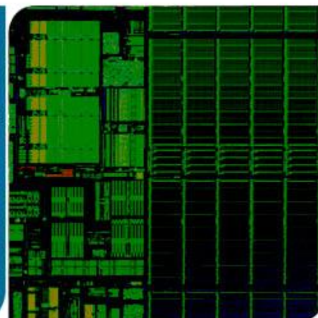
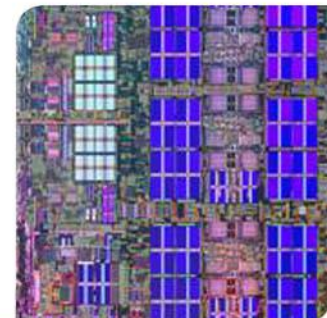
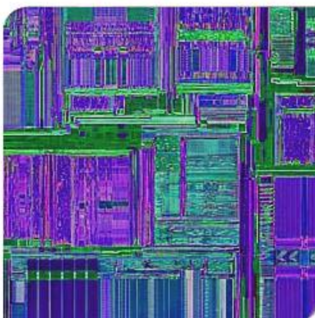
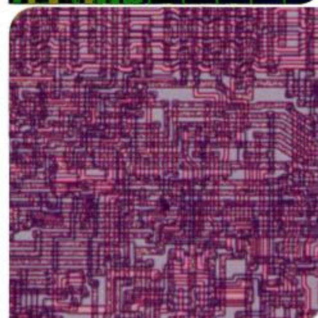
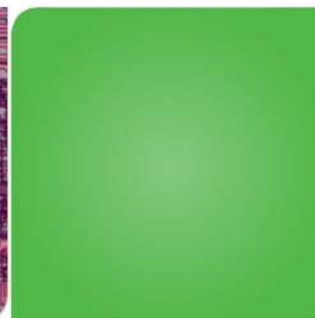
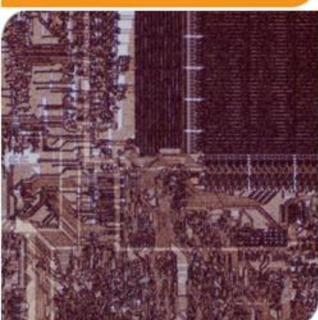






Strider: Architectures for Scalable Memory Centric Reduction of Sparse Data Streams



Sriseshan Srikanth, Tom Conte, Erik DeBenedictis

Georgia Tech

comparch

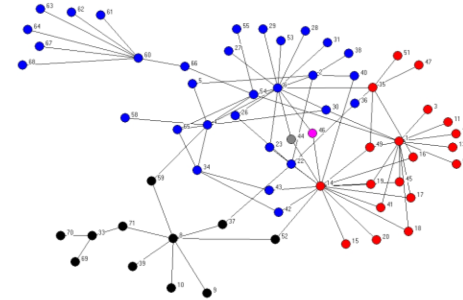
Reduction of Sparse Data Streams

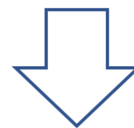
Graph Analytics

Cyber-security

HPC

ML





Sort and Reduce

Poor cache locality
Poor row locality
Low compute requirement

Abstraction: (key, value, $\oplus$)
Associative Array Reduction

Example of Sparse Reduction SpGEMM

1	-	4	-	-
-	-	-	-	-
2	3	-	1	-
-	-	5	-	-
-	-	2	2	2

×

1	-	4	-	-
-	-	-	-	-
2	3	-	1	-
-	-	5	-	-
-	-	2	2	2

=

1	-	4	-	-
-	-	-	-	-
2	-	8	-	-
-	-	-	-	-
-	-	-	-	-

+

8	12	-	4	-
-	-	-	-	-
-	-	-	-	-
10	15	-	5	-
4	6	-	2	2

+

-	-	-	-	-
-	-	-	-	-
-	-	5	-	-
-	-	-	-	-
-	-	10	-	-

+

-	-	-	-	-
-	-	-	-	-
-	-	-	-	-
-	-	-	-	-
-	-	4	4	4

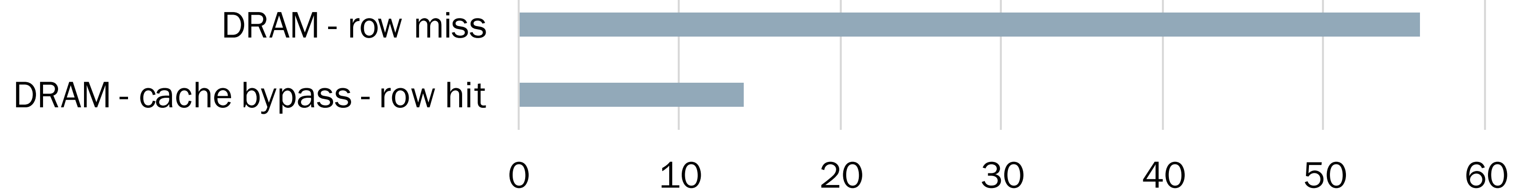
Key = Matrix Index

Value = Partial Product

$\oplus$ = Summation

Dense vs Sparse Applications

Optimized DRAM Row Performance results in <u>order of magnitude speedup</u> for sparse applications



Application	Typical LLC Miss Ratio	Potential Application Speedup due to Optimized DRAM Row Performance
SPEC 2006	< 10%	< 10%
SpGEMM *	> 50%	2x – 10x

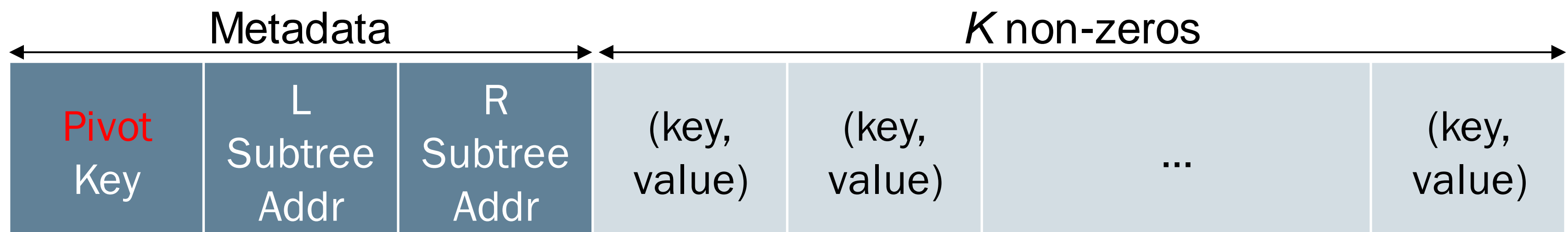
* SpGEMM: Key kernel in graph analytics, HPC

DRAM-Centric Design for Sparse Reductions

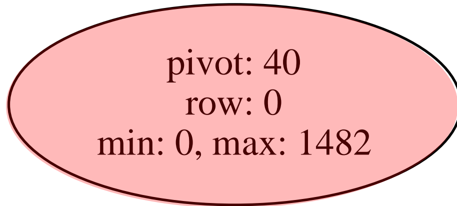
- Bypass caches
- Maximize DRAM row buffer utilization
- Increase algorithmic granularity of operation to 1 DRAM row, to avoid wasted reads and activations
  - Conventional approach: activate 2048 byte row, read 64 byte-worth cache block, perhaps use 8 bytes out of that

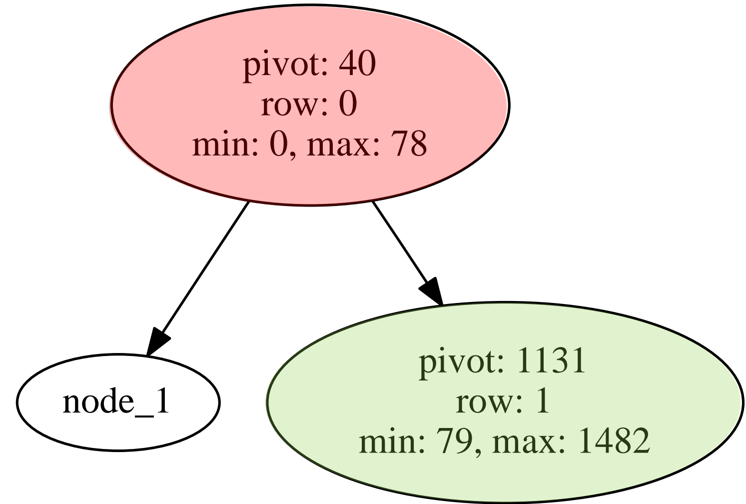
DRAM-Aware Sparse Representation

- Two-level hierarchy
- Lower Level:
  - (At most) *K* sorted (by key) non-zeros: node
  - Each node has a logical pivot
- Higher Level:
  - Pivots form a binary tree
  - Elements in left subtree have keys less that pivot
- Operation granularity: node



Addvec() Vector Insertion with Local Reduction





New	0	39	40	78	1482
Row 0	0	39	40	78	1482

New	40	79	936	1131	1132

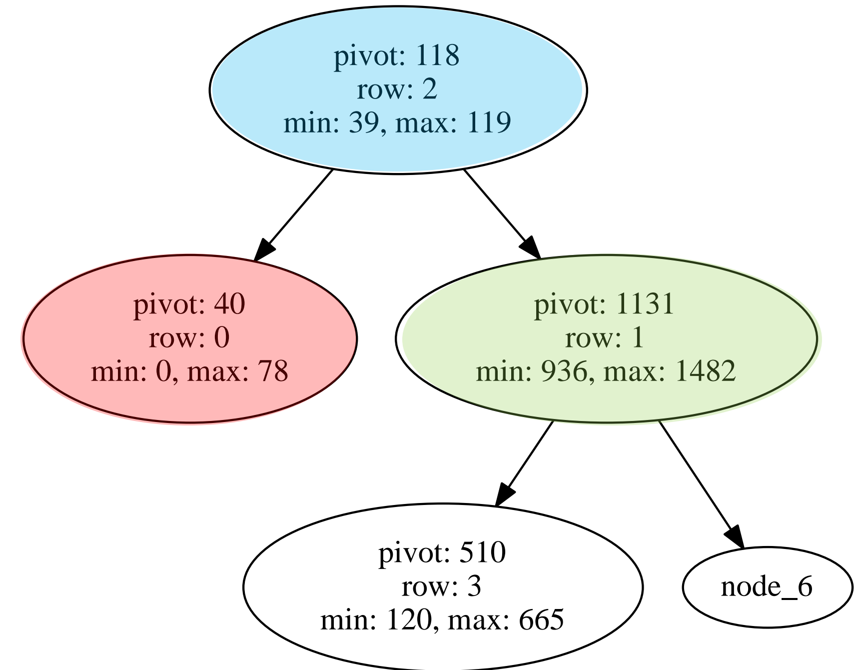
New	40	79	936	1131	1132
Row 0	0	39	40	78	
Row 1	79	936	1131	1132	1482

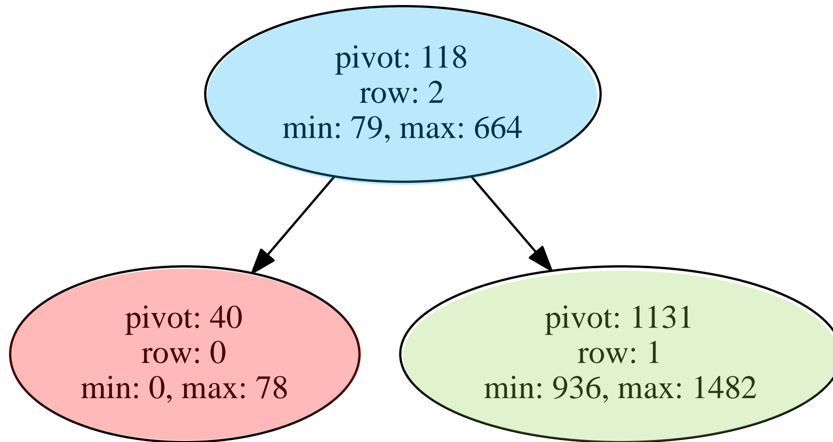
0	39	40	40	78	79	936	1131	1132	1482

Send *K* higher records to right subtree

Addvec()
Vector Insertion with Local Reduction

New	40	79	936	1131	1132
Row 0	0	39	40	78	
Row 1	79	936	1131	1132	1482





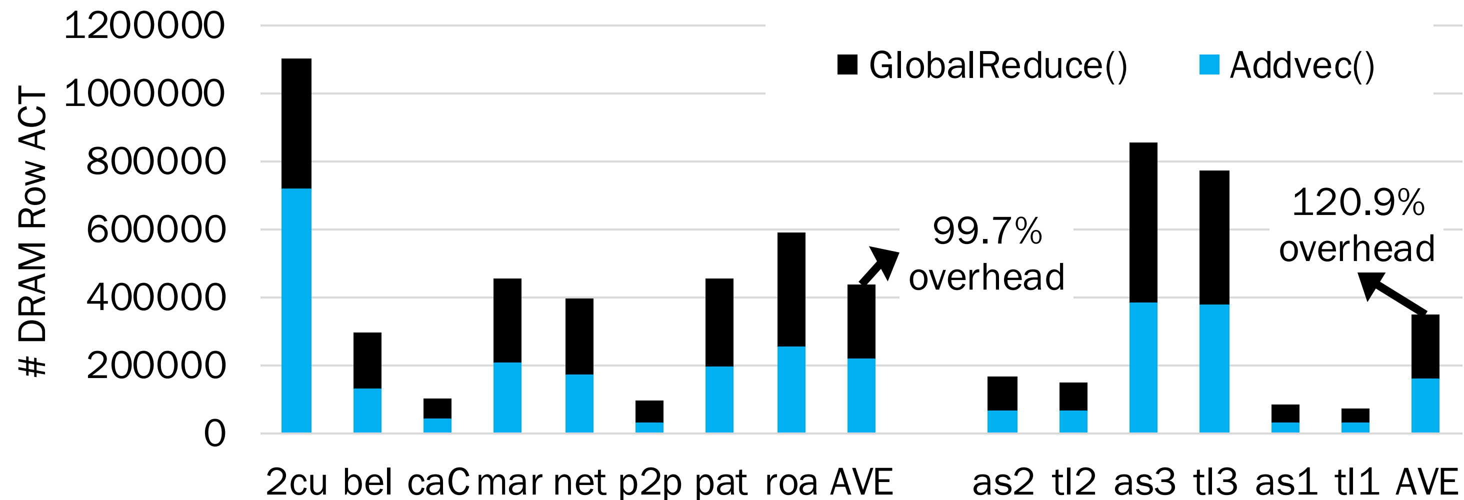
New	79	80	118	119	664
Row 0	0	39	40	78	
Row 1	936	1131	1132	1482	
Row 2	79	80	118	119	664

New	39	120	509	510	665
Row 0	0	39	40	78	
Row 1	936	1131	1132	1482	
Row 2	39	79	80	118	119
Row 3	*120*	509	510	664	665

GlobalReduce()

- Addvec() recurses over exactly one of L/R subtrees
  - Necessary for performance as tree grows
  - Disadvantage: duplicates may occur along different paths of tree
    - However, this is NOT the common case
- GlobalReduce(): Final dedup pass
  - Upon DFS traversal, detect and fix violations:
    if (max of L subtree) > (min of R subtree), then
    extract argmax row from L subtree and perform Addvec()

GlobalReduce() Overhead



- Testing for violation requires row ACT
  - This is redundant in the common case
  - Solution:
    - Include min, max info in metadata
    - Decouple metadata

Architecture v1 – Bitonic Merge

Indexable SRAM 1 (K records)

Indexable SRAM 2 (K records)

Buffer (C records)

Buffer (C records)

Mux

Mux

FU FU FU FU FU FU FU FU

FU FU FU FU FU FU FU FU

FU FU FU FU FU FU FU FU

Buffer (C records)

Buffer (C records)

Near Data Processing Unit

Merge Compress Unit

Control State Machine

MetaData Store

CPU

L1

L2

MetaStrider Port

System Interconnect (Crossbar)

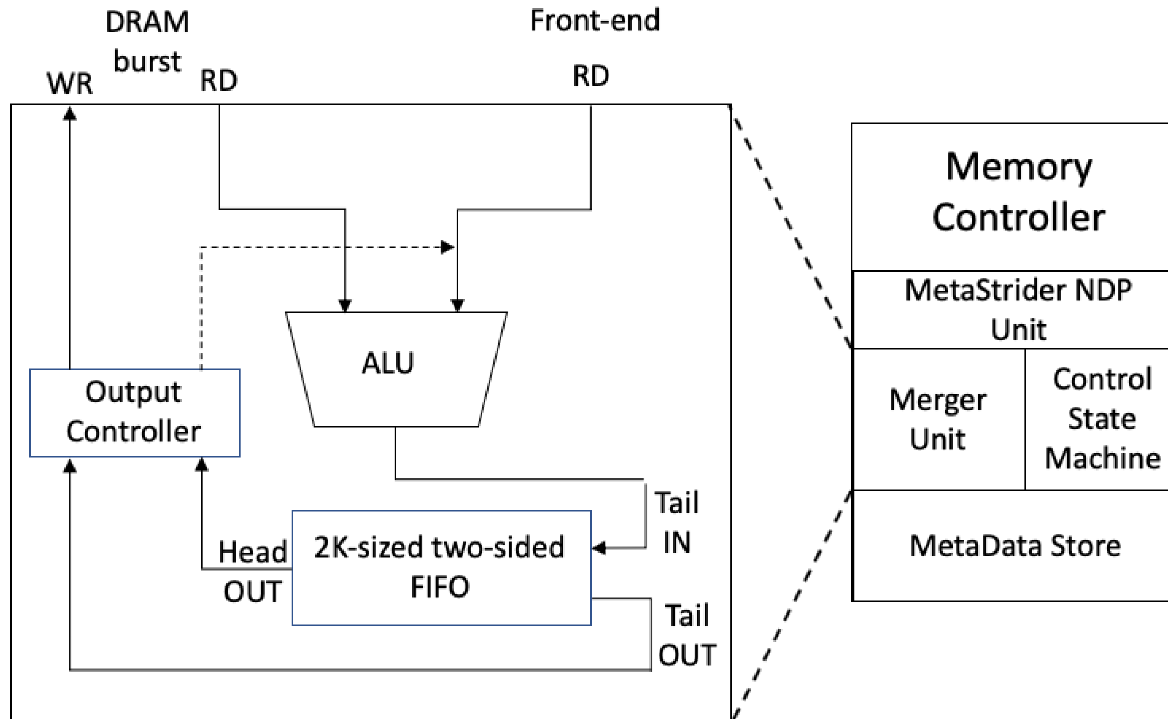
MetaStrider Mem Ctrl

NDP

Memory Controller

Architecture v2 – Linear-Merge

- However, observe that most of the input to bitonic network is from existing nodes (due to Addvec() recursion)
  - Each node is sorted
  - Don't need bitonic network, simple linear merge sufficient



- Linear-merge interleaves logic and memory at fine granularity (DRAM burst)
- Linear speedup with increased burst width (DDR vs HBM)

V1: Bitonic network can be used for efficient sorting. Merging in O(N lg N).
V2: If input vectors are pre-sorted, merging in O(N).

Strider Variants

- SuperStrider
  - Bitonic merge network
  - No decoupled metadata
  - No tree balancing
- MetaStrider
  - Linear merge network
  - Decoupled metadata
  - Tree balancing
  - Hashing
  - Can be implemented with or without dedicated hardware at memory controller (NDP)

API

```
1  // A and B may be stored in conventional sparse formats or in the MetaStrider format.
2  SpGEMM_OuterProduct (Matrix A, Matrix B):
3    // Initialize MetaStrider memory.
4    K = MetaStrider.Init(12, 1, "+")
5    colA = A.readNextNonZeroColumn()
6    rowB = B.readNextNonZeroRow()
7    // Assume that the column index of colA and the row index of rowB are identical (k). If
         not, advance accordingly until they are (not shown).
8    for (i, k, A_ik) in colA:
9      for (k, j, B_kj) in rowB:
10       key = makeKey(i, j)
11       value = A_ik * B_kj
12       sortedQueue.pushAndSort((key, value))
13       if (sortedQueue.size() == K or EOF):
14         // Stream records for Addvec().
15         while (sortedQueue.size() > 0):
16           MetaStrider.EnqReduce(sortedQueue.popMin())
17         MetaStrider.Addvec()
18   // Repeat lines 5-17 until A or B consumed. Then:
19   MetaStrider.GlobalReduce()
20   // Perform gather for downstream operations as needed.
21   MetaStrider.Lookup()
```

Evaluation

- Workloads
  - (8) SpGEMM – Undirected/directed graphs, 2D/3D, Electromagnetics, Road networks
  - (9) Firehose – Cybersecurity benchmark
    - Key: IPv6 address
    - Value: bias bit, ground truth (for reference)
    - Reduction:
      - Count incidence of key (+1 logic)
      - If count exceeds static threshold, flag future incidence and test bias (compare logic)
    - Requires support for incremental updates
- Gem5 - single HBM channel simulation
- In-house – rapid design space exploration

Space of Reducers

- Software
  - – Scalar balanced red-black binary tree (std::map) – CPU only (baseline)
  - – Kokkos hashmap accumulator – CPU or GPU
- Hardware
  - – GraFBoost – hierarchical merge-sort – unable to handle incremental updates (limited to SpGEMM)
  - – SuperStrider
  - – MetaStrider

DRAM Performance

Scalar Binary Tree | Kokkos | SuperStrider | MetaStrider

Reduction in Row Activations

60
50
40
30
20
10
0

Reduction in DRAM Traffic

8
6
4
2
0

2cu bel caC mar net p2p pat roa HM pl1 pl2 pl3 as1 as2 as3 tl1 tl2 tl3 HM w/o pl

MetaStrider improves DRAM performance and energy efficiency by an <u>order of magnitude</u>

Uncore Energy

Average Un-core Energy (mJ)

0
20
40
60
80
100

Scalar Binary Tree
Kokkos
SuperStrider
Non-NDP MetaStrider
MetaStrider

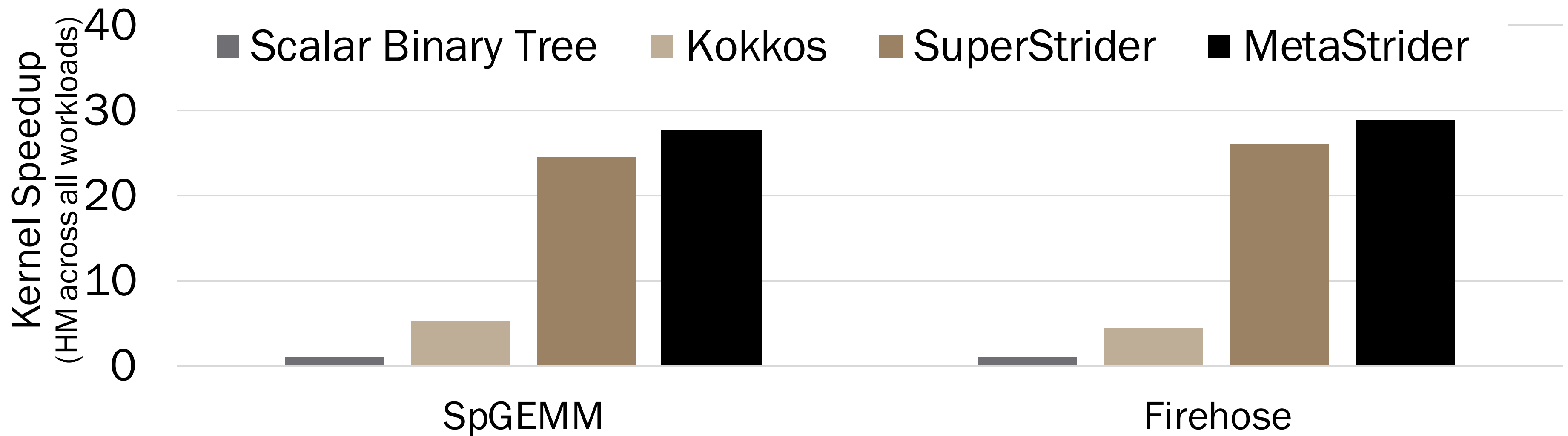
L1-L2 Bus
Merger Unit
Metadata Store
LLC
System Interconnect
DRAM

Any energy overheads due to MetaStrider's Merger and Metadata are overshadowed by energy reductions due to efficient data management

Sriseshan Srikanth

Georgia Tech comparch

System Performance



Order of magnitude performance improvement on the reduction kernel renders application performance that is **within 8% of an ideal** hypothetical accelerator that performs reduction in zero time

Batch Mode – SpGEMM only

	Grafboost	MetaStrider (Batch)
Reduction in Row Activations	1	~1.35
Reduction in DRAM Traffic	1	~1.35
Kernel Speedup	1	~1.27
Reduction in Un-core Energy	1	14.9

Even when ability to handle incremental updates is discarded, MetaStrider is relatively faster and more efficient

Georgia Tech comparch

Data Movement on System Interconnect

Reduction in Crossbar Traffic

120
100
80
60
40
20
0

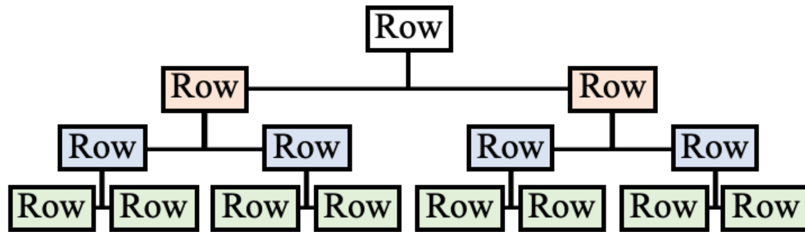
■ Non-NDP MetaStrider ■ MetaStrider

34.82
1.54
75.19
3.24

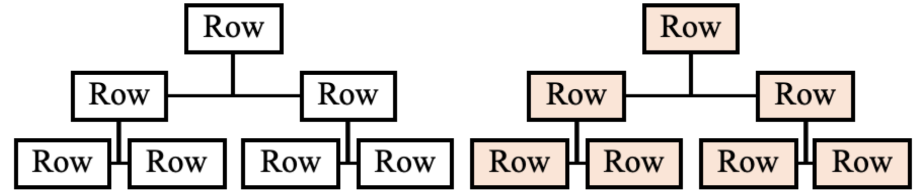
2cu bel caC mar net p2p pat roa HM pl1 pl2 pl3 as1 as2 as3 tl1 tl2 tl3 HM w/o pl

MetaStrider reduces data movement even without extra hardware
With NDP, over an <u>order of magnitude reduction</u> is seen

Scaling with MLP, Parallel Front-end



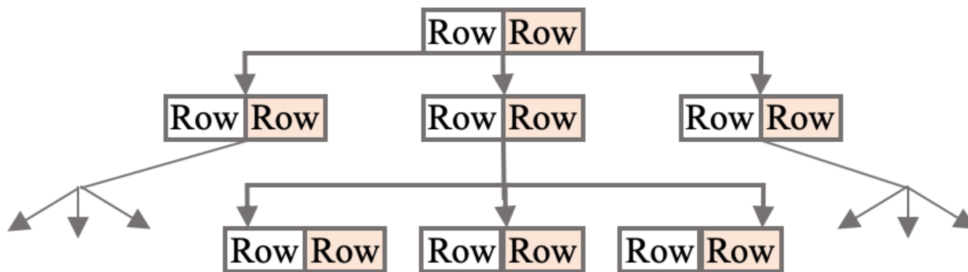
Pipelining



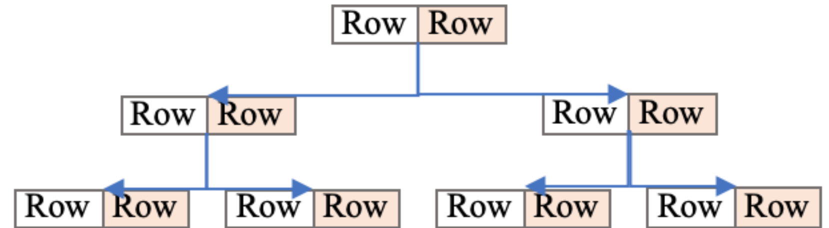
Partitioning

Input	T = 0	1	2	3	4	5	6	7	8
I0	Level 0	Level 1	Level 2	Level 3	Level 4	Retire			
I1		Level 0	Level 1	Level 2	Level 3	Level 4	Retire		
I2			Level 0	Level 1	Level 2	Level 3	Level 4	Retire	
I3				Level 0	Level 1	Level 2	Level 3	Level 4	Retire

Fanout

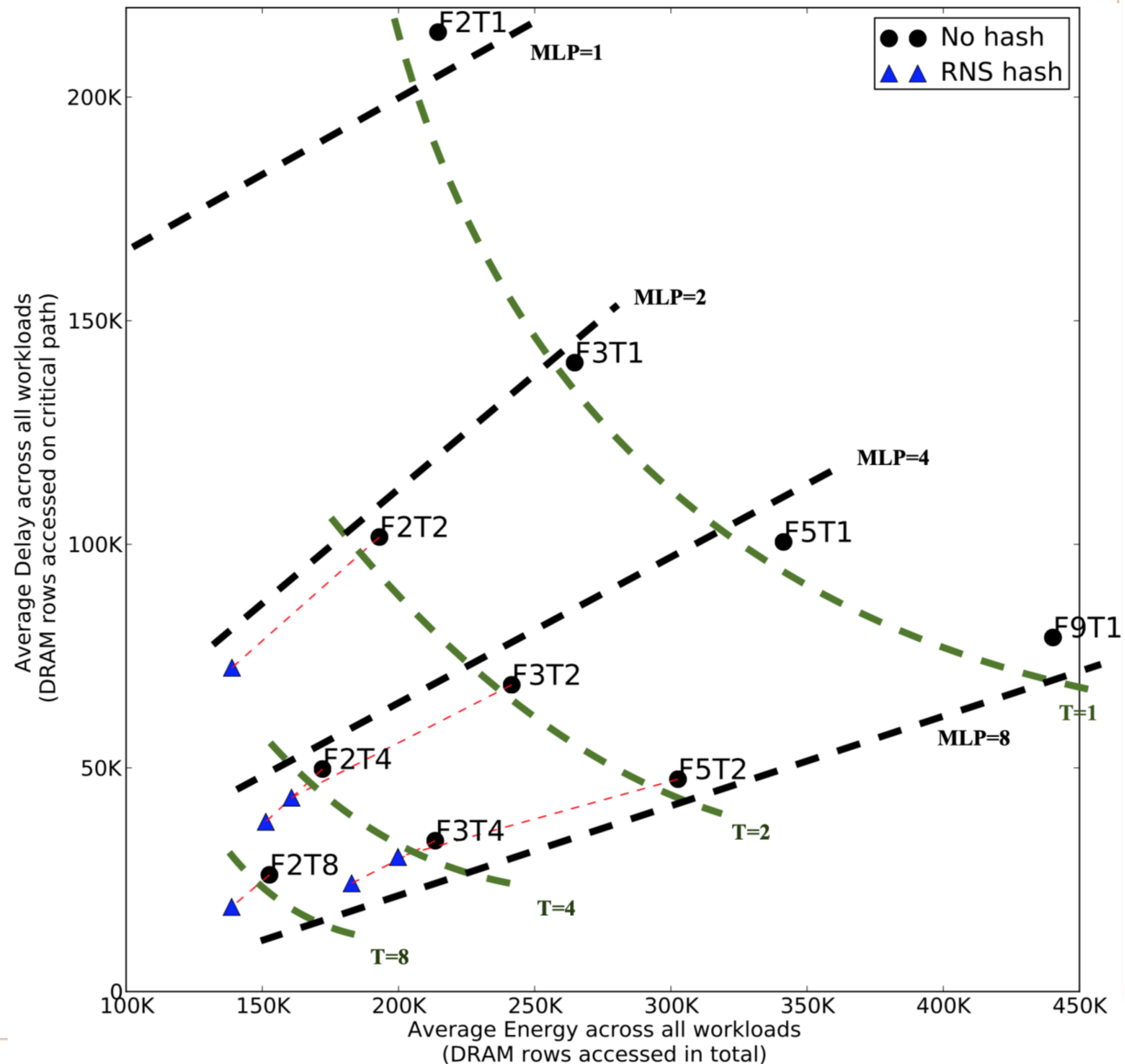


Grouping



Scaling with MLP, multi-core

- <u>Takeaway</u>: Performance ↑ ~**linearly** using our strategies.
- <u>Pipelining</u>: 80% of available MLP achievable using simple heuristics.
- <u>Grouping</u>: Requires high frequency front-end cores.
- <u>Partitioning vs Fanout</u>: see chart on right. Partitioning is better if parallel front-end available.



Conclusion

- Sparse data applications widely used in graph analytics, cybersecurity, HPC, ML.
  - Low compute to communicate ratio, low locality of reference.
- Traditional architectures NOT energy-efficient:
  - Significant / redundant data-movement through memory hierarchy.
- Solution: Scalable memory-centric architectures
  - SuperStrider
  - MetaStrider
    - Scales with tighter logic-memory integration, MLP

Thank You

seshan@gatech.edu

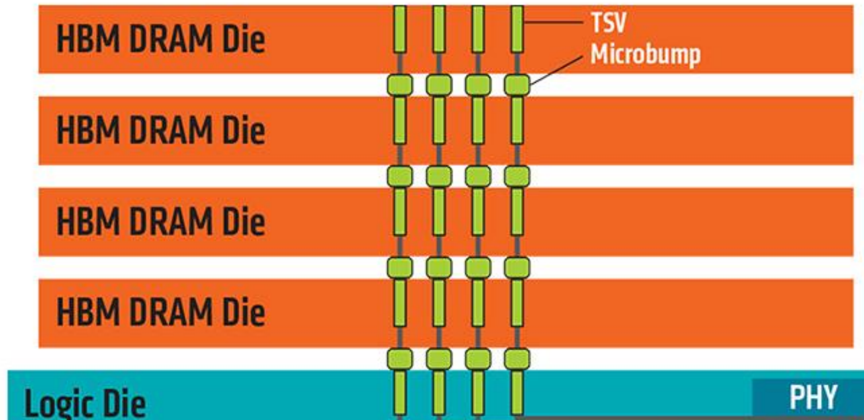
Georgia Tech

comparch

Typical Memory Latencies

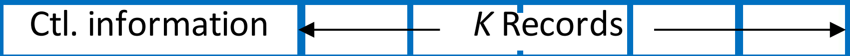
- L1: 1ns
- L2: 1 + 3 = 4ns
- L3: 1 + 3 + 10 = 14ns
- DRAM (no cache bypass)
  - 14 + 14 = 28ns (hit)
  - 14 + 28 = 42ns (empty)
  - 14 + 42 = 56ns (miss)
- Similar trends for energy (nJ); off-chip data movement ~2 orders of magnitude higher than FLOP.

Superstrider Architecture

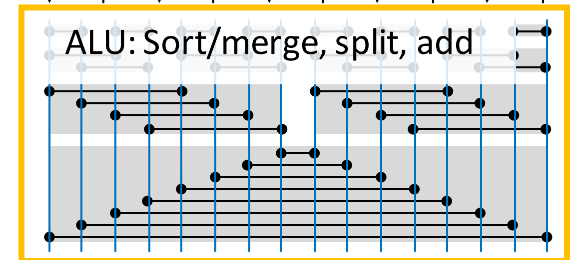


DRAM bank:

Addr:(size)	Pivot	Addr(size)	Addr(size)	22 22 Results				
4:(5)	6	0(0)	0(0)	[1]=0.52	[2]=0.02	[3]=0.80	[6]=0.28	[7]=1.61
2:(10)	13	4(5)	0(0)	[8]=0.59	[10]=0.14	[11]=1.66	[12]=0.27	[13]=0.61
0:(22)	17	2(10)	1(7)	[14]=0.09	[16]=0.77	[17]=2.75	[18]=1.46	[19]=0.47
1:(7)	20	0(0)	3(2)	[20]=1.31	[21]=0.26	[22]=0.45	[25]=0.07	[26]=1.59
3:(2)	28	0(0)	0(0)	[27]=0.20	[28]=1.56			



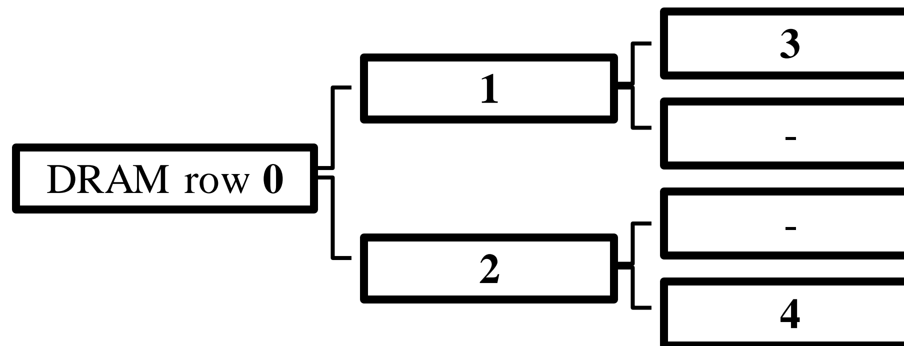
Control logic: Scalar arithmetic, includes stack and length



https://www.amd.com/en/technologies/hbm

Superstrider Data Organization

Addr:(size)	Pivot	L subtree	R subtree	Records (*K* = 5)				
3:(2)	28	0(0)	0(0)	[27]=0.20	[28]=1.56			
1:(7)	20	0(0)	3(2)	[20]=1.31	[21]=0.26	[22]=0.45	[25]=0.07	[26]=1.59
0:(22)	17	2(10)	1(7)	[14]=0.09	[16]=0.77	[17]=2.75	[18]=1.46	[19]=0.47
2:(10)	13	4(5)	0(0)	[8]=0.59	[10]=0.14	[11]=1.66	[12]=0.27	[13]=0.61
4:(5)	6	0(0)	0(0)	[1]=0.52	[2]=0.02	[3]=0.80	[6]=0.28	[7]=1.61



Superstrider: Key Principles

- Memory rows organized as a binary tree with *K* records (sorted by key) and a *pivot* (key) per row.
- Memory access and computation granularity:
  - 1 memory row of **sorted** records.
  - SIMD-style operation tightly integrated with wide memory words.
- Sorted invariant
  - Two *N*/2 length pre-sorted vectors can be merged in $\log_2(N)$ stages.
  - Novel algorithms: Fast insertion/lookup into binary tree; fast compression (later).

Superstrider Operation

Pre-sorted input				[2]=0.02	[14]=0.09	[17]=0.49	[20]=0.27	[22]=0.45
Addr:(size)	**Pivot**	**L subtree**	**R subtree**	**Records ($K = 5$)**				
Accumulator				[2]=0.02	[14]=0.09	[17]=0.49	[20]=0.27	[22]=0.45
Addr:(size)	**Pivot**	**L subtree**	**R subtree**	**Records ($K = 5$)**				
0:(5)	17	0(0)	0(0)	[2]=0.02	[14]=0.09	[17]=0.49	[20]=0.27	[22]=0.45
Accumulator				[2]=0.02	[14]=0.09	[17]=0.49	[20]=0.27	[22]=0.45

Pre-sorted input				[13]=0.50	[13]=0.04	[18]=0.63	[20]=0.61	[27]=0.20
Addr:(size)	**Pivot**	**L subtree**	**R subtree**	**Records ($K = 5$)**				
0:(5)	17	0(0)	0(0)	[2]=0.02	[14]=0.09	[17]=0.49	[20]=0.27	[22]=0.45
Accumulator				[13]=0.50	[13]=0.04	[18]=0.63	[20]=0.61	[27]=0.20

Addvec: Merge and Compress

Addr:(size)	Pivot	L subtree	R subtree	Records ($K = 5$)				
0:(5)	17	0(0)	0(0)	[2]=0.02	[14]=0.09	[17]=0.49	[20]=0.27	[22]=0.45
Accumulator				[13]=0.50	[13]=0.04	[18]=0.63	[20]=0.61	[27]=0.20

- Row 0 and Accumulator:

[2]=0.02	[14]=0.09	[17]=0.49	[20]=0.27	[22]=0.45	[13]=0.50	[13]=0.04	[18]=0.63	[20]=0.61	[27]=0.20

- Merge:

[2]=0.02	[13]=0.50	[13]=0.04	[14]=0.09	[17]=0.49	[18]=0.63	[20]=0.27	[20]=0.61	[22]=0.45	[27]=0.20

- Compress

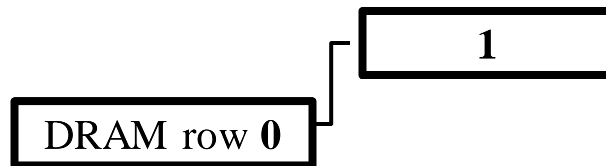
[2]=0.02	[13] = 0.50 + 0.04	[14]=0.09	[17]=0.49	[18]=0.63	[20] = 0.27 + 0.61	[22]=0.45	[27]=0.20

Addvec: Recurse

[2]=0.02	[13] = 0.54	[14]=0.09	[17]=0.49	[18]=0.63	[20] = 0.88	[22]=0.45	[27]=0.20

- Red: 3
- Green: 5
- Keep red in node, propogate (Addvec) green to “>” child.

Addr:(size)	**Pivot**	**L subtree**	**R subtree**	**Records (K = 5)**				
1:(5)	20	0(0)	0(0)	[17]=0.49	[18]=0.63	[20] = 0.88	[22]=0.45	[27]=0.20
0:(8)	17	0(0)	1(5)	[2]=0.02	[13] = 0.54	[14]=0.09		



Need for Normalization

Pre-sorted input				[7]=0.68	[13]=0.07	[18]=0.82	[26]=0.25	[28]=0.99
Addr:(size)	**Pivot**	**L subtree**	**R subtree**	**Records (K = 5)**				
1:(5)	20	0(0)	0(0)	[17]=0.49	[18]=0.63	[20] = 0.88	[22]=0.45	[27]=0.20
0:(8)	17	0(0)	1(5)	[2]=0.02	[13] = 0.54	[14]=0.09		

- Row 0 and accumulator:

[2]=0.02	[13] = 0.54	[14]=0.09	[7]=0.68	[13]=0.07	[18]=0.82	[26]=0.25	[28]=0.99

- Merge and Compress

[2]=0.02	[7]=0.68	[13]=0.61	[14]=0.09	[18]=0.82	[26]=0.25	[28]=0.99

- #(Red) > #(Green) => Propogate (Addvec) red to “<“ child.

Addr:(size)	**Pivot**	**L subtree**	**R subtree**	**Records (K = 5)**				
1:(5)	20	0(0)	0(0)	[17]=0.49	[18]=0.63	[20] = 0.88	[22]=0.45	[27]=0.20
0:(12)	17	2(4)	1(5)	[18]=0.82	[26]=0.25	[28]=0.99		
2:(4)	13	0(0)	0(0)	[2]=0.02	[7]=0.68	[13]=0.61	[14]=0.09	

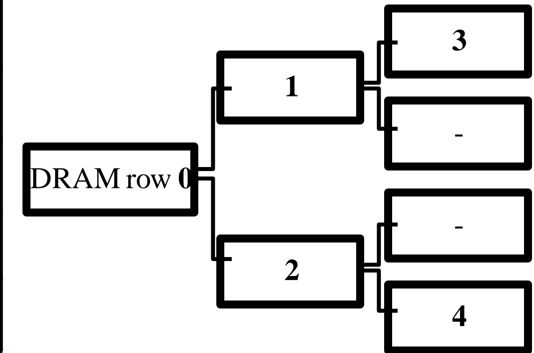
- Performance advantage: Addvec accesses L subtree OR R subtree, but NOT both.

Superstrider Architecture

Addr:(size)	Pivot	L subtree	R subtree	Records ($K = 5$)				
3:(2)	28	0(0)	0(0)	[27]=0.20	[28]=1.56			
1:(7)	20	0(0)	3(2)	[20]=1.31	[21]=0.26	[22]=0.45	[25]=0.07	[26]=1.59
0:(22)	17	2(10)	1(7)	[14]=0.09	[16]=0.77	[17]=2.75	[18]=1.46	[19]=0.47
2:(10)	13	4(5)	0(0)	[8]=0.59	[10]=0.14	[11]=1.66	[12]=0.27	[13]=0.61
4:(5)	6	0(0)	0(0)	[1]=0.52	[2]=0.02	(3)=0.80	(6)=0.28	(7)=1.61

Control information

Open row buffer



Merge networks
- Forward: Merges two sorted vectors.
- Backward: Deletes empty records after compression.

Function unit pool
Applies a collision function to records with identical keys

Control logic
sets which row to access next

Accumulator

Sorted stream of input records, with possibly repeated keys

Superstrider Architecture

Functional "schematic": Data flow "layout" →

Merge function, data flow →

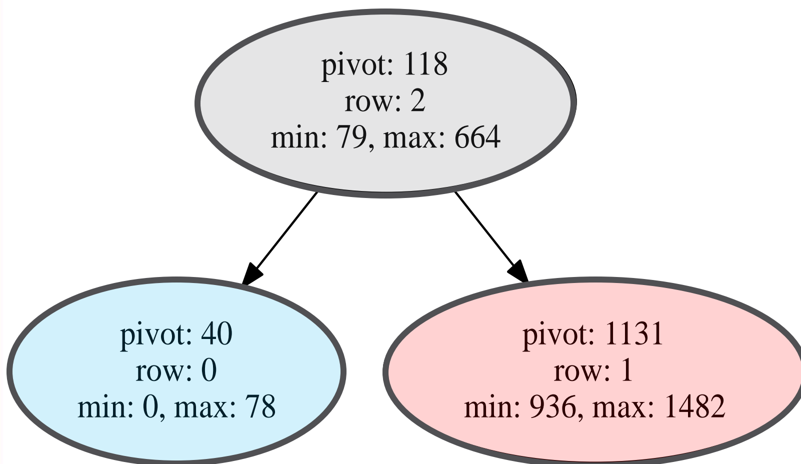
Physical module:
Logic layer →
Memory layer →

Short distance, parallel, interface

Georgia Tech comparch

SuperStrider / MetaStrider

- Memory-centric architecture operates on DRAM rows directly.
- Two-level, in-memory representation for efficient insertion and associative reduction.

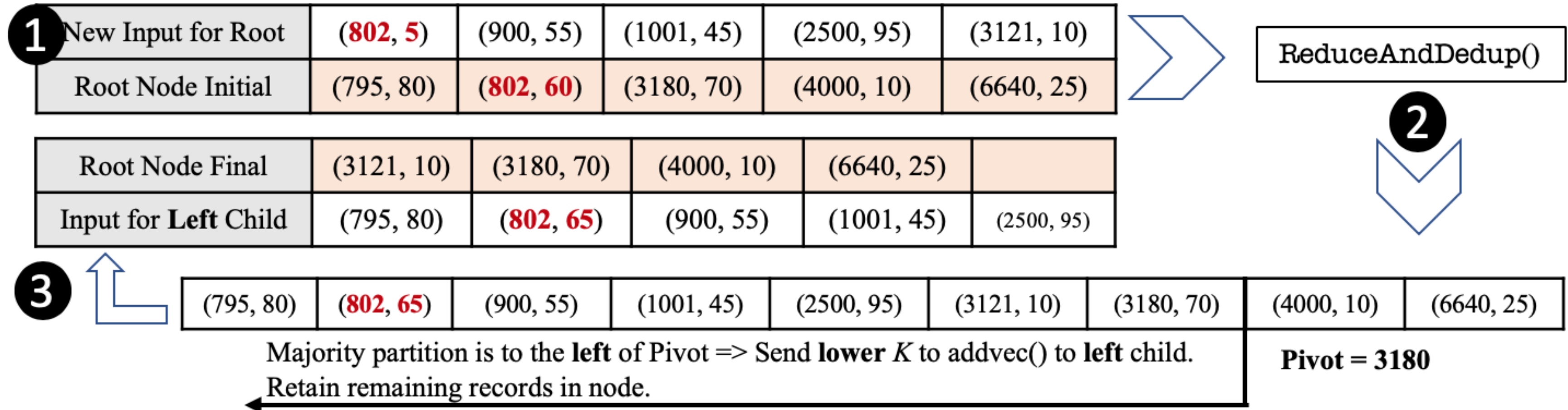


Row 0	0	39	40	78	
Row 1	936	1131	1132	1482	
Row 2	79	80	118	119	664

Example: 5 (key, value) pairs fit in each DRAM row, forming a node in a balanced binary tree

Georgia Tech comparch

Fundamental Operation

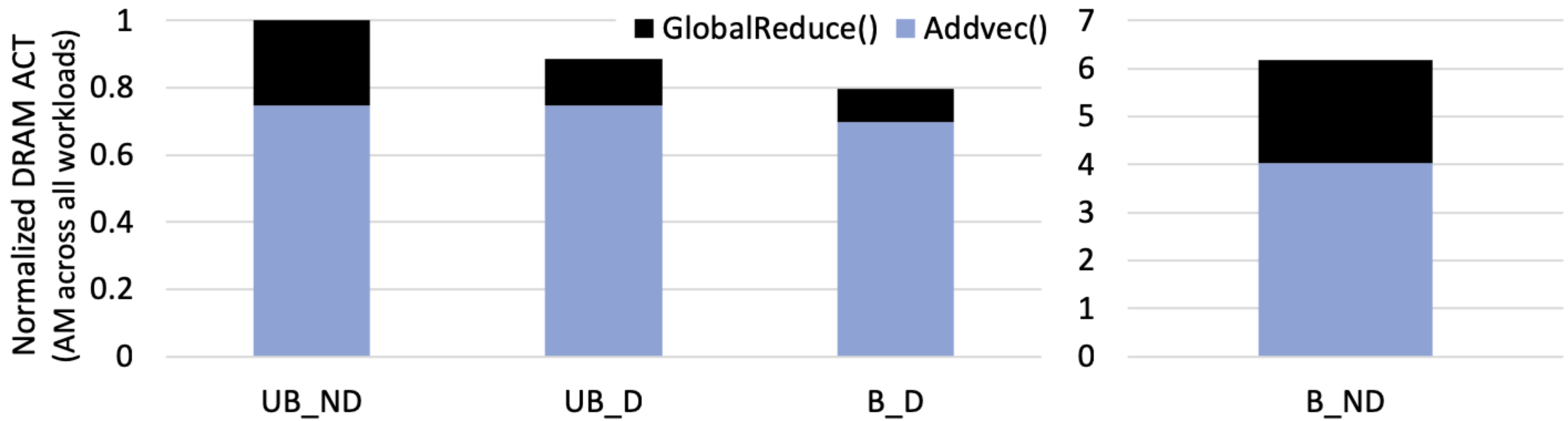


ReduceAndDedup() is called recursively along one path of the tree until all input records are inserted / reduced.

Realtime Capability

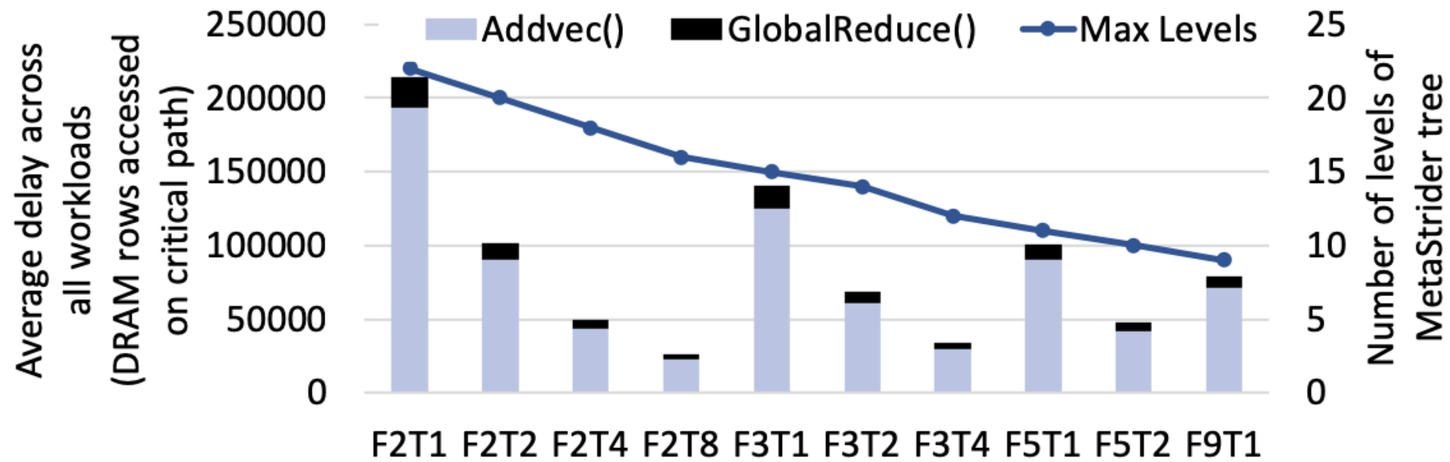
- Experiments performed at regular intervals of Addvec():
  - Exact match: 96.5%
  - Partial match: 2%
    - Omitted if GlobalReduce() called
  - No match: 1.5%
    - Omitted if no tree balance done
    - Omitted if GlobalReduce() called

Benefits of Decouple, Balance

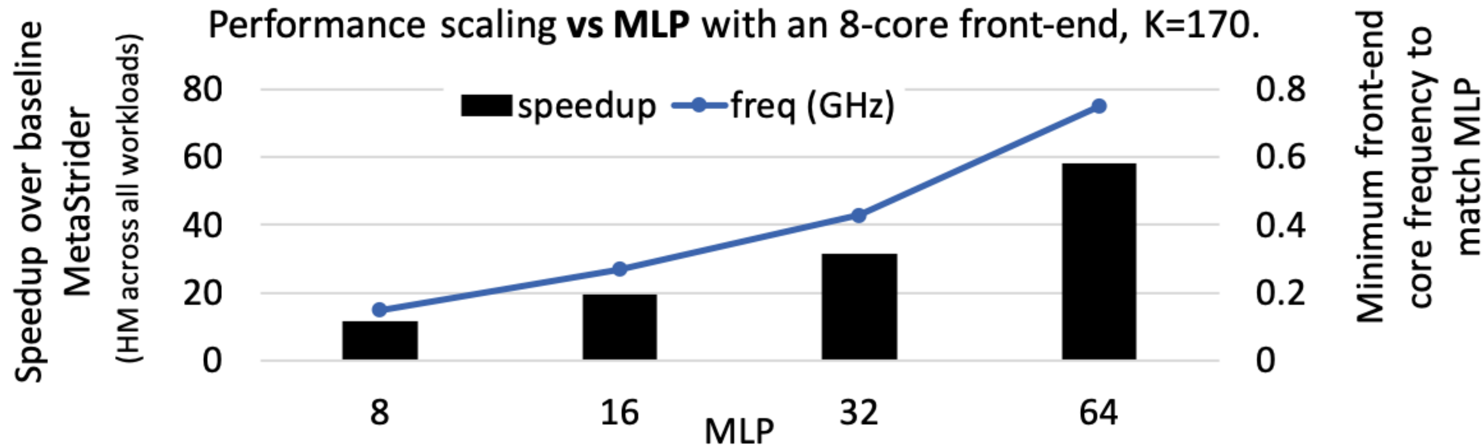


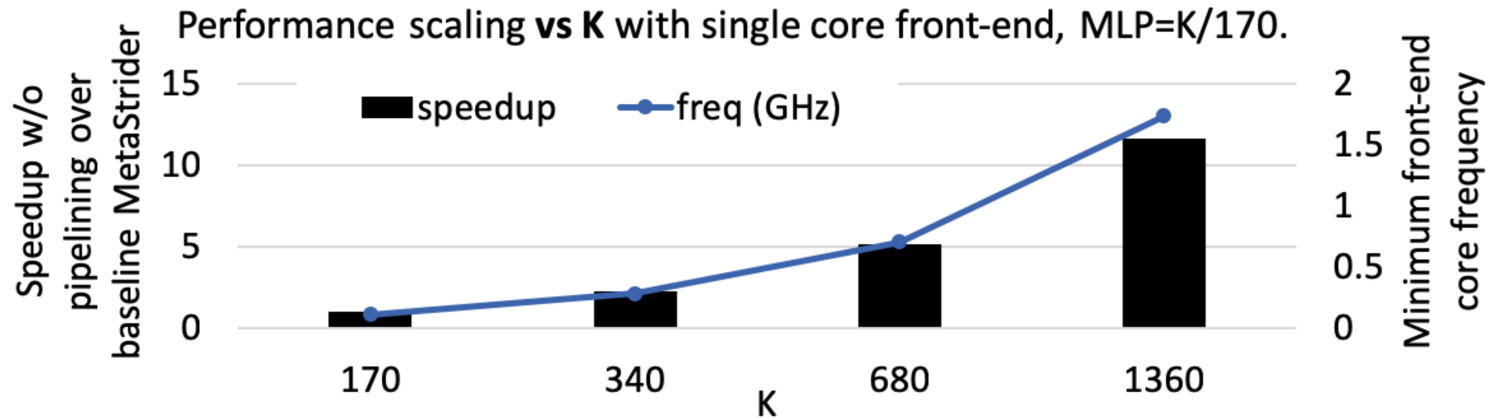
Georgia Tech comparch

Partition vs Fanout



Scalability Analysis





Scalability Analysis