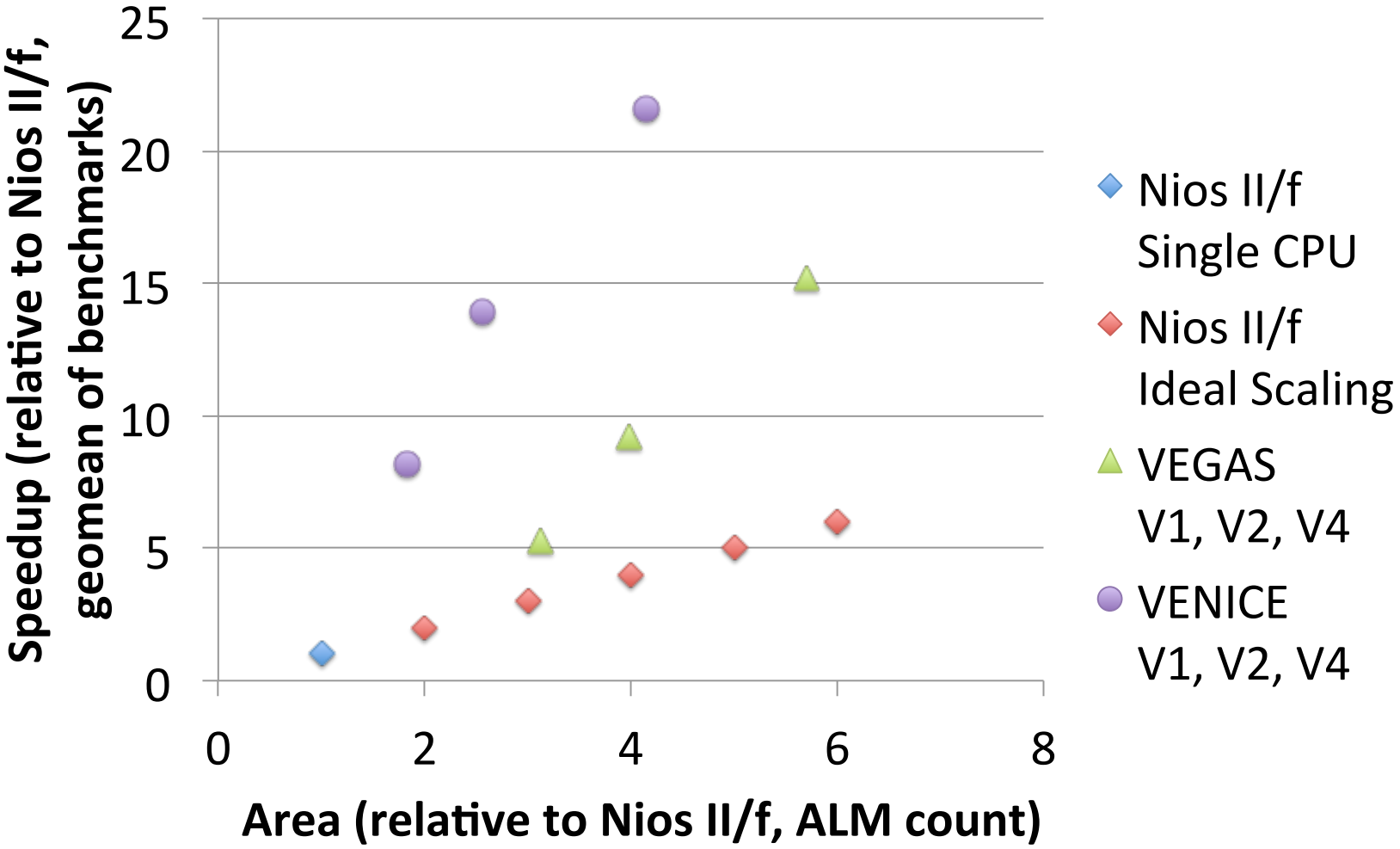
2 DSP Blocks (no extra logic)

Area Breakdown

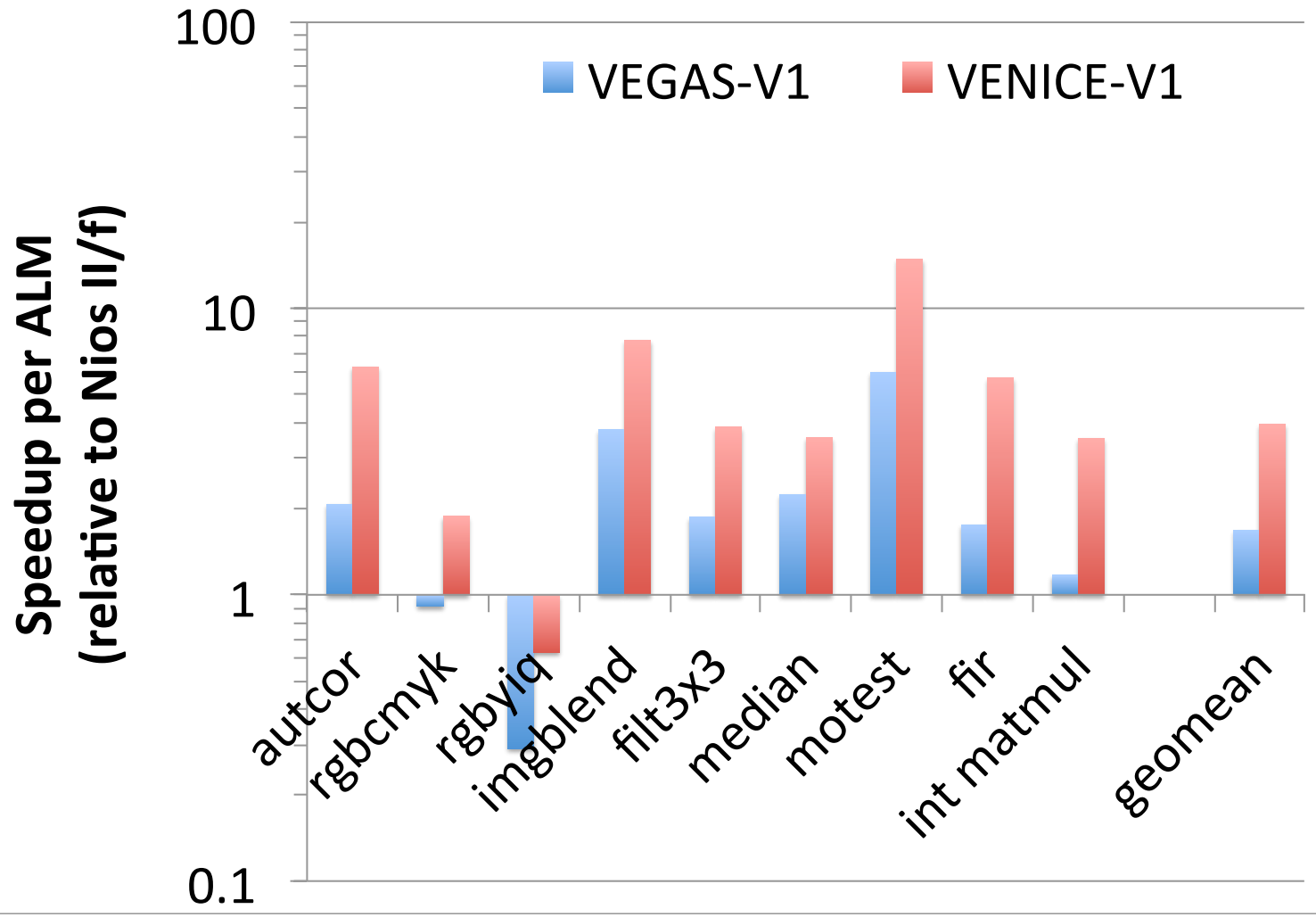
VENICE area greatly reduced.



VENICE Average Speedup vs. ALMs

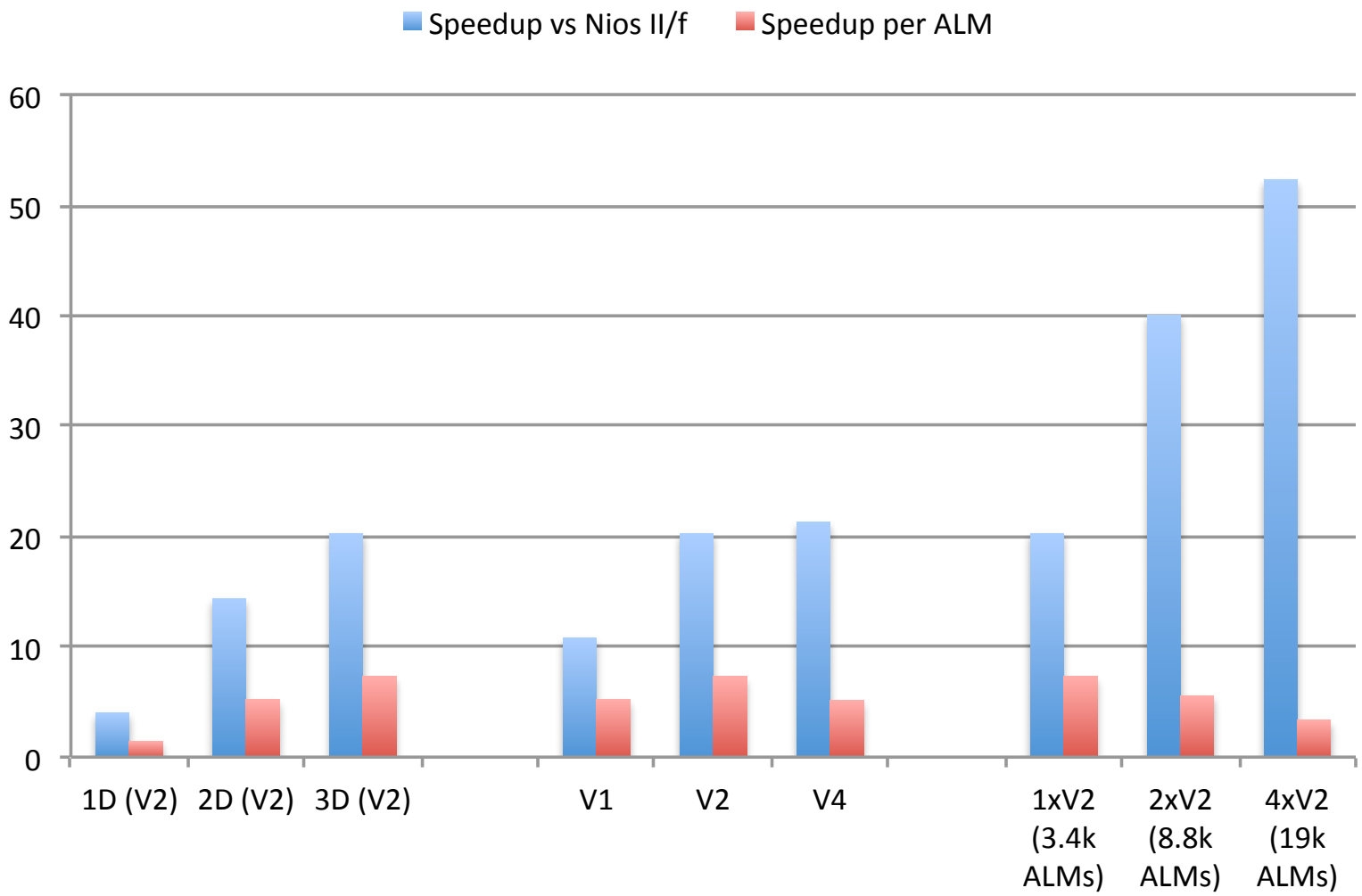


VENICE Computational Density



VENICE: Multiprocessor

16-bit 4x4 DCT



Conclusions

- Soft Vector Processors
  - Scalable performance
  - No hardware recompiling necessary
- VENICE
  - Optimized for FPGAs, 1 to 4 lanes
  - 5X Performance/Area of Nios II/f
- Future work
  - Hybrid vector/thread architectures
  - Commercial version (VectorBlox Computing)