Architecture exploration through FPGA acceleration

Rapid System Level Design and Evaluation of Near Memory Fixed Function Units

CASC
Center for Applied Scientific Computing

11/13/2020

Maya Gokhale
DMTS

LLNL-PRES-816381

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

Lawrence Livermore National Laboratory

Outline

- Trends in reconfigurable computing
  - Architectures
  - Tools
  - Applications
- Targeting fast architecture design space exploration
  - MPSoC to accelerate design and evaluation of heterogeneous function units
  - Mixed hardware/software approaches for scaling studies for complex design space scenarios
- The perennial tools problem
  - Need for a unified hardware/software development environment
  - Open source

FPGA architecture has evolved as dramatically as CPU

- Xilinx 3000 series
  - Configurable Logic Blocks “sea of gates”
  - I/O Blocks high speed programmable input-output
  - Interconnect combining mesh and long lines

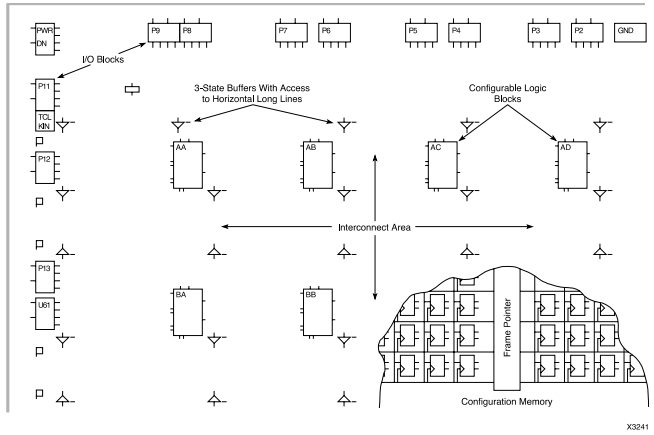


Figure 2: Field Programmable Gate Array Structure.
It consists of a perimeter of programmable I/O blocks, a core of configurable logic blocks and their interconnect resources. These are all controlled by the distributed array of configuration program memory cells.

https://www.xilinx.com/support/documentation/data_sheets/3000.pdf

- Xilinx Versal
  - Specialized DSP processors
  - “Fabric” for data acquisition/pre-processing
  - Control processor

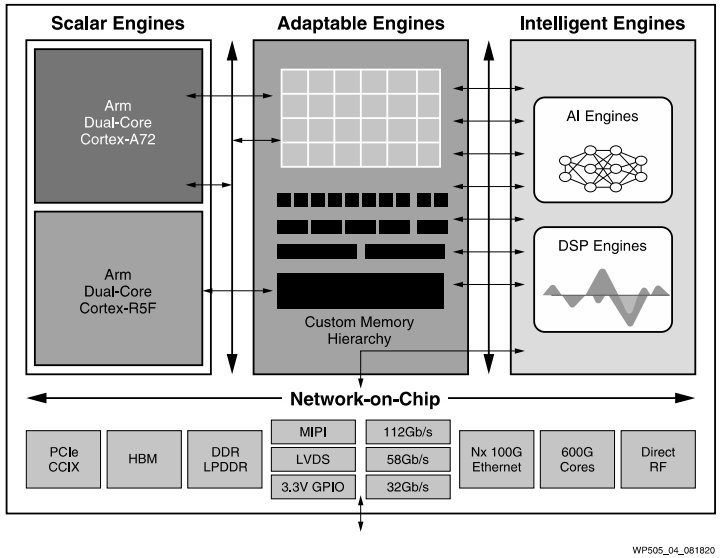
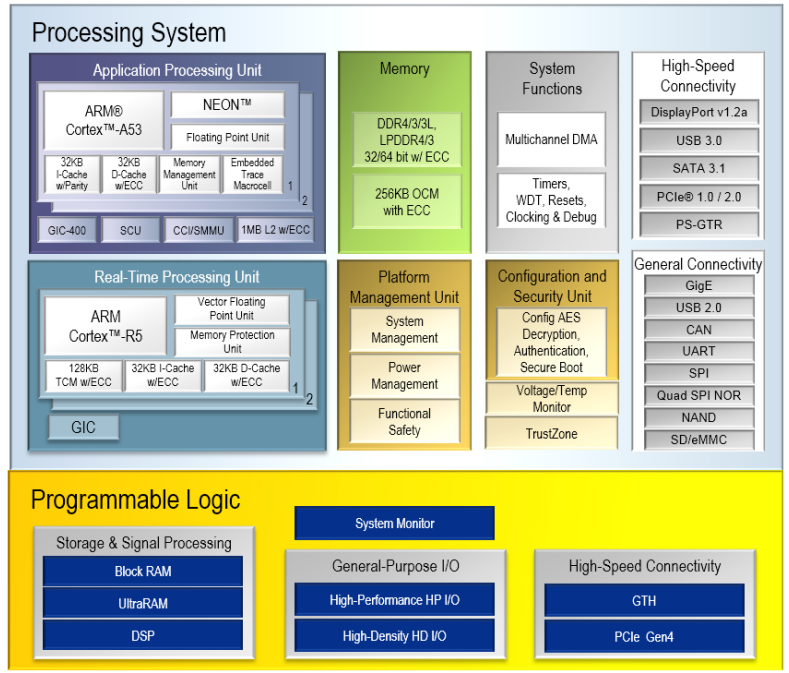


Figure 4: **Xilinx Versal ACAP Functional Diagram**

https://www.xilinx.com/support/documentation/white_papers/wp505-versal-acap.pdf

Progression of FPGA architecture evolution

- Embedded, distributed memories to store local state
- DSP blocks for fast fixed point arithmetic
- I/O architecture optimization for fast data ingest and generation
- Clock management for multiple clock domains
- Host CPU integration
  - HPC & ACP, CXL, CAPI



Specializations for application domains

Video codec

100 Gb EMAC, PCIe gen 4

FPGA tools have evolved from microprogramming to (highly annotated) C++

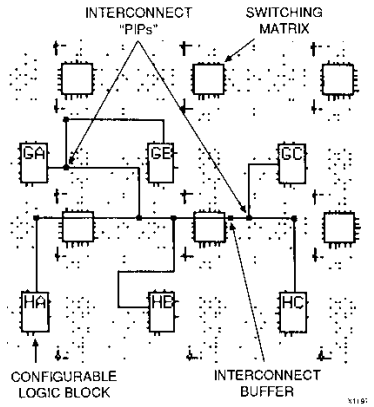
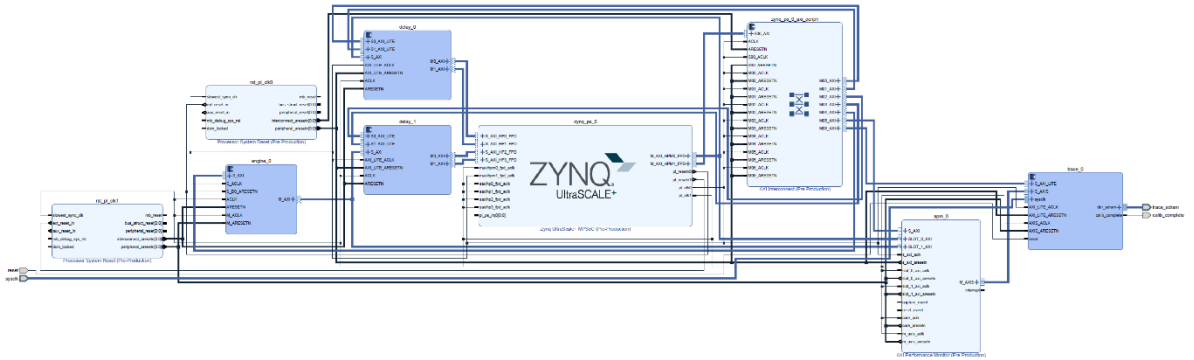


Figure 8: A Design Editor view of routing resources used to form a typical interconnection network from CLB GA.




```
// Ethernet FIFO interface
// Receives 128-bit wide data in
// Transmits a packet via PS Ethernet FIFO
// This version supports flushing out buffered data
void eth_fifo_interface(
                u1t dma_tx_end_tog,
                u1t tx_r_fixed_lat,
                u1t tx_r_rd,
                …) {
```

```
#pragma HLS PIPELINE II=1 enable_flush
#pragma HLS INTERFACE ap_ctrl_none port=return
#pragma HLS INTERFACE ap_none port=dma_tx_end_tog
#pragma HLS INTERFACE ap_none port=tx_r_fixed_lat
#pragma HLS INTERFACE ap_none port=tx_r_rd
#pragma HLS INTERFACE ap_none port=tx_r_status
…
// various state variables and useful constants
static enum state {IDLE, MAC_DST, MAC_SRC, TYPE, PAYLOAD, ZEROS, ID}
  current_state = IDLE;
        const u8t src_mac[6] = {0x00, 0x0A, 0x35, 0x03, 0x59, 0xF5};
#pragma HLS ARRAY_PARTITION variable=src_mac complete dim=1
…
        static u8st data_buffer;
#pragma HLS STREAM variable=data_buffer depth=16384
```

Reconfigurable computing applications are diverse

- Signal and image processing
  - Satellite, space application
  - Instrument sensor data streams
- Network packet processing
  - Routing
  - In-stream processing
  - Regular expression matching
- Finance
  - Integrated with network packet processing
  - High frequency trading
  - Risk analysis

CHIME Radio Telescope with F-Engine Containers

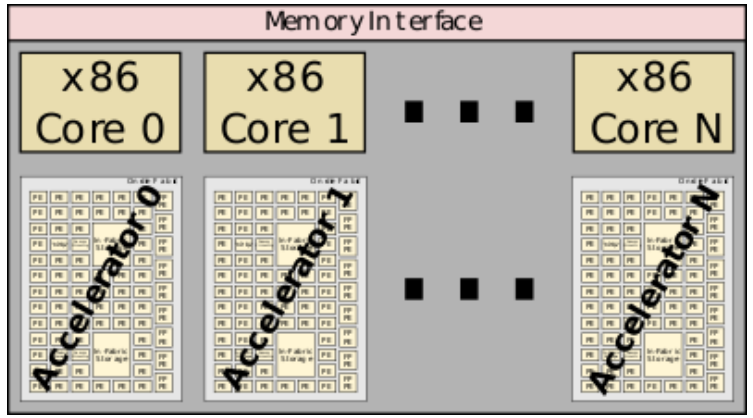
Mars Perseverance Rover

- Data center
  - Microsoft investment in FPGAs to accelerate search, ML, etc.: the FPGA sits between the datacenter's top-of-rack (ToR) network switches and the server's network interface chip (NIC). As a result, all network traffic is routed through the FPGA, which can perform line-rate computation on even high-bandwidth network flows.
  - Amazon F1 for individual, corporate, or FPGA as a service
- **Logic emulation**
  - **Use the sea of gates to emulate IP blocks, function units, full ASICs**

FPGAs can accelerate architecture exploration by orders of magnitude over software

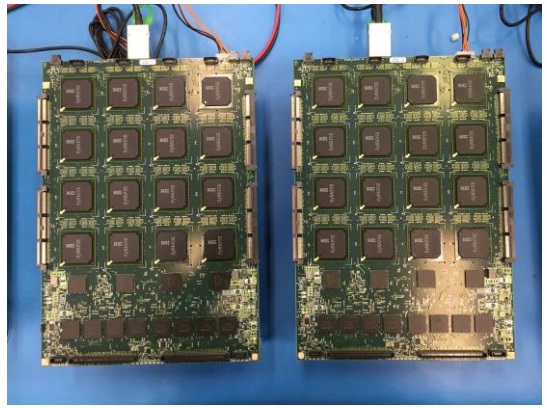
- M. Butts, J. Batcheller and J. Varghese, “An efficient logic emulation system,” *Proceedings 1992 IEEE International Conference on Computer Design: VLSI in Computers & Processors*, Cambridge, MA, 1992, pp. 138-141.
  - Realizer System: array of FPGAs for emulating large digital logic design
- Q. Wang et al., "An FPGA Based Hybrid Processor Emulation Platform," 2010 International Conference on Field Programmable Logic and Applications (https://ieeexplore.ieee.org/document/5694215)
  - Emulates Xeon processor on FPGA in a processor socket
- FireSim for many-core RISC-V simulation https://rise.cs.berkeley.edu/projects/firesim/
  - Amazon F1 cloud
  - Custom accelerators for RISC-V
- ESP for heterogeneous SoC design https://www.esp.cs.columbia.edu
  - tile-based architecture built on a multi-plane network-on-chip
  - prototype on FPGA
- Logic in Memory Emulator (LiME) follows a hybrid approach: keep the native hard IP cores/cache hierarchy for the CPU complex and use the programmable logic to emulate widely varying memory latencies and near memory accelerators

Shift to heterogeneous computing has generated innovation in purpose-built hardware blocks from exascale to IoT

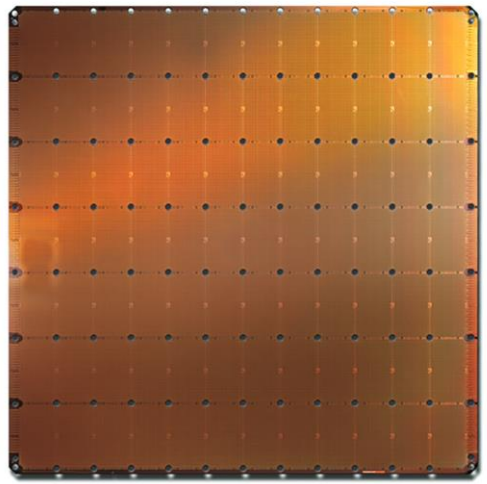


Intel CGRA
https://en.wikichip.org/wiki/intel/configurable_spatial_accelerator

Heterogeneous computing has been dominated by GPUs, but contenders abound:
For example, specialized tensor processing cores with embedded SRAM, HBM, fast network



LLNL NS61e True North boards with 16 TN chips



Cerebras WSE
1.2 Trillion transistors
46,225 mm² silicon

Largest GPU
21.1 Billion transistors
815 mm² silicon

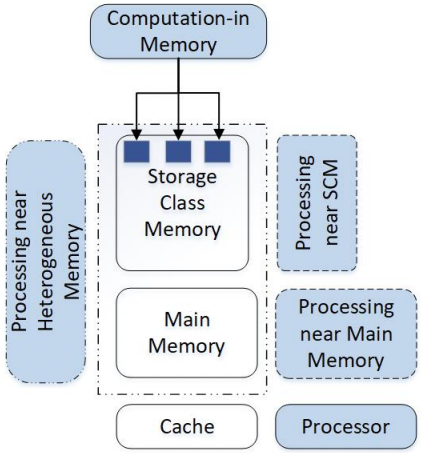
Habana Gaudi AI training chip

Focus on compute units

New memory technologies and packaging are needed to deliver data to the compute units

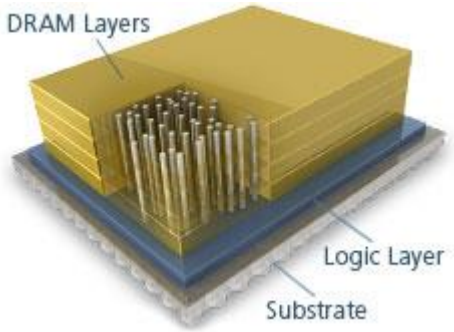
- Advances in memory technology and packaging
  - High bandwidth memories – HBM, HMC
  - Non-volatile memory – 3D Xpoint

  - focuses attention on computer memory system design and evaluation
  - Potential for logic and compute functions co-located with the memory



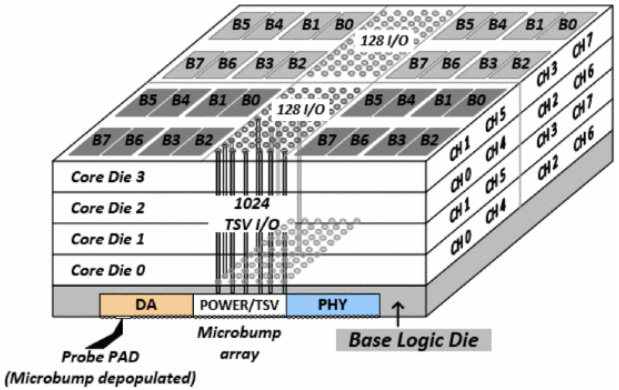
Singh, et. al. https://arxiv.org/pdf/1908.02640.pdf

HMC



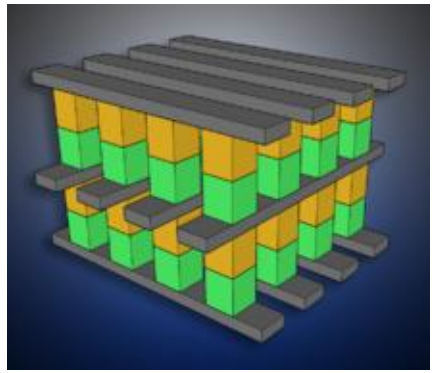
Micron Technology

HBM



Hongshin Jun, et. al. IMW 2017

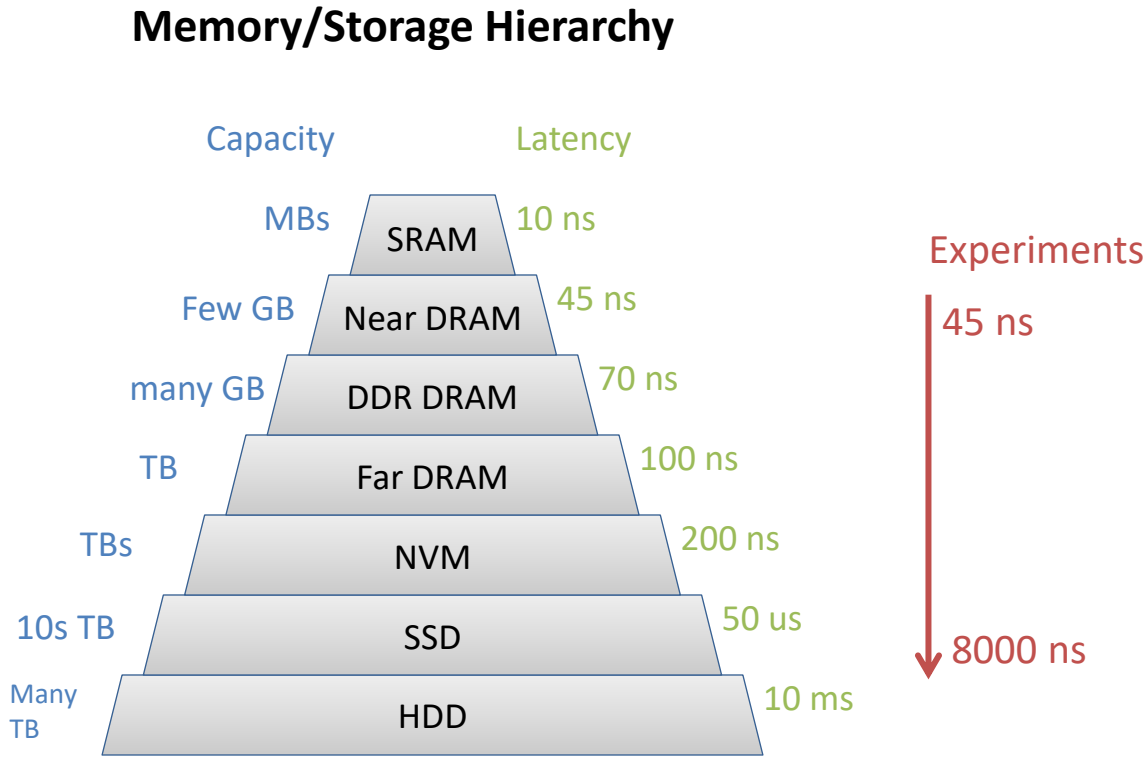
3D XPoint



Creative Commons Attribution

Memory landscape diversity presents challenges

- Emerging memories exhibit a wide range of bandwidths, latencies, and capacities
  - Challenge for the computer architects to navigate the design space
- Near-random and sparse access patterns make performance prediction difficult
  - Challenge for application developers to assess performance implications
- Opportunities for near memory acceleration emerge
  - Large design space must be investigated

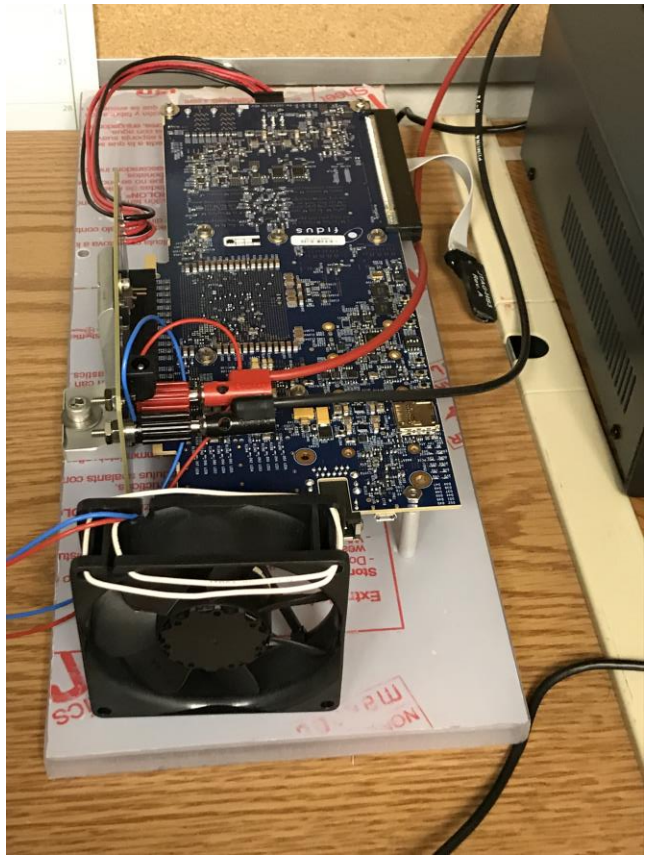


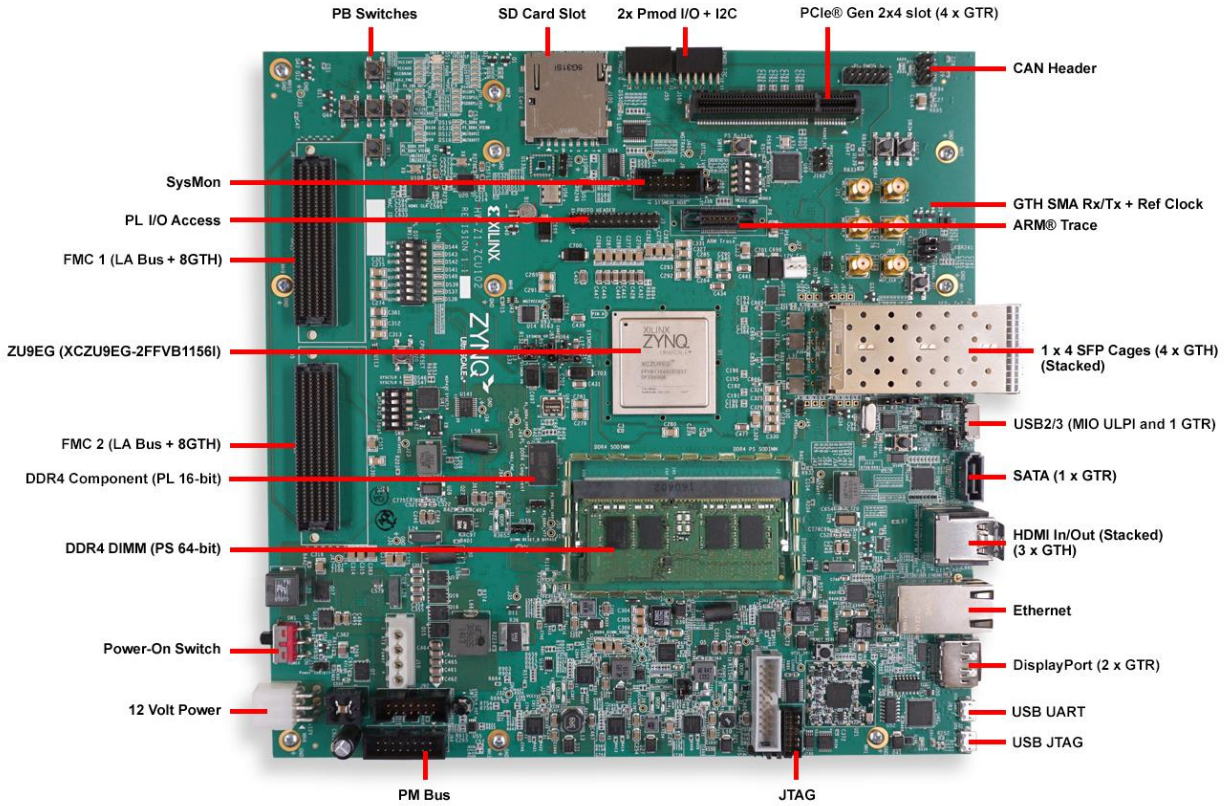
Quantifying impact of memory interactions requires a global view

- Need for ***system level*** exploration of the design space
  - Combinations of memory technology
  - Various memory hierarchies
  - Prototype architectural ideas in detail
  - Potential benefit of near-memory accelerators
- Need to quantitatively evaluate the performance impact on applications – beyond an isolated function
  - Latency impact
  - Scratchpad vs. Cache
  - Cache size to working data set size
  - Byte addressable vs. block addressable
  - Accelerator communication overhead
  - Cache management overhead
  - Operating System overhead

MPSoC can be an effective tool to accelerate memory system investigations

A. K. Jain, S. Lloyd and M. Gokhale, "Microscope on Memory: MPSoC-Enabled Computer Memory System Assessments," 2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), Boulder, CO, 2018, pp. 173-180, doi: 10.1109/FCCM.2018.00035.



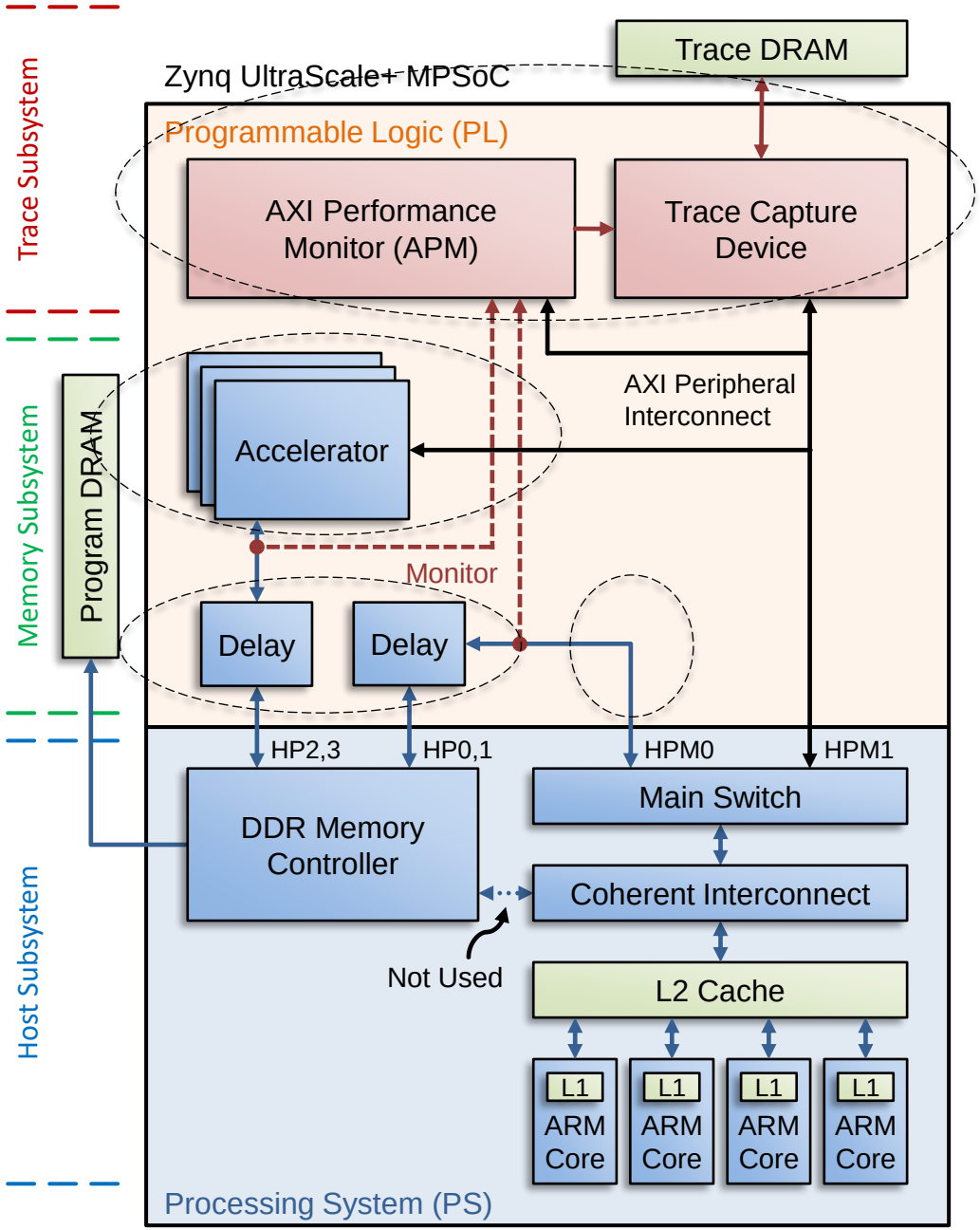


Fidus Sidewinder and ZCU102 development boards with Xilinx Zynq UltraScale+ MPSoC device
Desktop, dedicated evaluation environment

LiME (Logic in Memory Emulator) approach

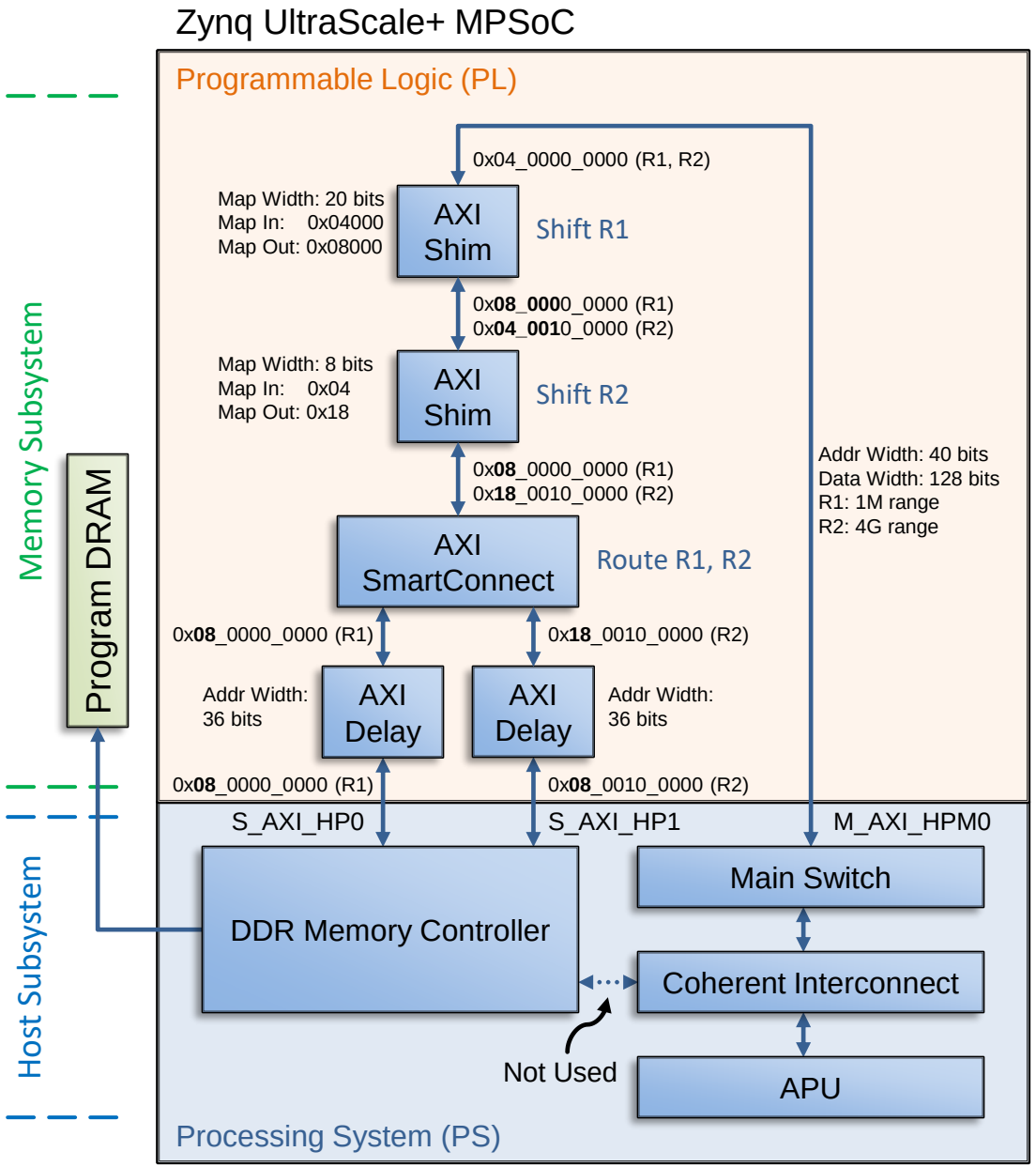
- Use **embedded CPU** and cache hierarchy in Zynq MPSoC to save FPGA logic and development time
- **Loopback** path to route CPU memory traffic through hardware IP blocks
- Emulate the latencies of a wide range of memories by using programmable **delay** units in the loopback path
- Capture time-stamped memory transactions using **trace** subsystem
- Emulate Accelerator, including CPU/Accelerator interactions

Open Source:
https://github.com/LLNL/lime and lime-apps



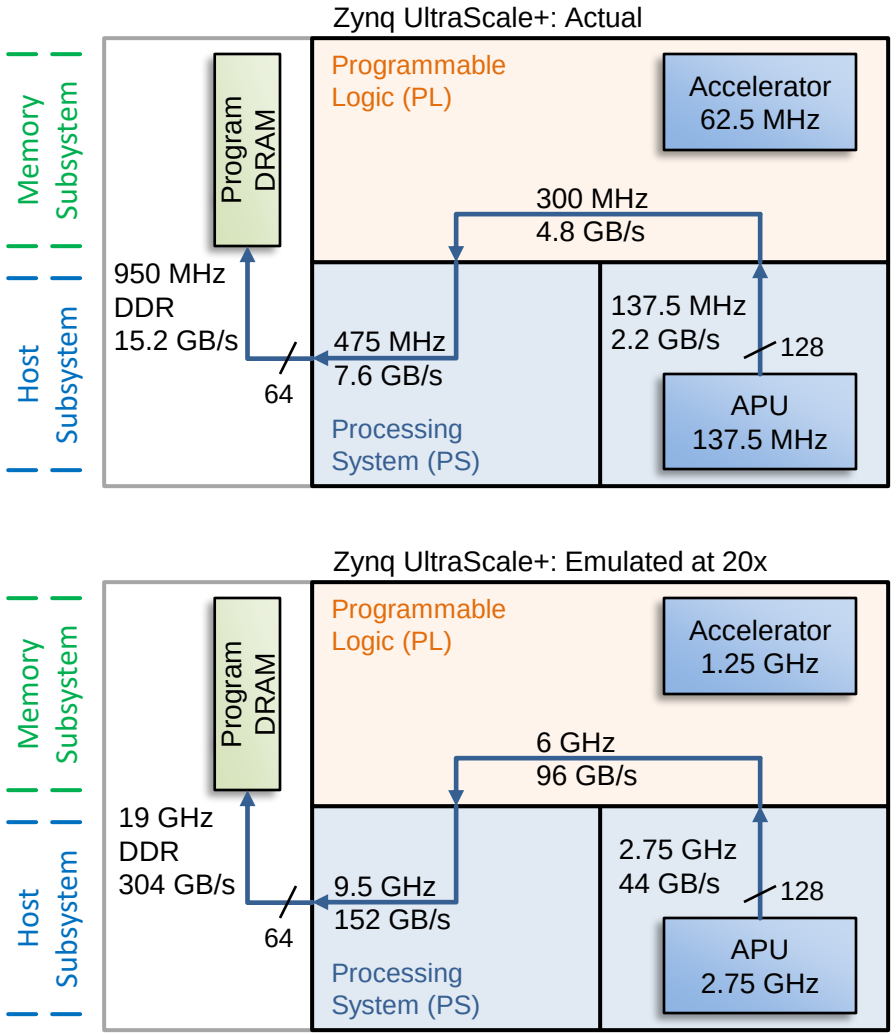
Emulation Method
Delay & Loopback

- Address ranges R1, R2 intended to have different access latencies (e.g. SRAM, DRAM)
- Shims shift and separate address ranges (R1, R2) for easier routing
- Standard AXI Interconnect routes requests through different delay units
- Delay units have separate programmable delays for read and write access



Emulation Method
Clock Domains

- ARM cores are slowed to run at a frequency similar to programmable logic
- A scaling factor of 20x is applied to the entire system
- Other scaling factors can be used depending on the target peak bandwidth to memory
- CPU peak bandwidth is limited to 44 GB/s



Emulation Method
Scaling by 20 Example

Component	Actual	Emulated
Memory Bandwidth (PL)	4.8 GB/s	96 GB/s
Memory Latency (PL)	230 ns	12 ns (too low)
Memory Latency (PL) w/delay	230 ns	12+88 = 100 ns
CPU Frequency	137.5 MHz	2.75 GHz
CPU Bandwidth	2.2 GB/s	44 GB/s
Accelerator Frequency	62.5 MHz	1.25 GHz
Accelerator Bandwidth	Up to 4.8 GB/s	Up to 96 GB/s

Delay is programmable over a wide range: 0 - 174 us in 0.16 ns increments

Emulation Method
Macro Insertion

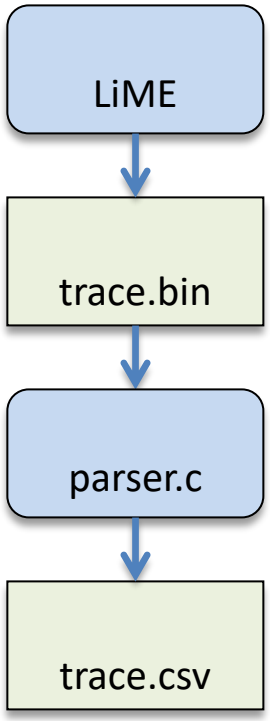
- Insert macros at the start and end of the region of interest (ROI)
- CLOCKS_EMULATE/CLOCKS_NORMAL
  - Modify the clock frequencies and configure the delay units
- TRACE_START/TRACE_STOP
  - Trigger the hardware to start/stop recording memory events in Trace DRAM
- STATS_START/STATS_STOP
  - Trigger the hardware to start/stop the performance monitor counters
- TRACE_CAP
  - Save captured trace from Trace DRAM to SD card

```
CLOCKS_EMULATE
TRACE_START
STATS_START
{
    /* --- MAIN LOOP --- repeat test cases NTIMES times --- */
    scalar = 3.0;
    for (k=0; k<NTIMES; k++)
    {
    times[0][k] = mysecond();
#ifdef _OPENMP
#pragma omp parallel for
#endif
    for (j=0; j<STREAM_ARRAY_SIZE; j++)
        c[j] = a[j];

    times[0][k] = mysecond() - times[0][k];

    times[1][k] = mysecond();
#ifdef _OPENMP
#pragma omp parallel for
#endif
    for (j=0; j<STREAM_ARRAY_SIZE; j++)
        b[j] = scalar*c[j];

    times[1][k] = mysecond() - times[1][k];

    times[2][k] = mysecond();
#ifdef _OPENMP
#pragma omp parallel for
#endif
    for (j=0; j<STREAM_ARRAY_SIZE; j++)
        c[j] = a[j]+b[j];

    times[2][k] = mysecond() - times[2][k];

    times[3][k] = mysecond();
#ifdef _OPENMP
#pragma omp parallel for
#endif
    for (j=0; j<STREAM_ARRAY_SIZE; j++)
        a[j] = b[j]+scalar*c[j];

    times[3][k] = mysecond() - times[3][k];
    }
}
STATS_STOP
TRACE_STOP
CLOCKS_NORMAL
TRACE_CAP
```

Trace capture subsystem stores memory accesses for analysis

- Uses 2nd DRAM so that memory system of device under test is unaffected
- Captures
  - Timestamp
  - Transaction type
  - Source of request (CPU core, cache pre-fetch, accelerator)
  - Type of memory
    - Emulator supports two memory regions with separate read and write latencies
    - Total of 8 individual delays

Memory Trace Capture



Source	Type	Address	Length	AXI ID	Time
0	W	0x400082F80	64	525	481545
0	W	0x400082FC0	64	525	482101
0	R	0x4002011C0	64	1165	482432
0	R	0x400201200	64	1293	482441
0	R	0x400201240	64	1037	487379
0	R	0x400201280	64	1037	492539
1	W	0x400080000	8	3	498523
1	R	0x400082000	16	0	493495
1	W	0x400080008	8	3	498557
1	W	0x400080010	8	3	503270
1	W	0x400080018	8	3	503304
1	W	0x400080020	8	3	503400

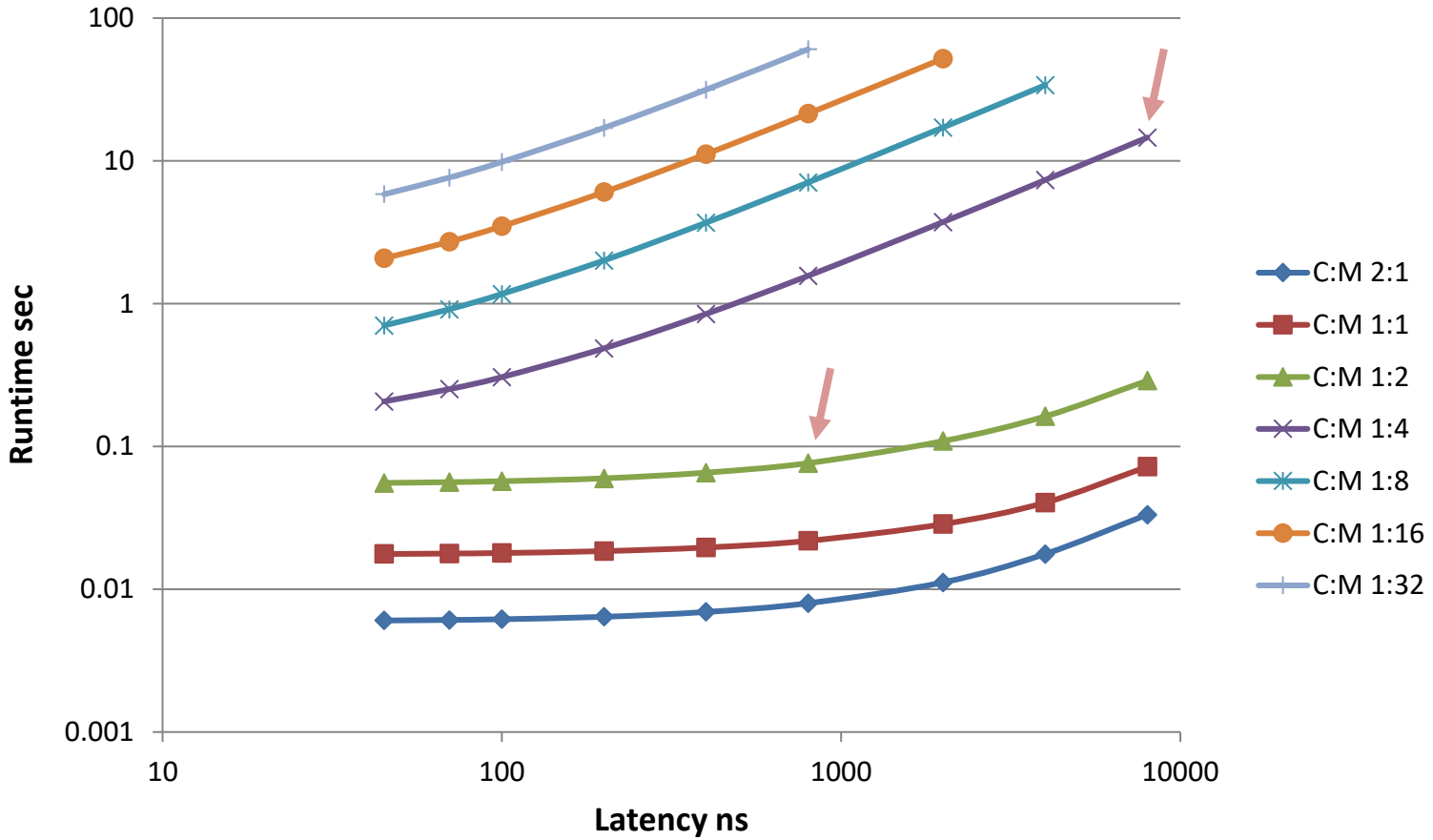
CPU = 0, Accelerator = 1

Each count represents 0.16 ns

Can persistent memory serve as main memory for dense, regular access patterns?

R:W 1:1, Regular Access Pattern

DGEMM



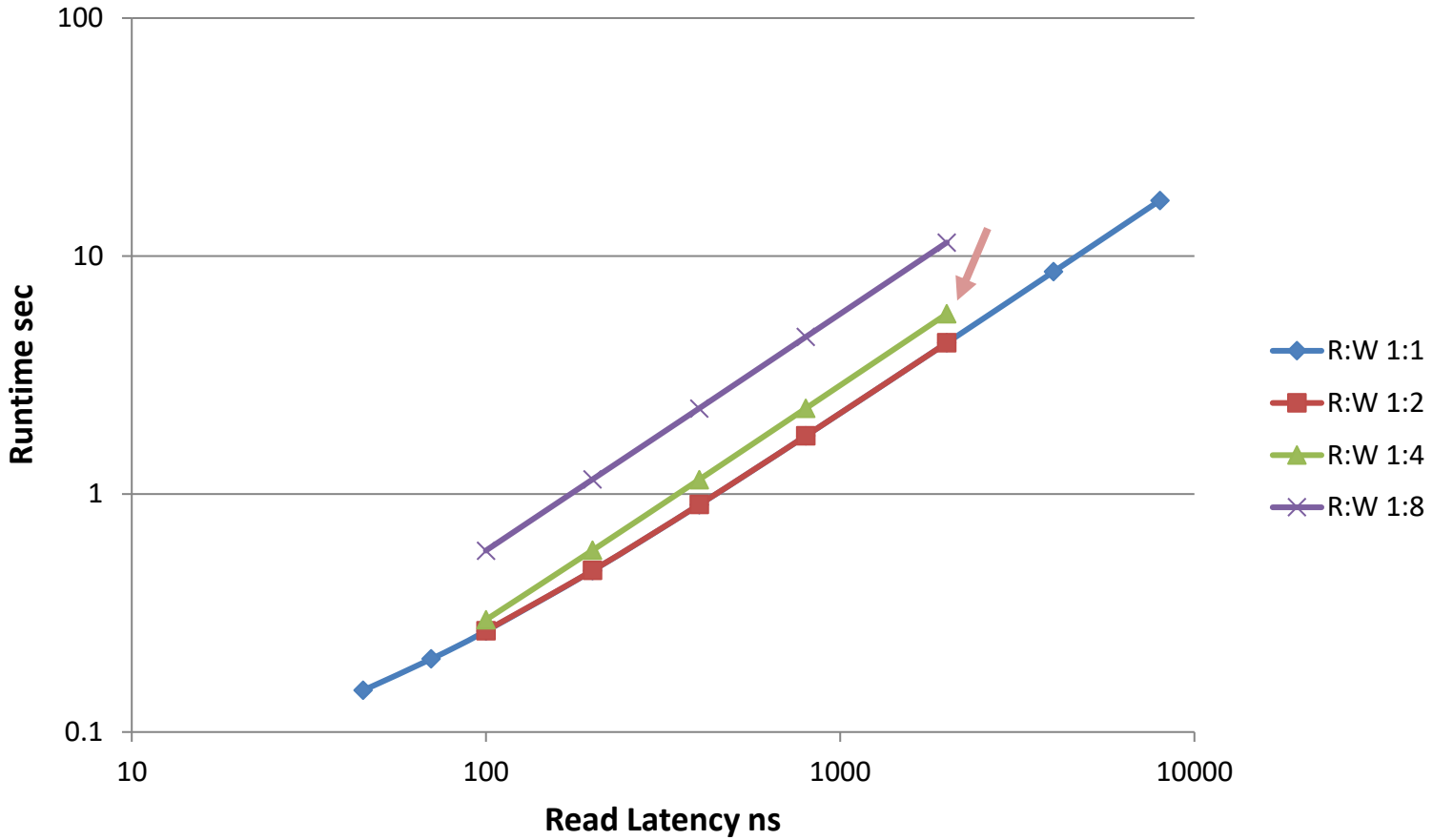
FPGA-accelerated design space evaluation:
9 latency levels and 7 cache-to-memory ratios

- At cache to memory ratios of 1:2 and lower, latency up to 800 ns can be tolerated
- At cache to memory ratios of 1:4 and higher, runtime increases linearly with latency
- Matrix multiply kernels for small (perhaps blocked) matrices can tolerate SCM latency

Can persistent memory serve as main memory for dense, regular access patterns?

C:M 1:1024, Irregular Access Pattern

RandomAccess



9 latencies, 4 read-write ratios

- A read to write latency ratio up to 1:4 has little impact on performance
- Direct linear correlation between memory latency and runtime
- Concurrent threads could in aggregate compensate for longer SCM latency

Let’s add accelerators to the mix

- Near memory data rearrangement engine for gather/scatter
  - Batch operation
  - Indexed A[B[i]]
  - Strided A[i+c]
- Key/Value Store lookup accelerator
  - Gather values for batch of keys
- Floating point compression pipeline
  - Tailored to scientific 1D, 2D, 3D data arrays
  - Based on zfp library

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

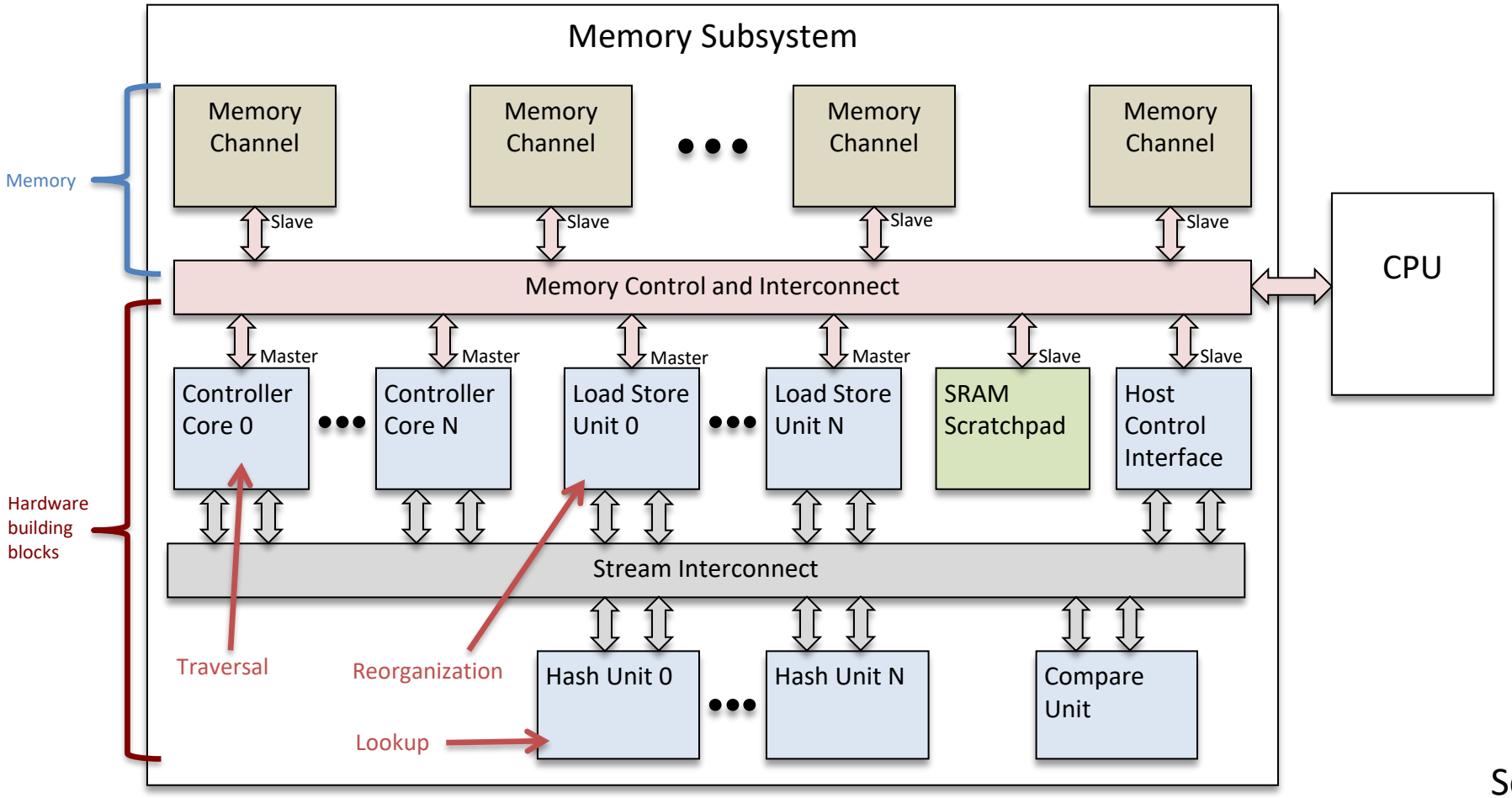
A. K. Jain, S. Lloyd and M. Gokhale, "Performance Assessment of Emerging Memories Through FPGA Emulation," in IEEE Micro, vol. 39, no. 1, pp. 8-16, Jan.-Feb. 2019, doi: 10.1109/MM.2018.2877291.

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

Near memory data rearrangement can help applications with sparse, irregular access patterns

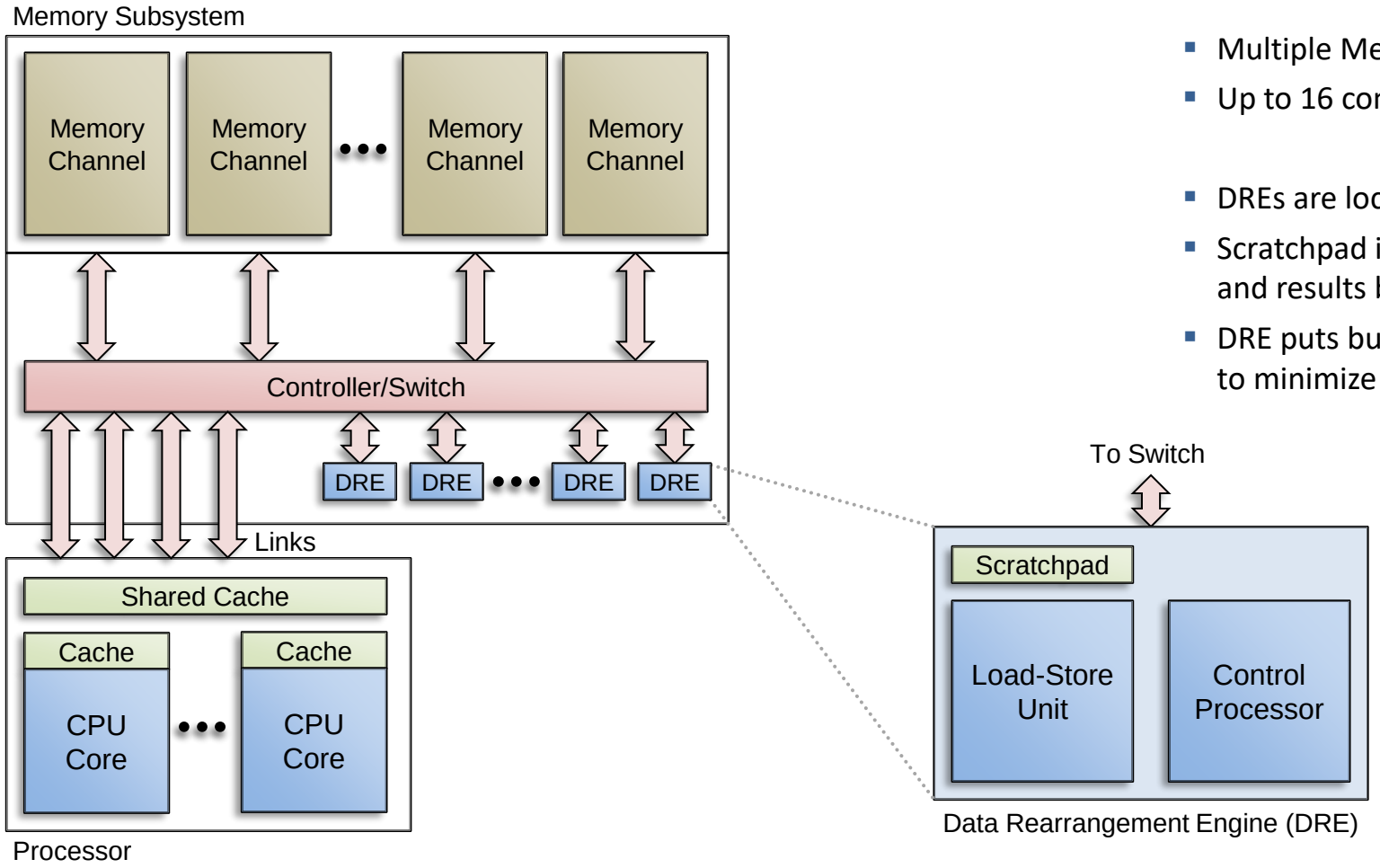
- Memory bandwidth to CPU limiting many applications
  - Trend is downward with many-core processors
  - 8 GB/s per core Intel Xeon X5550, Q1'09
  - 5.6 GB/s per core Intel Xeon E7-4890 v2, Q1'14
  - Large caches and more memory channels may help some applications
- Data-intensive applications
  - Large application working sets
  - Unstructured and irregular data access patterns
  - Manipulate complex, linked data structures
  - Benefit less from CPU caches
  - Small portion of cache line actually used by CPU
- Approach
  - Rearrange and reduce data near the source
  - Move less data to CPU for energy and performance benefit
  - Rearrangement hardware is generally applicable

Heterogeneous architecture targets interconnected, near memory, configurable fixed function units



Scott Lloyd

Use Cases
Evaluation of Near-Memory Data Rearrangement Engine

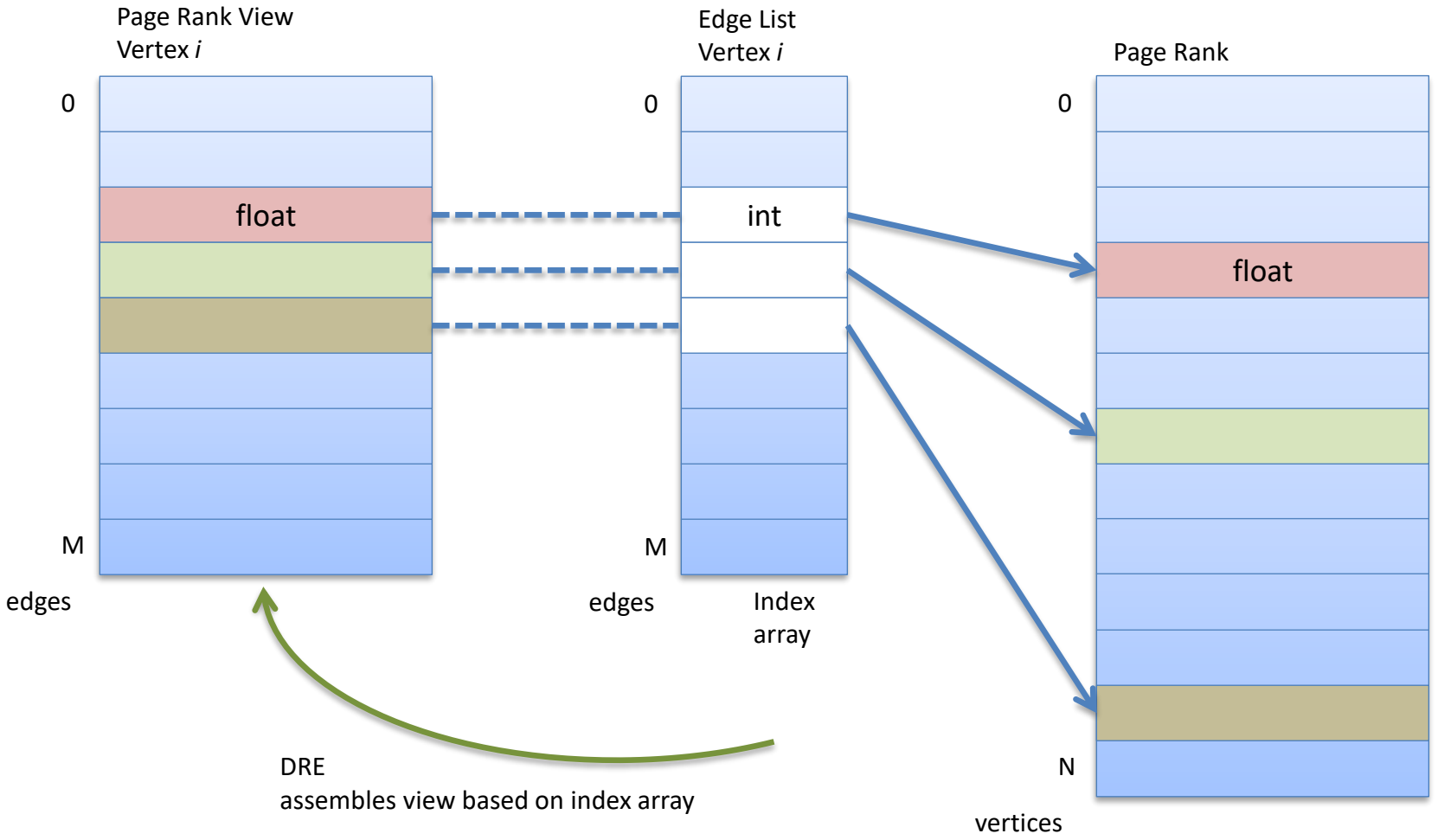


- Multiple Memory Channels
- Up to 16 concurrent memory requests

- DREs are located in the Memory Subsystem
- Scratchpad is used to communicate parameters and results between CPU and accelerator
- DRE puts buffer data into a cache-friendly layout to minimize wasted memory bandwidth

Scott Lloyd and Maya Gokhale, "In-memory data rearrangement for irregular, data intensive computing," IEEE Computer, August 2015, v. 48, no. 8, pp. 18–25.

PageRank: DRE gathers a “view” of page ranks G[E[i]]

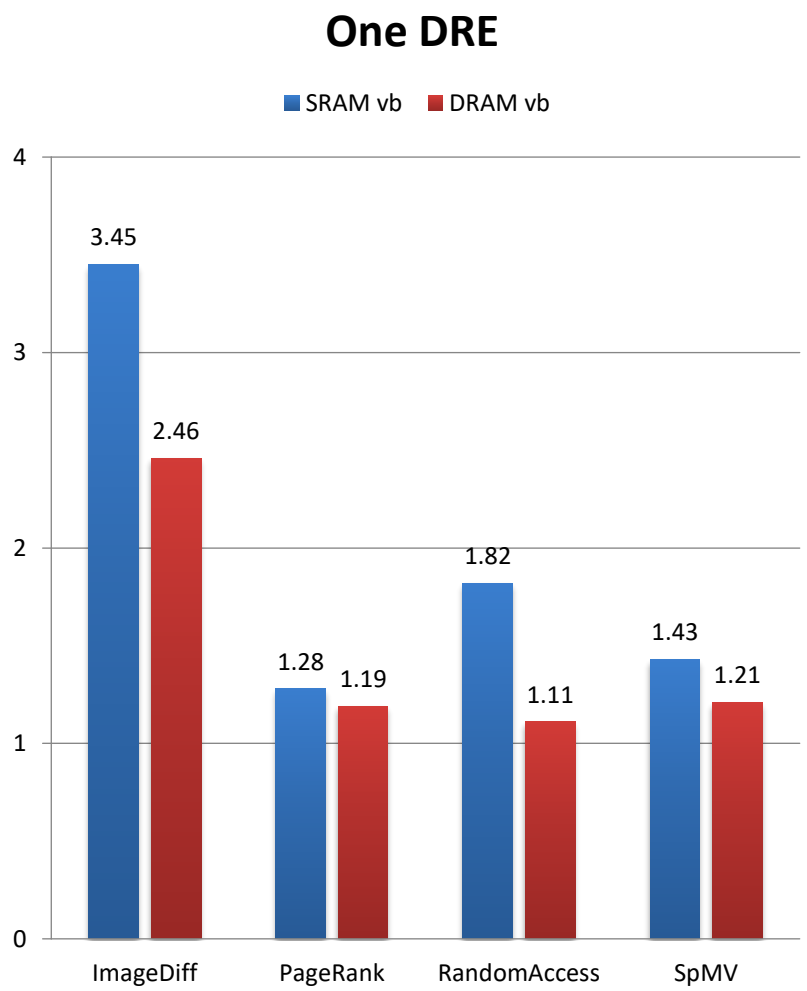


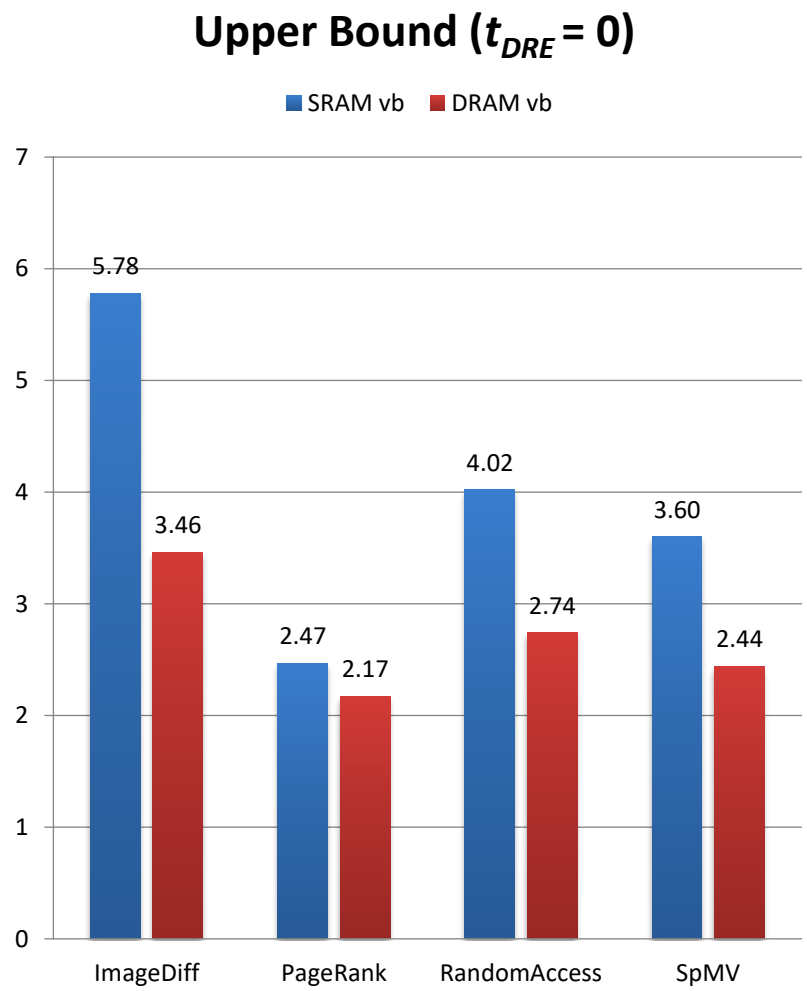
API

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);

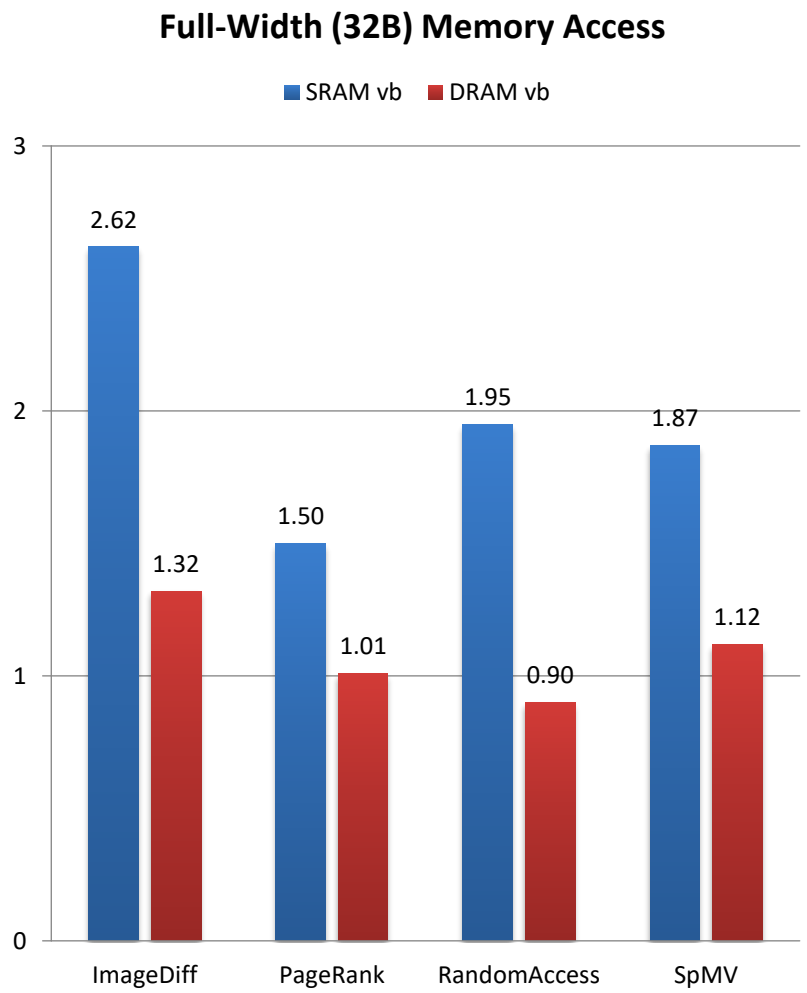
On HMC-like memory, should near memory buffer be SRAM or DRAM?

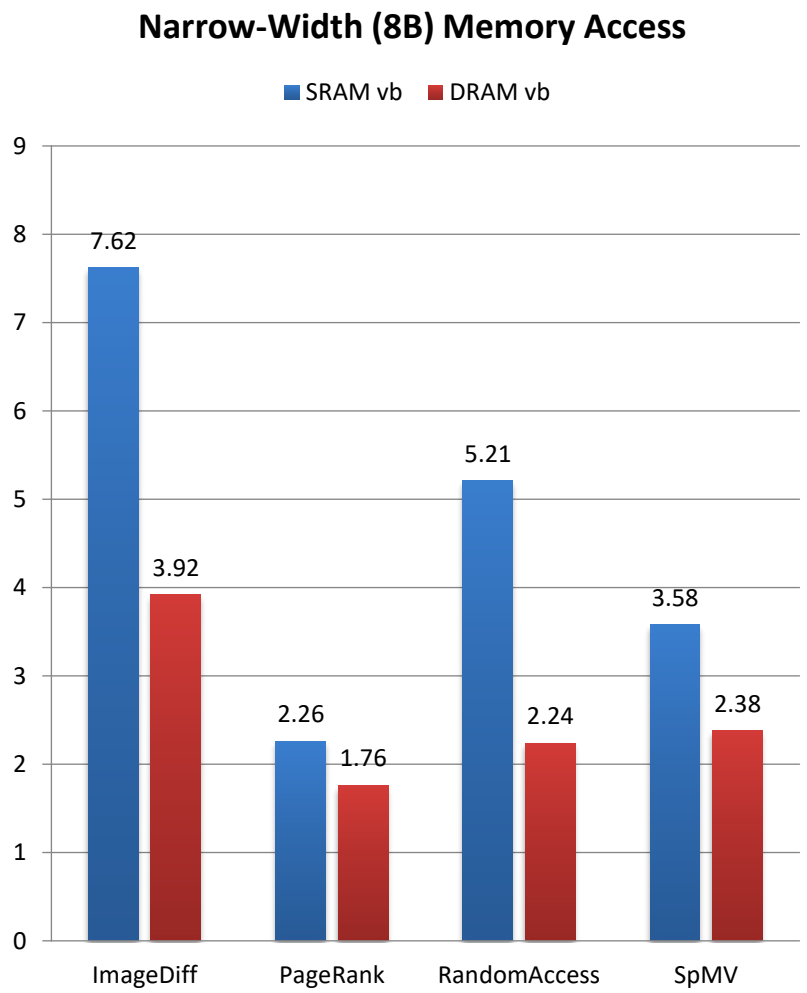




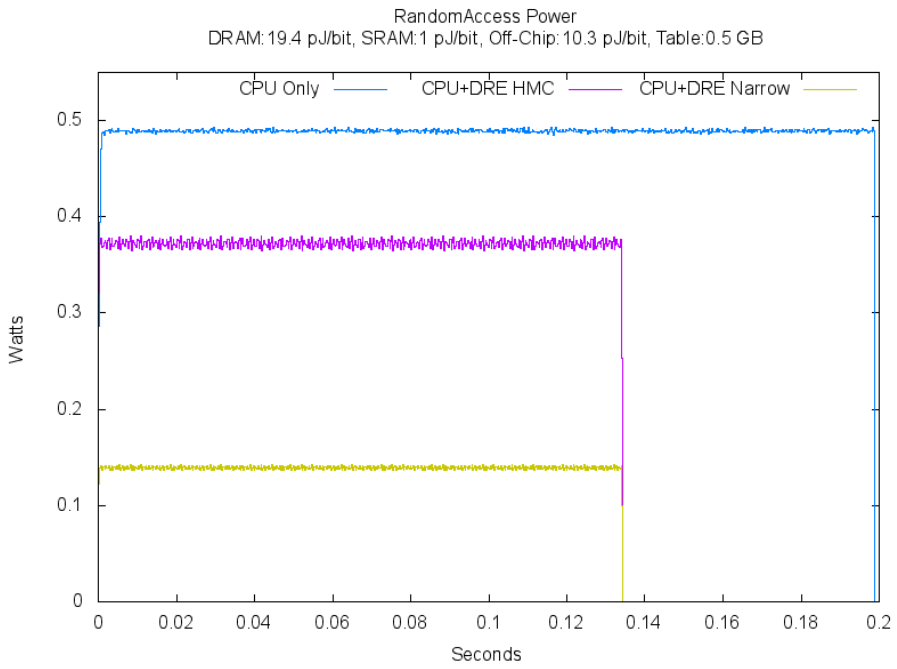
Is there energy savings in using a narrow width memory?

Simple model: 19.4 pJ/bit for DRAM, 1.0 pJ/bit for SRAM, and 10.3 pJ/bit for off-chip traversal

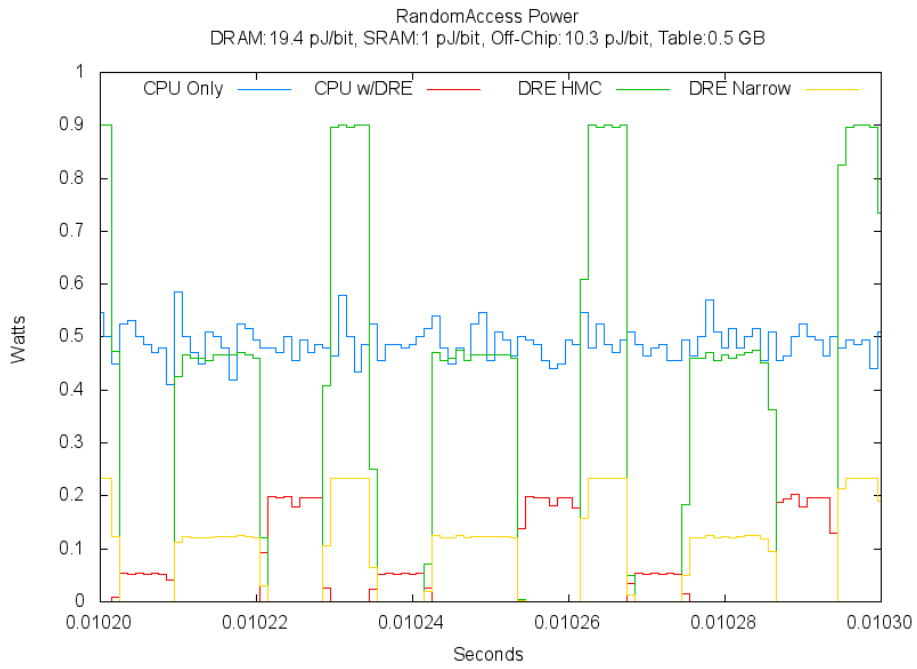




RandomAccess Power Profile



(a) The entire run.

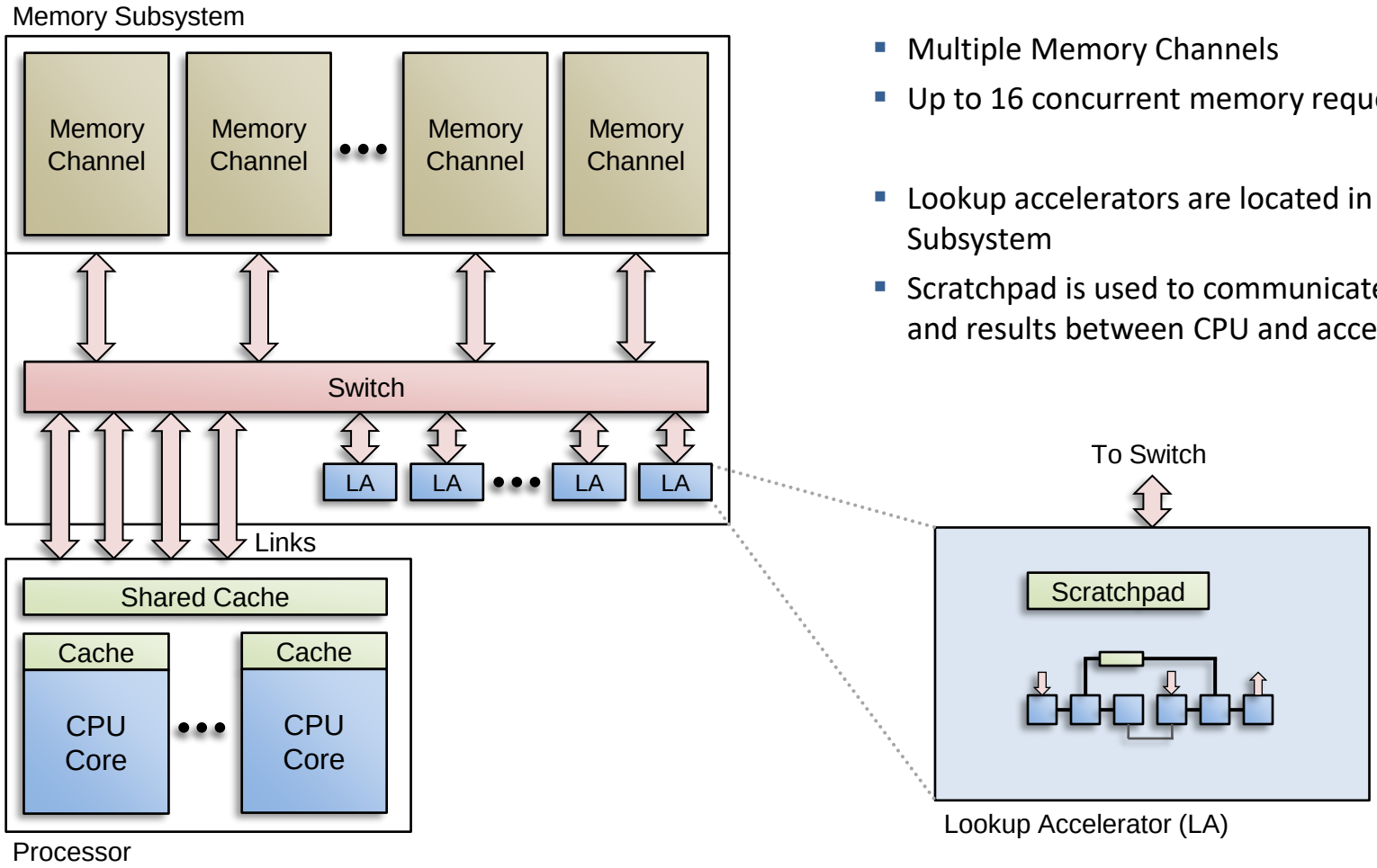


(b) Enlarged segment of the run.

What have we learned about Data Rearrangement Engine?

- One DRE provides a benefit
  - Even when data rearrangement is not overlapped with CPU computation
  - Computation can take advantage of vector and SIMD units
  - View buffer contains only data that is needed by the CPU
  - Speedup – up to 3.45x (SRAM view buffer)
  - Reduces energy – up to 7.62x (Narrow DRAM access)
- An SRAM view buffer provides an advantage over DRAM
  - Speedup – up to 1.64x
  - Reduces energy – up to 2.17x
- Narrow-width (8B) memory access uses less energy than Full (32B)
  - Reduces energy – up to 2.91x
- Further speedup expected based on upper-bound results
  - Multiple cores
  - Multiple DREs
  - Overlapped computation with data rearrangement

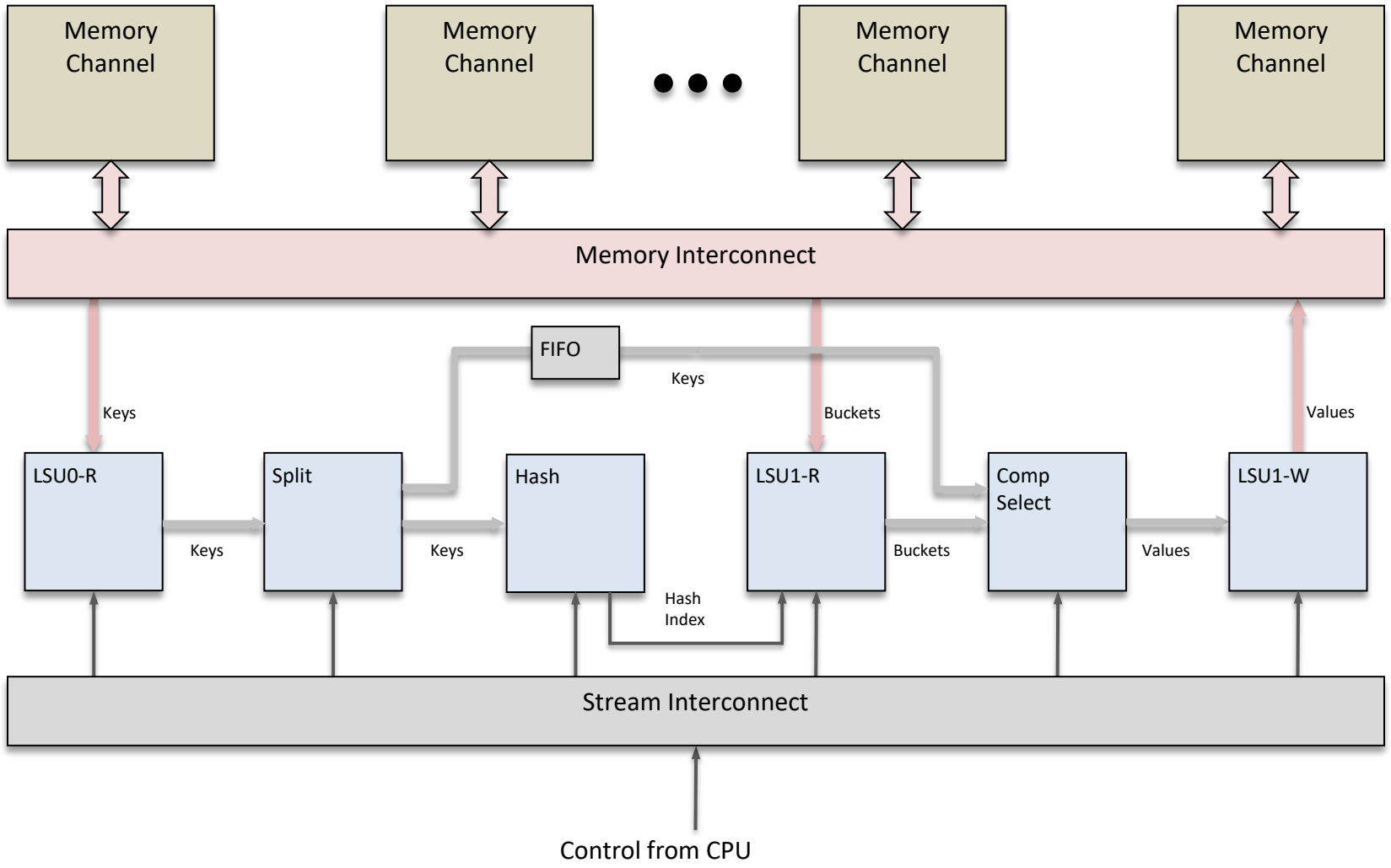
Near memory key/value store lookup accelerator



- Multiple Memory Channels
- Up to 16 concurrent memory requests

- Lookup accelerators are located in the Memory Subsystem
- Scratchpad is used to communicate parameters and results between CPU and accelerator

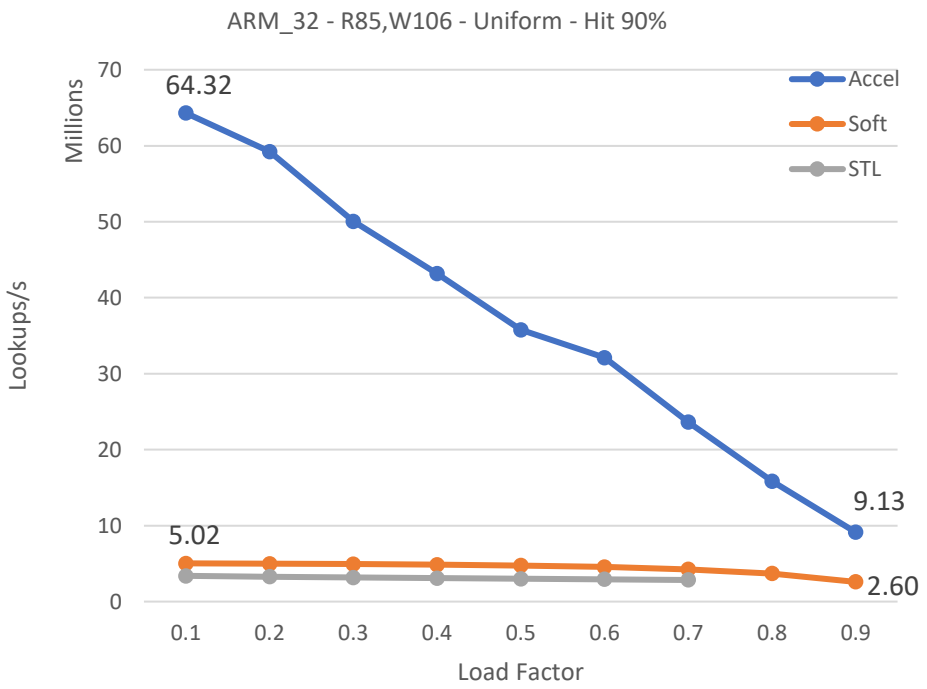
Lookup pipeline connects simple IP blocks



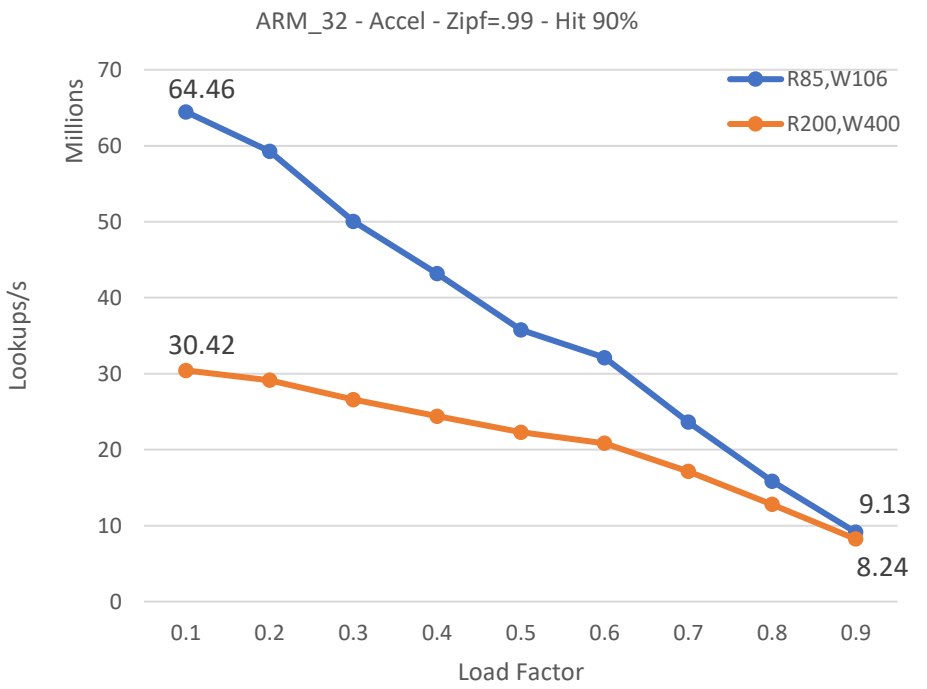
Emulator predicts lookup performance over large design space

90% hit rate

Accelerator vs. Software



Low vs. Moderate Latency



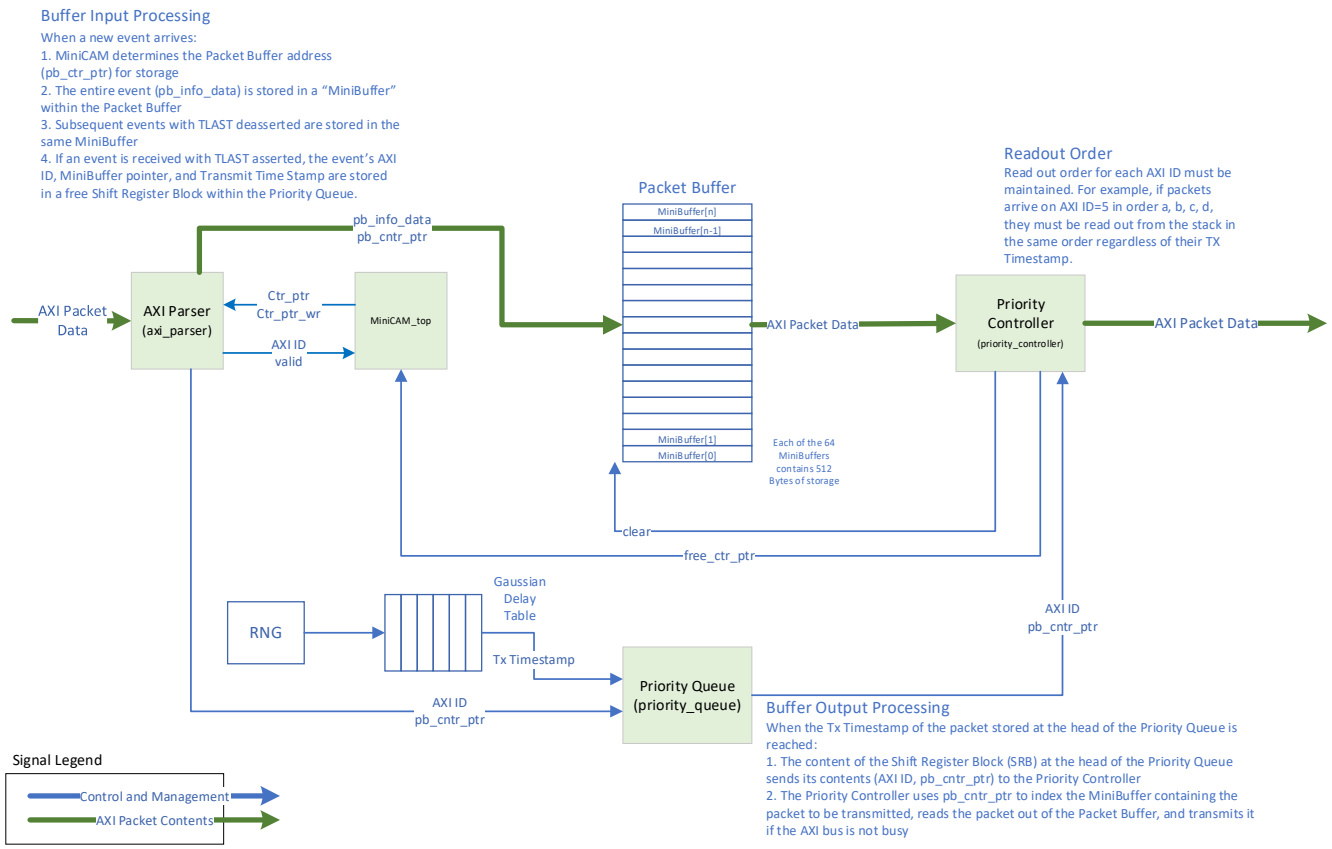
- Accel. performance does not vary with hit rate or key repeat frequency (scans entire PSL)
- Accel. performance decreases with increasing load (PSL) and memory latency
- Accel. performance comes from parallelism and more outstanding near memory requests
- Software is slower because of serialization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

Great insights, but what about …

- Emulator models an idealized memory.
  - Can we generalize?
- Emulator studies focused on single core + single accelerator.
  - What about multiple accelerators with realistic memory behavior?
- Building accelerators in RTL is time consuming
  - Can we have a higher level of abstraction and still get meaningful, quantitative answers?

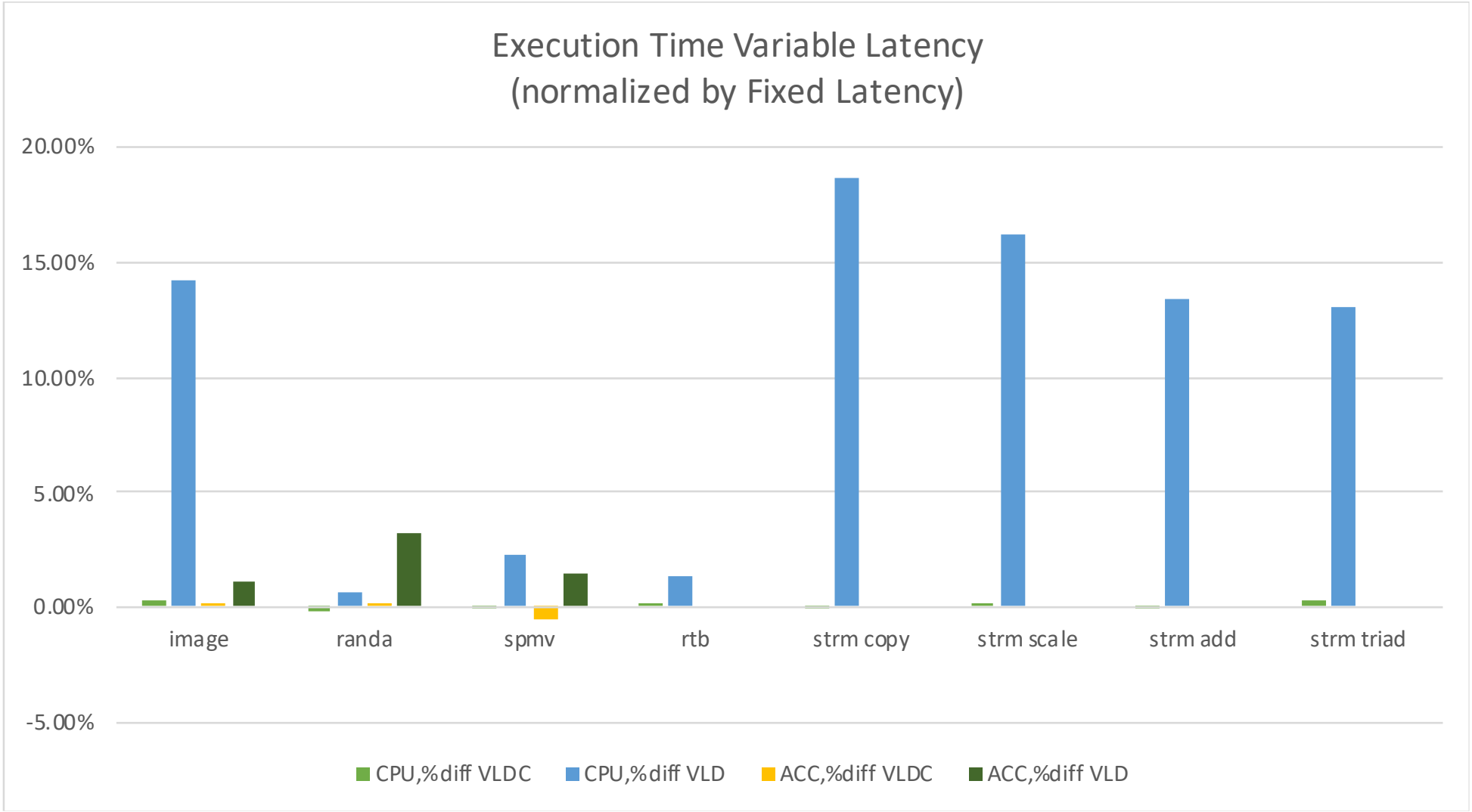
Variable Latency Model improves accuracy of predictions

- Fixed latency delay model is simplistic (simulation 101!)
- Memories show considerable variability in access latency
- Variable latency delay (VLD) unit can improve prediction accuracy
- Delay profiles stored in table
- Each memory access delay amount is chosen randomly from a table



Chris Macaraeg

Variable latency reduces performance of some applications



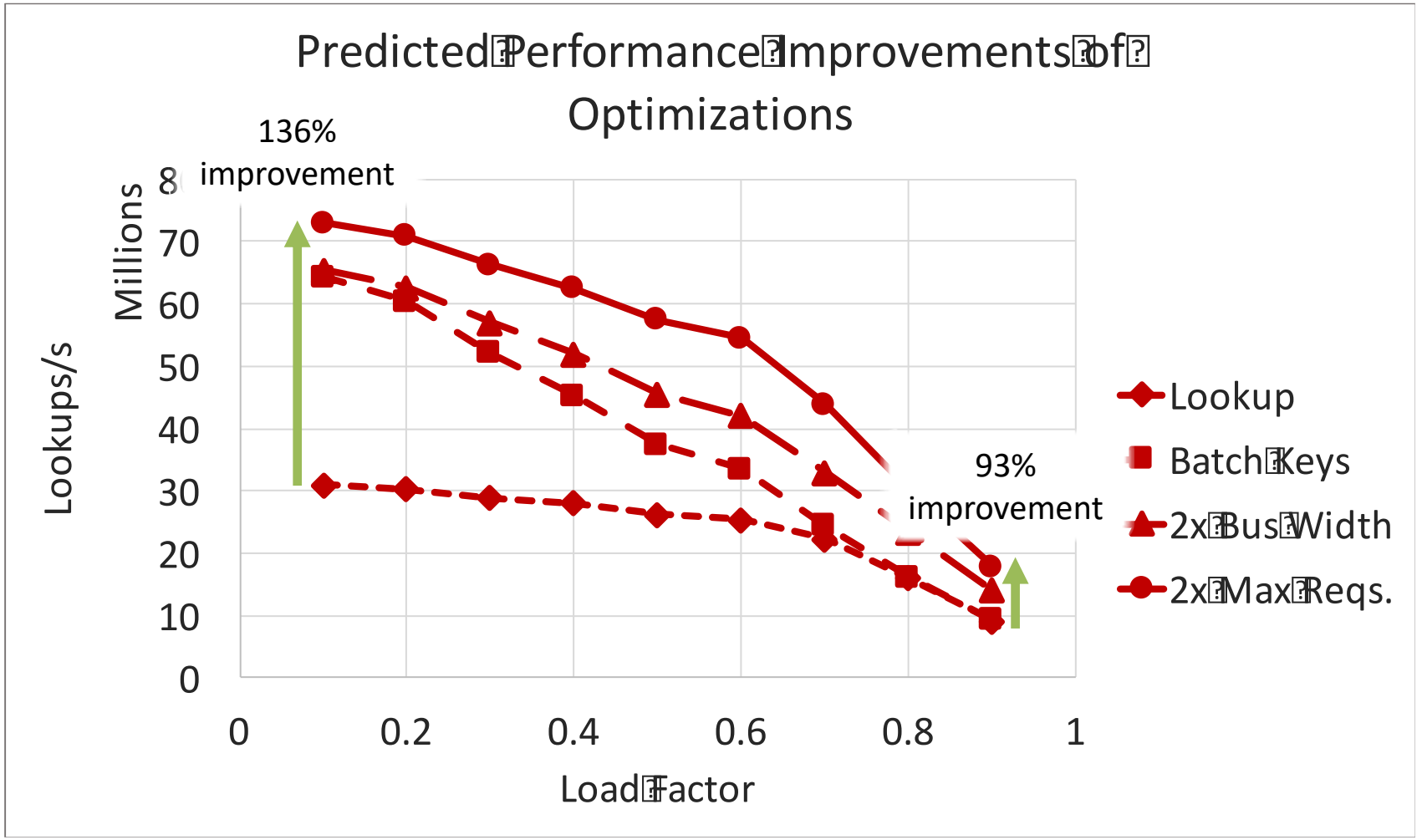
Best of both worlds: combine insights from FPGA emulator with software simulator to study complex scenarios

- Model accelerator, CPU, and LiME memory model in Structural Simulation Toolkit (SST)
- LiME data
  - Capture memory access traces through LiME
  - Use memory traces to determine model parameters
- SST capabilities
  - Plugin detailed memory model: use HMC-Sim to simulate a Hybrid Memory Cube (HMC)
  - Can scale up to an arbitrary number of CPUs and accelerators

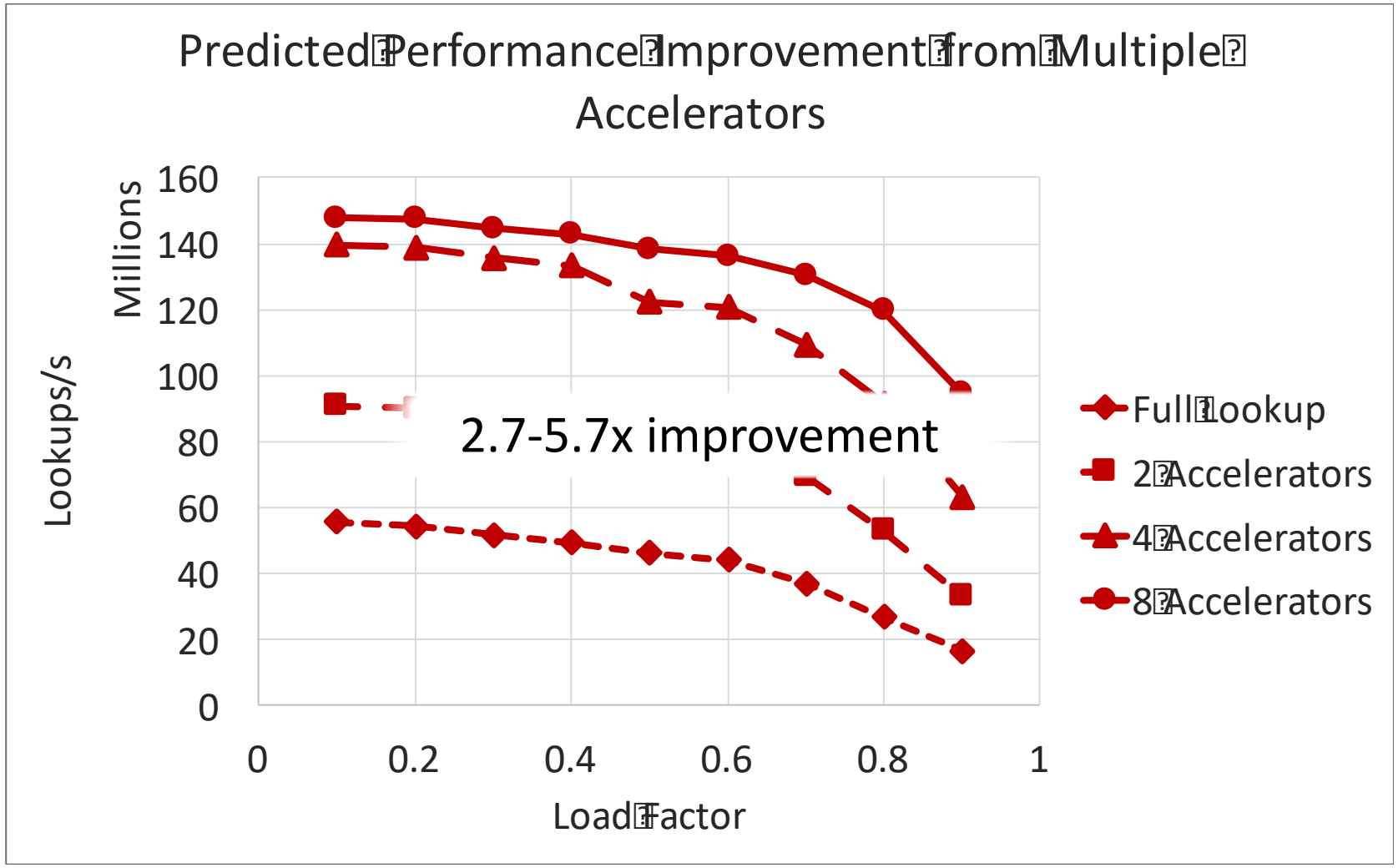
https://github.com/sstsimulator

Joshua Landgraf, Scott Lloyd and Maya Gokhale. 2017. Combining Emulation and Simulation to Evaluate a Near Memory Key/Value Lookup Accelerator. In Open Source Computing Workshop, SC17. Available at https://www.researchgate.net/publication/330369517_Combining_Emulation_and_Simulation_to_Evaluate_a_Near_Memory_KeyValue_Lookup_Accelerator

Exploit low level memory features: optimization predictions for lookup accelerator



Scaling Predictions



The tool problem: how can we speed up design of accelerators?

- Goal is to find a simulation to emulation path:
  - Enable full system evaluation combining software and hardware
  - Provide flexible simulation during initial design
  - Offer synthesis of promising design options for fast emulation
  - Avoid writing two models, one for simulation and another for emulation
- LiME was implemented in RTL with heavy inclusion of Xilinx IP blocks: fifos, data mover, APM, AXI stream, AXI lite
  - Continual battle with the tools
  - C++ HLS didn't work well for our use case
    - We want to design a system of communicating processes
    - We want to develop a library of building blocks stitched together with custom stream interconnect
    - Our fixed function units need independent, concurrent accesses from multiple modules to shared DRAM
  - We need a communicating process model
    - HLS from C/C++ tries to parallelize sequential programming model
    - OpenCL and other parallel languages with data parallel model make it difficult to describe fixed function units

SystemC Language

https://www.accellera.org/downloads/standards/systemc
https://github.com/accellera-official/systemc

- Modeling and simulation language of complex System on Chip hardware architectures
  - Event-driven simulation hardware components
- Multiple levels of simulation
  - **Register/Transfer level**
  - Behavioral
  - Transaction
- Parallel communicating process model
  - Timing, event sequencing, process concurrency
- C++ library of classes and macros
- Hierarchical model
  - Modules, ports
- Scheduling and synchronization of concurrent processes
- Separation of computation (process) and communication (channel)
- Hardware oriented data types
  - Digital logic
  - Fixed point arithmetic

SystemC in action

```
SC_MODULE(find_emax)
{
      typedef typename FP::expo_t expo_t;
      /*-------- ports --------*/
      sc_in<bool> clk;
      sc_in<bool> reset;
      sc_stream_in <FP>     s_fp;
      sc_stream_out<FP>     m_fp;
      sc_stream_out<expo_t> m_ex;
/*-------- modules --------*/
      sfifo_cc<FP,2*DIM+1,RLEVEL> u_que_fp;
      sreg<expo_t,FWD_REV,RLEVEL> u_reg_ex;
```

```
{
 bool last = (count.read() == 0);
 FP fp = s_fp.data_r();
  expo_t expo;
  if (fp.expo == 0 && fp.frac == 0) {
    expo = fp.expo;
  } else {
    expo = fp.expo + expo_t(1);
  }
  if (c_sync) {
    if (last) {count = fpblk_sz(DIM)-1;}
      else {count = count.read() - 1;}
  }
  if (emax_v && c_ex.ready_r()) {
    if (s_fp.valid_r()) emax = expo; else emax = 0;
  } else if (s_fp.valid_r() && expo > emax) {
    emax = expo;
  }
  if (emax_v && c_ex.ready_r()) emax_v = false;
  else if (c_sync && last) emax_v = true;
  }
```

SystemC for FPGA System on Chip

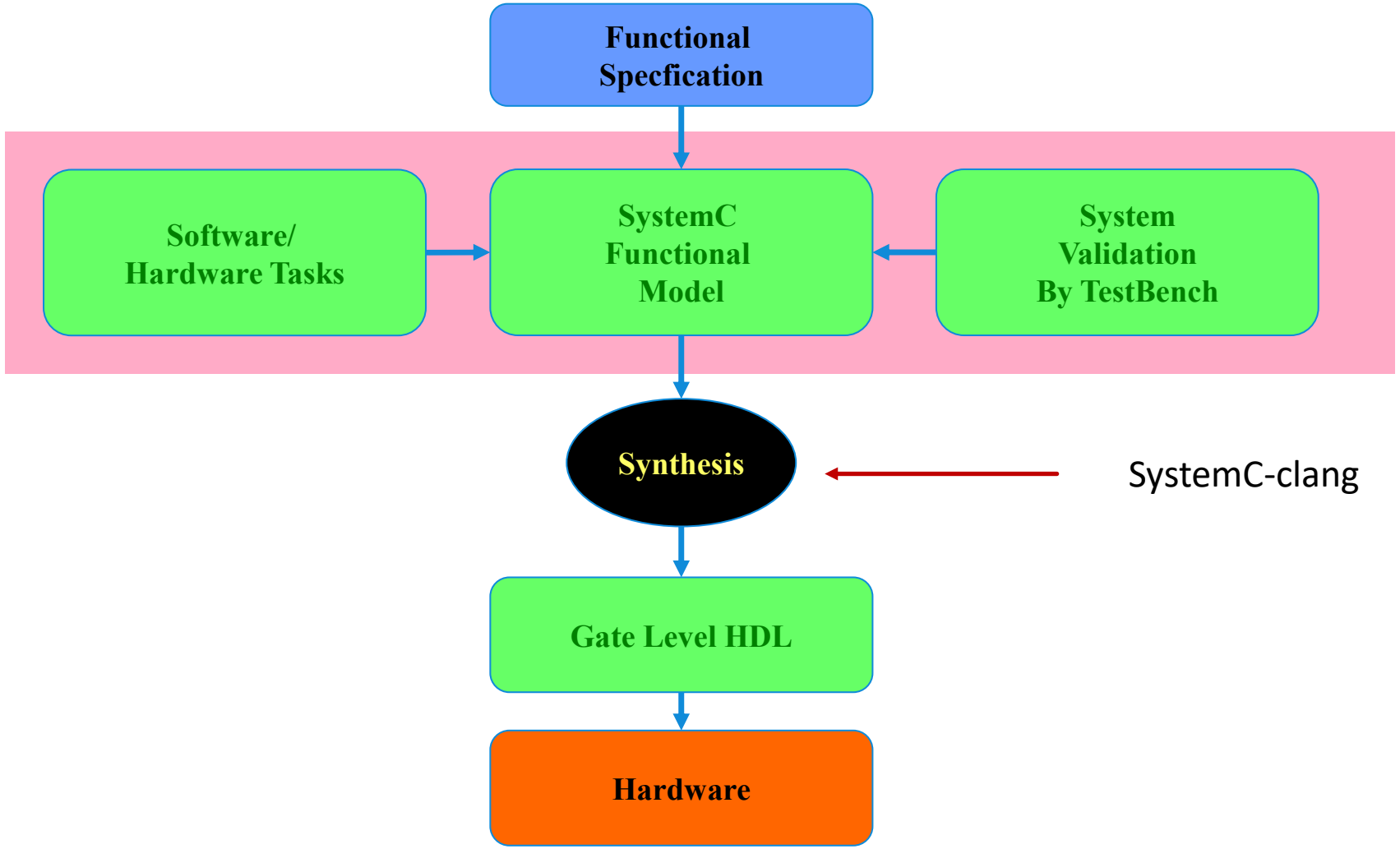
- FPGA HLS tools focus on C/C++/OpenCL, lack equivalent robustness for SystemC
- Industrial strength SystemC synthesis tools cost $$$$$
- Let's work on a community effort on an open source SystemC to RTL compiler!
- Leverage LLVM/CLANG C++ front end
- Identify and consolidate synthesizable SystemC constructs in CLANG AST
- Translate SystemC processes to RTL
- On-going open source effort

Clang: front end for LLVM
https://clang.llvm.org

- Language front-end and tooling infrastructure for languages in the C language family
- Supports C++11, C++14, C++17
- Modular library based architecture
- Well documented internal data structures and AST
- Tools to process AST: visitor pattern, traverse, matchers
- Code examples of clang usage

SystemC Design Flow

https://cas.tudelft.nl/Education/courses/et4351/SystemC1.pdf



Benefits:

- Open source
- Iterative refinement
  - Functional to RTL
- Suitable for SoC design
  - Not just CPU/Accelerator
- C++ “carrier” language enables easy sw/hw co-design

Issues:

- Simulation language
- Synthesis requires vendor tools
- FPGA tools immature and buggy
- ASIC tools $$$$$

Vendor tools don’t handle complex C++ patterns very well: type hierarchies, typedefs, constexpr, etc.). We leverage Clang technology.

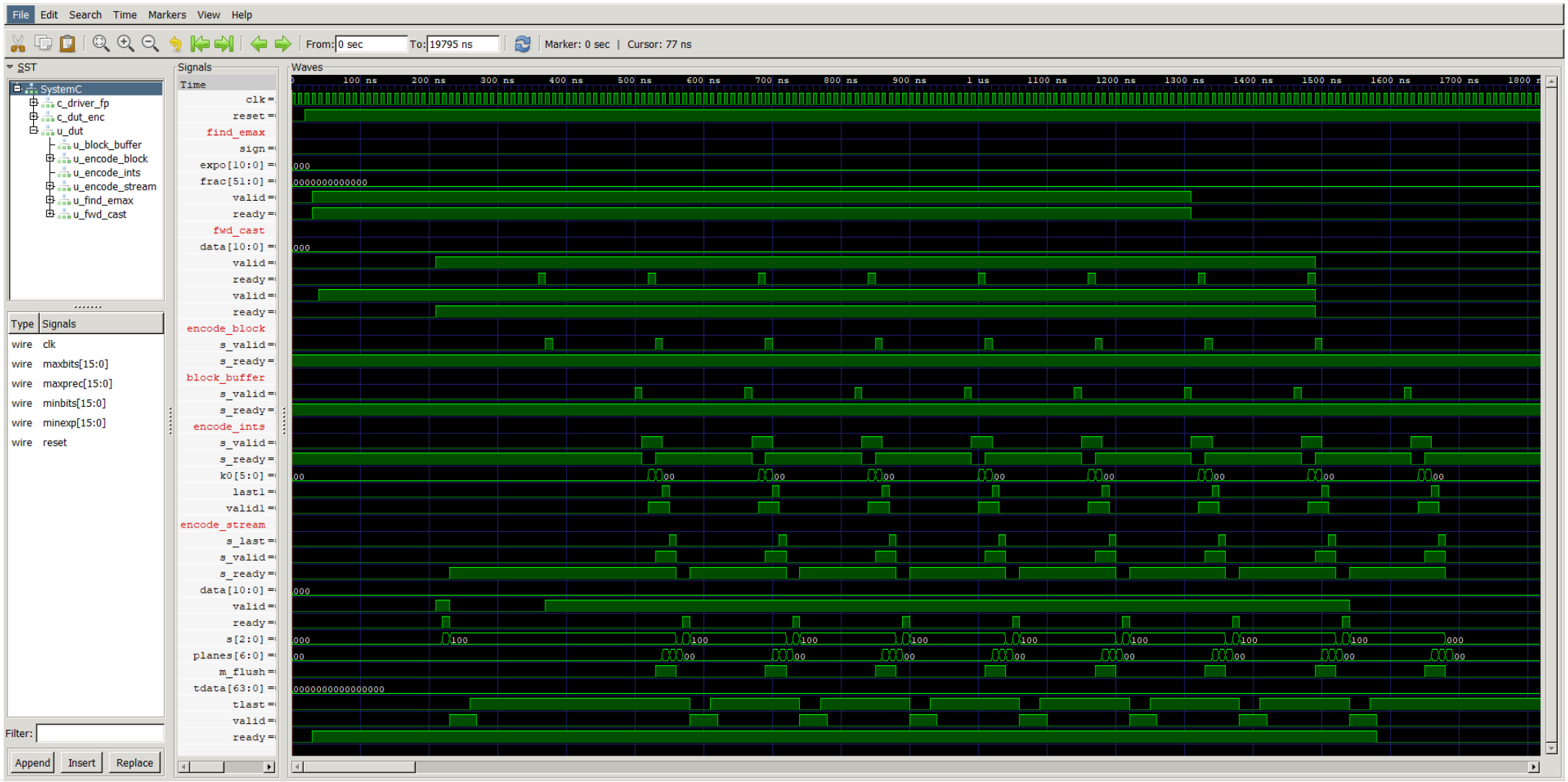
Systemc-clang with HDL plugin: open-source translator based on clang

https://github.com/anikau31/systemc-clang

- Translate synthesizable SystemC to HDL
- Build from prior work by U Waterloo
  - Leverage clang parsing and semantic analysis
    - Parse and build AST, type info from complex templated data types
  - Traverse AST to identify SystemC constructs
    - Objects: SC_MODULE, SC_METHOD
    - Templated data types: sc_in, sc_out, sc_signal
  - Optimize simulation
- Team with Waterloo to extend systemc-clang for synthesis
  - (Waterloo) Improved template class handling, type infrastructure
  - (LLNL) Add HDL plugin to generate HDL IR for modules and methods
  - (Waterloo) Translate HDL IR to Verilog, test on FPGA board

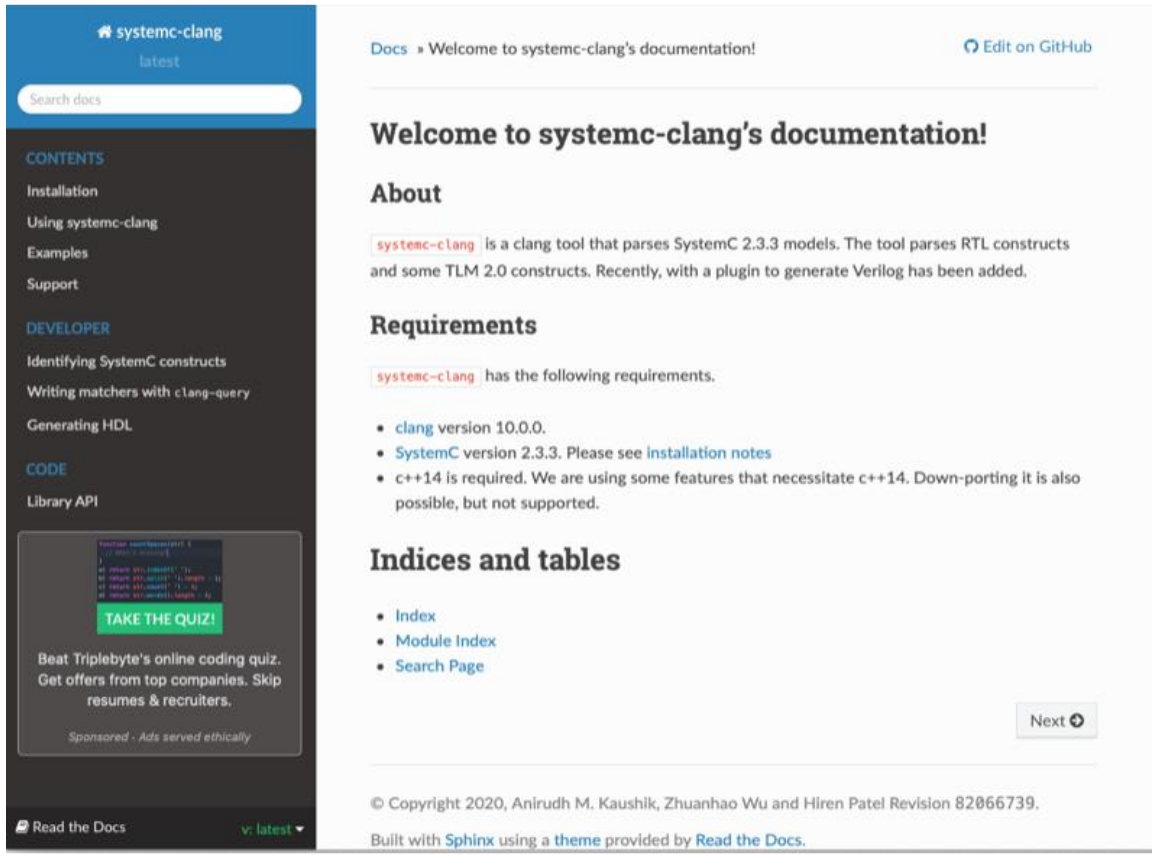
A. Kaushik and H. D. Patel, "Systemc-clang: An open-source framework for analyzing mixed-abstraction SystemC models," Proceedings of the 2013 Forum on specification and Design Languages (FDL), Paris, France, 2013, pp. 1-8.

Full open source tool chain to generate RTL



Recent progress

- Our test cases are taken from a floating point compression hardware IP library by Scott Lloyd https://github.com/LLNL/zhw
  - Complex pipeline in synthesizable SystemC
  - FPGA vendor tool was unable to synthesize simple library components to hardware
- Systemc-clang automatically translates SC_METHODs in modules of "zhw_encode" to hardware that runs correctly on FPGA
  - Xilinx tool fails on this and simpler modules
- Collaborators welcome!



Summary

- FPGAs have seen as dramatic innovation in architecture as CPUs
- FPGA applications are diverse: no one killer app
- Leveraging MPSoC hard processors enables fast design space exploration of fixed function near memory units
- Emulation+simulation enables larger design space exploration
- Open source tools can enable wider adoption of reconfigurable computing technologies

Emulator Team

Scott Lloyd
All aspects of the implementation

Abhishek Jain
Port to Zynq UltraScale+

Chris Macaraeg
Trace Capture enhancements
Variable Latency Delay Unit

Team (2)

Hirel Patel and Zhuanhao Wu, Waterloo

- Joshua Landgraf (student intern): simulation + emulation
- Chris Hajas (student intern): DRE studies
- Prateek Srivastava (student intern): initial variable latency delay unit
- Nelson Ho (student intern, staff): Linux port
- Eric Green (student intern, staff): Linux support, trace collection

CASC

Center for Applied
Scientific Computing

Lawrence Livermore
National Laboratory

CHIME radio telescope

Canadian Hydrogen Intensity Mapping Experiment

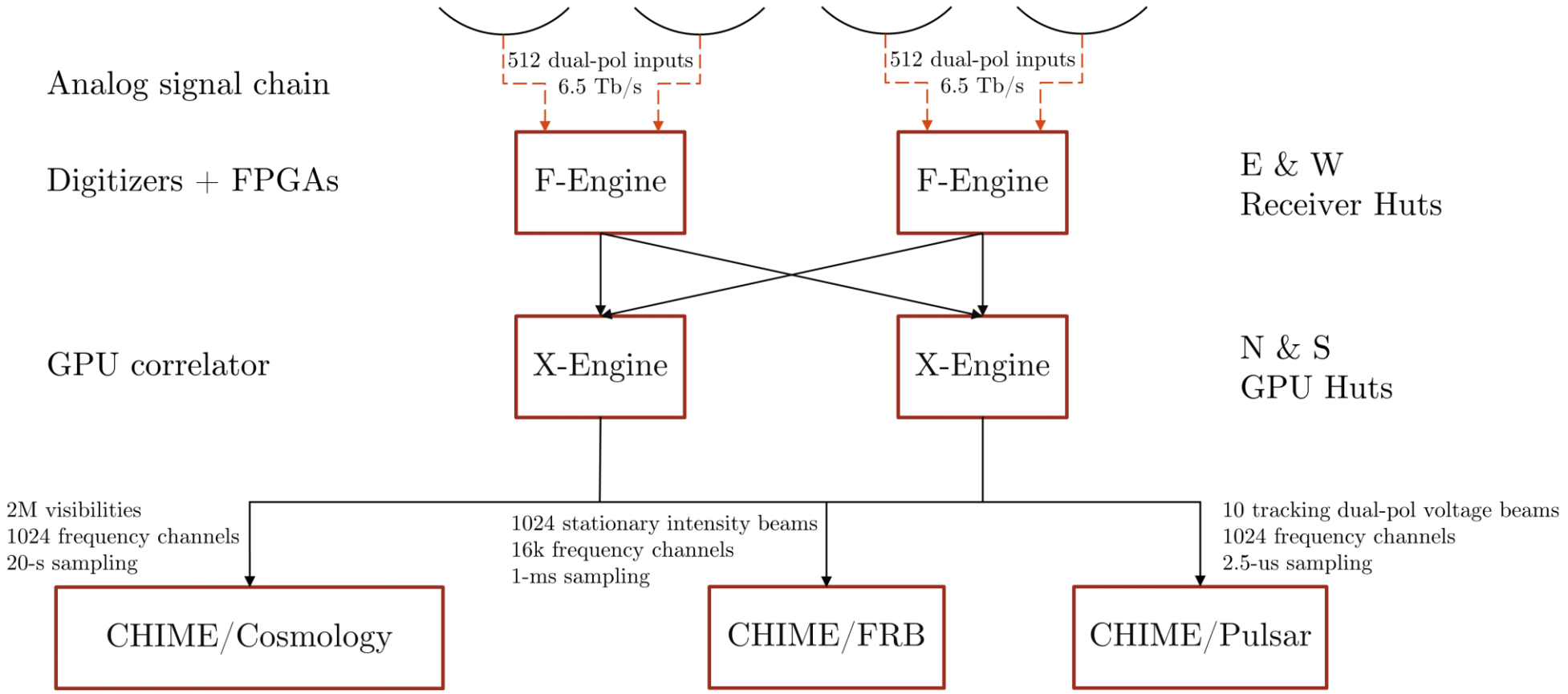
- Map the history of the expansion rate of the Universe by observing hydrogen gas in distant galaxies that were very strongly affected by dark energy.
- Detect FRBs (fast radio bursts) to act as an early warning system for the wider astrophysical community.
- Monitor known pulsars in the Northern sky to investigate the properties of neutron stars and ionized gas in the interstellar medium to help verify the predictions of general relativity and the search for gravitational waves.

Other than electrons, the CHIME radio telescope has no moving parts. Instead, the telescope consists of four parallel, adjacent cylindrical cylinders measuring 20x100m and oriented north-to-south. The telescope scans the heavens as the Earth turns. CHIME's four reflectors feed 256 focal-point antennas located along each cylindrical axis (for a total of 1024 antennas) and each antenna generates signal feeds from two polarizations for a total of 2048 signal feeds. CHIME's front-end electronics then sample each signal at 800Msamples/sec, resulting in 1.6384 Tsamples/sec, resulting in a front-end feed of 13Tbps.

CHIME Radio Telescope with F-Engine Containers

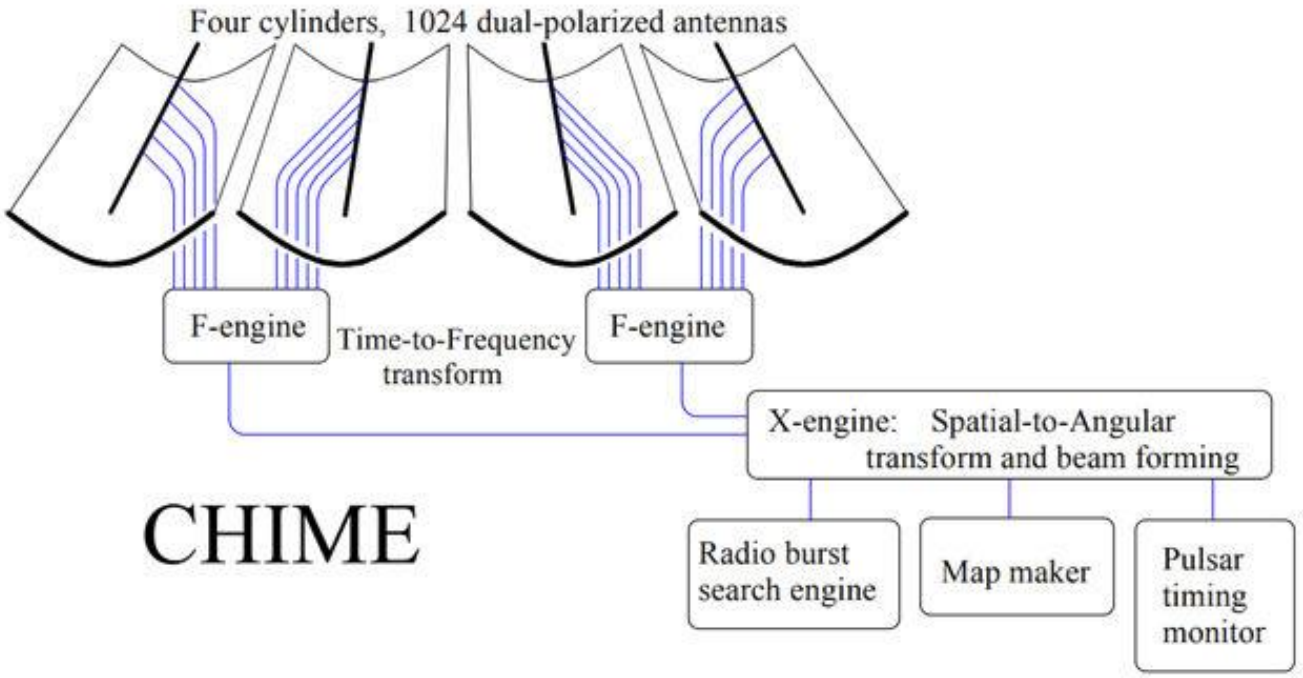
The CHIME F-Engine

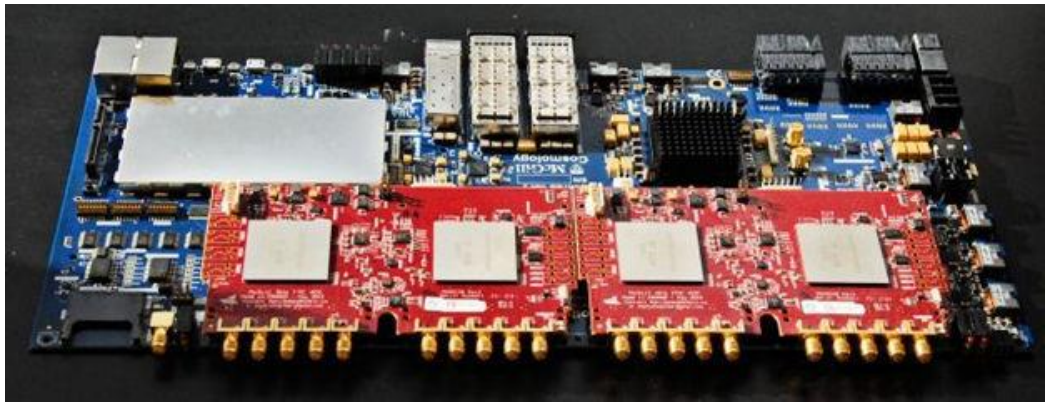
CHIME processing architecture



F-Engine hardware and algorithms

The ICE motherboard incorporates a Kintex-7 FPGA connected to 16 ADCs mounted on the two FMC daughter cards





Kintex-7 provides twenty-eight 10Gbps serial ports for inter-board networking and data offload.
On-board ARM running Linux manages MB functions, runs user code algorithms.

- F-Engines convert each microsecond of raw data (2048 samples/usec) into spectral range spanning 400MHz-500MHz with frequency resolution of .39MHz. The binned spectral data is shipped to GPU-based X-Engine via optical fiber.

Reconfigurable computing in space: Mars Perseverance Rover

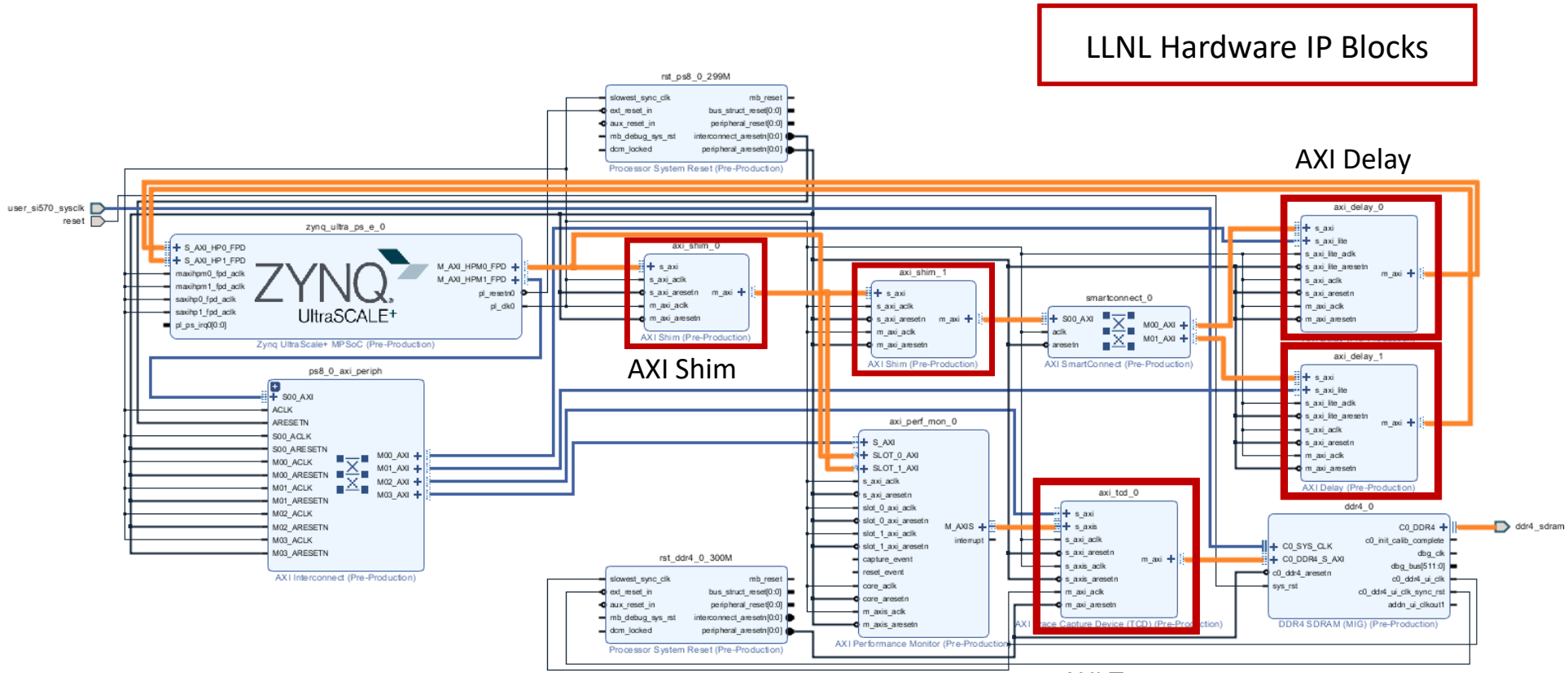
The Mars rover Perseverance illustrated here will carry a lunchbox-size PIXL device to analyze rocks and soil quickly in hopes of finding evidence of ancient life on the Red Planet. Virtex 5 accelerates specific stereo and visual tasks like image rectification, filtering, detection, and matching (NASA)

Virtex 2 Pro chips in multiple instruments

- Electra-lite instrument maintains UHF Transceiver and runs relay telecommunications and navigation.
- Radar Terminal Descent Sensor (TDS) is a Ka-band radar that provides range and velocity measurements through all phases of (post-heatshield separation, including Entry, Descent, and Landing (EDL).
- Mastcam-Z is a mast-mounted camera system that can zoom in, focus, and take 3D pictures and video at high speed to allow detailed examination of distant objects.
- SHERLOC (Scanning Habitable Environments with Raman & Luminescence for Organics & Chemicals) is for the fine-scale detection of minerals, organic molecules, and potential biosignatures.

https://www.fierceelectronics.com/electronics/nasa-mars-rover-perseverance-launches-thursday-to-find-evidence-life-red-planet

LiME (Logic in Memory Emulator) Implementation



LiME uses only 13% of the device resources

DRE Architecture