

Purpose-built IP for High Performance Computing?

Maya Gokhale
Computing Directorate

September 9, 2024

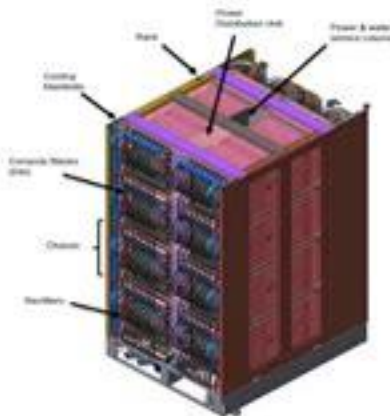


Exascale supercomputer nodes: commodity commercial CPUs and GPUs



Olympus rack

- 128 AMD nodes
- 8,000 lbs
- Supports 400 kW



All water cooled, even DIMMS and NICs

AMD node

- 1 AMD "Trento" CPU
- 4 AMD MI250X GPUs
- 512 GiB DDR4 memory on CPU
- 512 GiB HBM2e total per node (128 GiB HBM per GPU)
- Coherent memory across the node
- 4 TB NVM
- GPUs & CPU fully connected with AMD Infinity Fabric
- 4 Cassini NICs, 100 GB/s network BW

Compute blade

- 2 AMD nodes



System

- 2 EF Peak DP FLOPS
- 74 compute racks
- 29 MW Power Consumption
- 9,408 nodes
- 9.2 PB memory (4.6 PB HBM, 4.6 PB DDR4)
- Cray Slingshot network with dragonfly topology
- 37 PB Node Local Storage
- 716 PB Center-wide storage
- 4000 ft² foot print

Exascale Computing Project

Develop exascale-ready applications and solutions that address currently intractable problems of strategic importance and national interest.

Create and deploy an expanded and vertically integrated software stack on DOE HPC pre-exascale and exascale systems. Deliver **US HPC vendor technology advances and deploy ECP products** to DOE HPC pre-exascale and exascale systems.

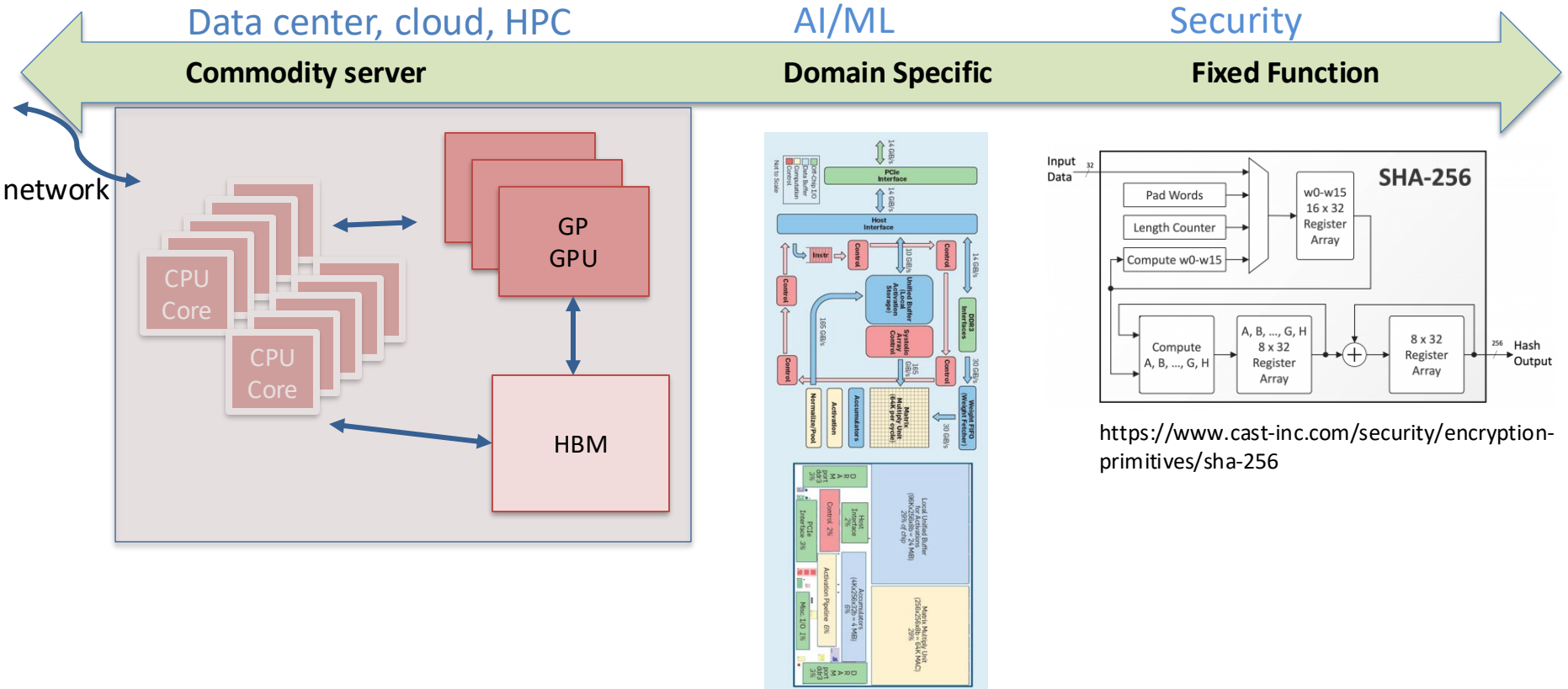


General purpose vs purpose built

US Department of Energy perspective

- HPC nodes use commercial data center server architectures
 - Invest in specific components such as low latency, high bandwidth interconnection network
 - Influence architectural direction and accelerate timeline with investment programs: FastForward, PathForward, Advanced Memory Technologies
 - Continue to pursue code refactor and rewrite
 - From vector to distributed memory parallel to GPU offload ... to AI engines?
- Can microelectronics fabrication and packaging innovations facilitate developing IP specialized to HPC?
 - 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
 - Can it be cost-effective?

Landscape of architectures



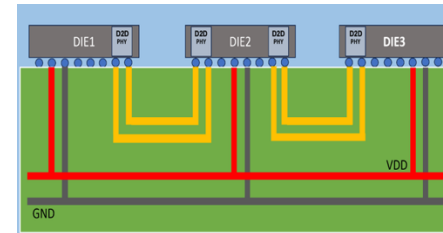
TPU: source nextplatform.com

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

Customize with specialized hardware blocks

- Standard operating procedure for large volume uses

- Hash units
- Compression
- Encryption
- AI matmul in low precision



<https://www.keysight.com/blogs/en/tech/sim-des/2024/2/8/what-is-a-chiplet-and-why-should-you-care>

- Can it work for HPC?

- Challenge is the huge range of science applications and techniques (MD → radiation transport)
- Select widely applicable kernels
 - Dense and sparse matmul, mat-vec operations
 - FFT
 - **Programmable Gather/Scatter Engine, K/V lookup accelerator**
 - **Floating point compression for ZFP (fixed rate)**
- Chiplet mix and match might make it feasible

Chiplet-based plug and play

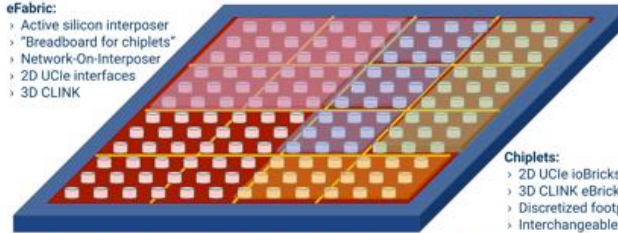
z e r |o|

Emulation Chiplets Team Partners News

- Embedded FPGA generator
- Application class RISC-V CPU
- AI accelerator (2-20 TOPS)
- Network-On-Chip
- Silicon Compiler
- Digital Twin System Emulator

eFabric:

- › Active silicon interposer
- › "Breadboard for chiplets"
- › Network-On-Interposer
- › 2D UCIe interfaces
- › 3D CLINK



Chiplets:

- › 2D UCIe ioBricks
- › 3D CLINK eBricks
- › Discretized footprints
- › Interchangeable



- Billions of unique System-In-Package assembly options
- 512 Gbps/mm on-fabric bisection bandwidth
- 128 Gbps/mm chiplet 2D bandwidth
- 128 Gbps/mm² chiplet 3D bandwidth
- <0.1 pJ/bit 3D interconnect energy efficiency

To enable plug-and-play chiplet composability, Zero ASIC has created a set of electrical and mechanical 3D chiplet interface standards and validated the standards through tapeouts of a canonical set of processing chiplets.

Outline: Augment server architectures with new IP blocks

- Memory-centric accelerators
- Scientific data compression accelerator
- CPU core interface options
- Tool chain

Memory-centric accelerators

- Data movement and memory access identified as key challenges to achieving high performance
 - Led to creation of US DOE Advanced Memory Technologies program
- Data movement is necessary – but only move necessary data
- Motivating applications include Sparse MatVec, found in HPCG benchmark
- Our approach: 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
 - Motivating application is bioinformatics: K-mer database lookup to identify genetic fragments in metagenomic sample
 - 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>

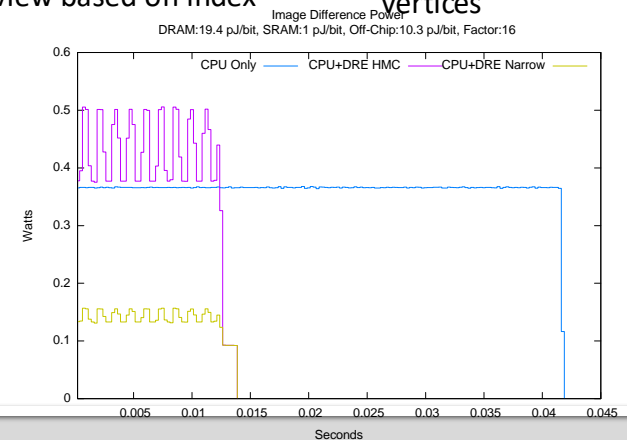
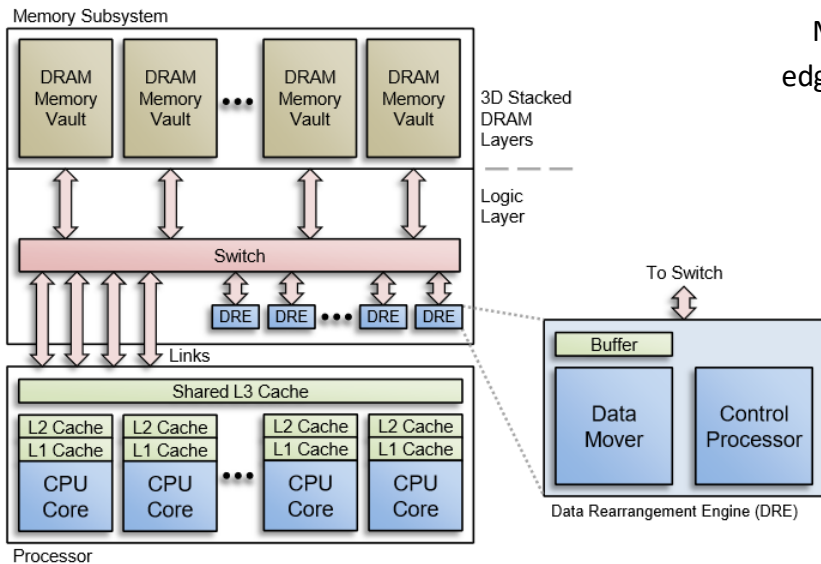
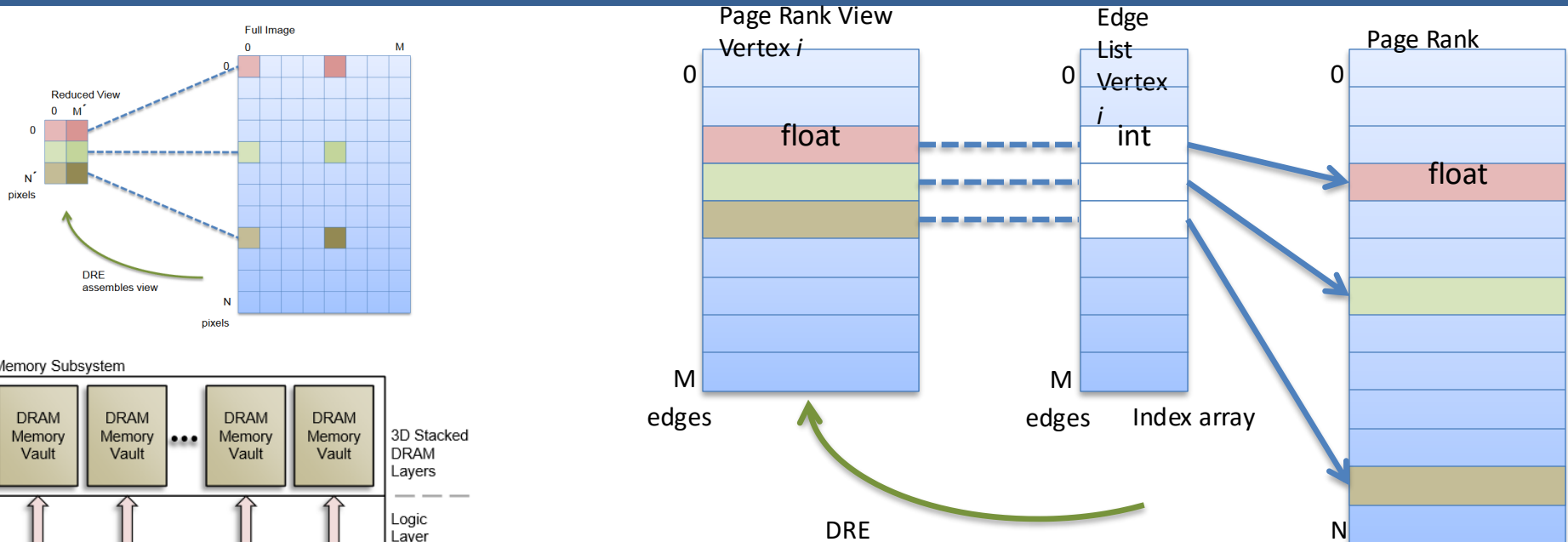
S. Lloyd and M. Gokhale, “In-memory data rearrangement for irregular, data intensive computing,” IEEE Computer, pp. 18–25, 2015.

Near-memory data reorganization engine
Patent number 9965187

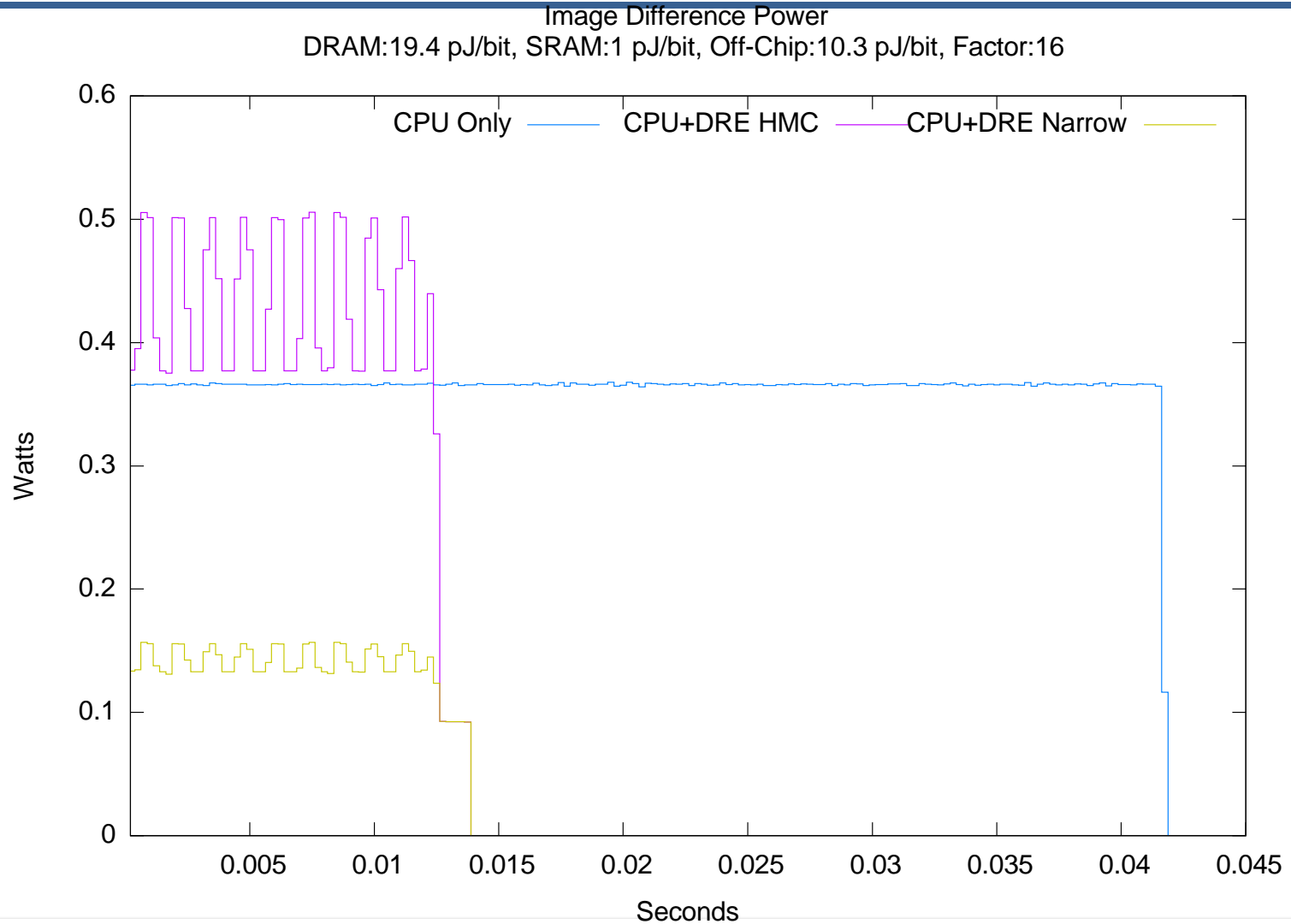
Near memory gather/scatter can help applications with irregular access patterns

- 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, are usually random access and can't benefit as much from increased bandwidth**
- 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

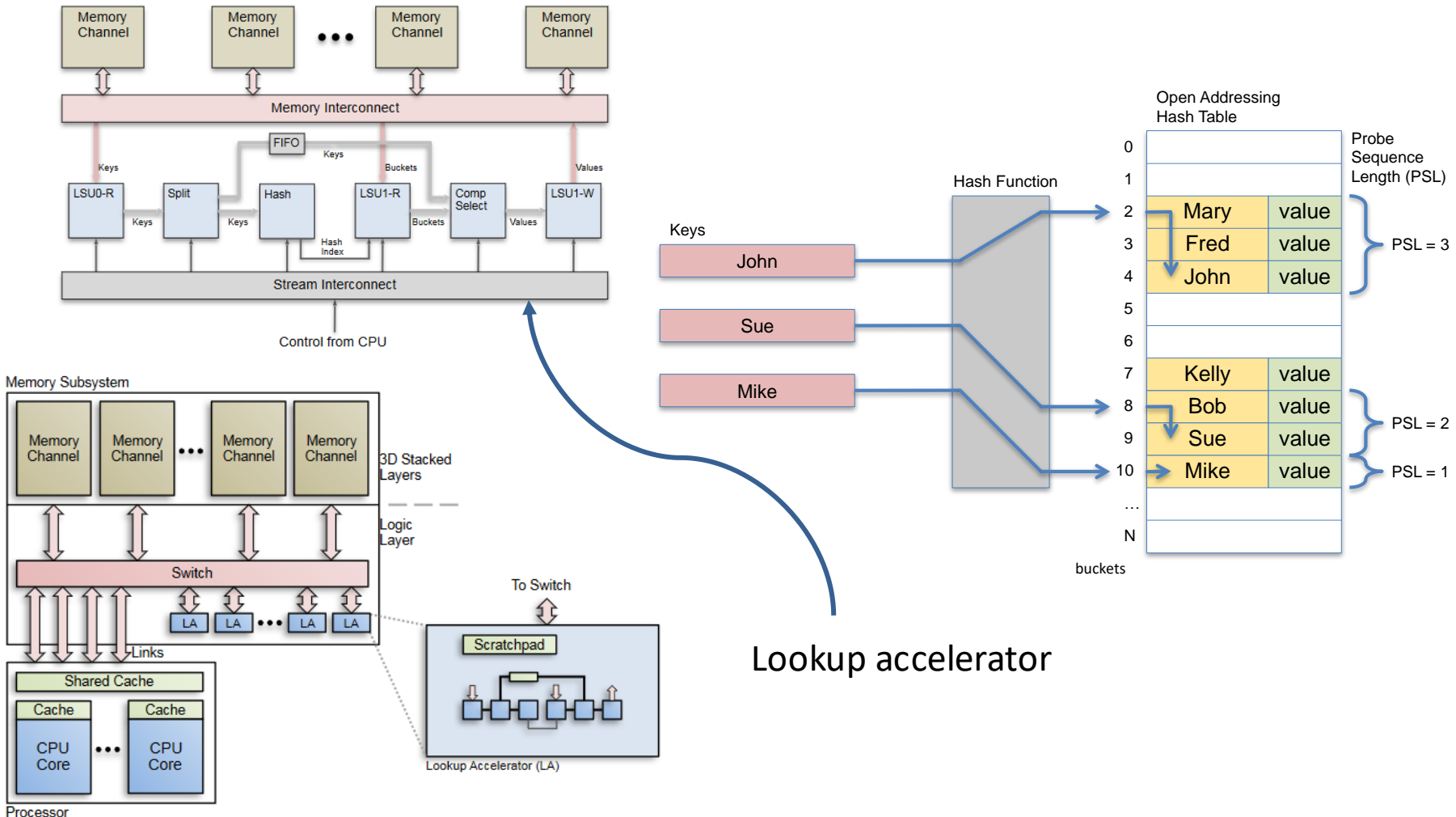
```
/* Specify view buffer and window offset */  
void drain(void *buf, size_t buf_sz, size_t offset);
```

Annoying details

- Abstract view of memory modeled on FPGA
 - Modeled on Hybrid Memory Cube, precursor of HBM
 - Latency-centric characterization
 - asymmetric read/write latencies
 - Model two different memory module latencies
 - Simplified memory model
 - Fixed latency or statistical distribution
 - Memory sees stream of read or write to physical addresses
- Where does **address translation** occur from virtual to physical
 - Scratchpad buffer on memory side holds physical addresses
 - Option 1: Route addresses written to or read from scratchpad through MMU
 - Option 2: Data is contiguous in physical memory
 - We used CMA
 - More general approach recently published: K. Zhao, et al., "Contiguitas: The Pursuit of Physical Memory Contiguity in Data Centers" in IEEE Micro, vol. 44, no. 04, pp. 44-51, 2024.
- Improve fidelity of simulation for specific memory type, specific interface
 - E.g. <https://arxiv.org/pdf/2311.10378> "Near-Memory Parallel Indexing and Coalescing: Enabling Highly Efficient Indirect Access for SpMV"
 - Model multiple independent channels as with HBM
- Detect gather/scatter pattern in instruction stream
 - .g. A. Naithani, J. Roelandts, S. Ainsworth, T. Jones and L. Eeckhout, "Decoupled Vector Runahead for Prefetching Nested Memory-Access Chains" in IEEE Micro, vol. 44, no. 04, pp. 20-26, 2024.

Key/Value Store Lookup Accelerator

Use gather/scatter engine as component (LSU)



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

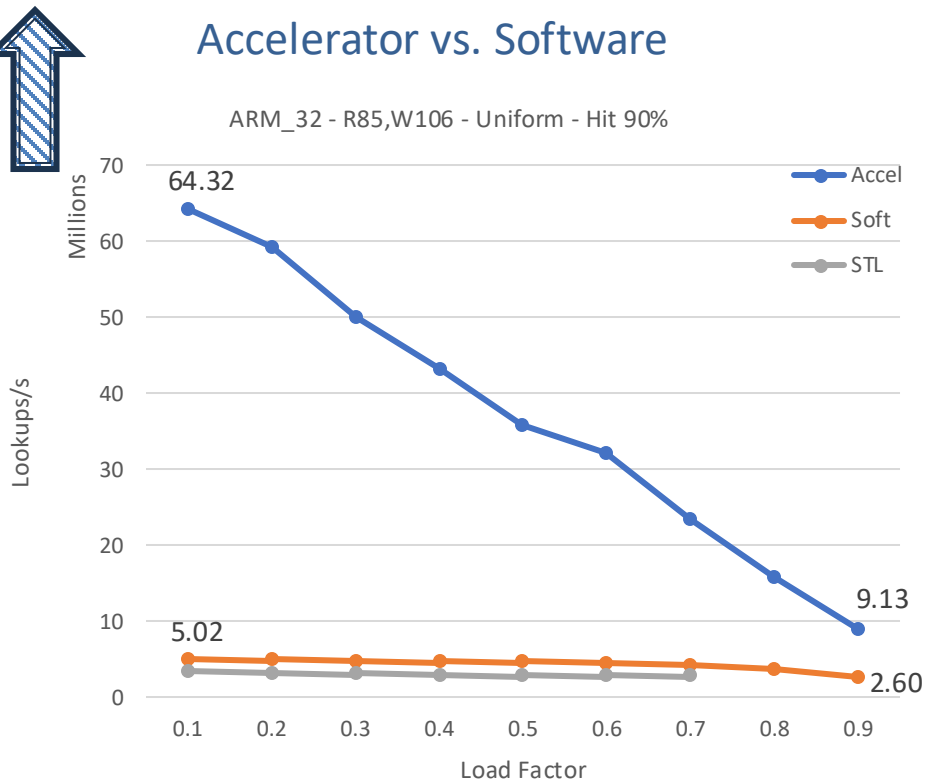
- **Accel**
 - Near memory hardware lookup accelerator
 - Collision resolution: open addressing and Robin Hood hashing
 - Hash function: adapted from SpookyHash
 - Lookup uses linear probing
- **Soft**
 - Software version of the hardware lookup algorithm
 - Collision resolution: same as Accel
 - Hash 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

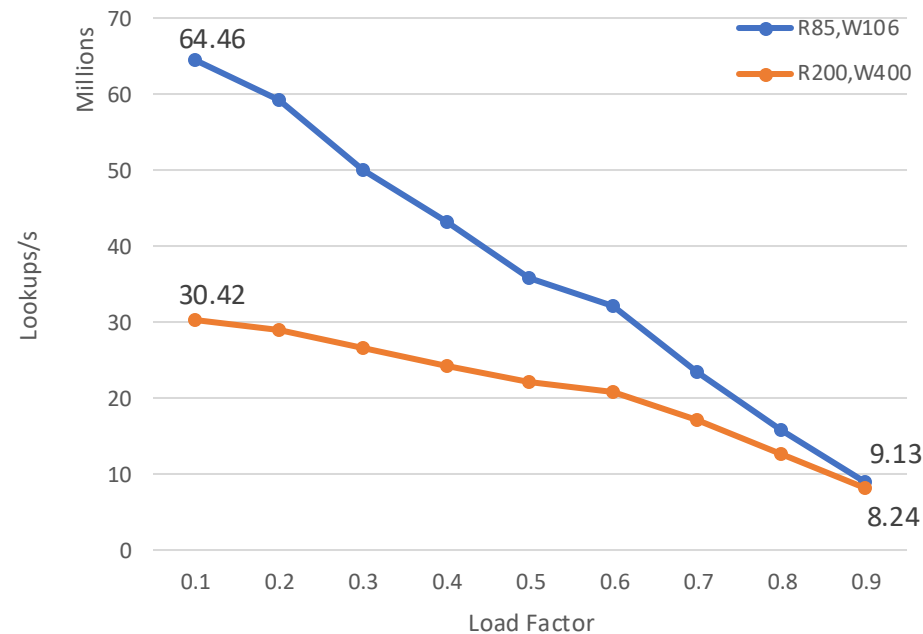
Accelerator vs. Software

ARM_32 - R85,W106 - Uniform - Hit 90%



Low vs. Moderate Latency

ARM_32 - Accel - Zipf=.99 - Hit 90%



- 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

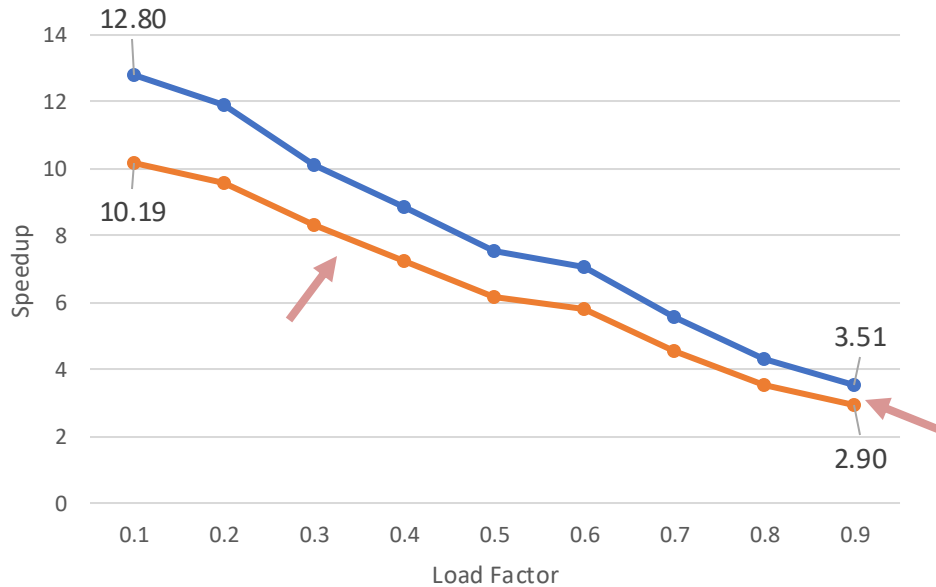
Speedup of Uniform and Zipfian Key Distributions

90% hit rate

Low Latency (DRAM)

ARM_32 - Accel/Soft - R85,W106 - Hit 90%

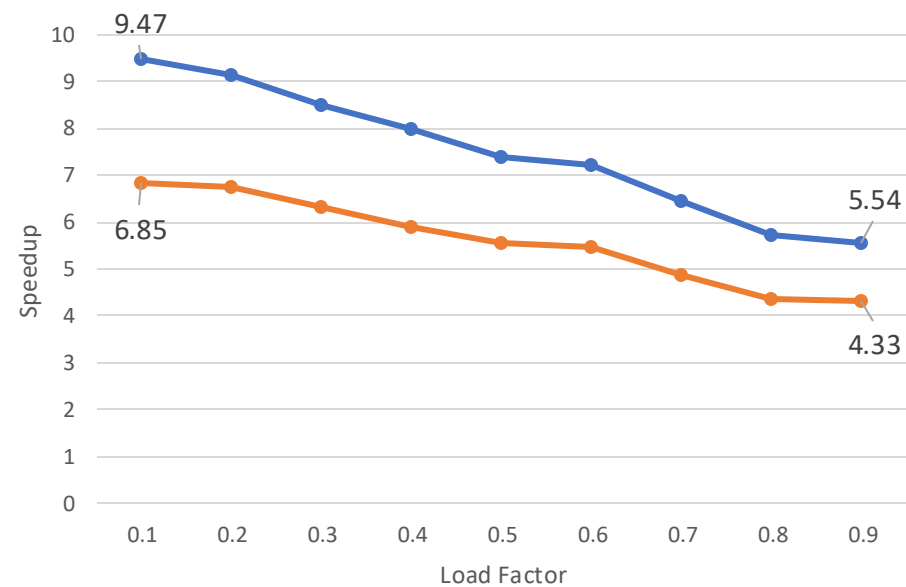
Uniform Zipf



Moderate Latency (SCM)

ARM_32 - Accel/Soft - R200,W400 - Hit 90%

Uniform Zipf



- 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)

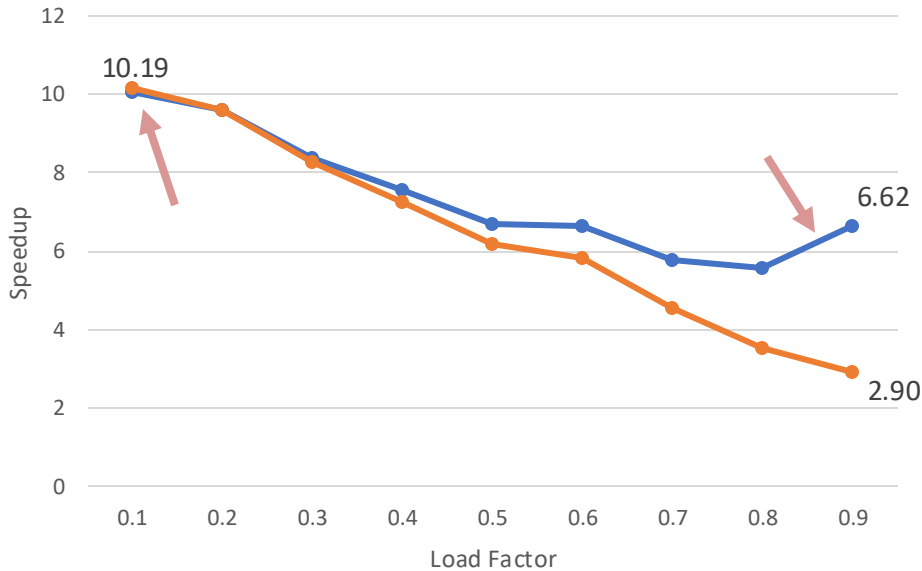
Speedup of 10% and 90% Hit Rate

Zipf skew factor 0.99

Low Latency (DRAM)

ARM_32 - Accel/Soft - R85,W106 - Zipf=.99

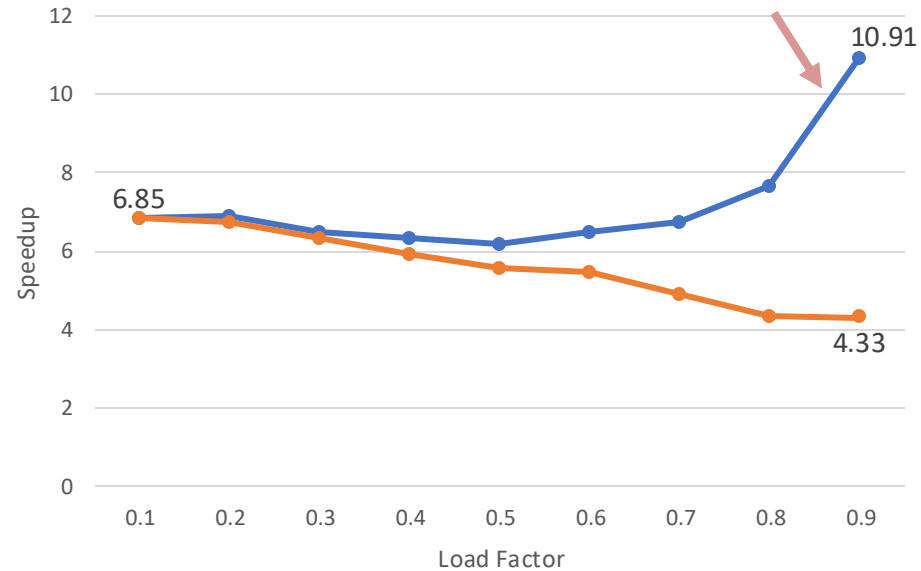
hit 10% hit 90%



Moderate Latency (SCM)

ARM_32 - Accel/Soft - R200,W400 - Zipf=.99

hit 10% hit 90%



- 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

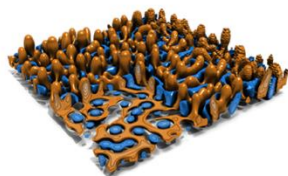
- Interface matters
 - Blocks of requests/responses to enable efficient pipelining
- Evaluation matters
 - Under what conditions will the hardware IP block be worthwhile?
 - FPGA-based emulator was a big investment in time
 - Fast
 - High visibility
 - But ...
 - Combining with software SST simulator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

<https://github.com/llnl/lime>

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

Compression for HPC: block compression of 1D, 2D, 3D floating point arrays

zfp: Compressed Floating-Point and Integer Arrays



Open-source software for compressed floating-point arrays

zfp

zfp Versions
zfp Compression
zfp and Derivatives
zfp Arrays
Floating Point Compression
Publications
Related Projects


zfp Video

This video describes how zfp saves storage, time, and compute power. Watch on YouTube.

Learn More

- For bug reports and questions, contact: zfp@llnl.gov
- GitHub repository and latest version 1.0.0
- See also the menu above including lists of publications and related projects
- Science & Technology Review: The Laboratory's Habit of Innovations

zfp is an R&D 100 winner



Home / Projects: Inspired R&D at the Heart of LLNL Computing

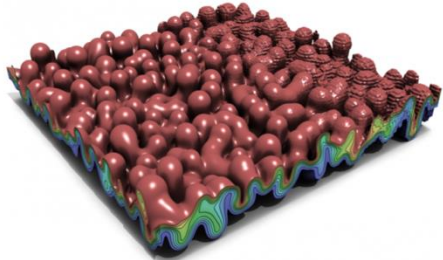
zfp is a BSD licensed open-source library for compressed floating-point and integer arrays that support high throughput read and write random access. One of zfp's unique features is its support for efficient in-memory representation of multidimensional numerical data for computations like differential equation solvers, data analysis, and visualization, with significant reductions in memory usage. zfp is primarily written in C and C++ but also has Python, Fortran, and other bindings. zfp is loosely based on the algorithm described in the following paper:

Peter Lindstrom, "Fixed-Rate Compressed Floating-Point Arrays," *IEEE Transactions on Visualization and Computer Graphics*, 20(12): 2674–2683, December 2014, doi:10.1109/TVCG.2014.2346458.

zfp was designed to achieve high compression ratios and therefore uses lossy but optionally error-bounded compression. Bit-for-bit lossless compression of integer and floating-point arrays is also supported. zfp provides high-quality compression and is fast, achieving throughputs of up to 2 GB/s per CPU core and 800 GB/s aggregate throughput on recent GPUs. zfp supports several different back-ends, including OpenMP, CUDA, and HIP. An FPGA Implementation is also available.

zfp is hosted as open source on GitHub and can also be installed with package managers like conda, spack, RPM, and MacPorts. Separate conda and pip packages are available for zfp, the Python interface to zfp. HDF5 users may be interested in the HSZ-ZFP compression plugin. zfp is supported by software tools and libraries like Intel IPP, HDF5, ADIOS, BLOSC, E4S, MVAPICH2, OpenInventor, TTK, VTK-m, and Zarr.

zfp development is supported by the U.S. Department of Energy's Exascale Computing Project, by the Advanced Scientific Computing Research Program, and by the Advanced Simulation and Computing Program. Advanced features such as variable-rate random-access arrays were investigated on LLNL's Variable Precision Computing Project.



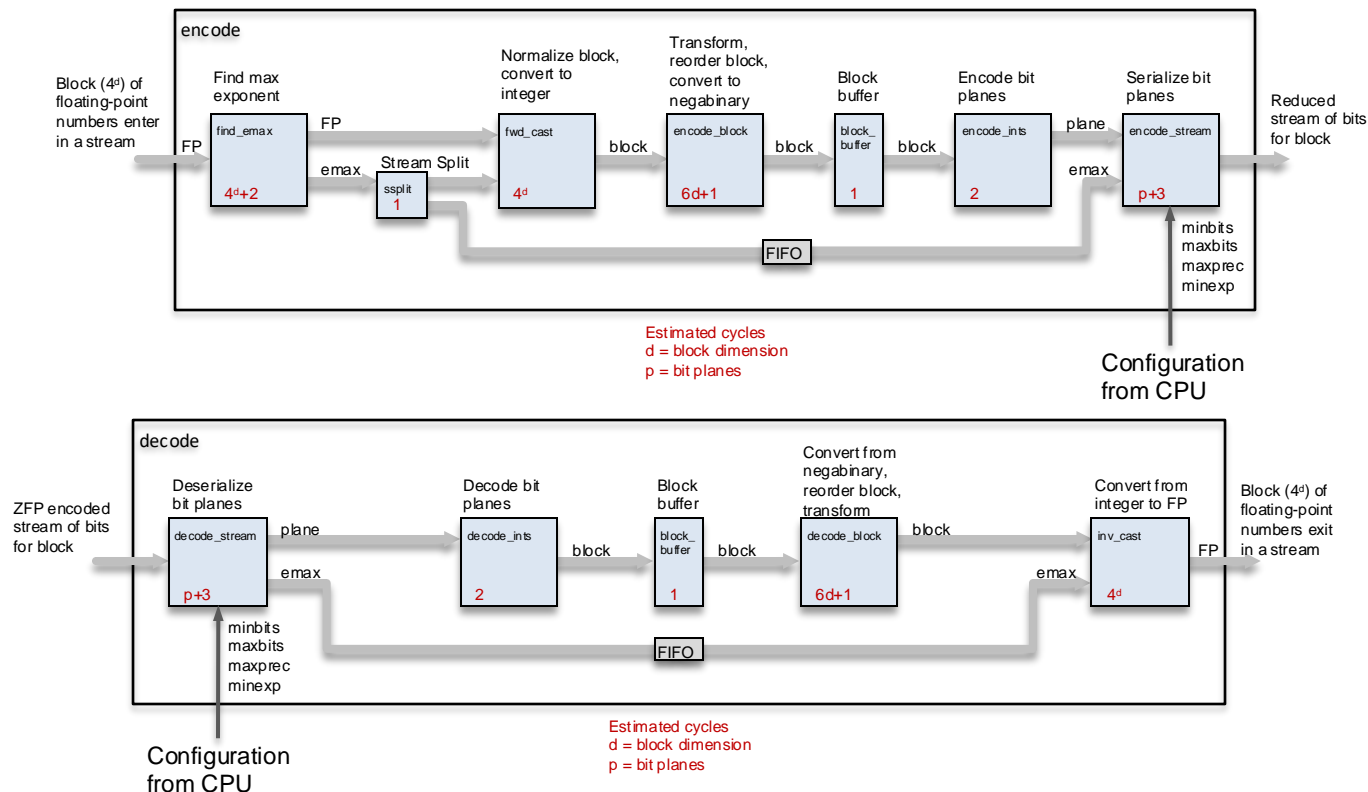
ZHW hardware codec fully interoperable with zfp

- Encode with zhw, decode with zfp
- Encode with zfp, decode with zhw

ZHW supports fixed rate 1, 2, 3D arrays organized as blocks

ZHW implemented in SystemC, using templating features of C++ to parameterize floating point format (32b or 64b), block size, and compression factor

ZHW: hardware ZFP compression pipeline for floating point arrays



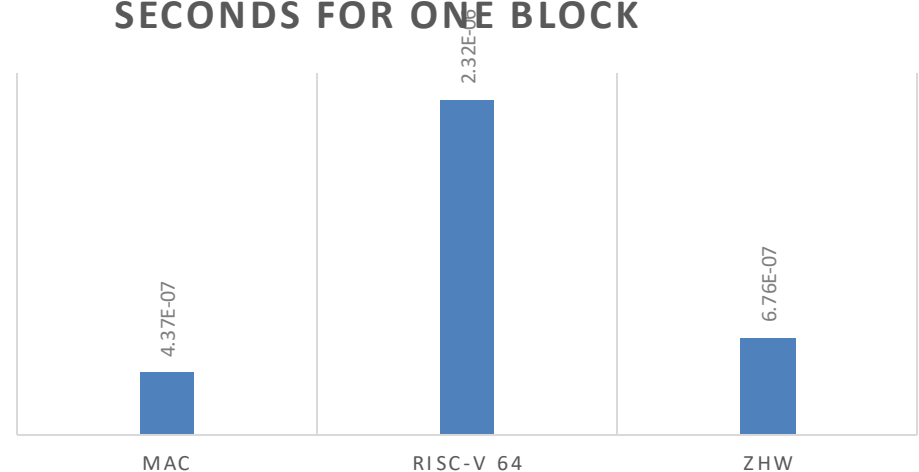
ZFP: software floating point library for scientific floating point arrays. 2023 R&D 100 award winner.

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.

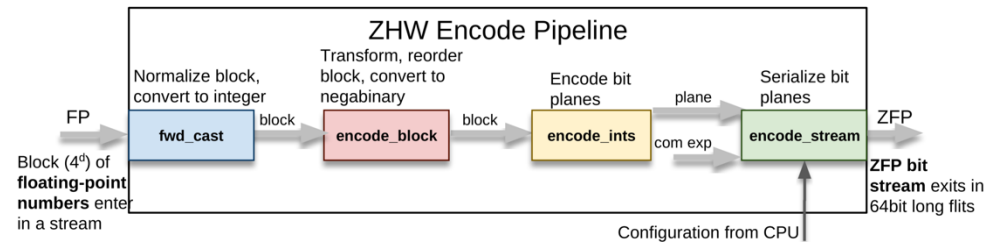
ZHW Encoder as an IP block

- Encoder is synthesizable at 293MHz
- IP block outperforms Vision 5 RISC-V64 core by 3.43X
- Slower than x86 Mac laptop
- Provide encoder as custom instruction issued by RISC-V core in SoC

ENCODER ELAPSED TIME IN SECONDS FOR ONE BLOCK

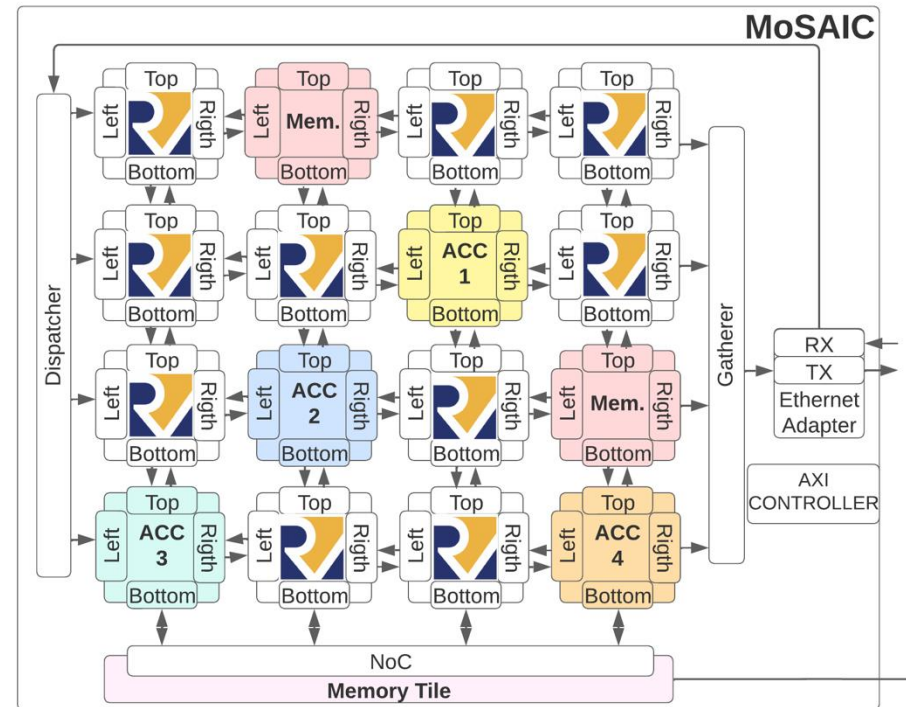


X. Liu, P. Gonzalez-Guerrero, I. B. Peng, R. Minnich, and M. B. Gokhale, "Accelerator integration in a tile-based soc: lessons learned with a hardware floating point compression engine," SC-W '23: Proceedings of the SC '23 Workshops of The International Conference on High Performance Computing, Network,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



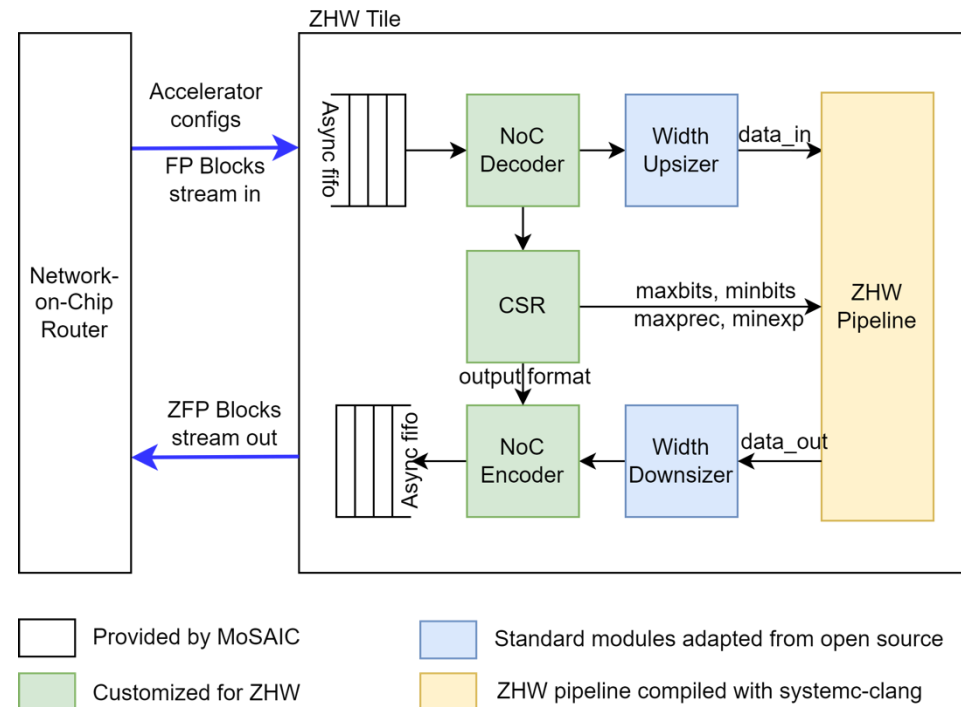
Encoder uses LBNL Mosaic SoC

- Tile architecture, heterogeneous and configurable
 - HW: RISC-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 encoder 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

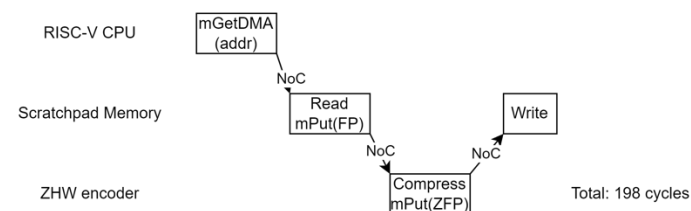
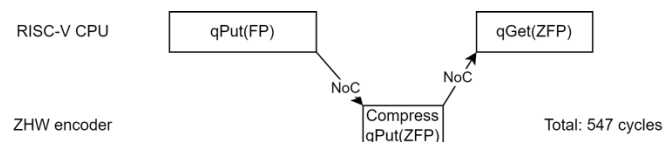
```
27 int rd_zhw(int tile_id, int *data, int size) {
28     int temp;
29     int received_data_poll;
30     qPoll(tile_id, received_data_poll);
31     /* Check if queue is empty */
32     if (received_data_poll == 1){
33         return -1;
34     }else{
35         /* Pop headers from queue */
36         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**

```

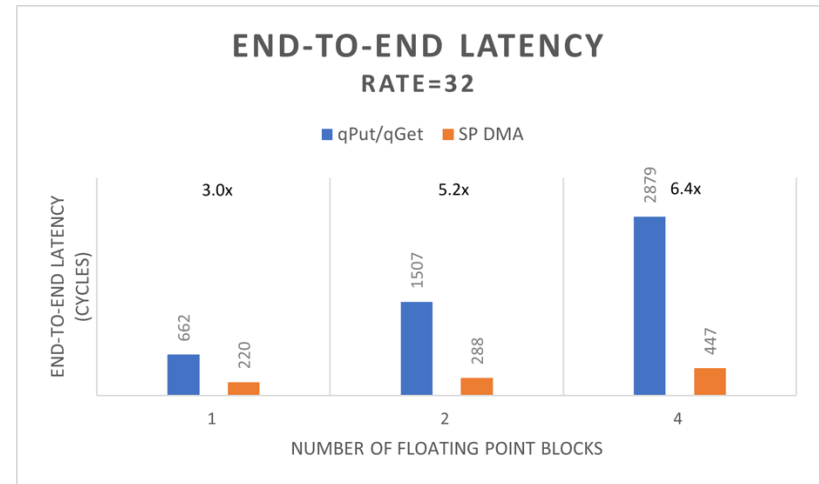
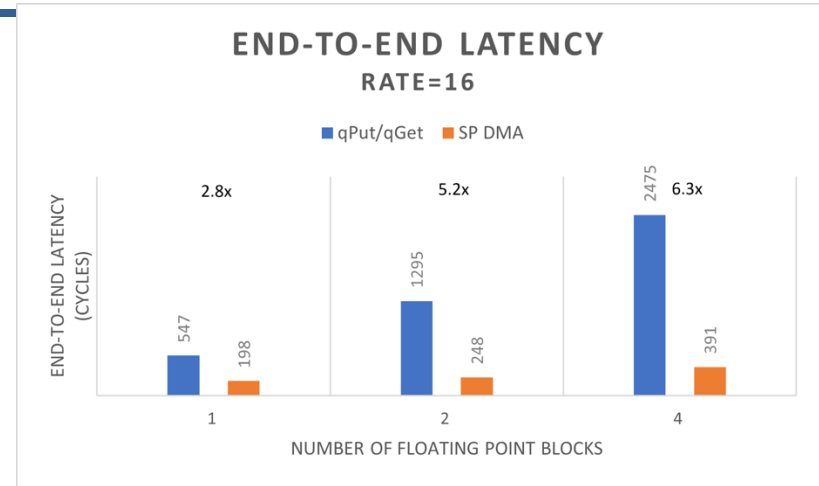
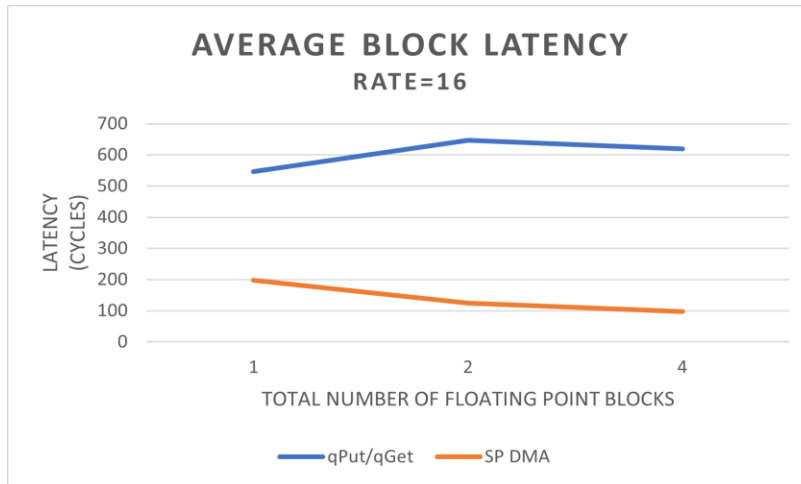
8  /* Retrieve data using SPAD DMA, data preloaded in
   spad2 */
9  mGetH((spad2_tile << 12), TEST_SIZE_LOG2);
10 mGetDMA((zhw_tile << 12) + ZHW_STREAM_ADDR, 0); // send
    to dedicated zhw address for input data streaming
  
```



<i>qPut/qGet</i> Operation	Latency	SP DMA Operation	Latency
<i>qPut(FP)</i>	302	<i>mGetDMA(addr)</i>	20
NoC	12	NoC	9
		SP read, <i>mPut(FP)</i>	38.5
		NoC	11.5
Compress, <i>qPut(ZFP)</i>	80	Compress, <i>mPut(ZFP)</i>	80
NoC	21.5	NoC	21.5
<i>qGet(ZFP)</i>	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

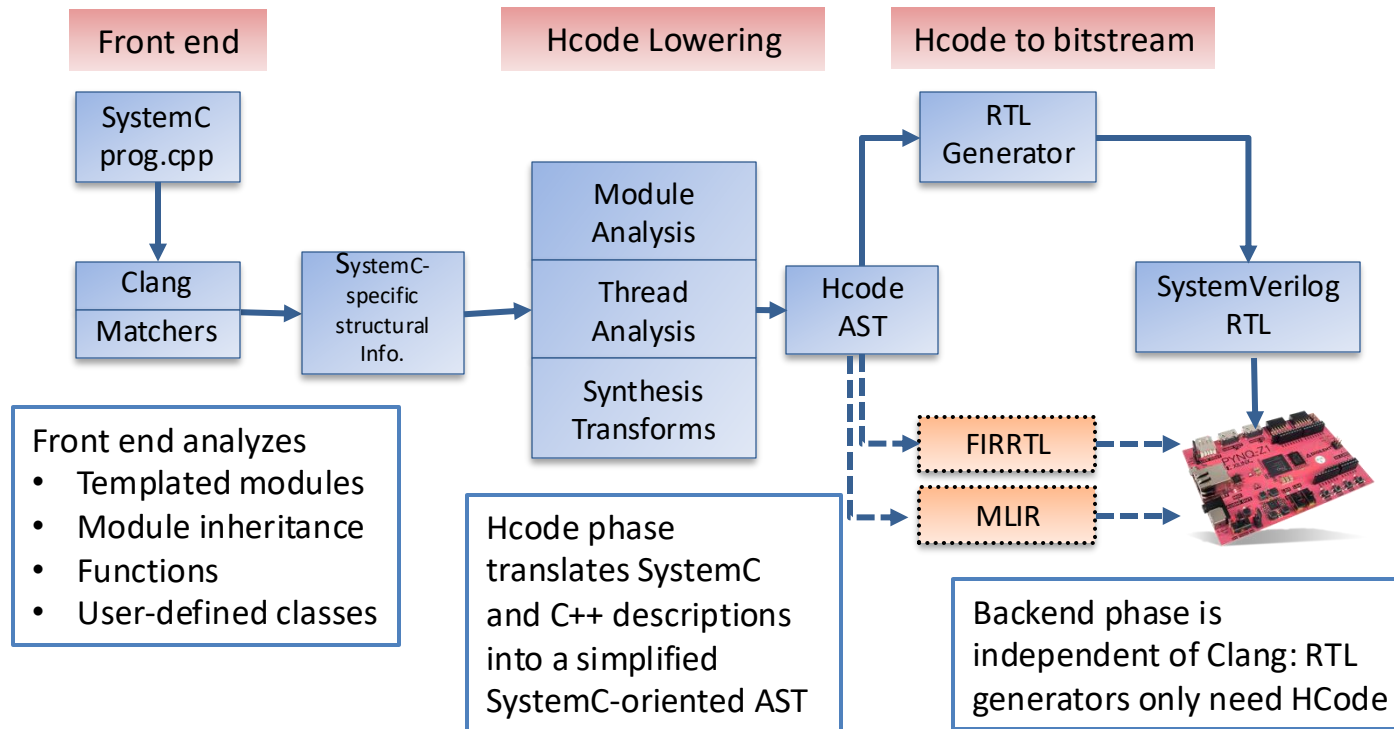
Heavily templated design

```
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 initially pass through open source ASIC tool chain OpenRoad automatically
 - Unsupported features
 - Bugs
- Using SiliconCompiler OpenRoad tool chain, synthesis, Place&Route completed with 7nm and 45nm technology nodes

ASIC synthesis results for decoder and encoder

- Decoder was compiled using SiliconCompiler and OpenRoad (<https://www.siliconcompiler.com/>):
 - Used Asap7 7 nm technology
 - f_max was 27.963 MHz
 - Area was 49,053.8 μm^2
- Encoder was compiled using SiliconCompiler:
 - Used Asap7 7 nm technology
 - Best f_max was 292.763 MHz
 - Area was 23,318.7 μm^2
- Decoder was compiled through CMC:
 - Used OpenPDK45 45 nm technology
 - Max frequency was 31.2 MHz
 - Area was 330,625 μm^2



Decoder ASIC generated by Siliconcompiler

U Waterloo collaboration

People!



Scott Lloyd:
DRE and Lookup
Accelerator, ZHW



Xueyang Liu:
ZHW on MoSaic



Patricia Gonzalez-Guerrero:
MoSaic

SystemC to SystemVerilog:
Zhuanhao Wu
Hiren Patel
Maya Gokhale

Michael Barrow: ZHW decoder
Michael Gionet: ZHW flow in SiliconCompiler, Boom core
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 can potentially improve performance
 - Coordination with cache hierarchy and MMU required
 - Zhw shows speedup over RISC-V core, but further improvements will require algorithm/hardware co-design
- 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
 - Multiple levels
 - SystemC valuable for hardware/software interface validation
 - RTL simulation uncovers lower layers of issues



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