System software and architecture assist for application-directed data views

Maya Gokhale

November 26, 2018

LLNL-PRES-951211

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

Data view: a window into the data

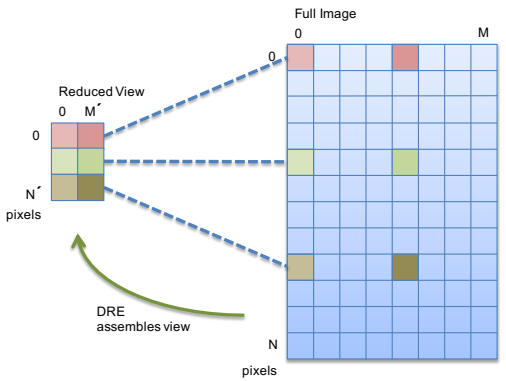
Station_ID	Date_Time	altimeter_se	air_temp_se	dew_point_t	relative_hum	wind_speed_	wind_directi	wind_gust_s	sea_level_pr	pressure_ten	air_temp_hi	air_temp_lo	dew_point_t	wind_chill_s	wind_cardin	pressure_set	sea_level_pr	heat_index_	weather_con
		INHG	Fahrenheit	Fahrenheit	%	Miles/hour	Degrees	Miles/hour	Millibars	code	Fahrenheit	Fahrenheit	Fahrenheit	Fahrenheit	Code	Millibars	Millibars	Fahrenheit	Code
KBUR	1/1/10 0:53	29.46	48.02		67.92	3.44	350		1025.3	8004			37.82		N	971.66	994.12		Clear
KBUR	1/1/10 1:53	29.43	46.04		70.64	0	0		1024.3				36.93		N	970.67	993.29		Clear
KBUR	1/1/10 2:53	29.42	46.04		70.64	0	0		1023.9				36.93		N	970.34	992.95		Clear
KBUR	1/1/10 3:53	29.41	42.98		79.37	6.91	360		1023.6	6017	51.98	42.98	36.96		N	970.01	992.86		Clear
KBUR	1/1/10 4:53	29.4	44.96		70.51	0	0		1023.2				35.85		N	969.68	992.39		Clear
KBUR	1/1/10 5:53	29.38	44.96		70.51	3.44	250		1022.8				35.85		WSW	969.02	991.71		Clear
KBUR	1/1/10 6:53	29.4	44.96		73.59	0	0		1023.3	5003			36.94		N	969.68	992.36		Clear
KBUR	1/1/10 7:53	29.4	46.94		65.42	3.44	300		1023.5				35.83		WNW	969.68	992.23		Clear
KBUR	1/1/10 8:53	29.41	53.96		50.42	3.44	300		1023.7				35.8		WNW	970.01	992.02		Clear
KBUR	1/1/10 9:53	29.42	57.92		37.29	0	0		1023.9	3007	57.92	42.98	31.83		N	970.34	992.17		Clear

A view is an abstraction

A view is assembled from disparate sources

A view filters extraneous data

air_temp_se	relative_hum	wind_speed_	wind_directi
Fahrenheit	%	Miles/hour	Degrees
48.02	67.92	3.44	350
46.04	70.64	0	0
46.04	70.64	0	0
42.98	79.37	6.91	360
44.96	70.51	0	0
44.96	70.51	3.44	250
44.96	73.59	0	0
46.94	65.42	3.44	300
53.96	50.42	3.44	300
57.92	37.29	0	0



Outline

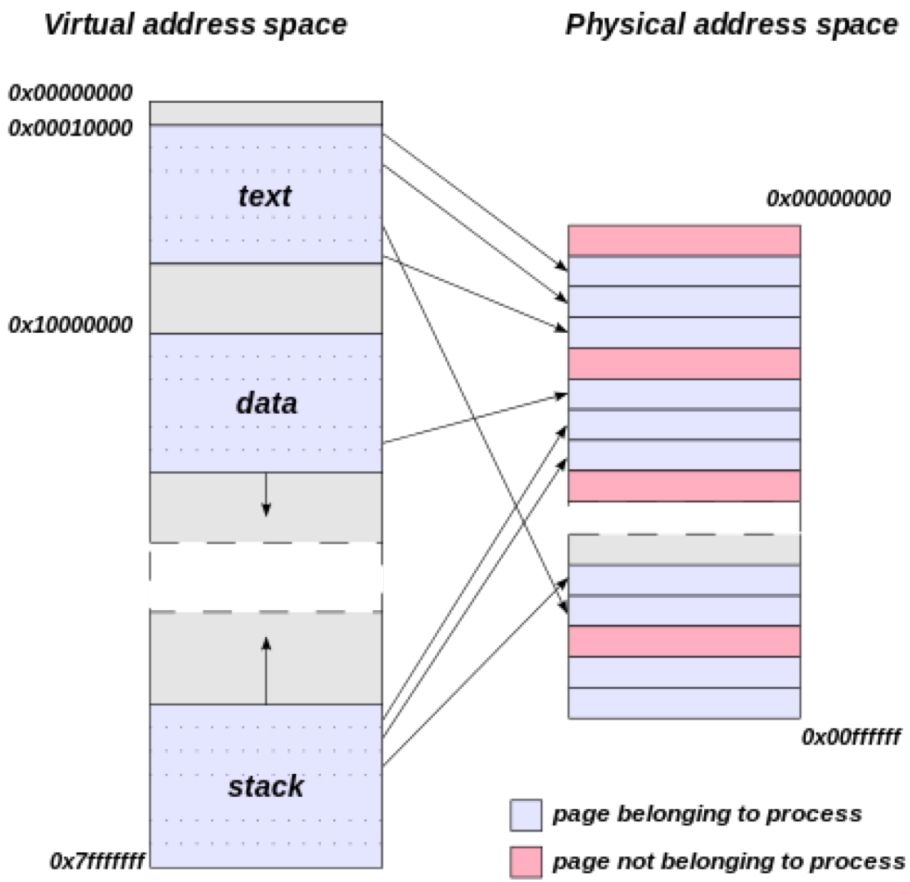
- Observations on views
- Memory mapping to support views
  - Exploit hardware page fault to materialize fragment of view on demand
- Near memory processing
  - Filter data near memory to materialize

Observations on views

- A view is an abstraction
  - It can be a filter, e.g. weather_data['relative_humidity']
  - It can be a composite lsst_dc1_10_3[i][j][k]
  - It can embody a search problem Genome['AGGTCGGAGTGAACGGATTCGGCCG']
- It is often advantageous to materialize a portion of a view
  - Create a fragment of the abstraction on demand rather than a bulk operation over the entire data set
  - Streaming access
- View semantics are widely applicable
  - Database (SQL or no SQL), dictionary, sparse array, search
- Implementations span the space of storage to compute
  - Database systems, file systems, persistent memory, compute in memory

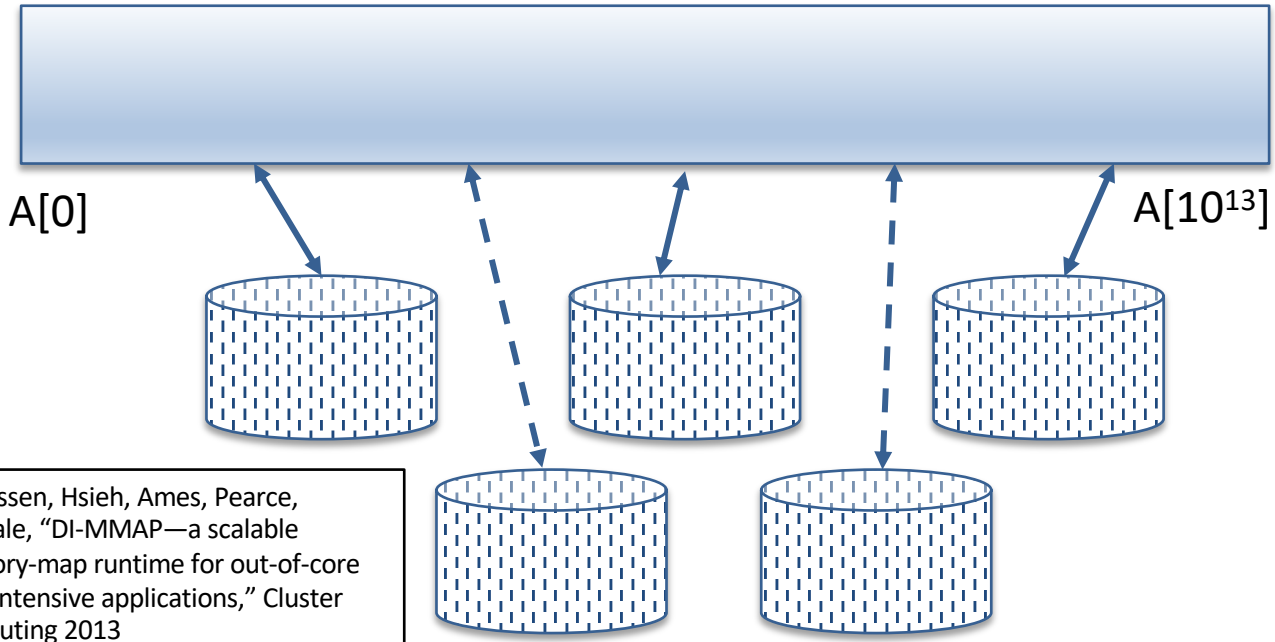
Assembling views at OS page granularity

- Paging basics
- Virtual and physical address space are divided into pages (often 4KB)
- Every virtual address generated by running process is translated into physical address by MMU
- If page containing referenced location isn't in memory, page fault occurs
- Page fault handling is usually done by OS, but can be forwarded to user code



Memory mapping to support views

- Goal is to define a large, virtual "in-memory" data structure A
  - Data structure is too big to fit in compute node main memory
  - Data structure may not exist anywhere in that form in entirety
- Mechanism is to use Linux mmap anonymous system call to assign a virtual address range for data structure A
- Reference to A[i] may generate a page fault if A[i]'s page not in physical memory

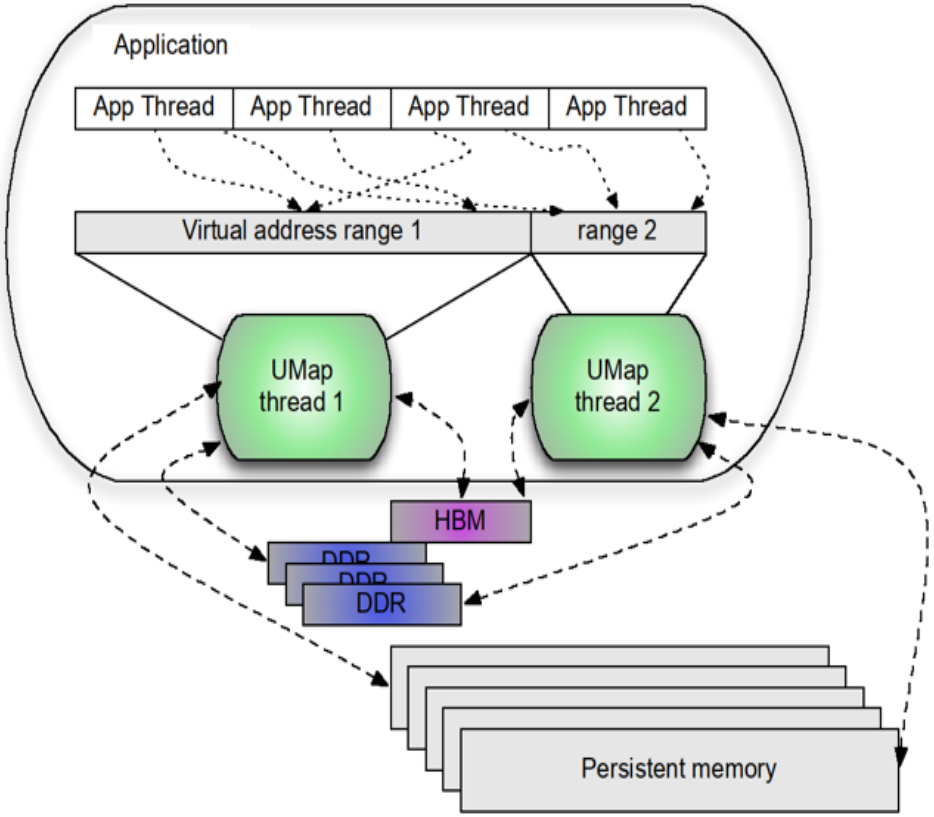


Van Essen, Hsieh, Ames, Pearce, Gokhale, "DI-MMAP—a scalable memory-map runtime for out-of-core data-intensive applications," Cluster Computing 2013

- Application maps A into a virtual address range
- Data may be assembled from many data sources
- Memory map handler fills pages of A on demand: A[i]

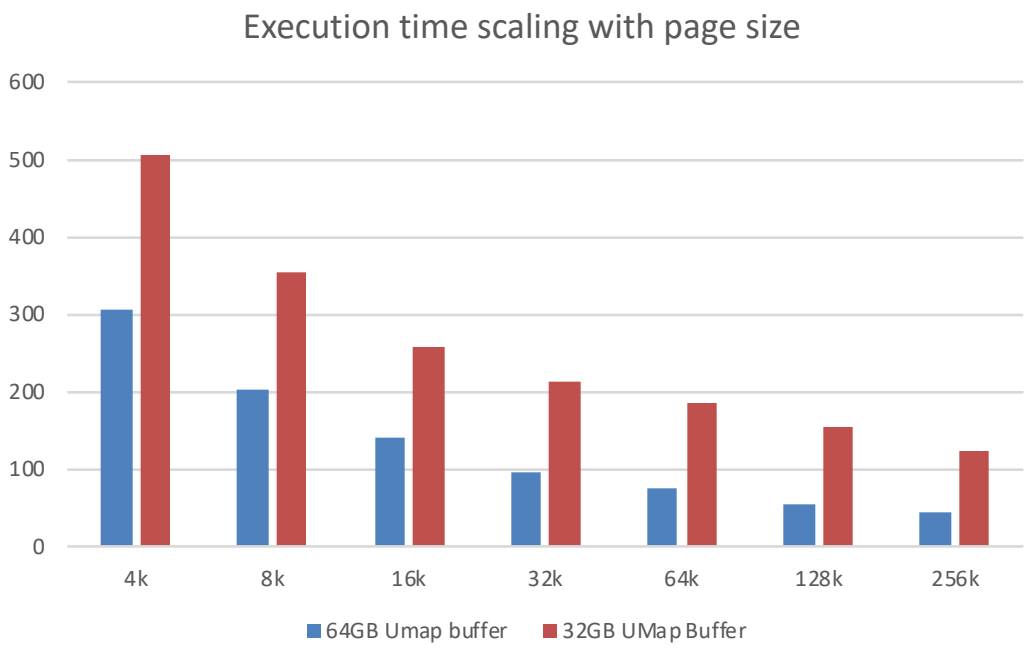
UMap: user mode library to handle page fault in user space

- Uses Linux 4.3+ userfaultfd messaging protocol to notify user space handler of page fault
  - Asynchronous, lower overhead than sigfault handler
- UMap library services faults in the registered virtual range
  - Notified through message queue from kernel
  - Provides page buffer to cache faulted pages
  - Evicts pages using LRU policy
  - Supports multiple page sizes without affecting OS page size
  - Supports multiple files/offsets to map into virtual range
- Application can have multiple UMap handlers for different data structures



https://github.com/LLNL/umap

Multiple page sizes with Graph500 BFS



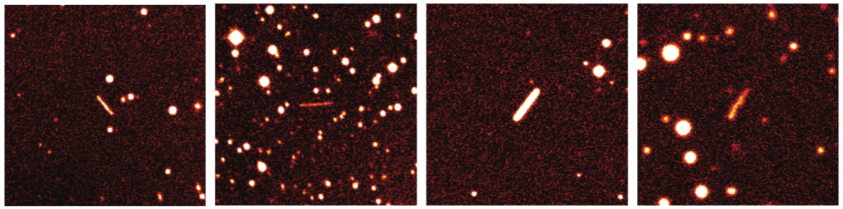
Scale 28, 64GB graph data
# of edge: 8,589,934,592
# of vertices: 268,435,443
24 bfs threads, 16 umap threads

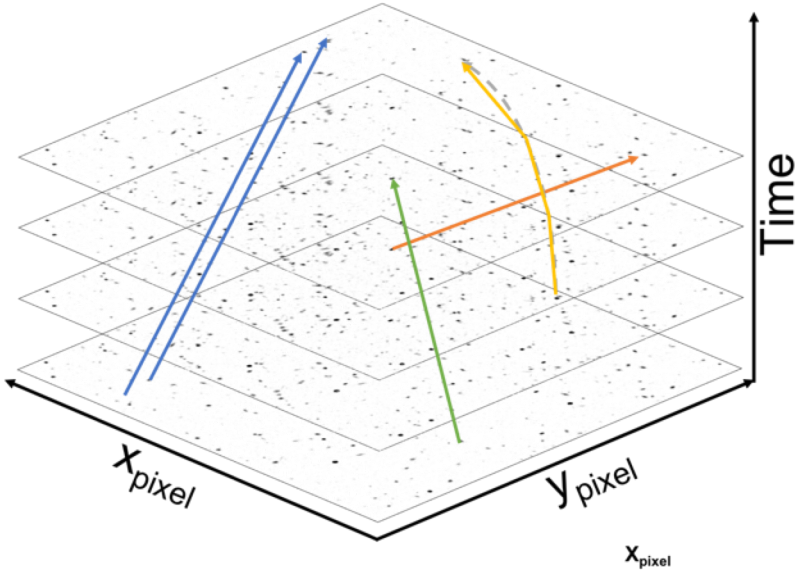
For this application, increasing page size improves performance.

Locality in data structure storage (CSR) and regular traversal pattern benefits from larger page size.

Top: 64GB UMap page buffer,
Bottom: 32GB

Application: detect faint near-earth asteroids in telescope imagery





- Near-Earth Asteroids (NEAs) of small size are estimated to be 1000 times more numerous than current 0.5~1000 kilometer diameter cataloged NEAs
- Our goal is detecting small NEAs with optical imaging data sets that are too faint to discover in any single image
- Data sets consist of Tera-bytes of FITS format files that are organized in time order as a 3D virtual cube
- We project vectors from (x,y) patches into the time dimension to detect possible NEAs
- Algorithm used is pixel-wise median calculation along vector
- Current research with LSST DC1 (150TB) data set with injected NEAs.

UMap on-going activities

- Umap runs on the LLNL Sierra supercomputer nodes
- Work with Linux kernel userfaultfd developers to upstream read/write mode
  - Current releases don't yet support write, so we develop the read/write version of UMap on an experimental kernel
- Restructure handler to support page fetch over network
  - Released version reads data from locally mounted file system
  - Support for distributed data sets or remote memory is in-progress
- Characterize and optimize performance for specific applications
  - Number of UMap handler threads
  - Tune algorithm to split virtual address range among handlers
  - Tune ratio of page buffer to available memory
  - Tune page size

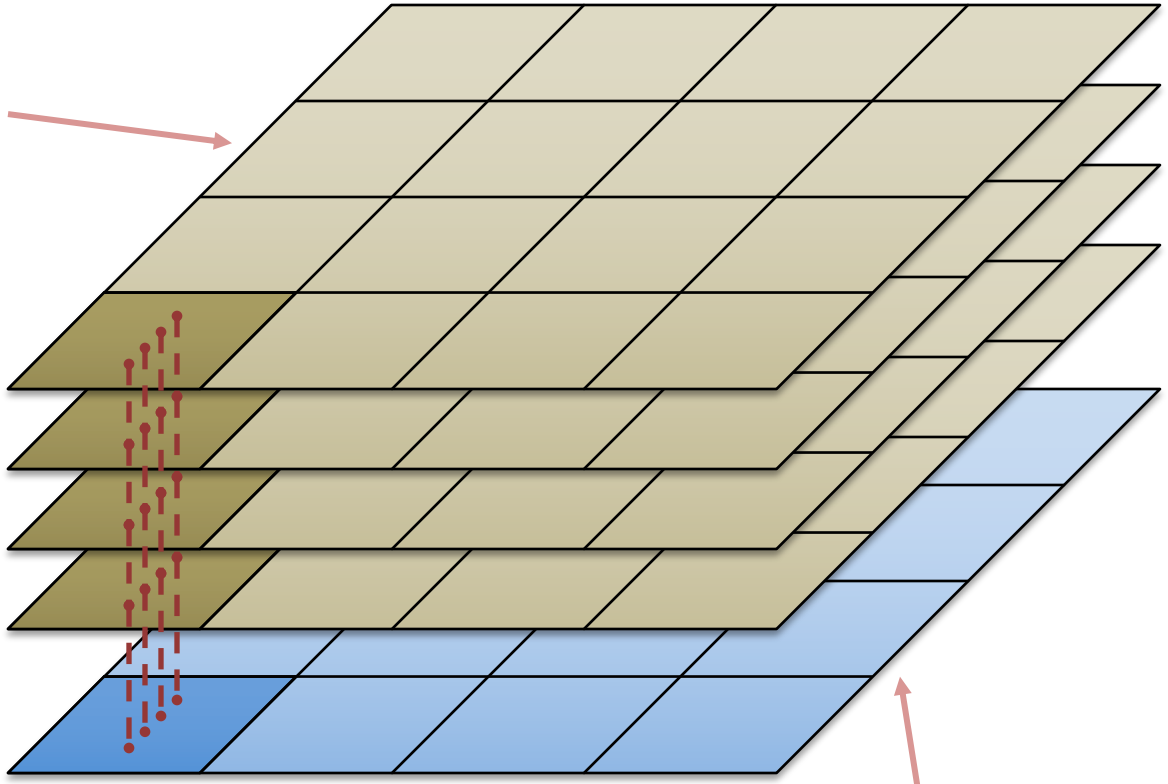
Page level views: summary

- Application defines data structures and maps them to a vacant virtual address range using OS memory map system call
- Reference to a particular part of a particular data structure in the virtual range may not be in physical memory
- UMap handler resolves the missing page reference using application specific method
  - Creates the page by reading or writing files or network resources
  - Subsequent references in the page occur at memory/cache speed through MMU
- Data view created by handler is ephemeral, triggered by page fault, created on the fly at a page granularity
- Method uses existing address translation hardware and OS services

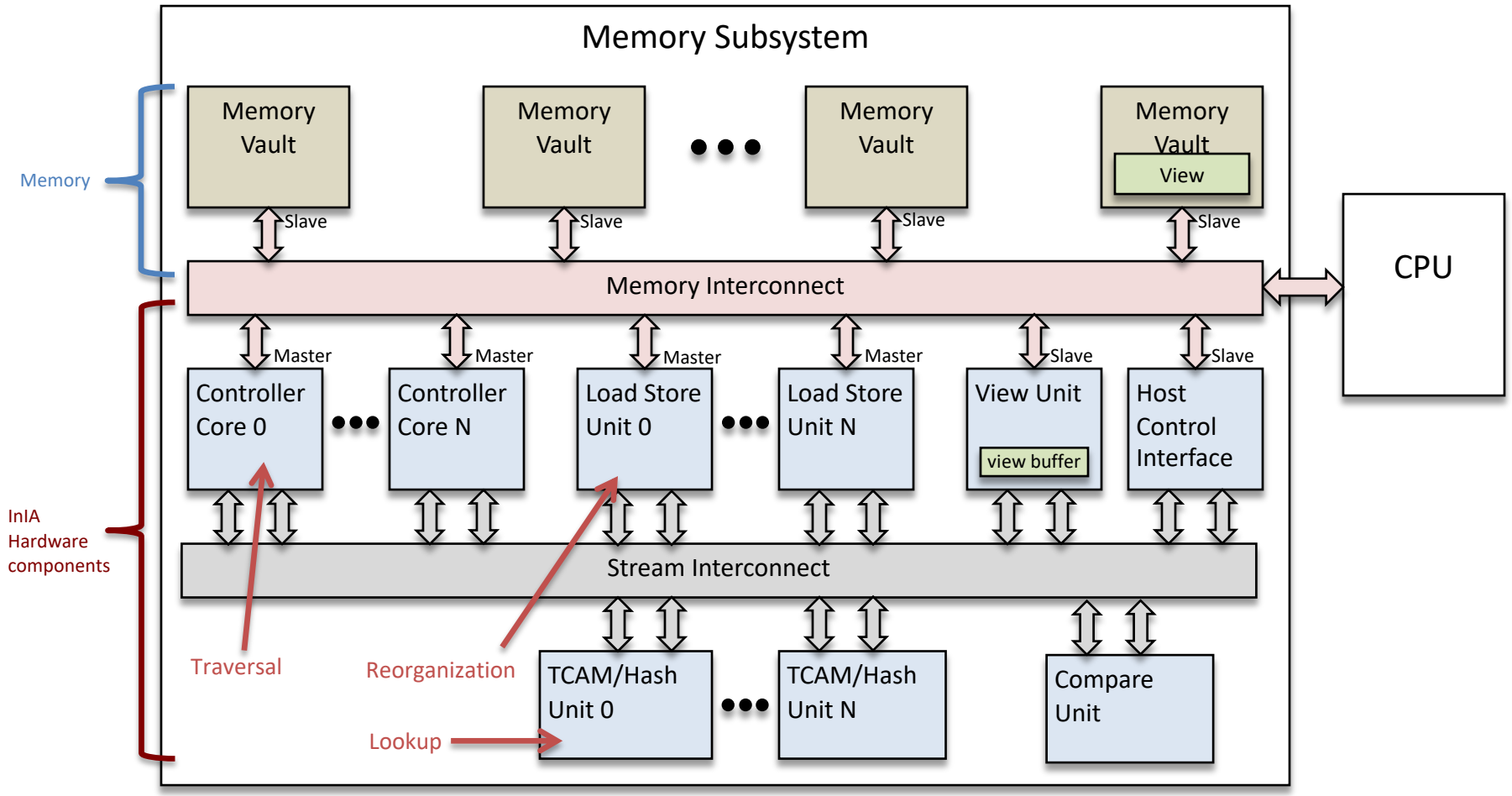
Creating data views with near memory hardware assist

- Design new hardware to assist assembling data views
- Hardware co-located with memory system
- Materialize view near memory
  - Exploit high bandwidth within memory system
  - Allow main processor to work directly with view rather than scattered fragments
- Generate view fragment
  - Referenced region
  - Application-defined granularity
- Virtual address to system address to physical address translation must be addressed
  - IOMMU
  - Allocate contiguous system address range and use base/limit registers
  - 1-1 virtual to system mapping

Enablers for near memory logic



Creating views with near memory accelerators

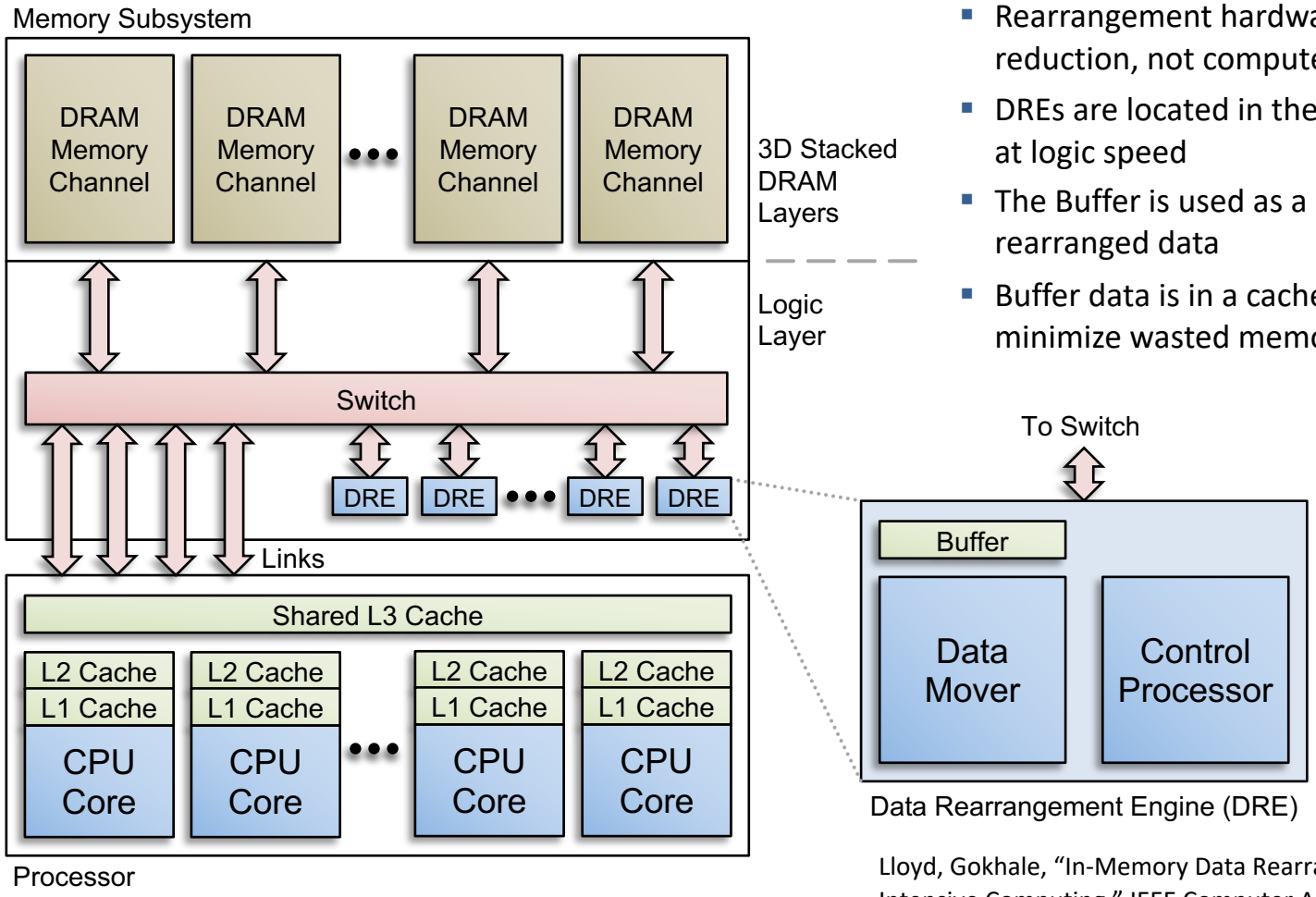


Near memory IP blocks with flexible interconnect

Creating data view with data rearrangement

- 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
- Memory bandwidth to CPU limits these 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 not as helpful to these applications
- Approach
  - **Rearrange and reduce data near the source**
  - Move less data to CPU for energy and performance benefit
  - Rearrangement hardware is generally applicable
  - Programmable gather/scatter units

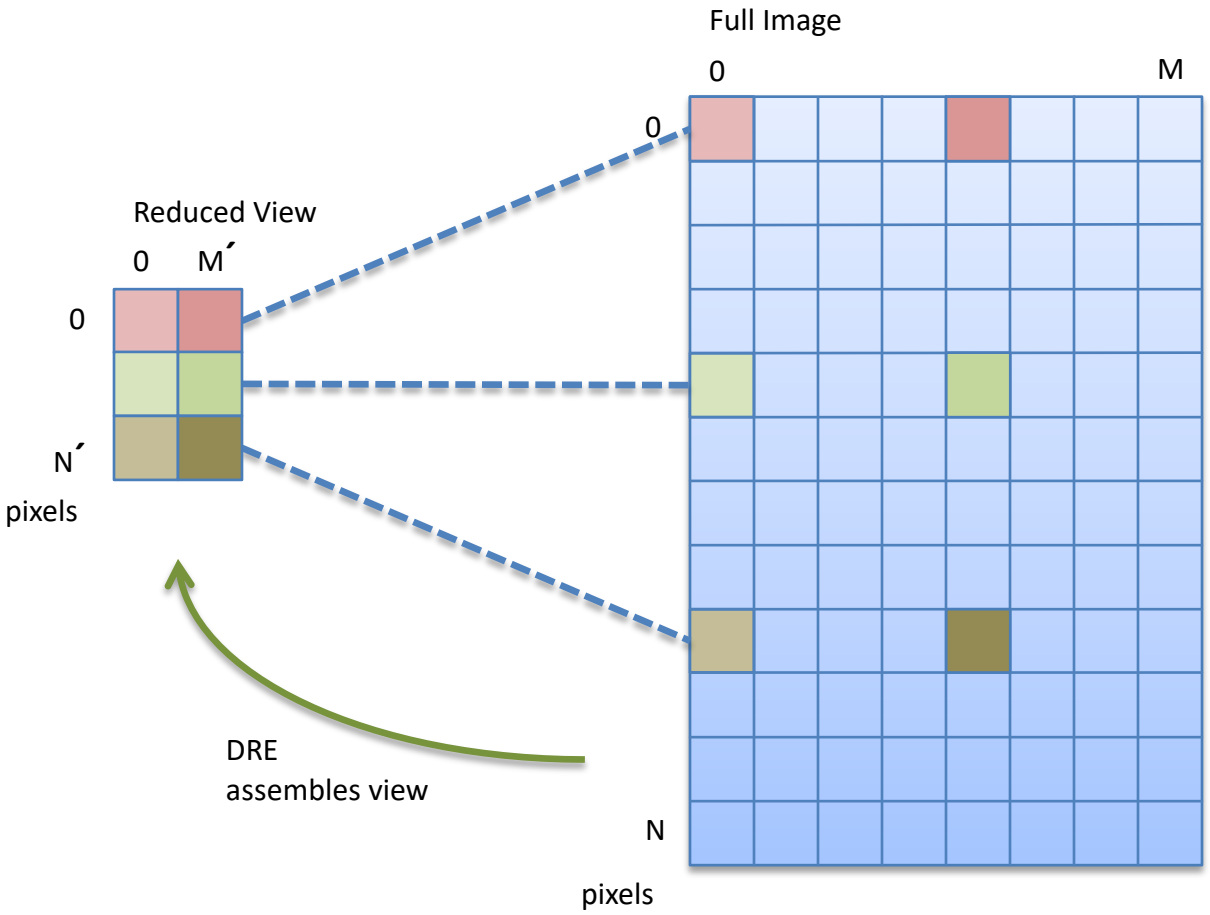
Data Rearrangement Engine (DRE) Near Memory



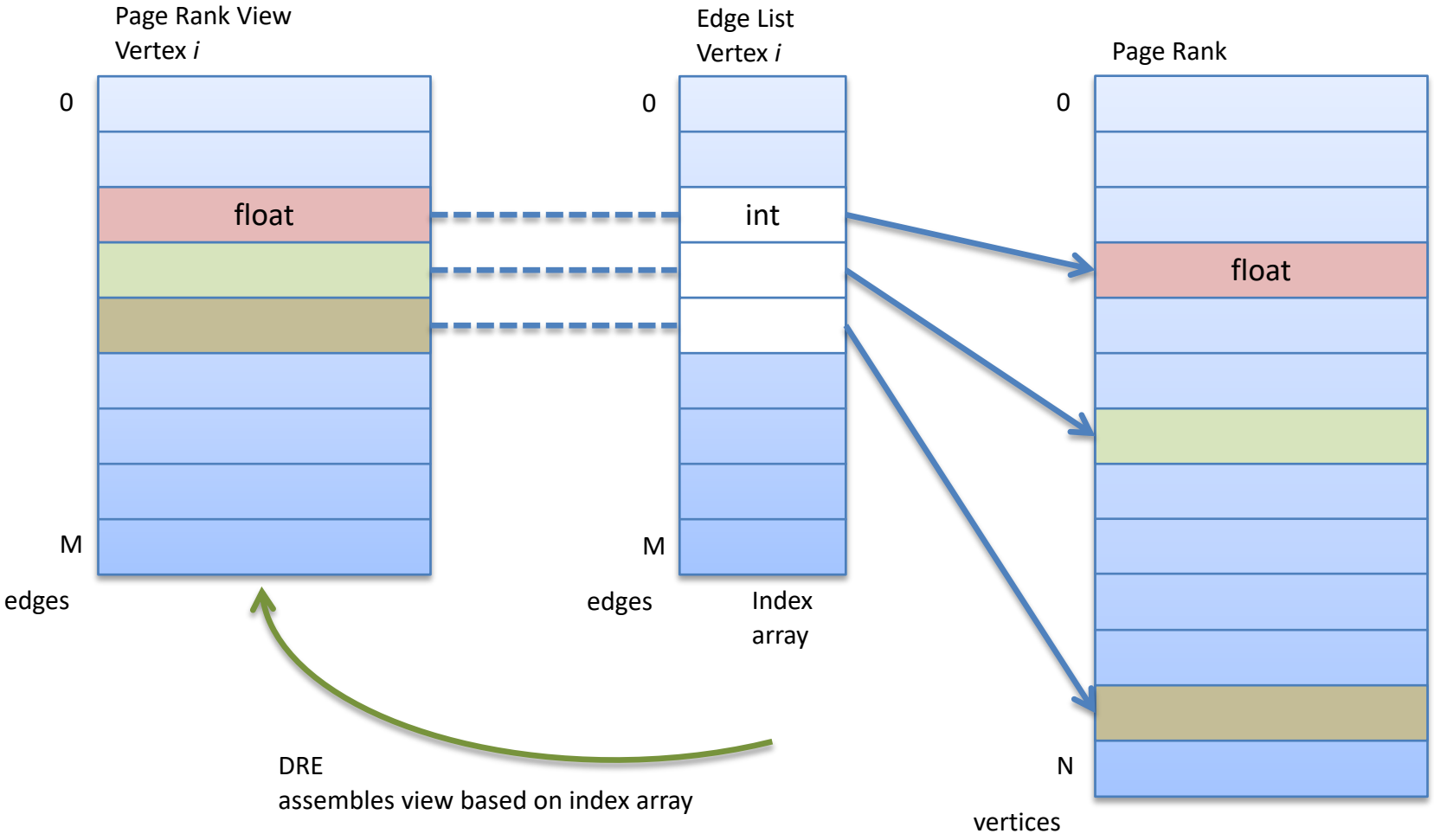
- Rearrangement hardware performs data reduction, not compute offload
- DREs are located in the logic layer and run at logic speed
- The Buffer is used as a scratchpad for rearranged data
- Buffer data is in a cache-friendly layout to minimize wasted memory bandwidth

Lloyd, Gokhale, "In-Memory Data Rearrangement for Irregular, Data-Intensive Computing," IEEE Computer Aug. 2015

Image View Assembled by DRE



PageRank View Assembled by DRE



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);
```

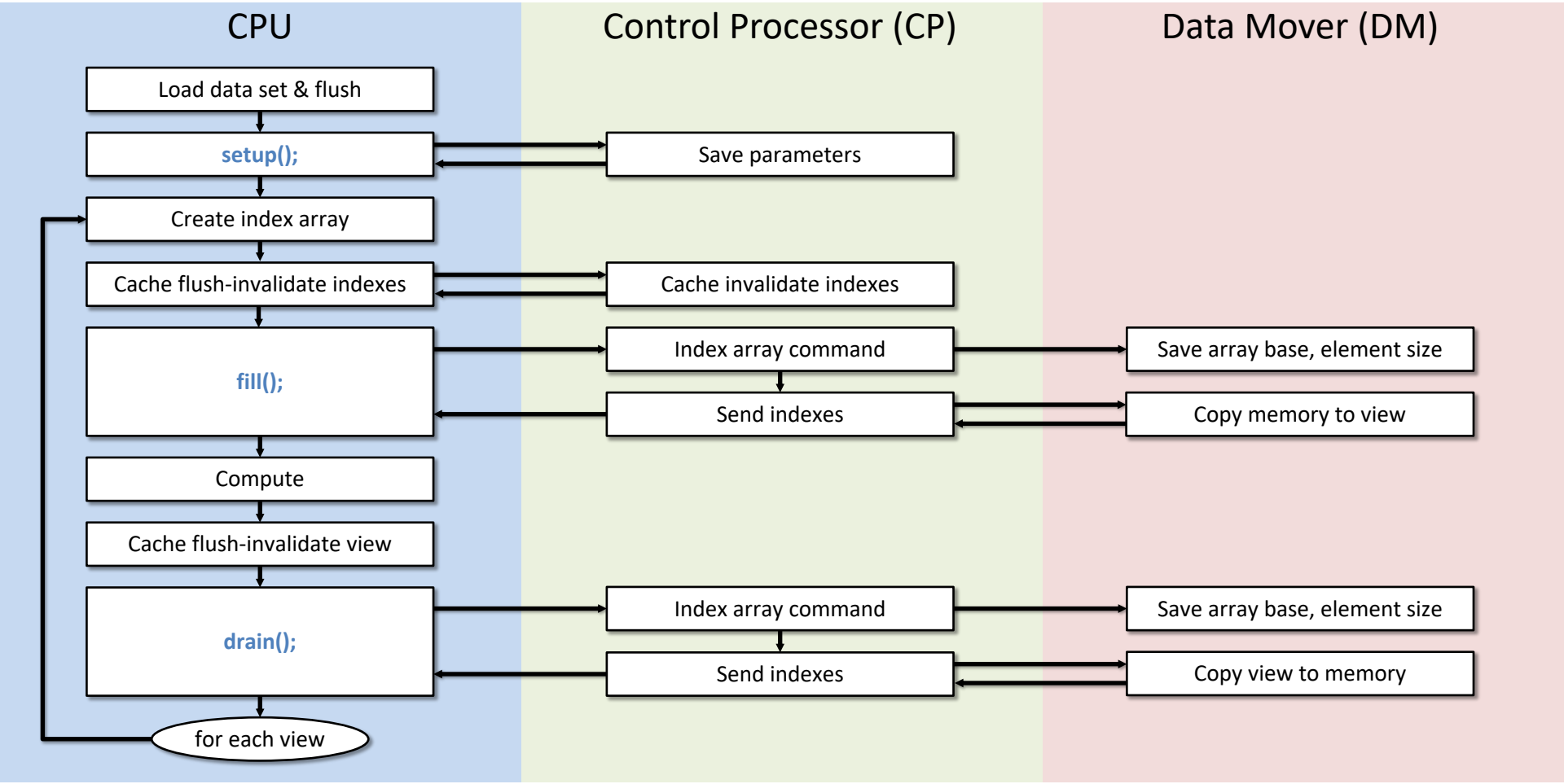
fill — Copy from bulk memory to the view buffer according to the access pattern established during setup

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

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

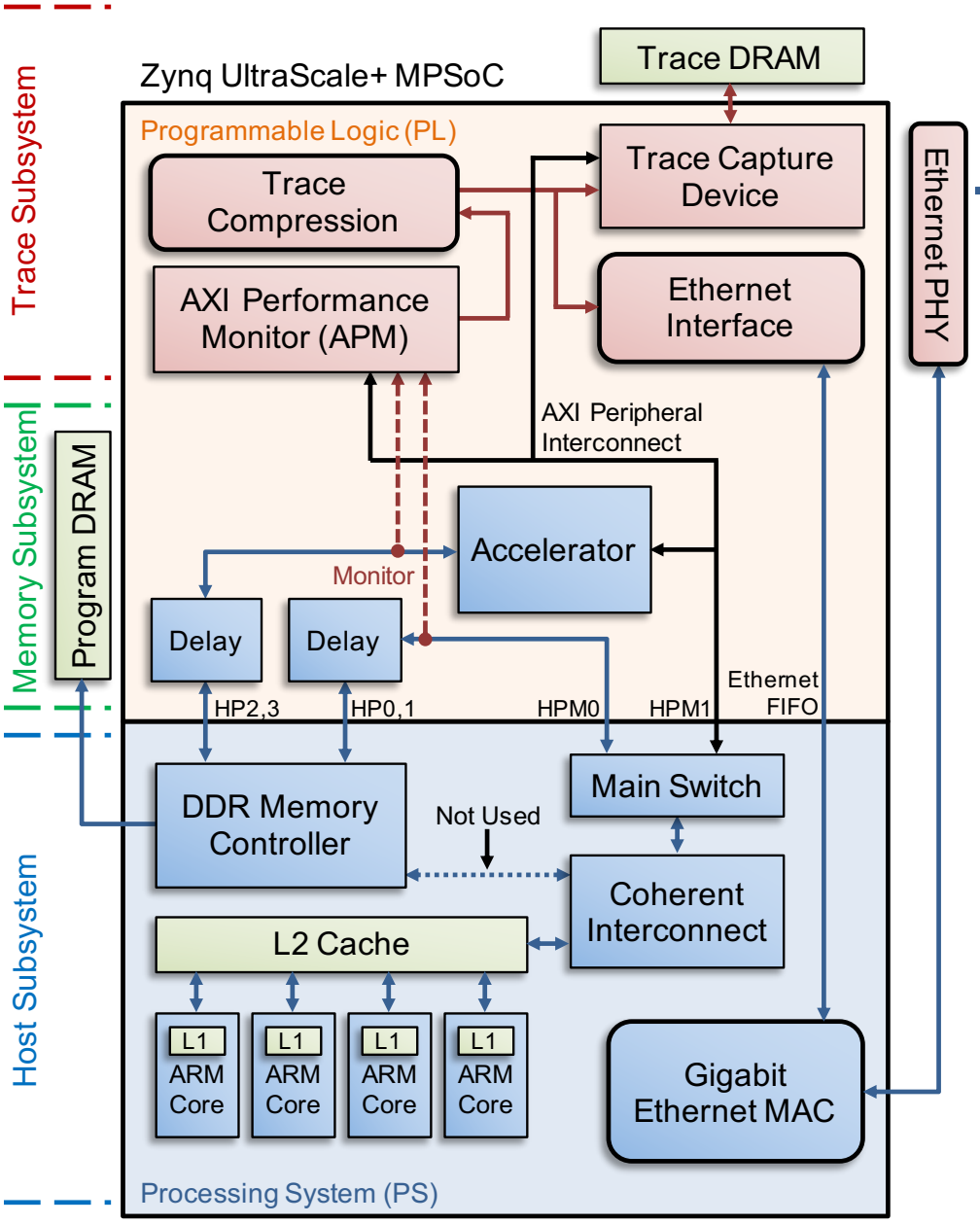
```
/* Specify view buffer and window offset */
void drain(void *buf, size_t buf_sz, size_t offset);
```

Control flow for gather-scatter operations

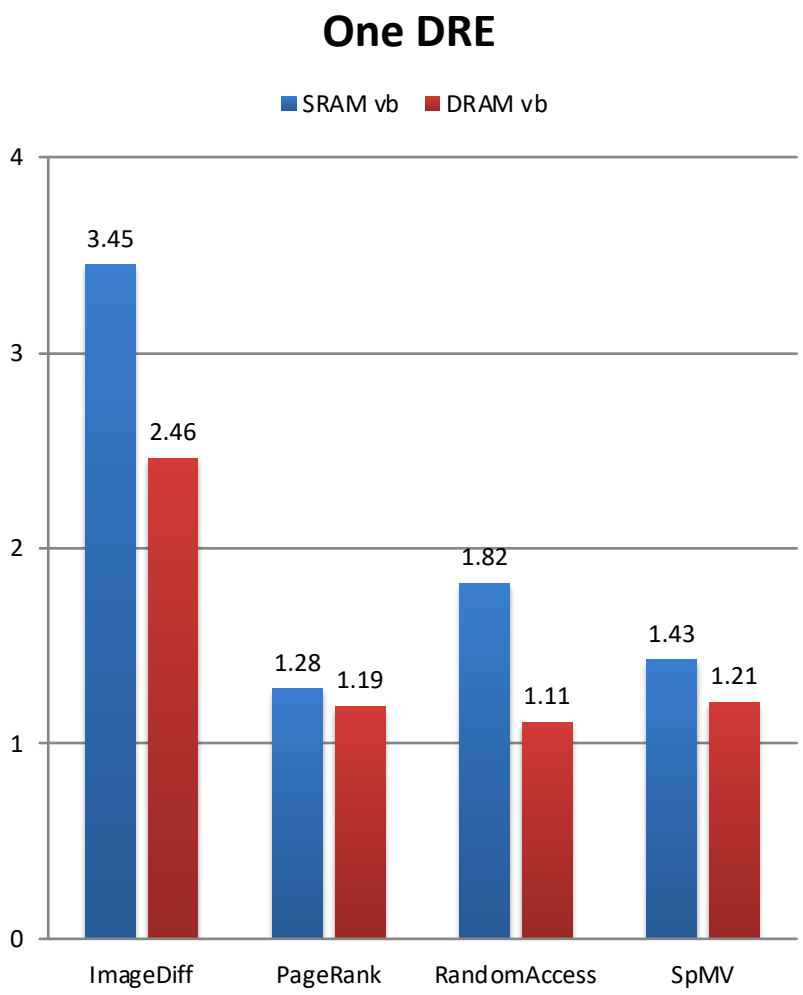


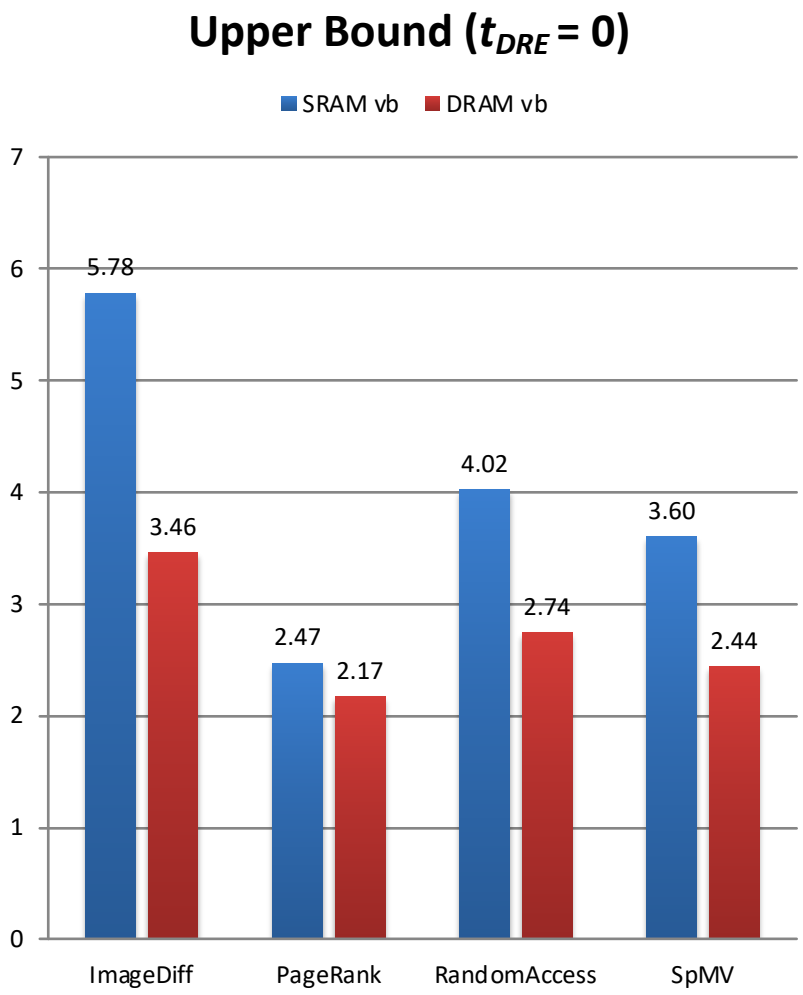
FPGA prototype for near memory DRE

- Use FPGA SoC with ARM processors and FPGA logic
- Delay units emulate off-chip access to multiple memory types
- Memory trace capture hardware blocks record time-stamped memory transactions
- Application runs on ARM, sends requests to DRE to gather or scatter block of data view ("view buffer")



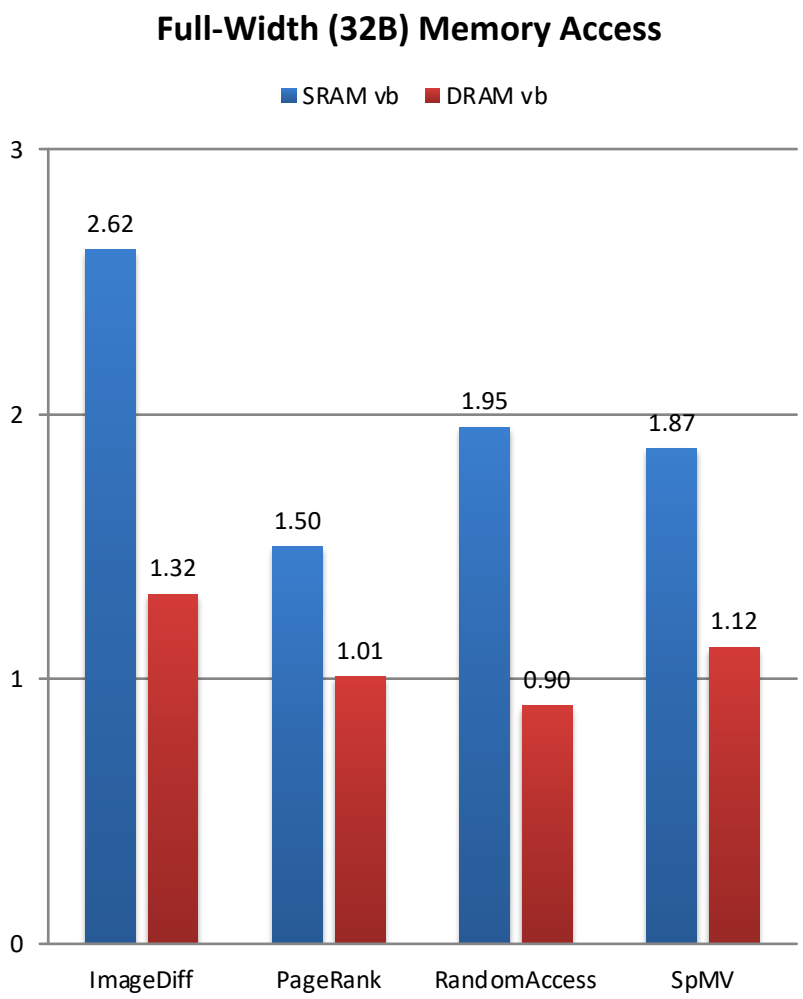
Speedup

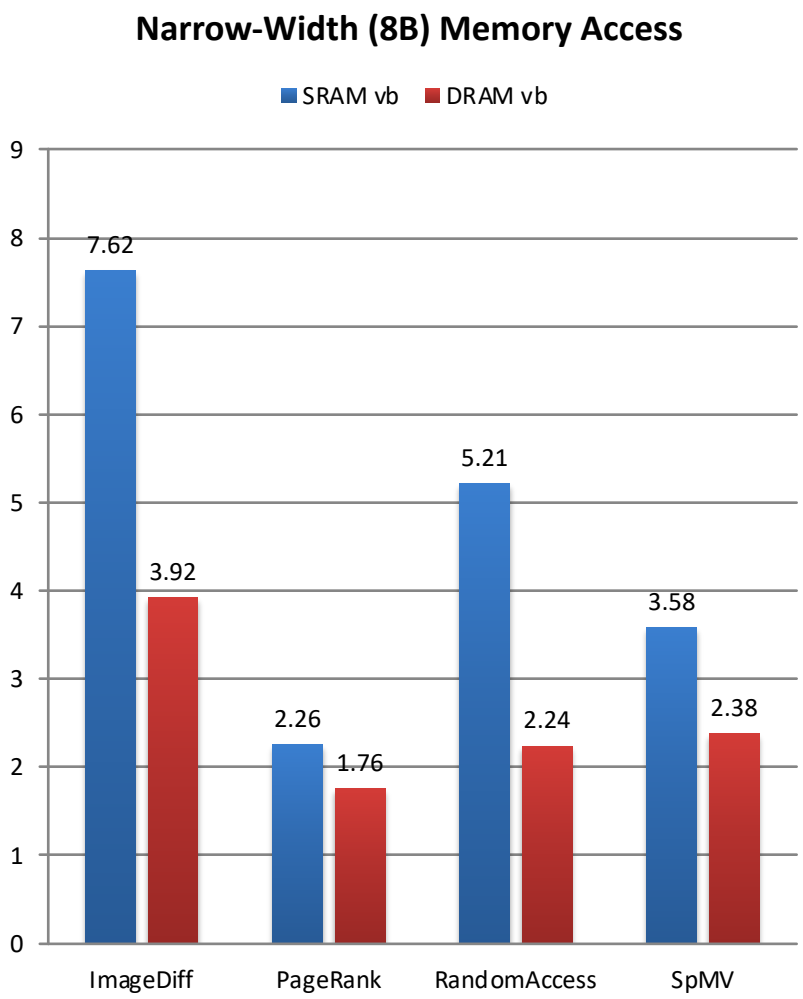

One DRE
SRAM vb
DRAM vb
4
3
2
1
0
3.45
2.46
1.28
1.19
1.82
1.11
1.43
1.21
ImageDiff
PageRank
RandomAccess
SpMV


Upper Bound (t_{DRE} = 0)
SRAM vb
DRAM vb
7
6
5
4
3
2
1
0
5.78
3.46
2.47
2.17
4.02
2.74
3.60
2.44
ImageDiff
PageRank
RandomAccess
SpMV

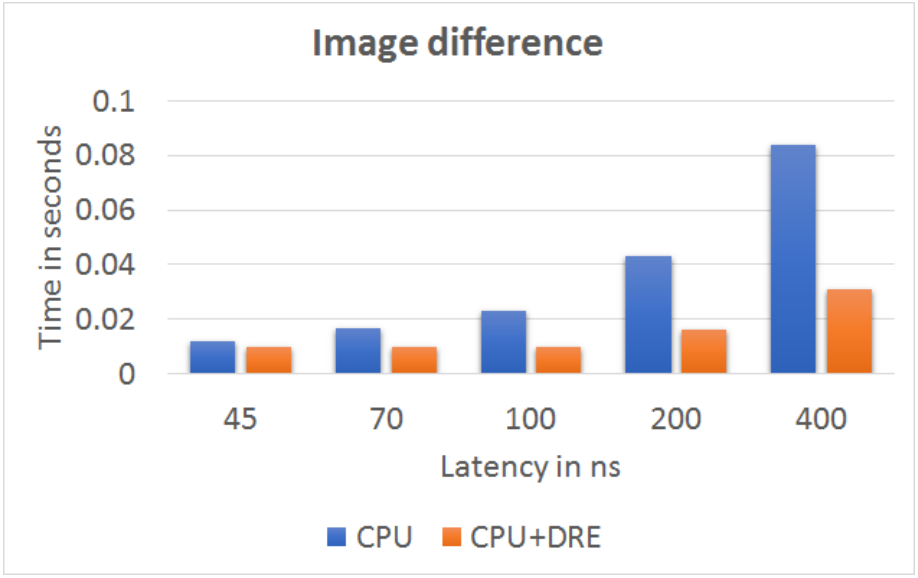
Energy Reduction

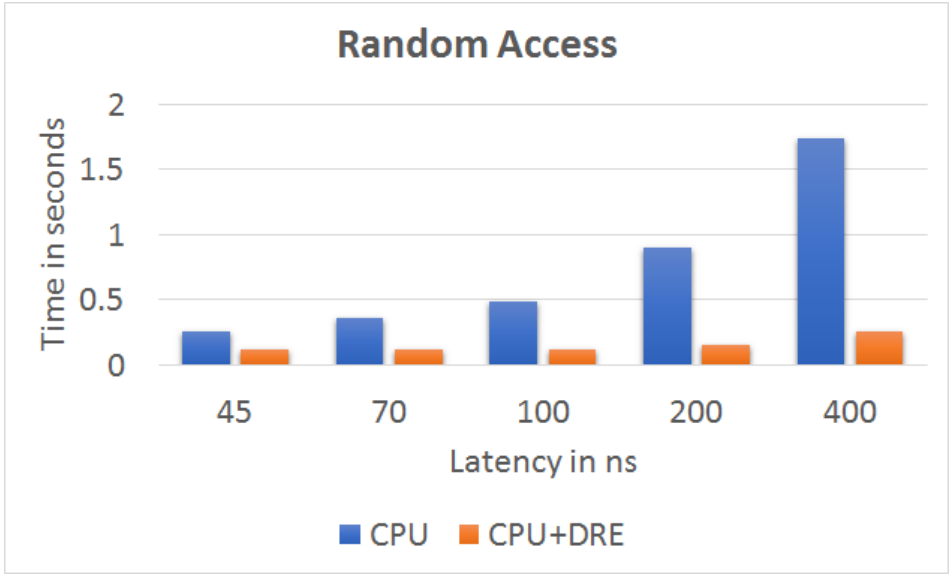
Simple model: 19.4 pJ/bit for DRAM, 1.0 pJ/bit for SRAM, and 10.3 pJ/bit for off-chip traversal





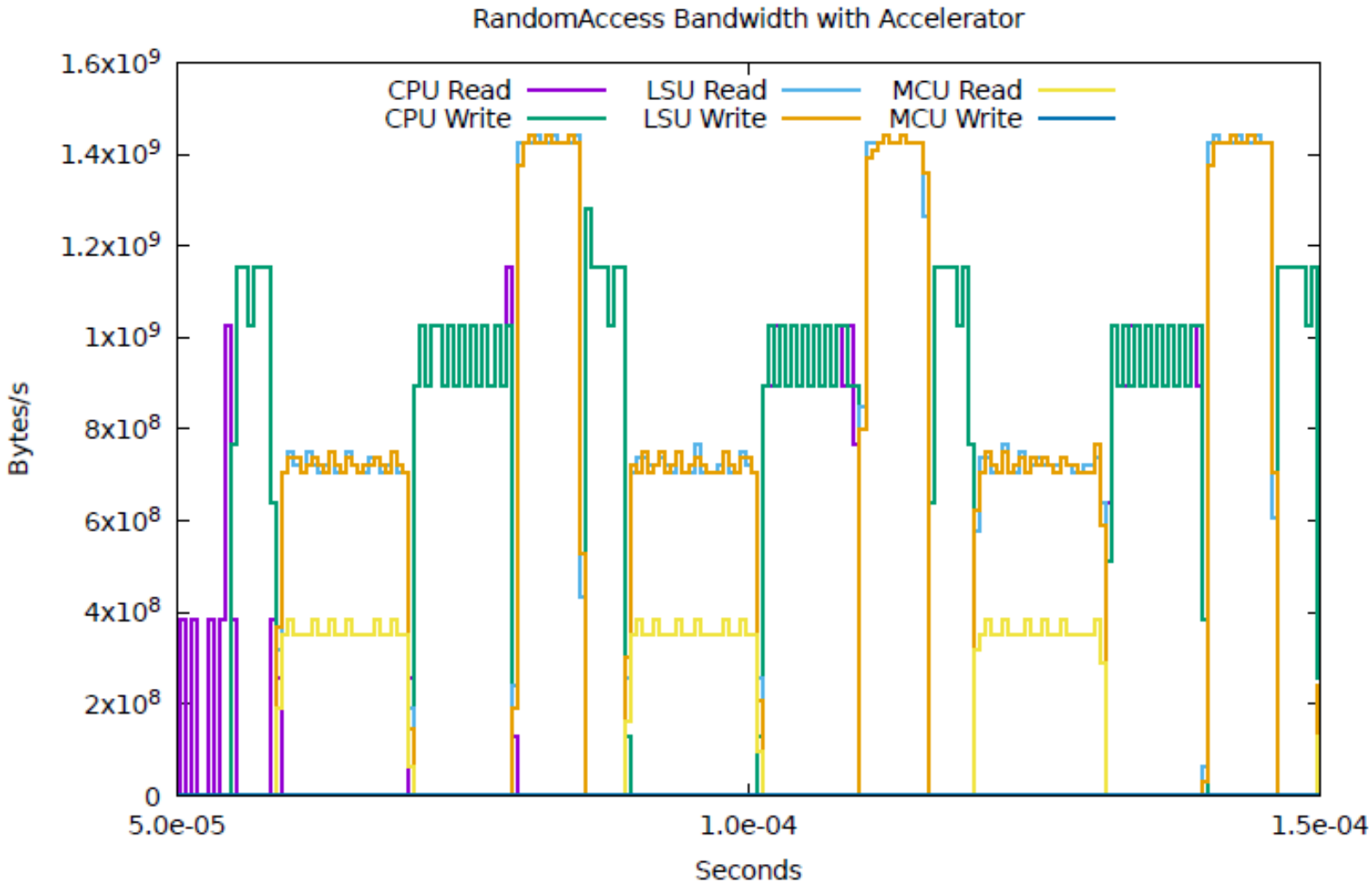
Benefit increases for single core and single DRE



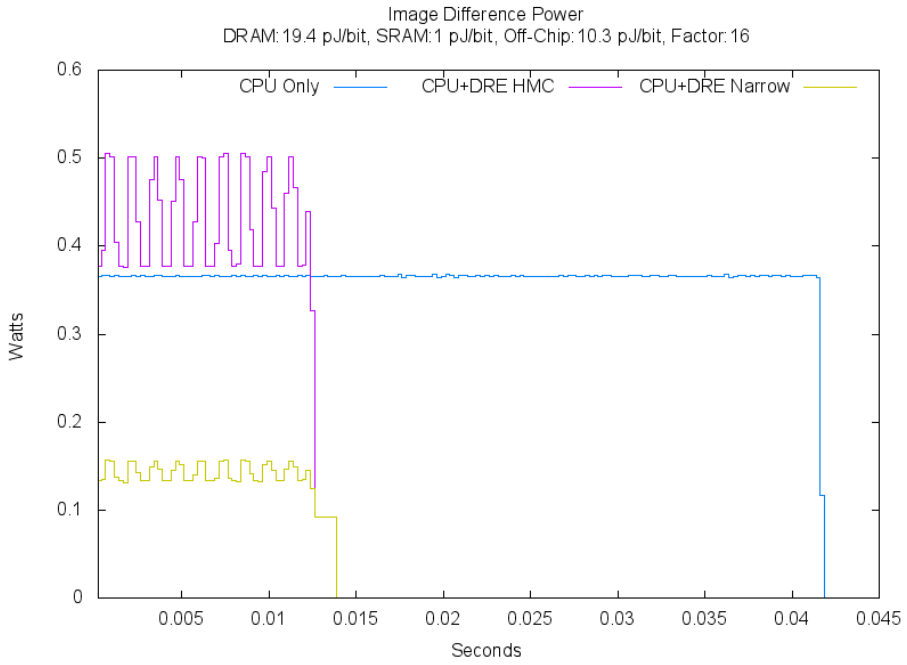


The results demonstrate that substantial speedup can be gained with a DRE due to the higher number of in-flight requests issued by the near-memory accelerator.

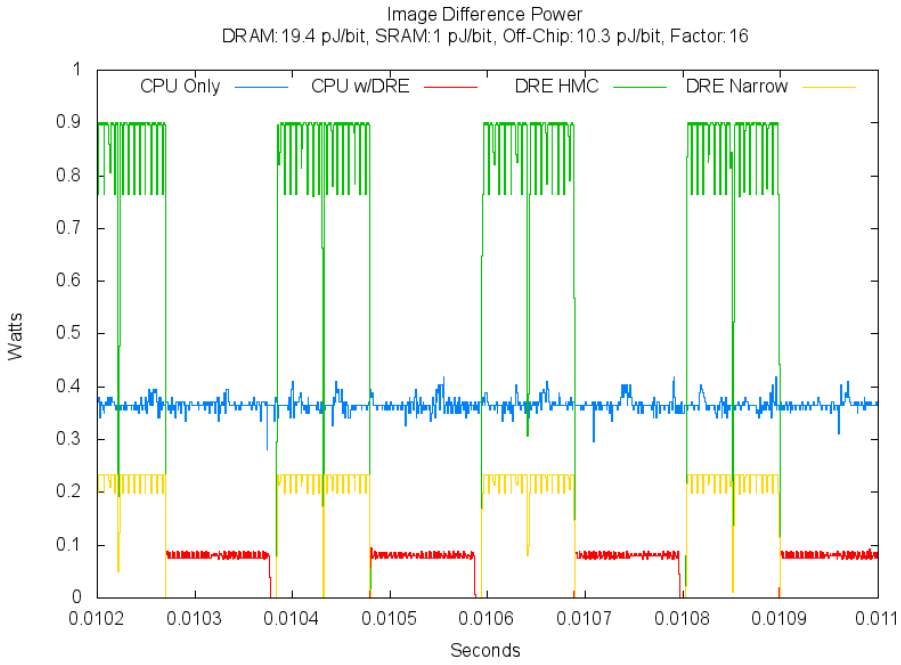
Bandwidth Analysis from Trace



ImageDiff Power Profile

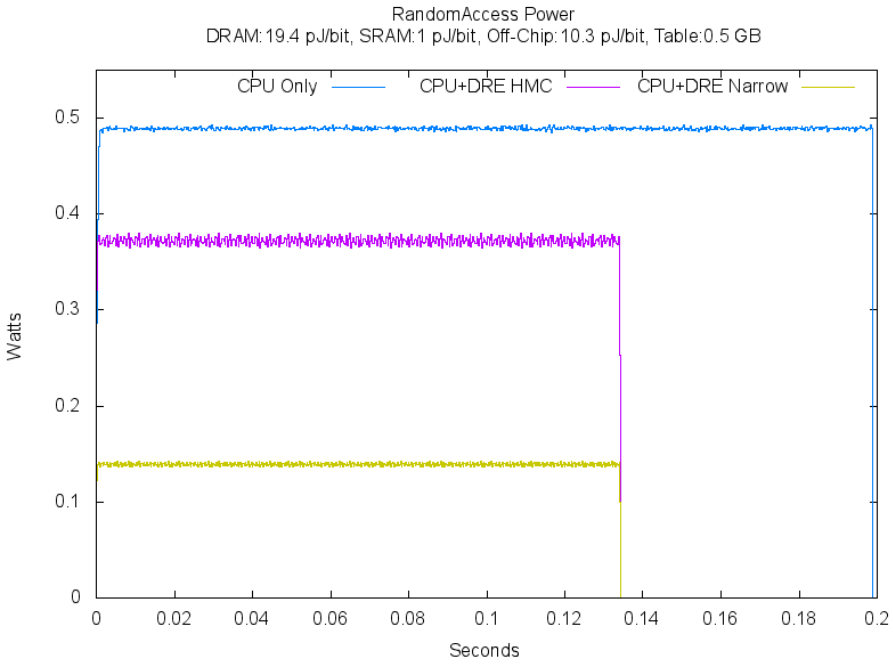


(a) The entire run.

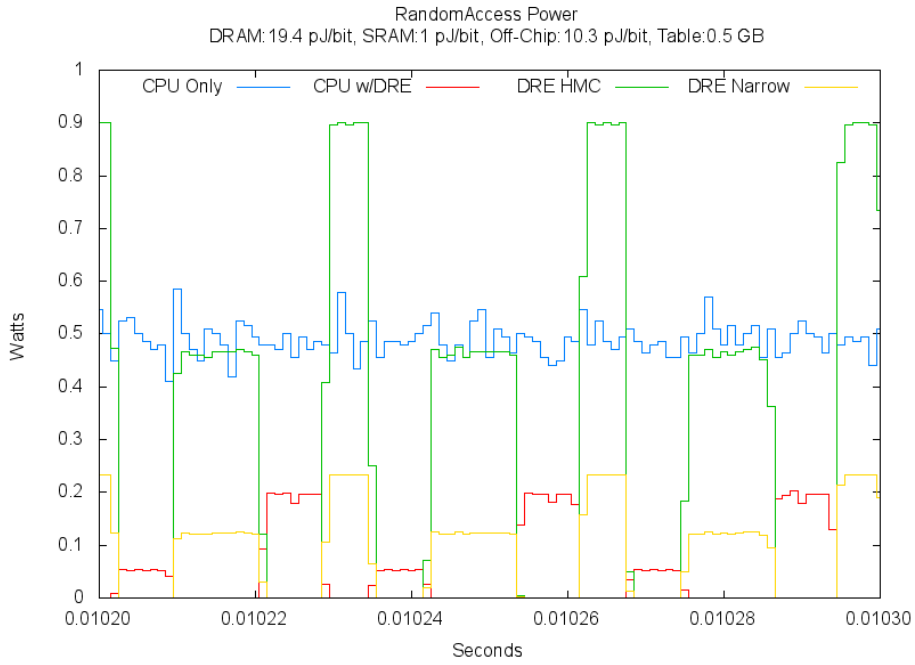


(b) Enlarged segment of the run.

RandomAccess Power Profile



(a) The entire run.



(b) Enlarged segment of the run.

DRE summary

- 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

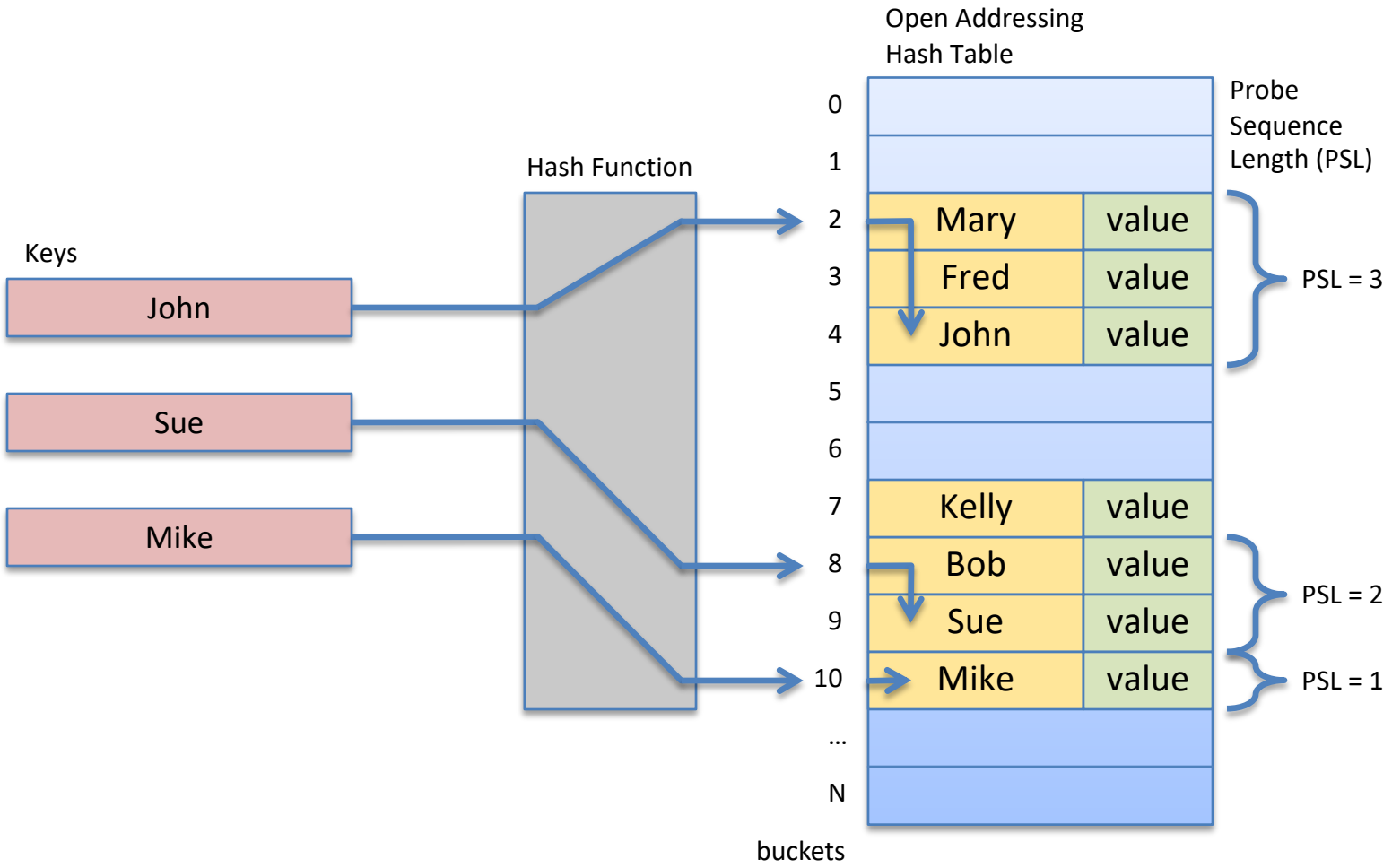
Associative view: key-value store

- Genome['AGGTCGGAGTGAACGGATTCGGCCG']
- Access data store through key
- Implementations
  - Content addressable memory
    - TCAM by folding comparators into RRAM
  - Generate index tables
    - B+-tree, hash tables
- We use hash table using open address hashing to resolve collision

L. Zheng, S. Shin, S. Lloyd, and M. Gokhale, "Rram-based TCAMs for pattern search," 2016 IEEE Int'l Symposium on Circuits and Systems, May 2016.

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

Open address hashing



Open Addressing Features

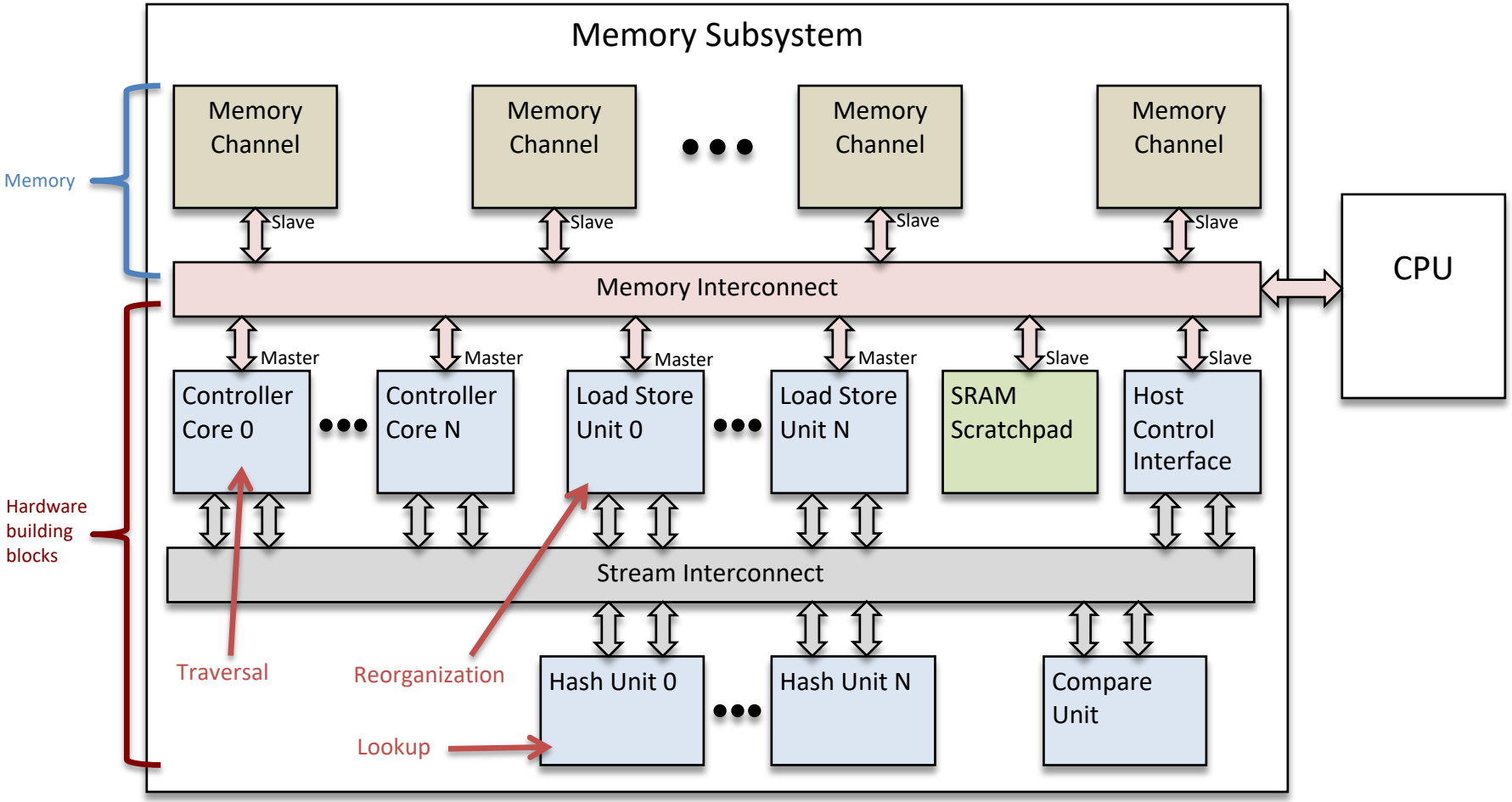
Supportive of Hardware Implementation

- Attractive for near memory hash tables
  - Allows probe sequence bytes to be streamed sequentially
  - Enables a constant rate, deterministic pipeline of lookups
- Beneficial for small keys and values
  - No extra space needed for linked lists or pointers
- Avoids indirection
  - Chaining approaches need extra memory accesses
- Table space must be reserved at the outset
  - Avoid overhead of allocating memory for each record
  - No need for a memory allocator
- Susceptible to clustering
  - Requires a high quality hash algorithm
  - Our hash algorithm is adapted from SpookyHash

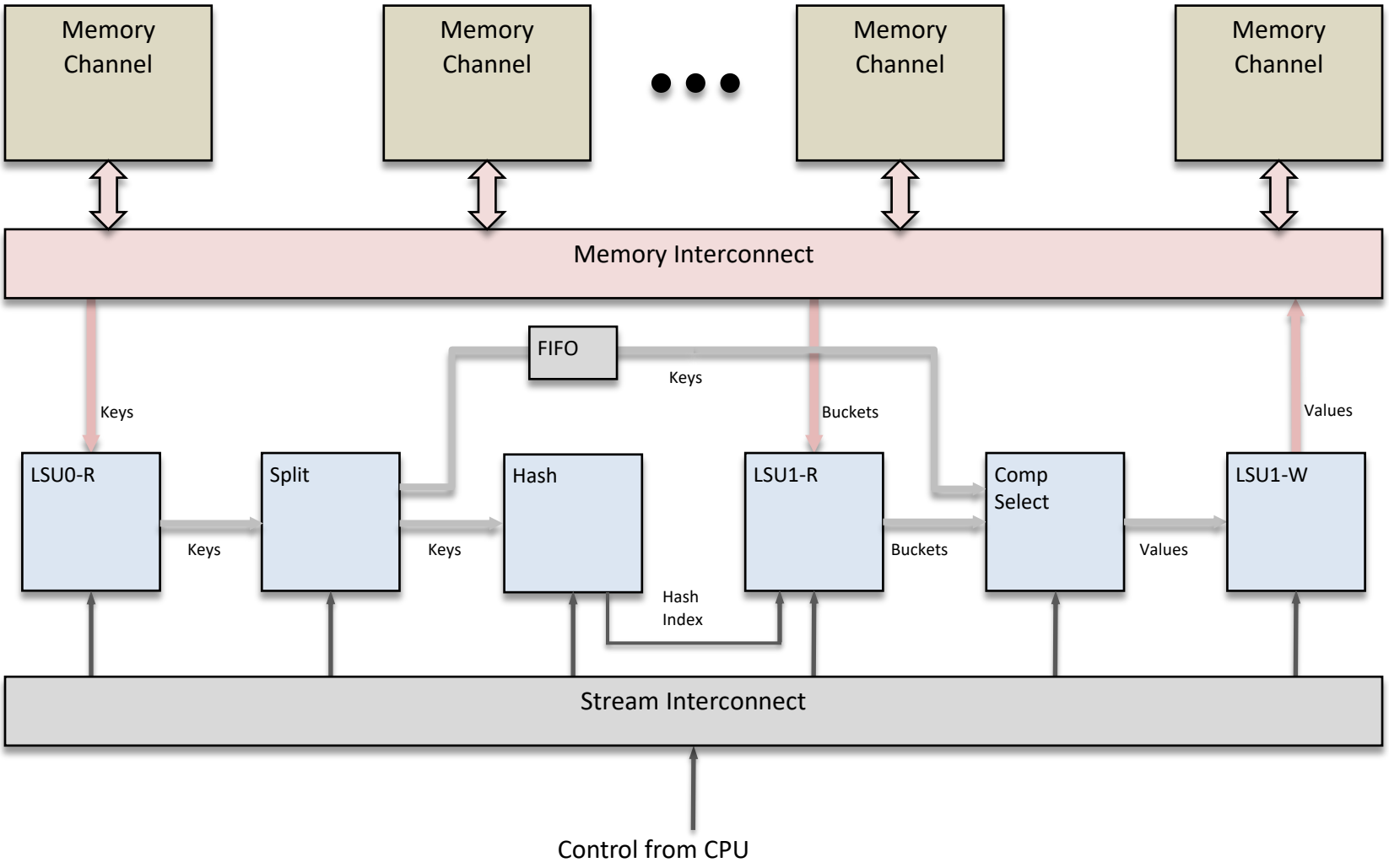
Insertion Algorithms

- Lookup pipeline is compatible with multiple open addressing insertion algorithms
- Linear insertion
  - Collisions are resolved by searching subsequent table locations for an empty slot
  - Can have large clusters (long probe sequence) at higher load factors
- Robin Hood hashing
  - Scans forward from initial probe point
  - Entries are swapped with one that has a shorter probe sequence length
  - Reduces the maximum probe sequence length
  - Our experiments use Robin Hood hashing

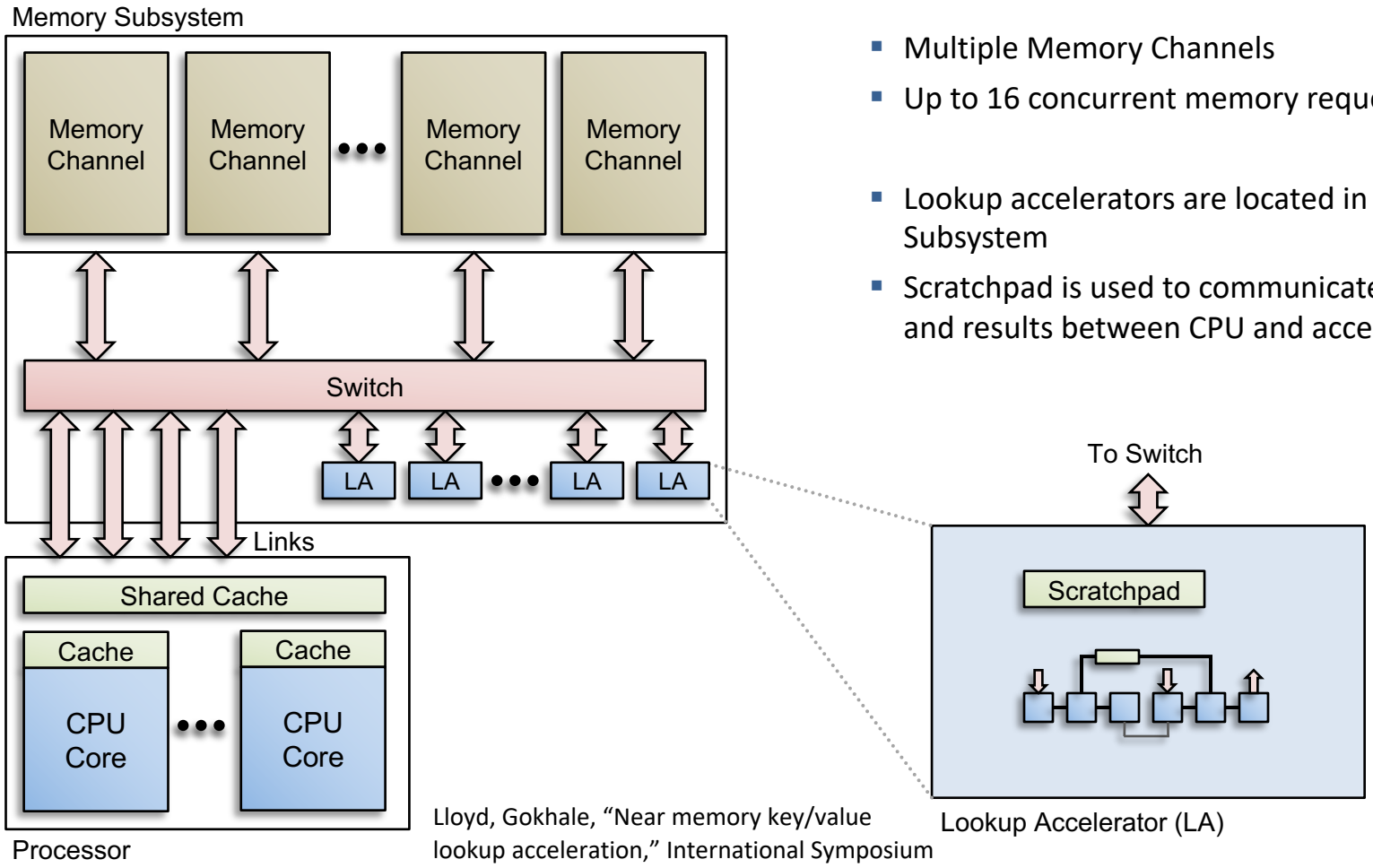
Near memory k/v building blocks



K/v lookup pipeline configuration



Lookup accelerator near memory



- 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

Lloyd, Gokhale, "Near memory key/value lookup acceleration," International Symposium on Memory Systems, 2017.

Evaluation: 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

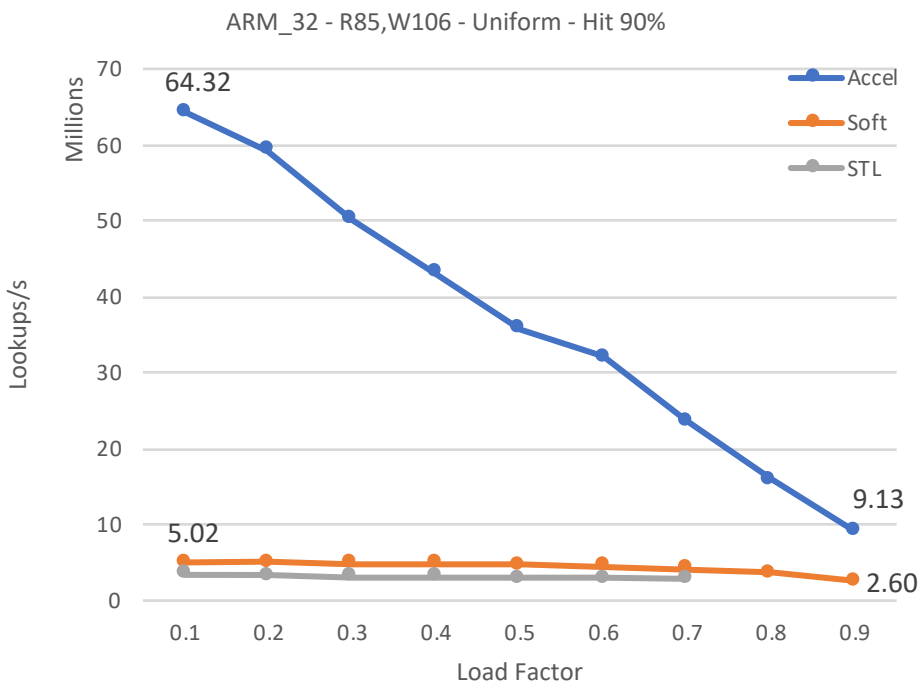
Lookup algorithms evaluated

- 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

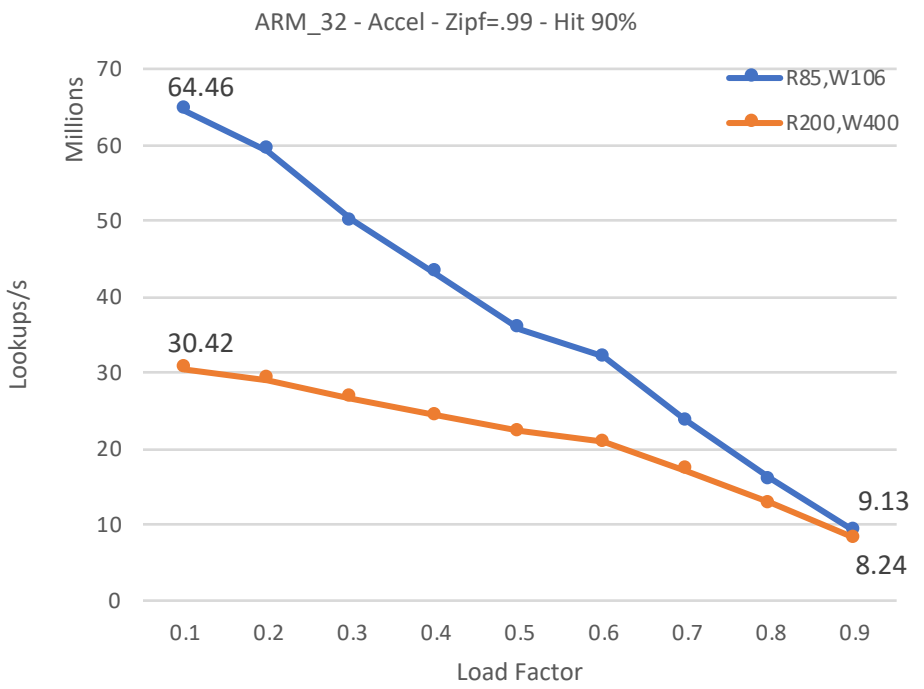
Lookup Performance

90% hit rate

Accelerator vs. Software



Low vs. Moderate Latency

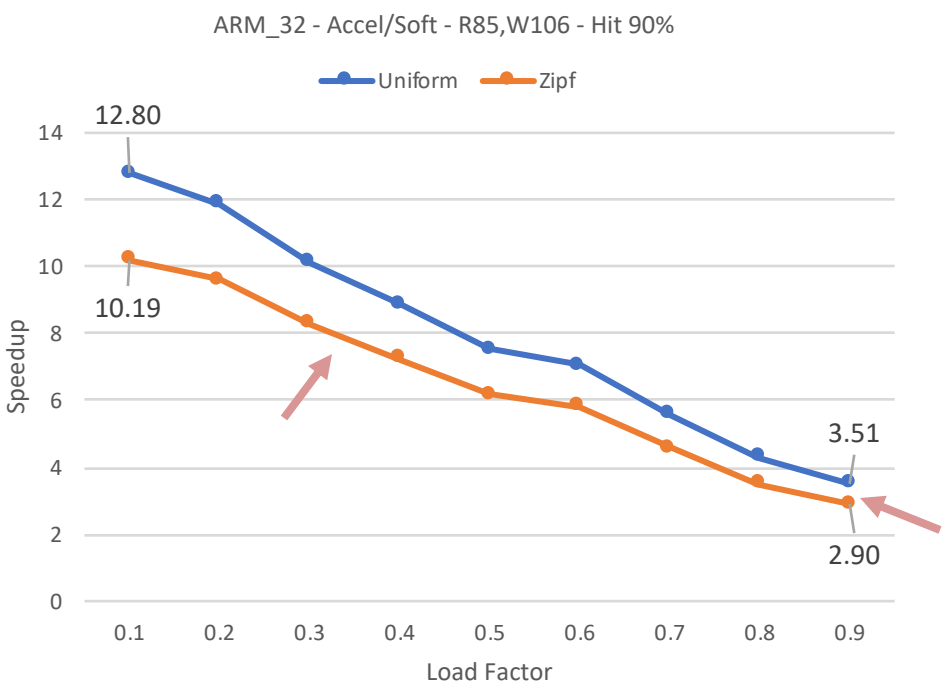


- Accel. performance does not vary with hit rate or key repeat frequency (scans entire PSL)
- Accel. performance decreases with increasing load (PSL) and memory latency
- Accel. performance comes from parallelism and more outstanding near memory requests
- Software is slower because of serialization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

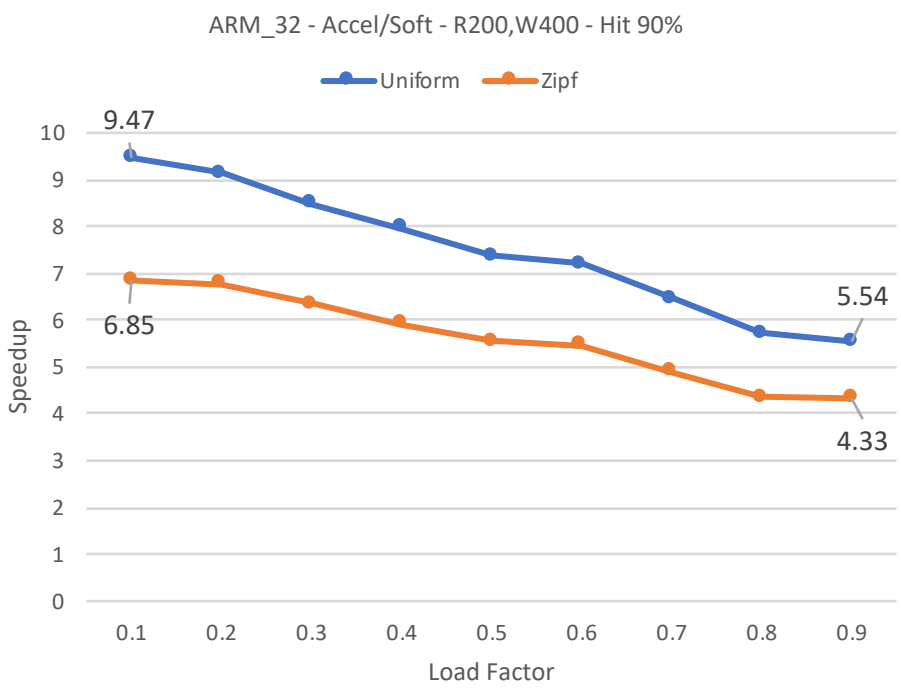
Speedup of Uniform and Zipfian Key Distributions

90% hit rate

Low Latency (DRAM)



Moderate Latency (SCM)

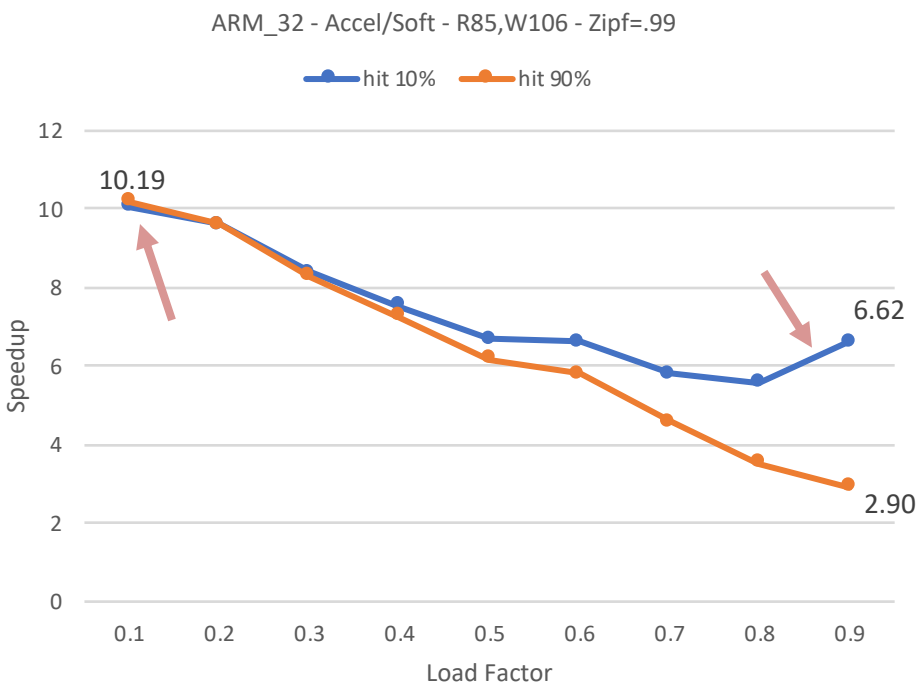


- Zipfian has less speedup because software has more query hits in CPU cache (lower)
- At higher load factors, the software is disadvantaged with more cache misses (convergence)

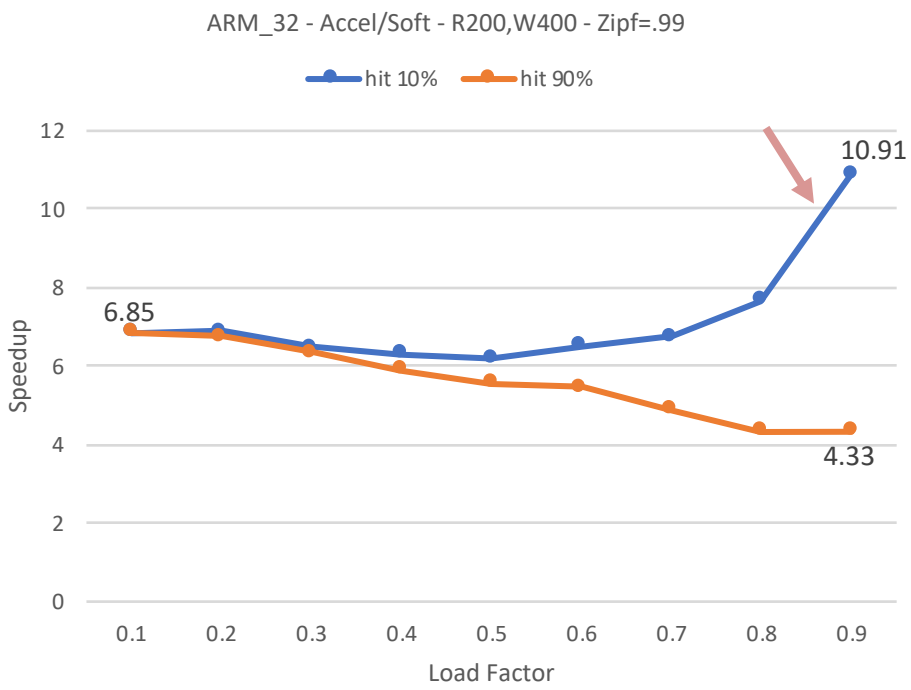
Speedup of 10% and 90% Hit Rate

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

CPU–accelerator communication overhead

- Cache flush and invalidate operations on shared buffers
  - 2.72 us per 1K block of keys
  - 2.65 ns avg. per key
- Command messages between CPU and Accelerator
  - Indicate the start and end of accelerator activity
  - Contain parameters and results
  - 1 us per 1K block of keys, ARM processor, no operating system
  - 1.5 us per message, x86 Linux platform, PCIe user-space driver
- Together, the communication overhead for a 1K block of keys ranges between 3% (high load) and 25% (low load)

Lookup accelerator summary

- Our simple, re-usable, interconnecting hardware building blocks are suitable for constructing a near memory associative lookup accelerator
- The building blocks have been composed in a synchronous high performance pipeline capable of delivering a result every few clock cycles
- Over a wide range of parameters, the accelerator has shown excellent speedup on both HMC/HBM memory and storage class memory
- Our lookup accelerator is tolerant to some memory latency, demonstrating up to an order of magnitude speedup with a pipelined architecture optimized for streaming, overlapped memory requests

Wrapup: multiple ways to materialize views

- Exploit page fault hardware to manipulate view at page (or greater) granularity
  - New asynchronous messaging protocol to eliminate kernel/user context switch
  - Materialize a view from multiple data sources incrementally, on demand
- Design new hardware to manipulate view at user-defined view buffer granularity
  - Gather/scatter engine
  - Associative lookup
- Research issues
  - Synchronization overhead
  - Granularity of view buffer
  - Coherence
  - Address mapping: virtual to system to physical

Team

- UMap: Marty McFadden, Roger Pearce, Keita Iwabuchi, Eric Green (all LLNL), Kai Wu, Dong Li (UC Merced)
- Near Memory Computing: Scott Lloyd, Chris Macaraeg, Joshua Landgraf, Nelson Ho, Eric Green (LLNL), Abhishek Jain (LLNL postdoc now at Xilinx)