Optimization through customization

One size doesn’t fit all

Maya Gokhale
Computing, CASC

April 25, 2024

LLNL-PRES-863073
This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract No. DE-AC52-07NA27344.

Lawrence Livermore National Laboratory

General purpose vs purpose built

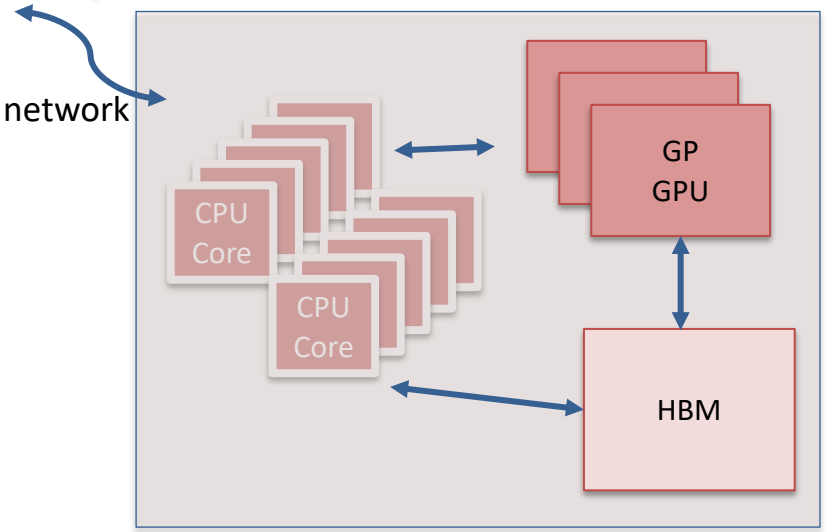
- For HPC, business as usual has been to adopt commercial architectures
  - Invest in specific components such as low latency, high bandwidth interconnection network
  - Influence architectural direction and timeline with NRE investments to vendors
  - Continue to pursue code refactor and rewrite
    - From vector to distributed memory parallel to GPU offload … to AI engines?
- How will business not as usual be achieved?
- Is it possible to “Develop purpose-built, advanced architectures that define new, perhaps disruptive, hardware designs?”
- Project 38 https://www.nitrd.gov/documents/HPC-Performance-Improvements-Project-38.pdf posed that problem

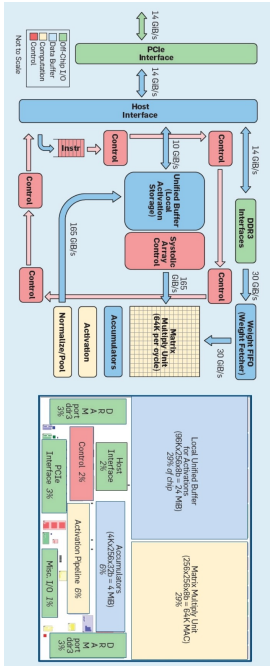
Landscape of choices

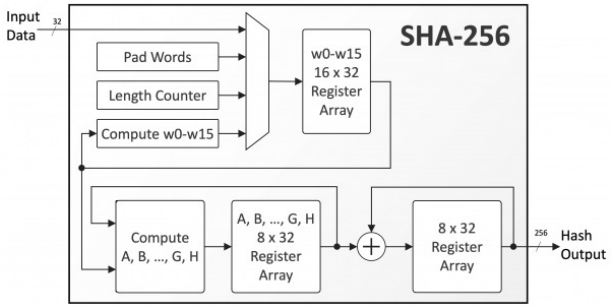
Business as usual:

Data center, cloud, HPC — AI/ML — Security

Fully Programmable — **Domain Specific** — **Fixed Function**







https://www.cast-inc.com/security/encryption-primitives/sha-256

TPU: source nextplatform.com

Outline

- Customize hardware with new instructions
  - Memory-centric accelerators
  - Scientific data compression accelerator
- Challenges
  - Tool chain
  - verification

Customize with specialized hardware blocks

- Standard operating procedure for large volume uses
  - Hash units
  - Compression
  - Encryption
  - AI matmul in low precision
- Can it work for HPC?
  - Important scientific kernels and applications are intensively analyzed for performance improvement opportunities, so that “compute” intensive regions are known
    - Dense and sparse matmul, mat-vec operations
    - FFT
    - **Programmable Gather/Scatter Engine, K/V lookup accerator**
    - **Floating point compression for ZFP (fixed rate)**

Memory-centric accelerators

- Data movement is necessary – but only move necessary data
- Near memory programmable gather/scatter engine “Data Rearrangement Engine (DRE)
  - Batch operation
  - Indexed A[B[i]]
  - Strided A[i+c]
- Key/Value Store query accelerator
  - Gather values for batch of keys

S. Lloyd and M. Gokhale, “Near memory key/value lookup acceleration,” International Symposium on Memory Systems MEMSYS17, 2017.

J. Landgraf, S. Lloyd, and M. Gokhale, “Combining emulation and simulation to evaluate a near memory key/value lookup accelerator,” arxiv.org/abs/2105.06594, 2021.

Maya Gokhale, Scott Lloyd, and Chris Hajas. 2015. Near memory data structure rearrangement. In Proceedings of the 2015 International Symposium on Memory Systems (MEMSYS '15). Association for Computing Machinery, New York, NY, USA, 283–290. DOI:https://doi.org/10.1145/2818950.2818986

A. K. Jain, S. Lloyd and M. Gokhale, "Performance Assessment of Emerging Memories Through FPGA Emulation," in IEEE Micro, vol. 39, no. 1, pp. 8-16, Jan.-Feb. 2019, doi: 10.1109/MM.2018.2877291.

G. Scott Lloyd and Maya Gokhale. 2017. Near memory key/value lookup acceleration. In Proceedings of the 2017 International Symposium on Memory Systems (MEMSYS ‘17). Association for Computing Machinery, New York, NY, USA, 26-33. https://doi.org/10.1145/3132402.3132434

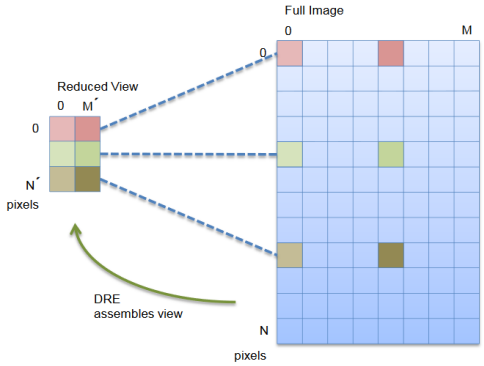
S. Lloyd and M. Gokhale, “In-memory data rearrangement for irregular, data intensive computing,” IEEE Computer, pp. 18–25, 2015.

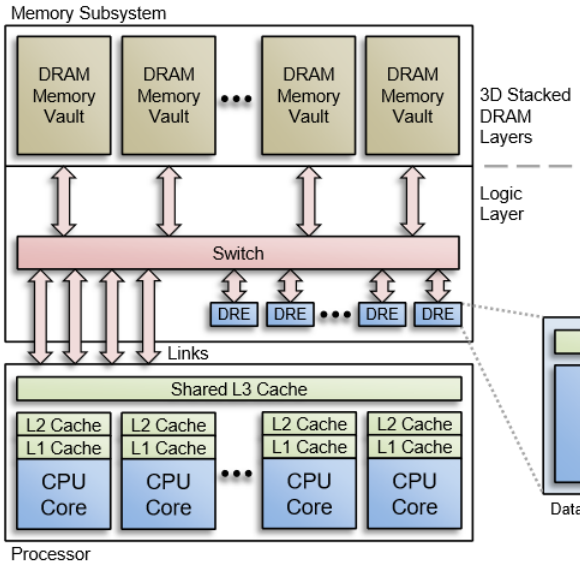
Near-memory data reorganization engine
Patent number 9965187

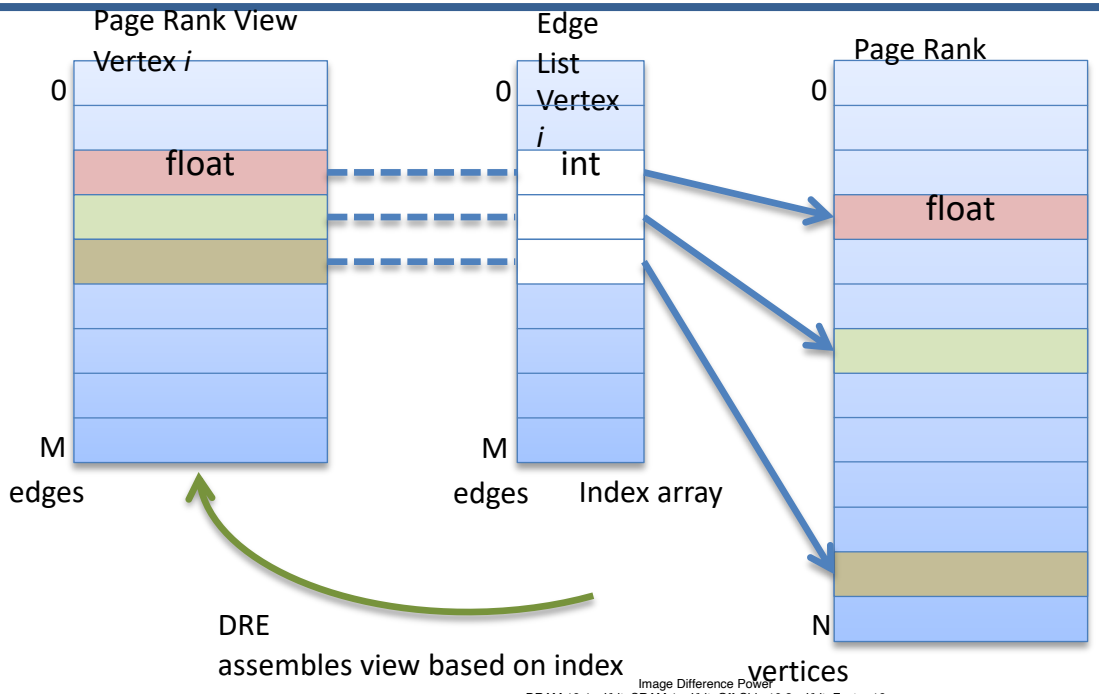
Near memory gather/scatter can help applications with irregular access patterns

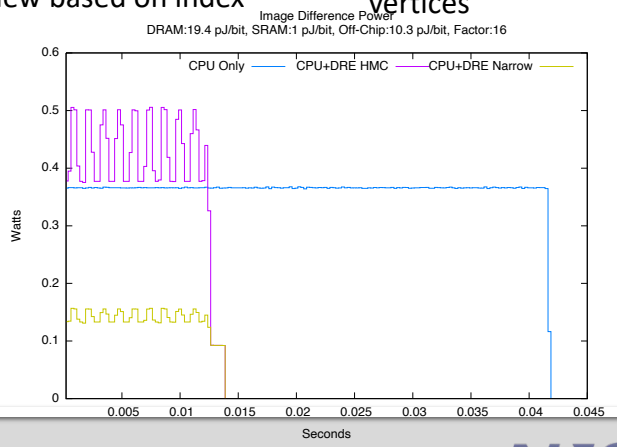
- Memory bandwidth to processors increasing
  - HBM channels with wide access amount benefit sequential, predictable load/store
  - Large caches and more memory channels may help some applications
  - **BUT irregular access such as A[B[i]] impose latency penalty and force random access and can't use the increased bandwidth effectively**
- Programmable gather/scatter hardware can help
  - Operate on a batch of indices
  - Gather a dense "view" into scratchpad
  - Application code can vectorize the dense representation

Data Rearrangement Engine (DRE)

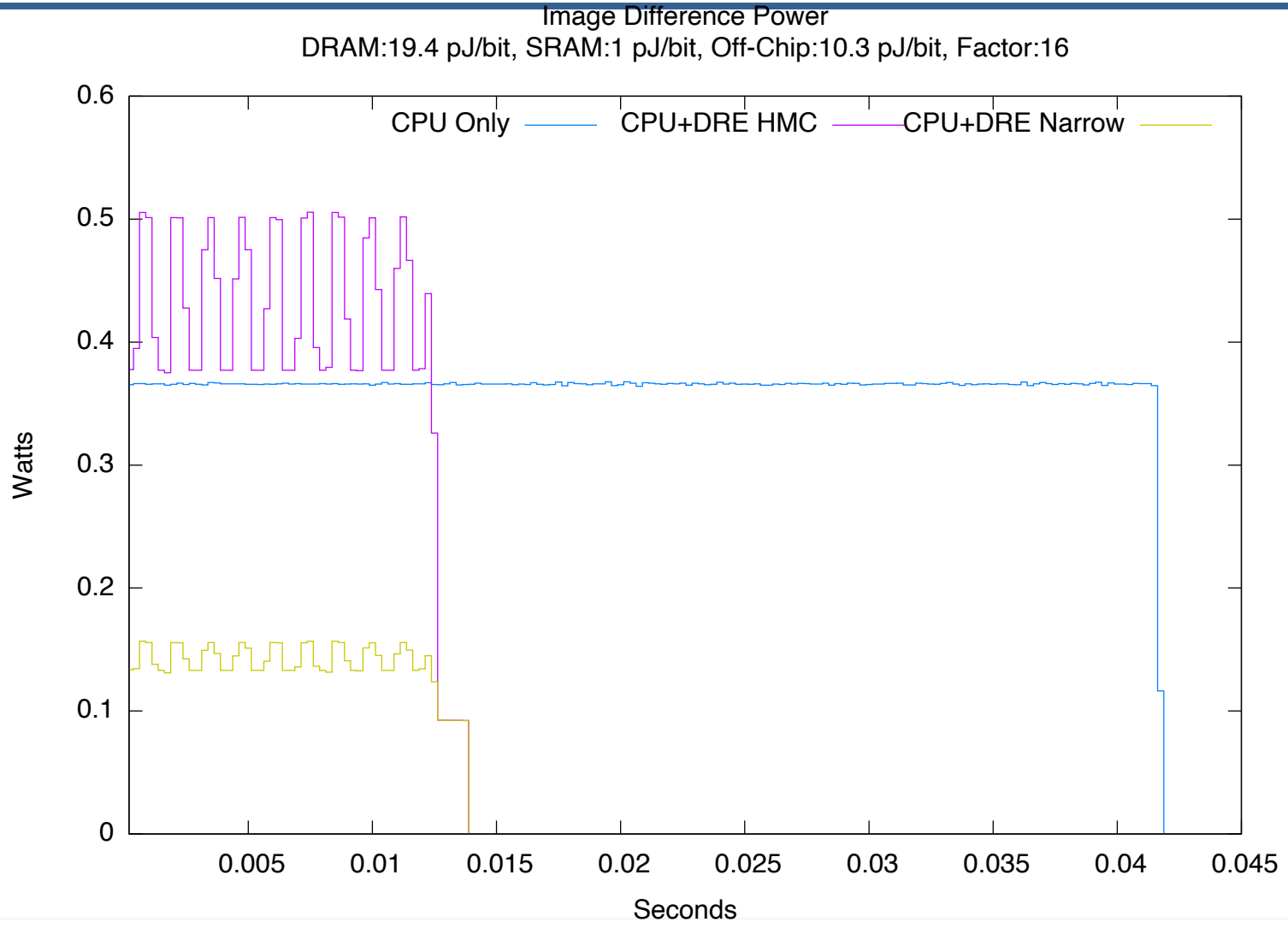








Execution trace in LiME FPGA-based emulator



API

setup — Specify the location and size of application data structures and other parameters for gather/scatter

```
/* ImageDiff: Specify image location, dimensions, and decimation factor */
void setup(void *ref, size_t ref_width, size_t ref_height, size_t elem_sz, size_t decimate);
/* PageRank, RandomAccess, SpMV: Specify reference table and index array */
void setup(void *ref, size_t elem_sz, const void *index, size_t len);
```

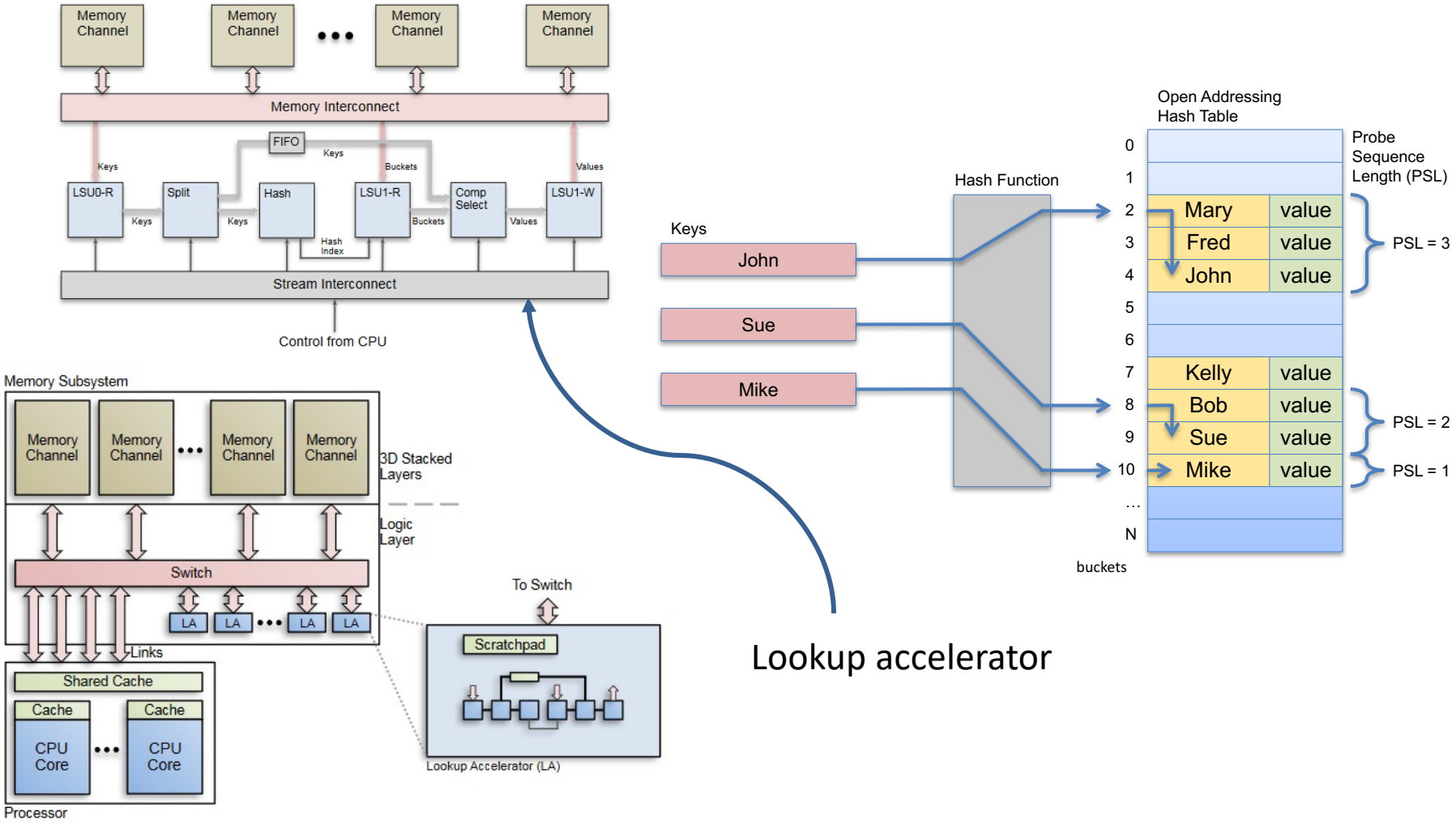
fill — Copy from DRAM to the view buffer according to the access pattern established during setup

```
/* Specify view buffer and window offset */
void fill(void *buf, size_t buf_sz, size_t offset);
```

drain — Copy from the view buffer into DRAM according to the access pattern established during setup

```
/* Specify view buffer and window offset */
void drain(void *buf, size_t buf_sz, size_t offset);
```

Key/Value Store Lookup Accelerator



Experiment Design

- Key/value table is filled with a scientific data set consisting of k-length genomic sequences (k-mers)
- 32 million entry table is allocated at first and filled to varying degrees
- Table entries consist of k-mers (64-bit keys) and sequence numbers (32-bit values)

Parameters	Values
Load factor	10%–90%
Hit ratio	10%, 50%, 90%
Key repeat frequency	Uniform, Zipf
Memory Latency (ns)	85R/106W, 200R/400W
Query block size	1024 keys

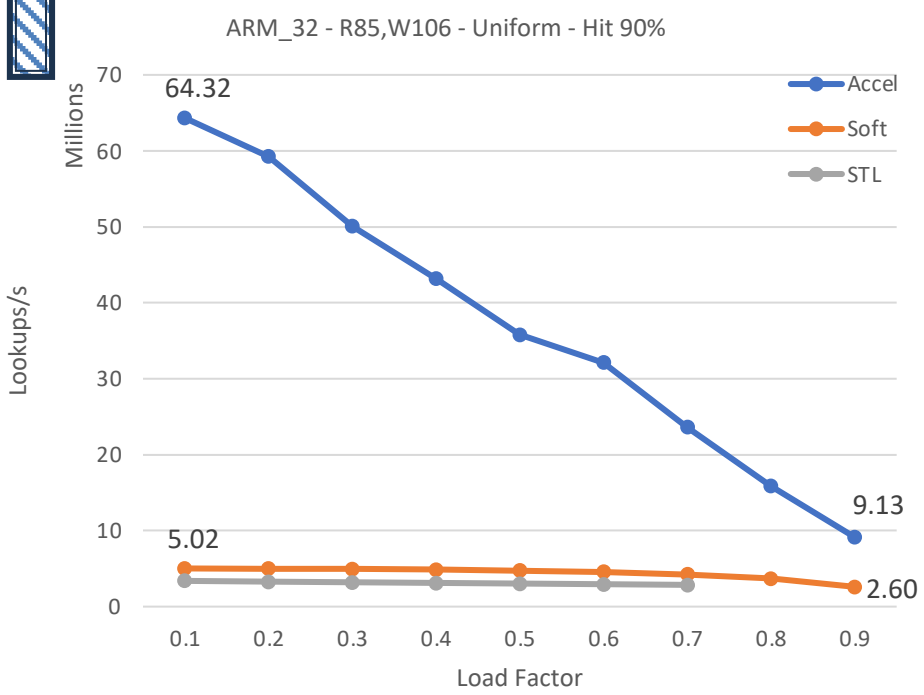
Lookup Algorithms Evaluated

- Accel
  - Near memory hardware lookup accelerator
  - Collision resolution: open addressing and Robin Hood hashing
  - Hash function: adapted from SpookyHash
  - Lookup uses linear probing
- Soft
  - Software version of the hardware lookup algorithm
  - Collision resolution: same as Accel
  - Hash function: same as Accel
  - Unlike the hardware, the software algorithm terminates probe sequence search as soon as a key has been found
- STL
  - Hash table uses the Standard Template Library (STL) unordered map
  - Collision resolution: separate chaining with linked lists
  - Hash function: simple

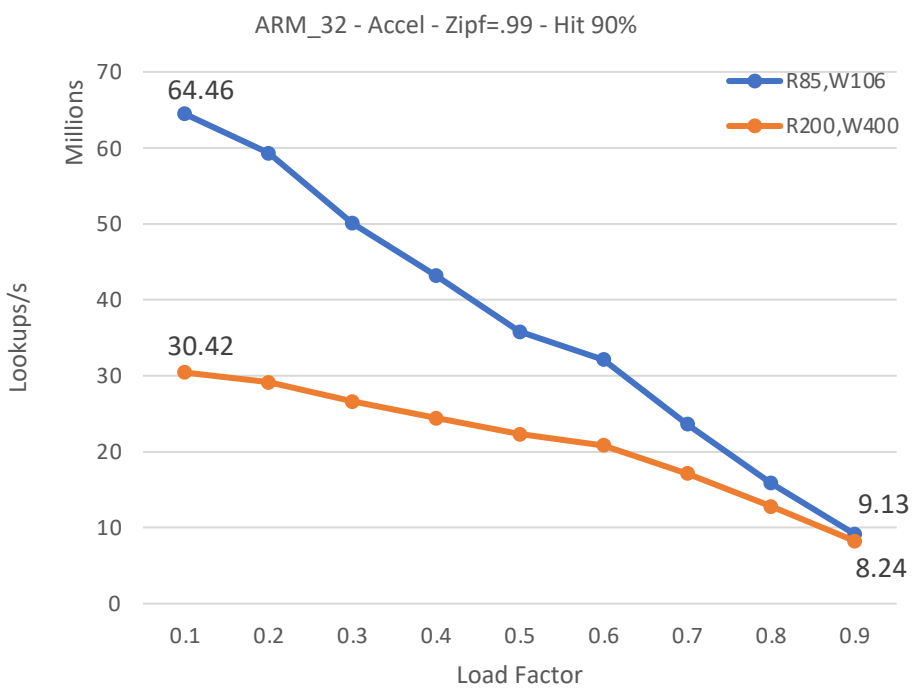
Lookup Performance

90% hit rate

Accelerator vs. Software



Low vs. Moderate Latency

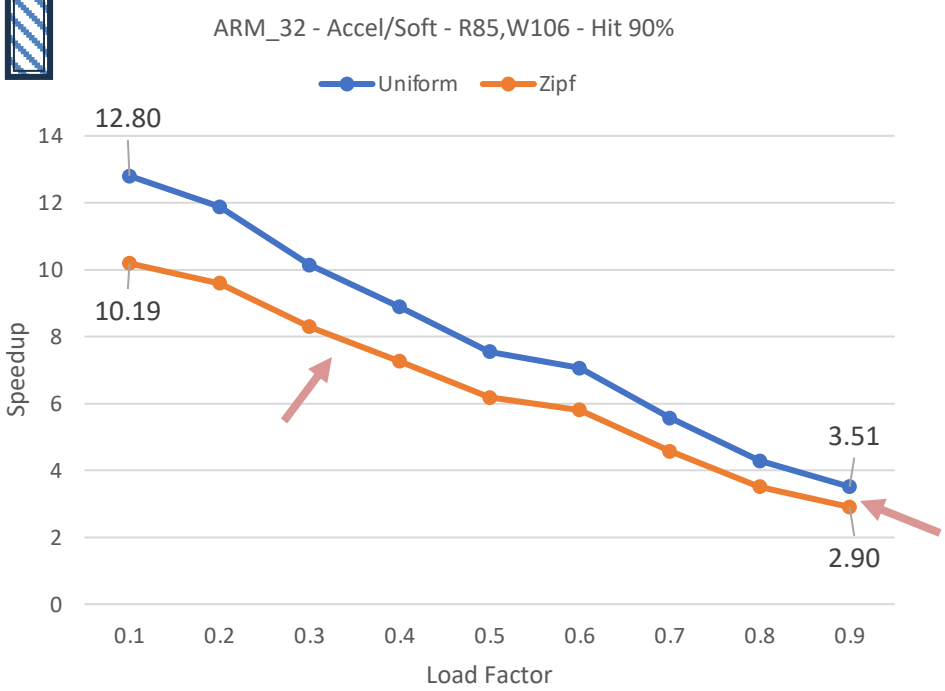


- Accel. performance does not vary with hit rate or key repeat frequency (scans entire PSL)
- Accel. performance decreases with increasing load (PSL) and memory latency
- Accel. performance comes from parallelism and more outstanding near memory requests
- Software is slower because of serialization and fewer outstanding far memory requests

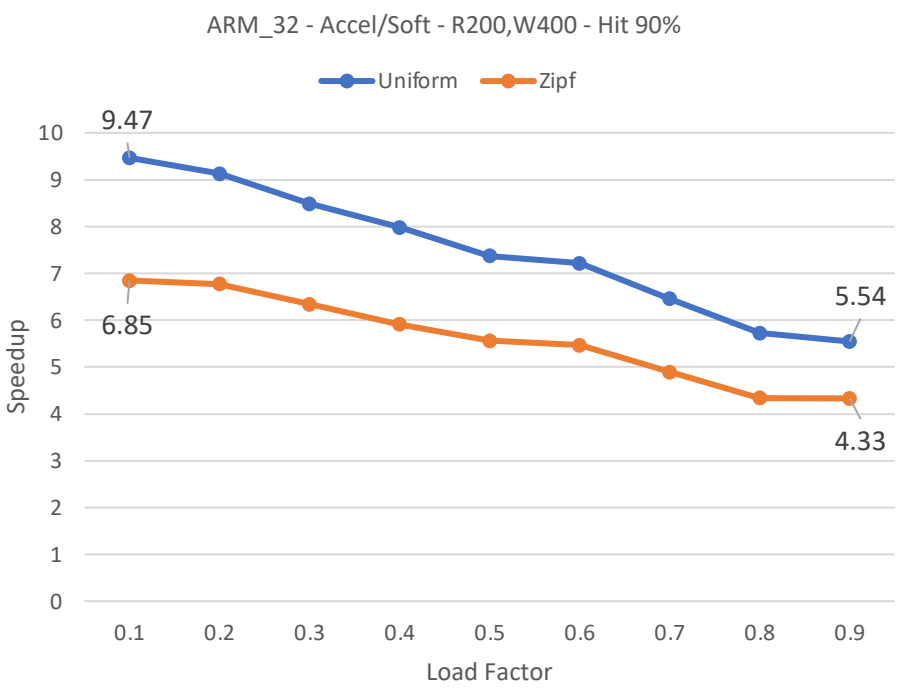
Speedup of Uniform and Zipfian Key Distributions

90% hit rate

Low Latency (DRAM)



Moderate Latency (SCM)

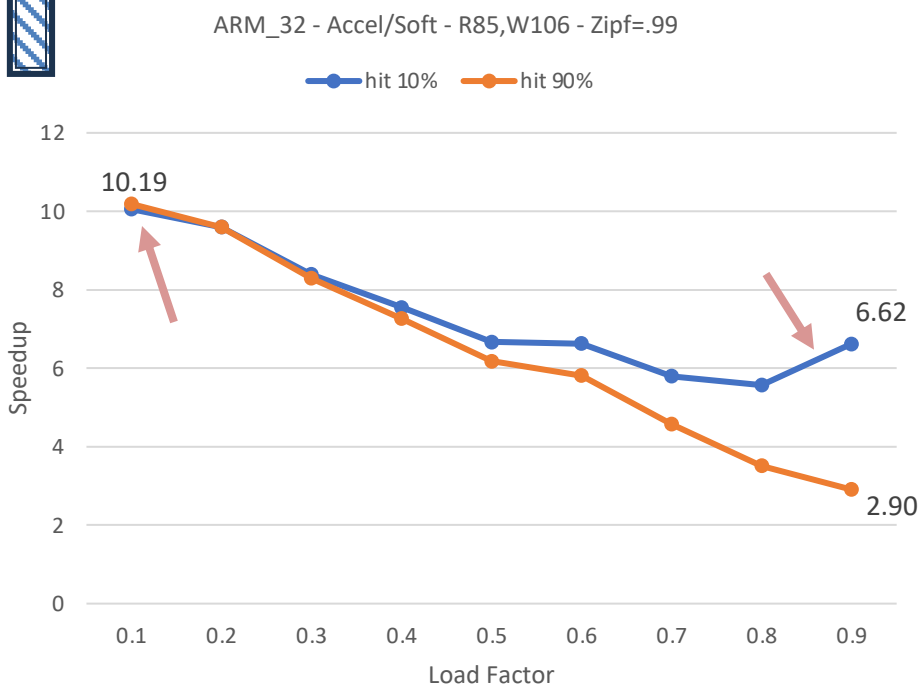


- Zipfian has less speedup because software has more query hits in CPU cache (lower)
- At higher load factors, the software is disadvantaged with more cache misses (convergence)

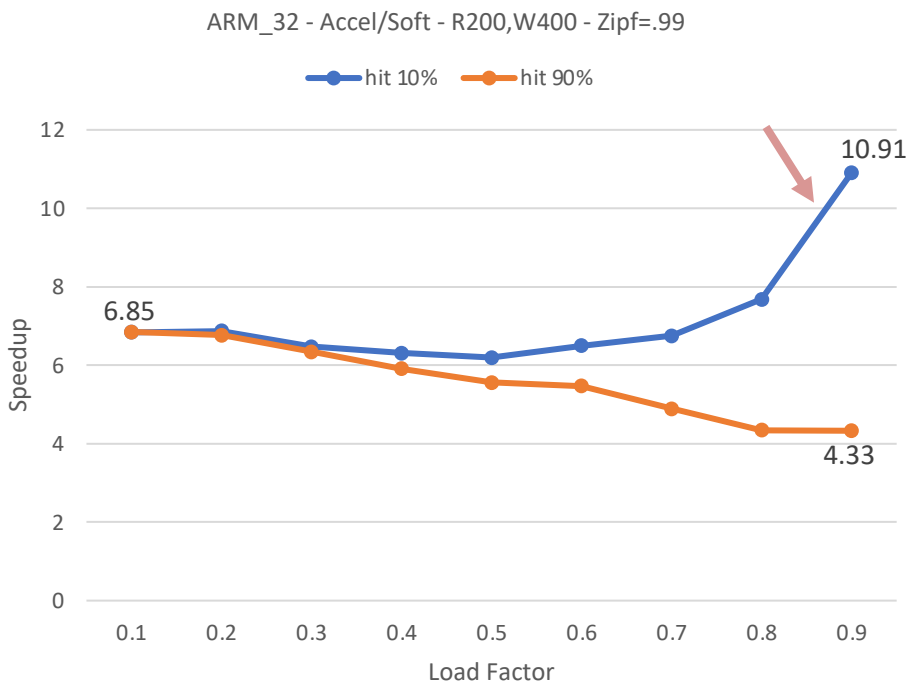
Speedup of 10% and 90% Hit Rate

Zipf skew factor 0.99

Low Latency (DRAM)



Moderate Latency (SCM)



- Hit rate does not affect speedup at low load factors since probe sequence is short
- Software is challenged on longer searches (low hit, high load) with more sequential memory accesses
- Higher latency pushes the trend even more

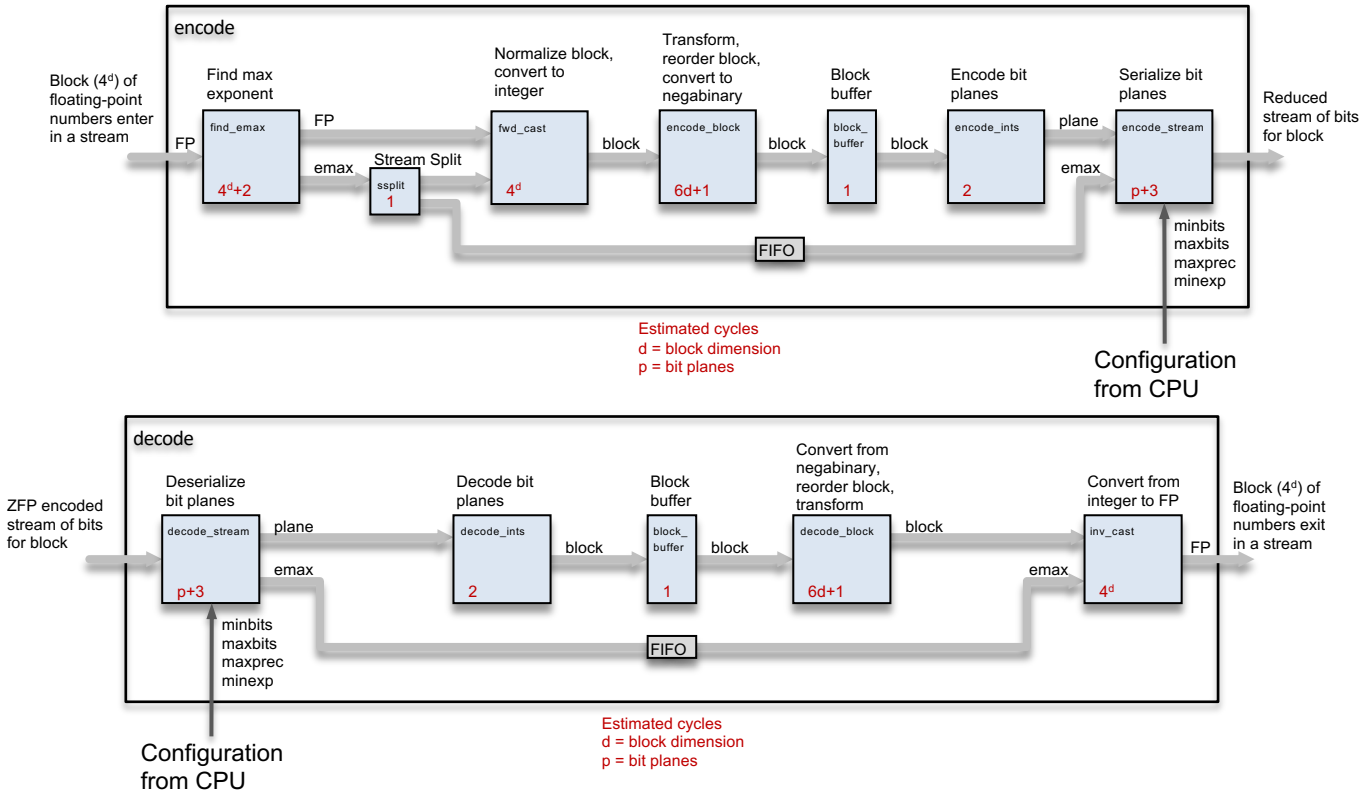
Key points

- Interface matters
  - Blocks of requests/responses to enable efficient pipelining
- Evaluation matters
  - Including a hardware IP block is a big investment
  - Under what conditions will it be worthwhile?
  - FPGA-based emulator was a big investment in time
    - Fast
    - High visibility
    - But …
  - Combining with software SST simulator gave new insights and adjustments to the design

https://github.com/llnl/lime

https://github.com/LLNL/lime-apps

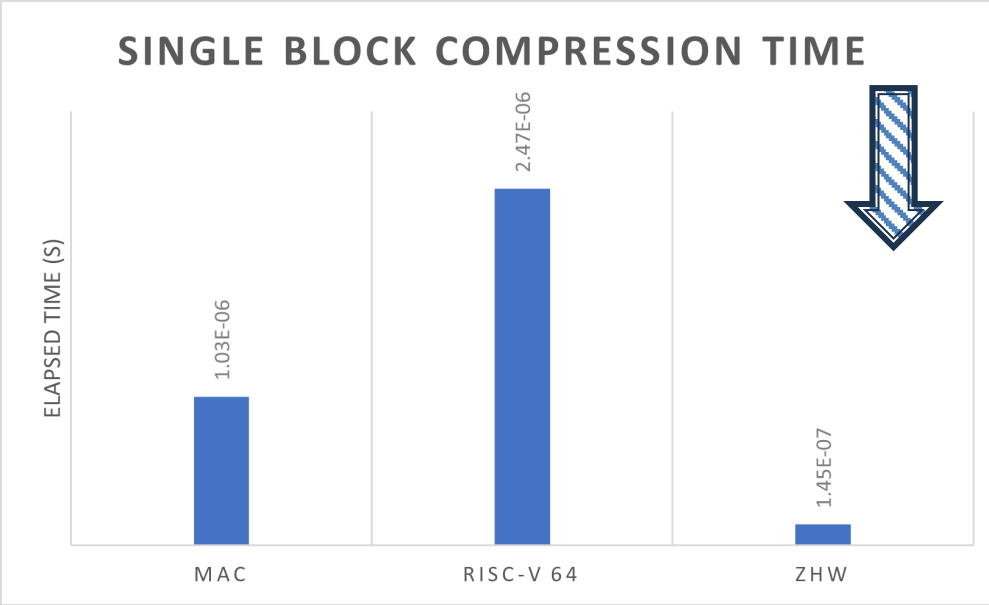
ZHW: hardware ZFP compression pipeline for floating point arrays

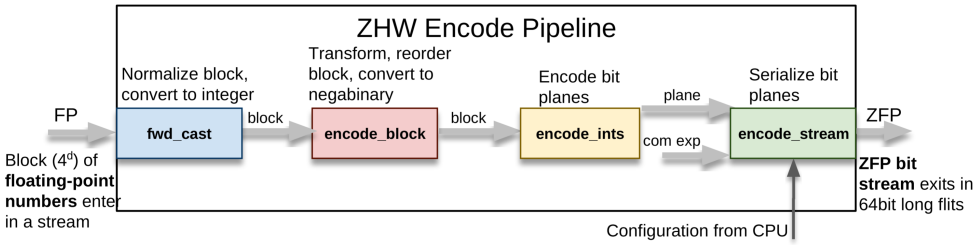


M. Barrow, Z. Wu, S. Lloyd, M. Gokhale, H. Patel, and P. Lindstrom, "Zhw: A numerical codec for big data scientific computation," Field Programmable Technology Conference (FPT '22), December 2022.

ZHW as an IP block

- Encoder is synthesizable at 550MHz
- 7X-17X speedup compared to Mac laptop and RISC-V64
  - Performance comparison of encoder only
  - What about integration with CPU cores in SoC?

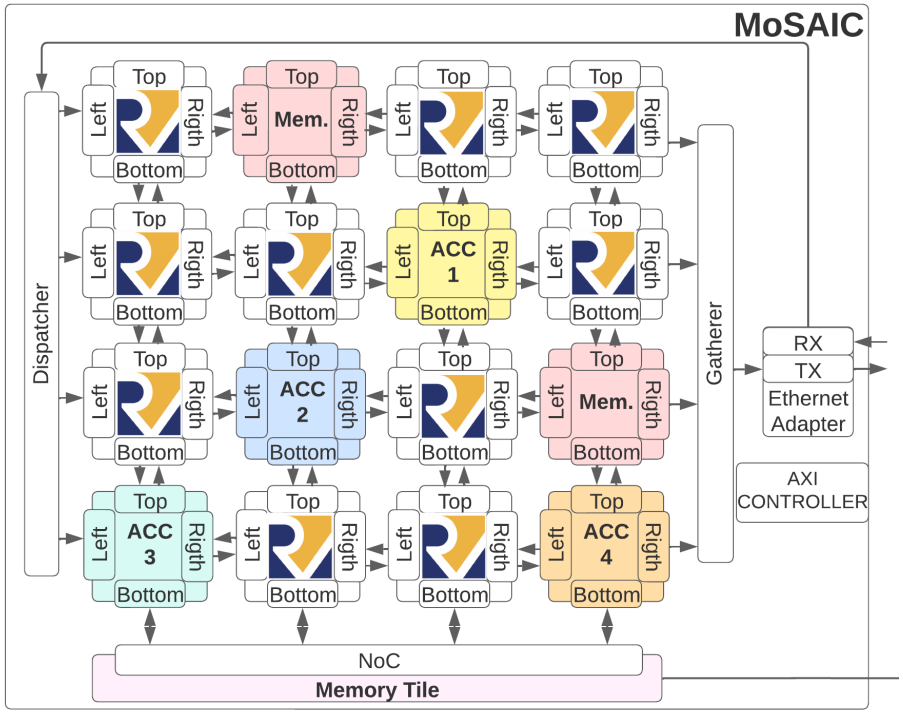




X. Liu, P. Gonzalez-Guerrero, I. B. Peng, R. Minnich, and M. B. Gokhale, "Accelerator integration in a tile-based soc: lessons learned with a hardware floating point compression engine," SC-W '23: Proceedings of the SC '23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis, 2023.

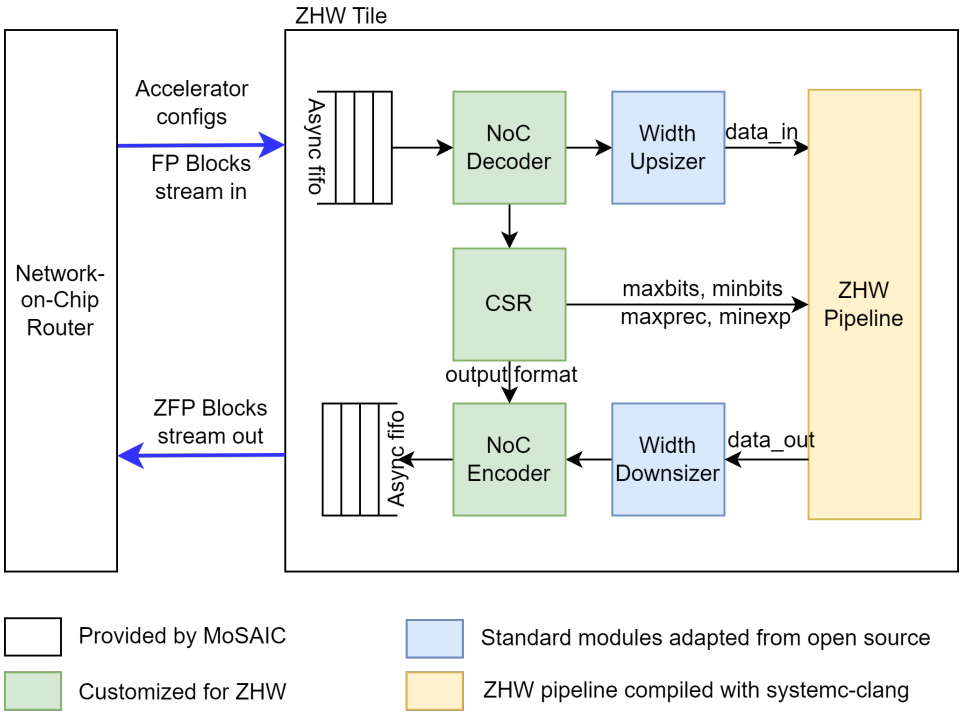
LBNL Mosaic SoC

- Tile architecture, heterogeneous and configurable
  - HW: RSIC-V CPU, scratchpad, NoC
  - SW: RISC-V tool chain with customized NoC message protocols
  - Configuration: tools to generate different tile sizes and layouts
- Full implementation in Verilog RTL, testing framework with FPGA



ZHW as an accelerator tile on NoC

- Connect ZHW RTL with standard NoC interface
  - NoC buffer to convert clock frequency in different domains
  - NoC decoder/encoder handles header metadata and transfers data to/from accelerator (header: input command, output destination and command)
  - Width converter (NoC data width is 32bits and ZHW is 64 bits)
- Control&Status Register programming
  - Accelerator config (maxbits, minbits)
  - Output NoC routing information (dest, op, size)



Accelerator Software Programming – CPU centric method

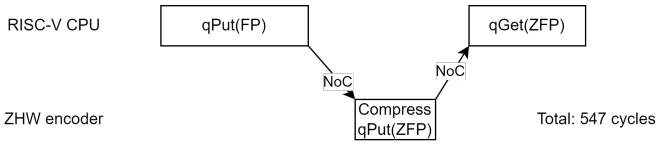
- Utilizing existing MoSAIC APIs with custom RISC-V instructions
- mPut/mGet – address-based communication
  - Originally designed to communicate with scratchpad tile
  - Used to configure CSRs in ZHW
- qPut/qGet – message queue bypass CPU cache/memory hierarchy
  - Use software for loop to send/retrieve data

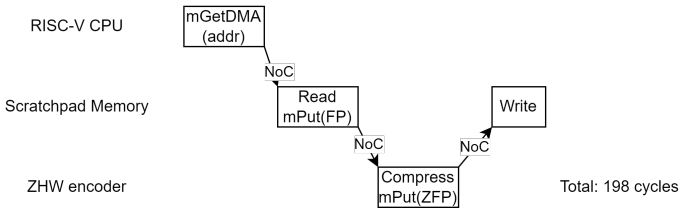
```
27 int rd_zhw(int tile_id, int *data, int size) {
28   int temp;
29   int received_data_poll;
30   qPoll(tile_id, received_data_poll);
31   /* Check if queue is empty */
32   if (received_data_poll == 1){
33     return -1;
34   }else{
35     /* Pop headers from queue */
36     qGet(tile_id, temp);
37     qGet(tile_id, temp);
38     /* Pop data from queue */
39     for (int i=0; i<size; i++)
40       qGet(tile_id, data[i]);
41     return 1;
42   }
43   return 1;
44 }
```

Scratchpad DMA optimization

- CPU-centric approach incurs significant CPU instruction overhead (80% for single block)
- Optimization: Utilize stand-alone scratchpad tile to transfer data
  - Create new instruction: mGetDMA
  - Revise scratchpad logic to send output to assigned memory tile/address
  - Reduce number of CPU instructions
  - **Offload memory operations to scratchpad**

```
8  /* Retrieve data using SPAD DMA, data preloaded in
      spad2 */
9  mGetH((spad2_tile << 12), TEST_SIZE_LOG2);
10 mGetDMA((zhw_tile << 12) + ZHW_STREAM_ADDR, 0); // send
       to dedicated zhw address for input data streaming
```

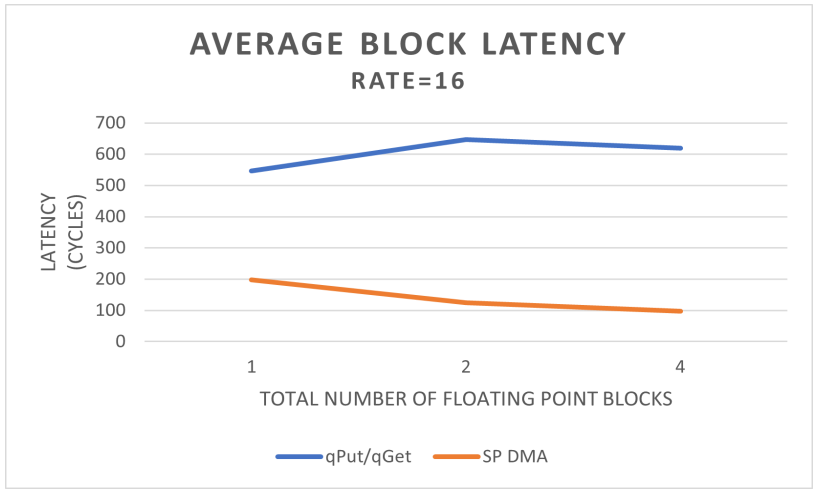


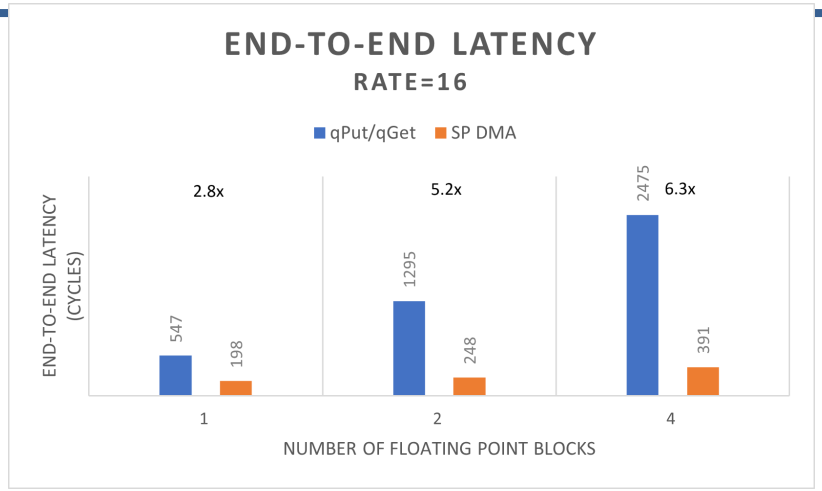


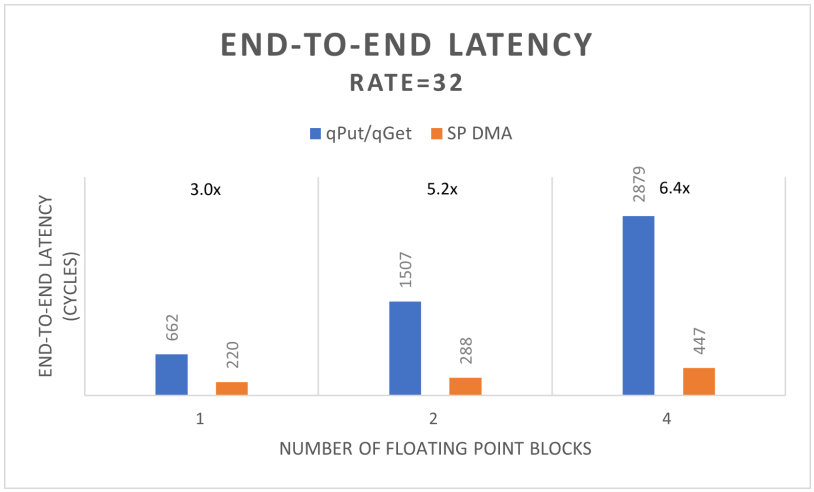
qPut/qGet Operation	Latency	SP DMA Operation	Latency
qPut(FP)	302	*mGetDMA(addr)*	20
NoC	12	NoC	9
		SP read, *mPut(FP)*	38.5
		NoC	11.5
Compress, *qPut(ZFP)*	80	Compress, *mPut(ZFP)*	80
NoC	21.5	NoC	21.5
qGet(ZFP)	131.5	SP data write	17.5
Total	547	Total	198

Evaluation

- Baseline performance scales linearly as number of floating point blocks increases
  - qPut/qGet instruction count scales linearly
- SP DMA has better speedup scalability
  - Average cycles per block decrease for higher number of blocks







Tool chain

ZHW uses C++ abstraction features

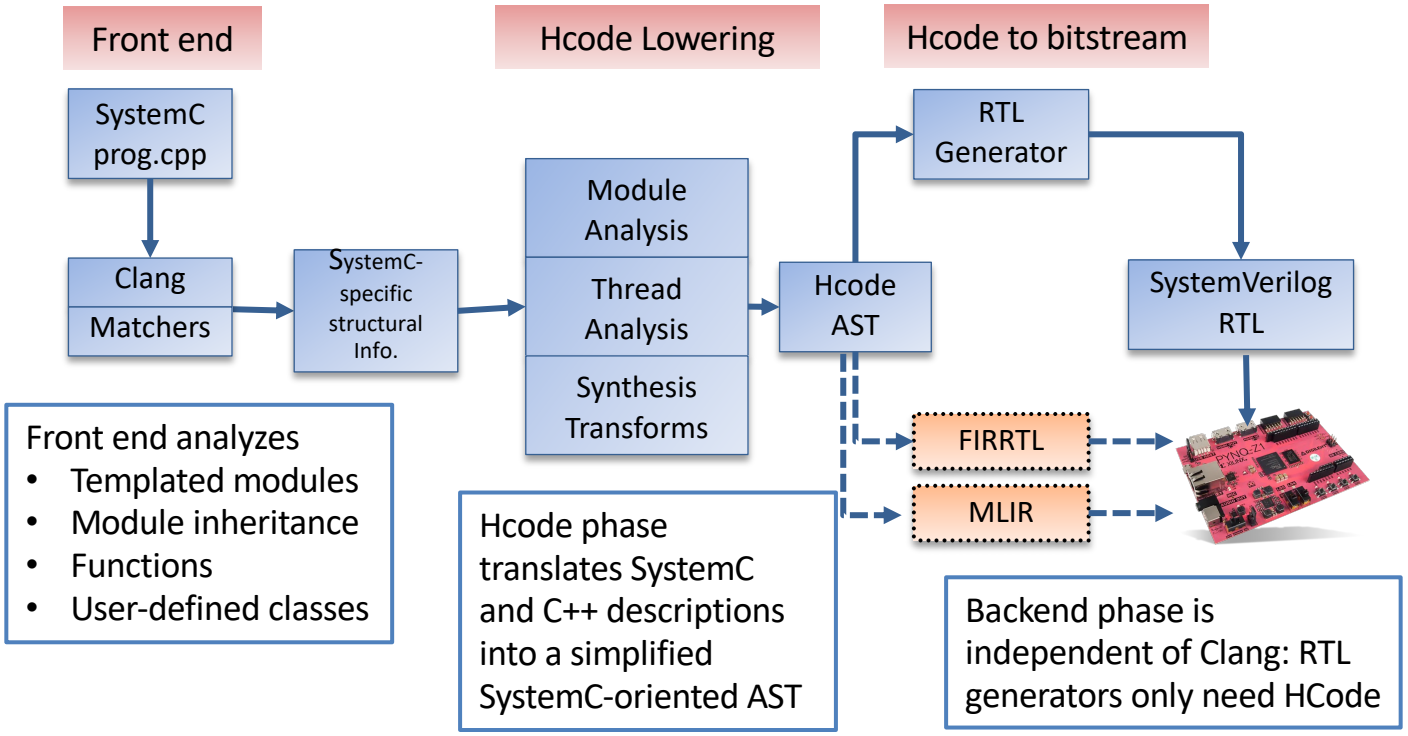
Heavily templated design

```
template<typename FP, int DIM, typename B>
SC_MODULE(encode)
```

Constant expressions, functions, enums, and internal structs

```
SC_MODULE(encode_stream) {
  static constexpr int tbc_w = log2rz(bp_w(DIM)*FP::bits+FP::ebits+1)+1;
  static constexpr int buf_w = max(bp_w(DIM),B::dbits)+B::dbits;
  enum state_e {START, ZERO, EXPO, PLANES, PAD};
  struct state_t { // ...
  };
  bool pack_bits(state_t &ts, sc_uint<tbc_w> bc, sc_bv<buf_w> bp) { // ...
  }
  bool out_bits(state_t &ts, bool done) { // ...
  }
};
```

SCCL Overview



Wu, Z., Gokhale, M.B., Lloyd, S., & Patel, H.D. (2023). SCCL: An open-source SystemC to RTL translator. 2023 IEEE 31st Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), 23-33.

From System Verilog to FPGA/ASIC

- SCCL-generate System Verilog compiled through FPGA tool chain
  - Hardware worked correctly on FPGA board
- SCCL-generate System Verilog did NOT pass through open source ASIC tool chain OpenRoad automatically
  - Many unsupported features
  - Bugs encountered (and fixed by the tool developers!)
  - Encoder required manual modification of generated System Verilog to get synthesis results
  - Decoder has features that don't pass through yosys
    - Modify SCCL to generate yosys-compliant System Verilog?

People!

Scott Lloyd:
DRE and Lookup Accelerator, ZHW

Xueyang Liu:
ZHW on MoSaic

Patricia Gonzalez-Guerrero:
MoSaic

SystemC to Verilog:
Zhuanhao Wu
Hiren Patel

Michael Barrow: ZHW decoder
Peter Lindstrom: ZFP library
Joshua Landgraf: Lookup accelerator in SST

Discussion

- Specialized HPC-centric hardware modules can improve performance
  - Programmable DMA engine for irregular memory access
    - What about address translation???
  - New instructions via RISC-V extensibility
- Large variety of scientific data, problems, approaches
  - Are there kernels/operations used widely enough to justify expense in labor, fab, maintenance?
- Tool chains continue to challenge
  - Hardware tool access should be as ubiquitous as software
    - Open source
    - Proprietary but free
    - Interoperable
- Verification/Validation of HW block, SW interface, HW/SW interactions

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC