Nested Parallelism PageRank on RISC-V Vector Multi-Processors

Alon Amid, Albert Ou, Krste Asanović, Borivoje Nikolić

Berkeley Architecture Research

ADEPT

Agenda

Problem Domain (Graphs/PageRank + Nested Parallelism)

+

Silicon-Proven Open Source Hardware and Software Implementations (Rocket + Hwacha + GraphMat + OpenMP)

+

FPGA-Accelerated Simulation (FireSim)

↓

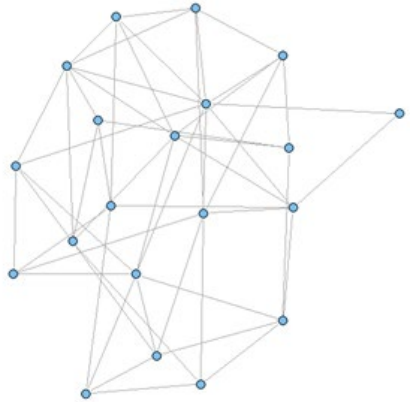
SW/HW Design Space Exploration

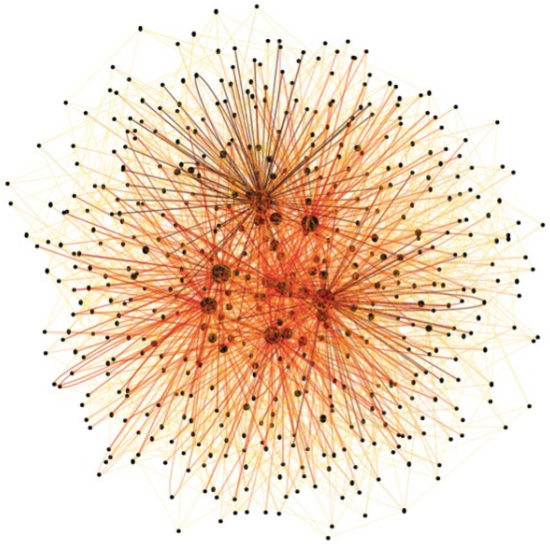
↓

Full-System Implications

Graphs

- Graph are everywhere
  - Implicit data-parallelism
  - Irregular data layout
- Usefulness of fixed-function acceleration of graph kernels is debatable
- Use general purpose data-parallel acceleration for graph workloads
  - Maximize the efficiency of data-parallel processors

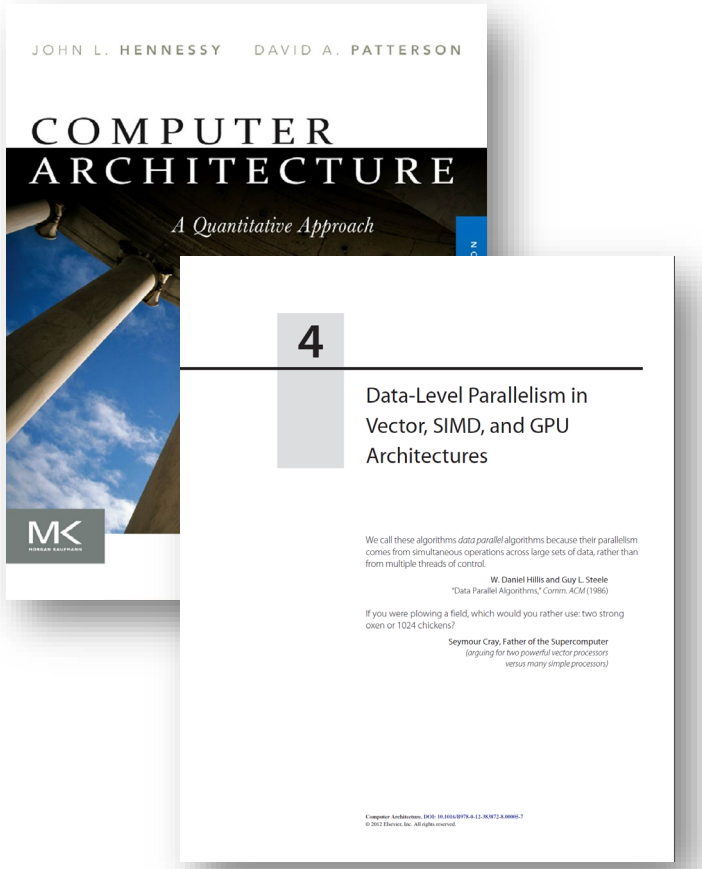




Images: http://netplexity.org/?p=809, http://horicky.blogspot.com/2012/04/basic-graph-analytics-using-igraph.html, http://mathworld.wolfram.com/GraphDiameter.html

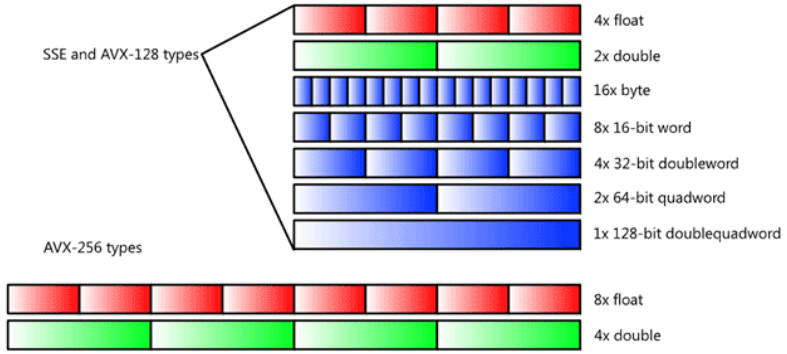
Common Data - Parallel Architectures

- Packed-SIMD
  - Register size exposed in the programming model
  - Direct bit-manipulation
  - ISA implications every technology generation change
- GPUs
  - SIMT programming model
  - Throughput-processors, scratchpad memories
- Vector Architectures
  - Vector-length agnostic programming model
  - Additional flexibility in μarch optimization



Graphs in Data - Parallel Architectures

- Intel AVX
  - Small parallelism factor
  - AVX register utilizations size alignments
    - Alternative sparse-matrix representations to fit AVX registers (Grazelle [1])



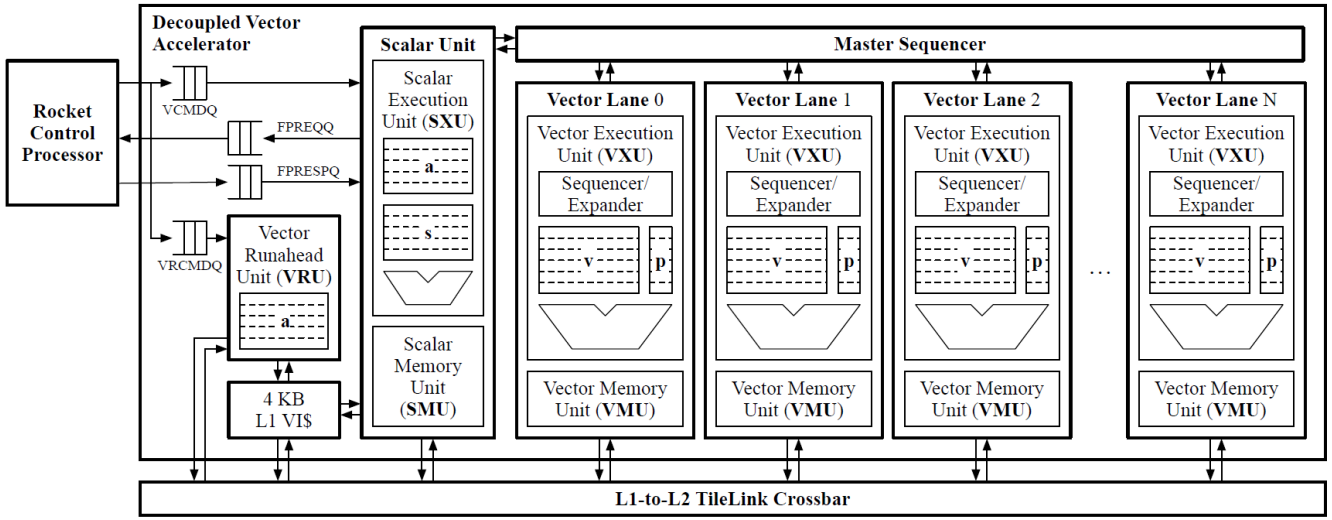
- GPUs [2][3]
  - Amortize data-movement between host memory and GPU memory
  - Load balancing between warps and threads

[1] Making Pull-Based Graph Processing Performant, Samuel Grossman, Heiner Litz and Christos Kozyrakis
[2] Scalable SIMD-Efficient Graph Processing on GPUs, Farzad Khorasani, Rajiv Gupta, Laxmi N. Bhuyan
[3] Multiple works by John Owens (UC Davis)

Photo credits:
https://software.intel.com/en-us/articles/introduction-to-intel-advanced-vector-extensions
https://www.tomshardware.co.uk/why-gpu-pricing-will-drop-further,news-58816.html

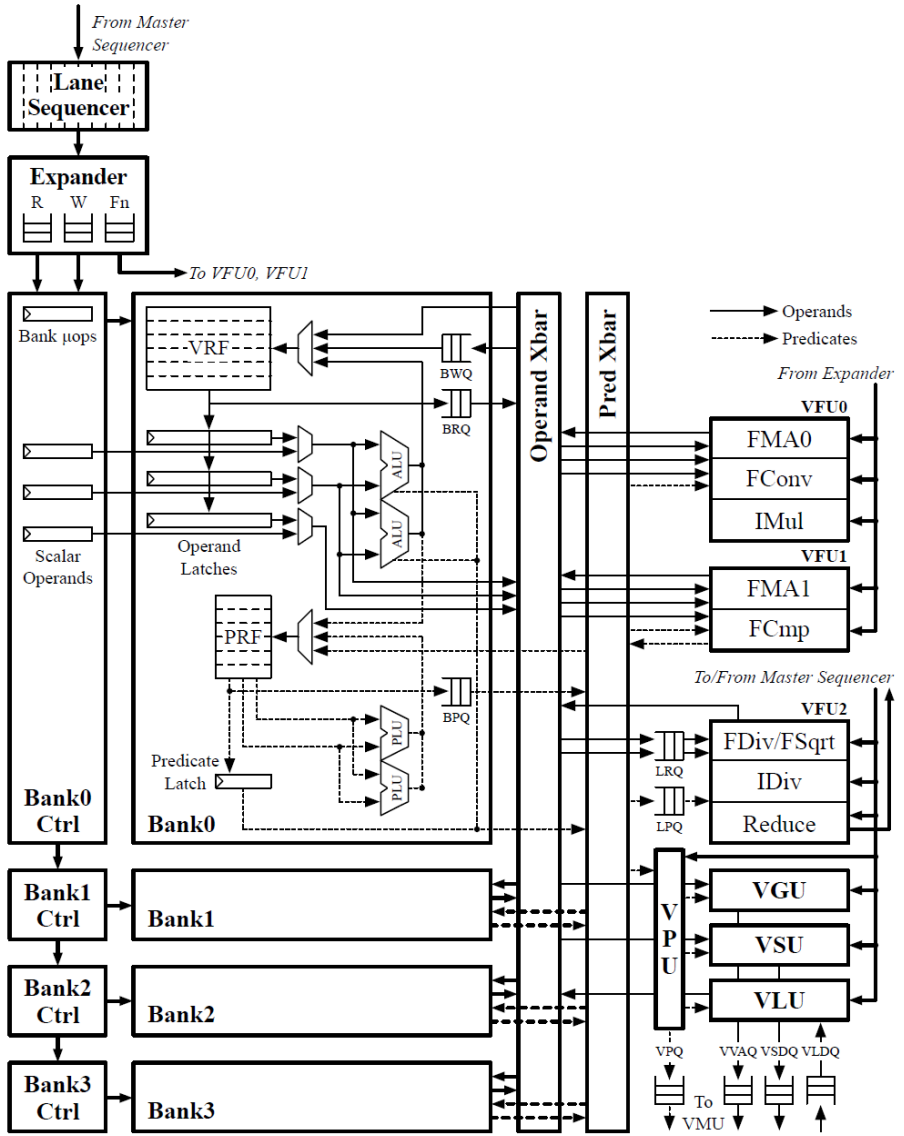
Hwacha Vector Architecture

- Non-standard RISC-V ISA extension
- Vector-length agnostic programming model
- **Silicon-proven, open-source** vector accelerator
  - Open-sourced at the 1st RISC-V Summit
- Integrated with Rocket chip generator
- TileLink cache-coherent memory system
- Parameterizable multi-lane design



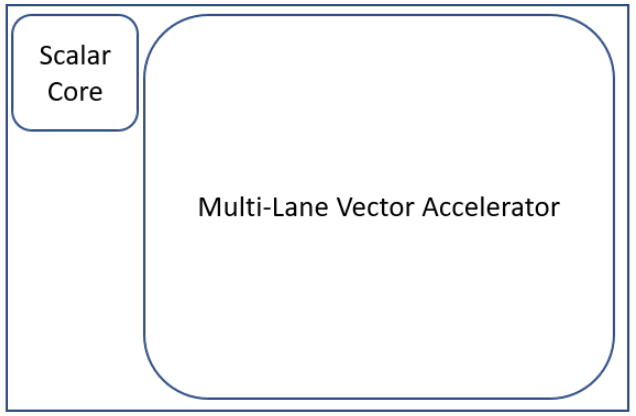
Hwacha Vector Architecture

- Decoupled access-execute
- 4 ops/cycle per lane average throughput
- 128 bits/cycle backing memory bandwidth
- 16 KiB SRAM banked register file per lane
  - Max vector length of 2048 double-width elements
  - Systolic-bank execution
  - 4x128 bits register file bandwidth

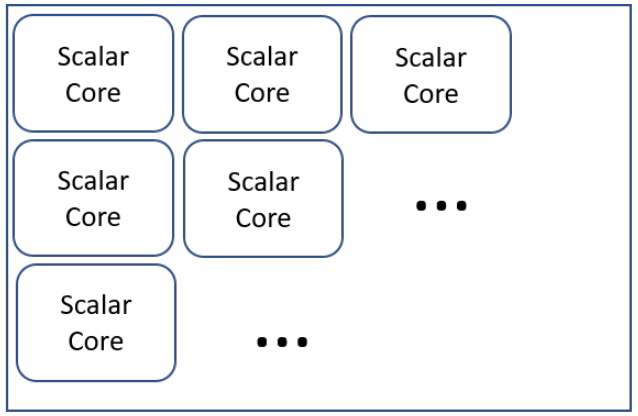


Nested Parallelism

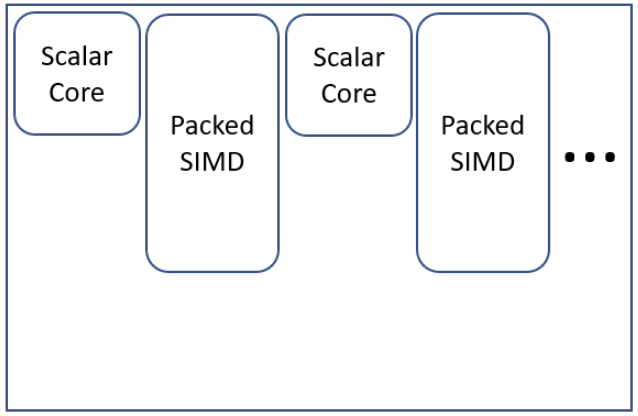
- Data-parallel accelerators + multi-processors
- Mixing parallelism properties
  - Task level parallelism – flexible, but expensive
  - Data level parallelism - efficient, but rigid
- Many design points, both SW and HW
- How to partition?



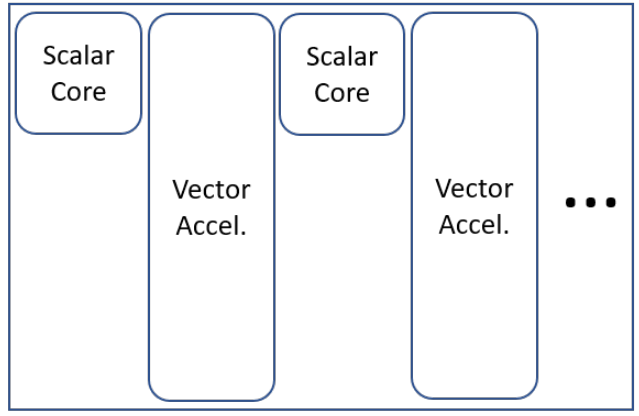
A: Scalar processor with on-chip vector accelerator



B: Chip Multi-Processor (CMP)



C: Chip Multi-Processor with packed-SIMD units



D: Chip Multi-Processor with vector accelerators

Graph and Sparse - Matrix Representations

- Graphs commonly represented as:
  - Adjacency lists
  - Adjacency matrices
- Adjacency matrix is usually a sparse matrix
- Sparse matrices can be compressed
  - Eliminating the zero values
  - Reduce storage in memory
- Variety of sparse matrix representations

0	81	0	0	0	0	0	0
0	5	0	0	0	0	0	0
0	0	0	0	0	0	0	0
61	0	9	0	0	0	34	11
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	42
0	0	0	0	0	0	17	0
0	92	0	0	0	0	0	70

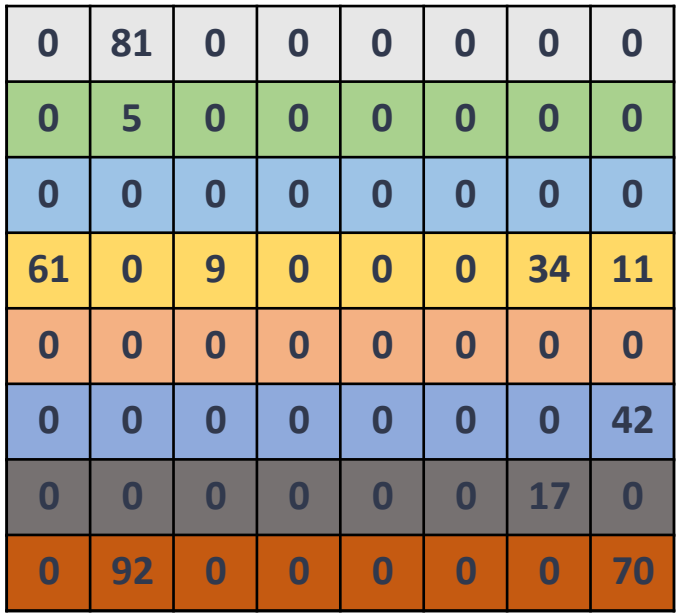
Graph and Sparse - Matrix Representations

0	81	0	0	0	0	0	0		
0	5	0	0	0	0	0	0		
0	0	0	0	0	0	0	0		
61	0	9	0	0	0	34	11		
0	0	0	0	0	0	0	0		
0	0	0	0	0	0	0	42		
0	0	0	0	0	0	17	0		
0	92	0	0	0	0	0	70		

row_indices	0	1	3	3	3	3	5	6	7	7
column_indices	1	1	0	2	6	7	7	6	1	7
values	81	5	61	9	34	11	42	17	92	70

COO

Graph and Sparse - Matrix Representations



0	81	0	0	0	0	0	0
0	5	0	0	0	0	0	0
0	0	0	0	0	0	0	0
61	0	9	0	0	0	34	11
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	42
0	0	0	0	0	0	17	0
0	92	0	0	0	0	0	70

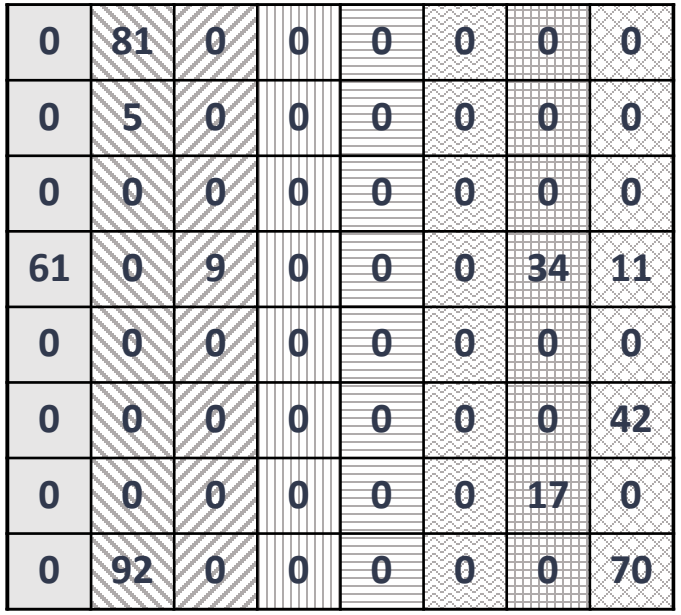
COO

row_indices	0	1	3	3	3	3	5	6	7	7
column_indices	1	1	0	2	6	7	7	6	1	7
values	81	5	61	9	34	11	42	17	92	70

CSR

row_pointers	0	1	2	2	6	6	7	8	10	
column_indices	1	1	0	2	6	7	7	6	1	7
values	81	5	61	9	34	11	42	17	92	70

Graph and Sparse - Matrix Representations



0	81	0	0	0	0	0	0
0	5	0	0	0	0	0	0
0	0	0	0	0	0	0	0
61	0	9	0	0	0	34	11
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	42
0	0	0	0	0	0	17	0
0	92	0	0	0	0	0	70

COO

row_indices	0	1	3	3	3	3	5	6	7	7
column_indices	1	1	0	2	6	7	7	6	1	7
values	81	5	61	9	34	11	42	17	92	70

CSR

row_pointers	0	1	2	2	6	6	7	8	10	
column_indices	1	1	0	2	6	7	7	6	1	7
values	81	5	61	9	34	11	42	17	92	70

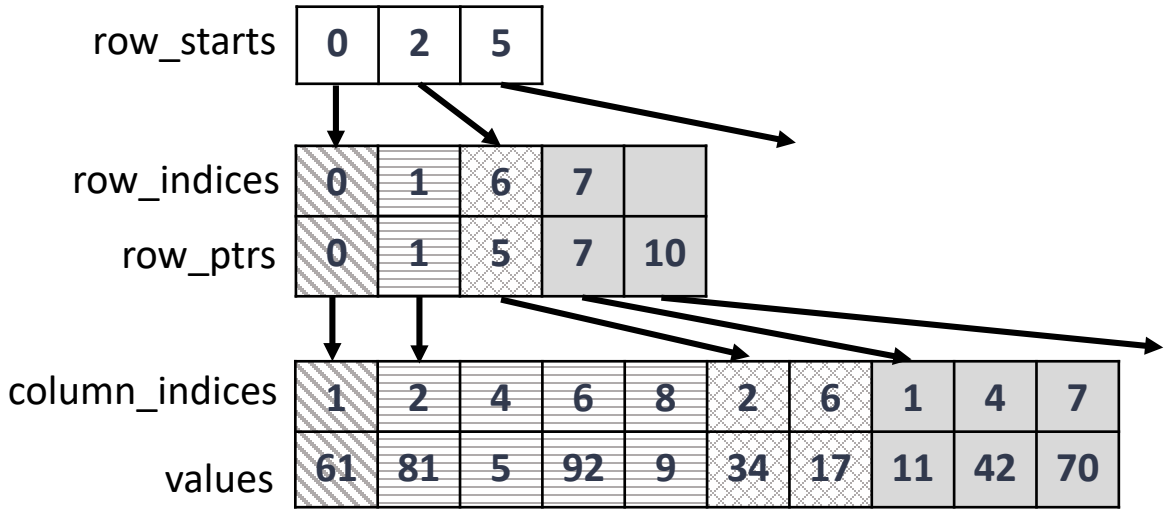
CSC

column_pointers	0	1	4	5	5	5	5	7	10	
row_indices	3	0	1	7	3	3	6	3	5	7
values	61	81	5	92	9	34	17	11	42	70

DCSR/DCSC Representation

- Compress across both dimensions
- Hyper-sparse matrices
  - Required to amortized the overhead of the additional indirection level
- Explicit nested parallelism

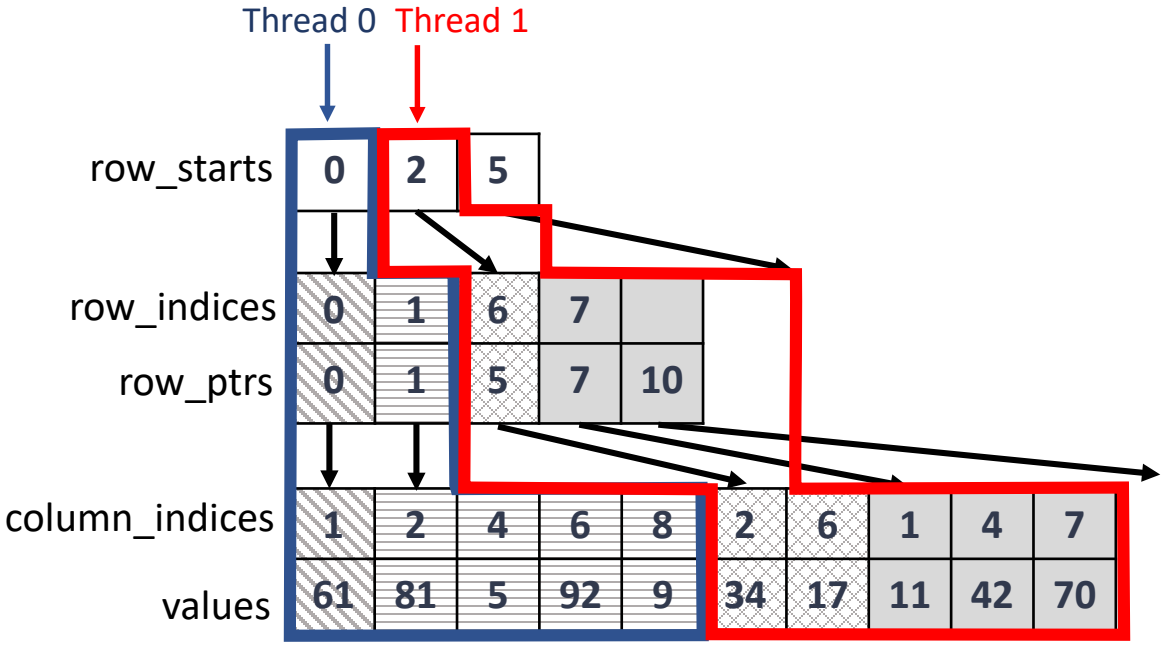
0	61	0	0	0	0	0	0
0	0	81	0	5	0	92	9
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	34	0	0	0	17	0
0	11	0	0	42	0	0	70



[1] Buluc, Aydin, and John R. Gilbert. "On the representation and multiplication of hypersparse matrices." 2008 IEEE International Symposium on Parallel and Distributed Processing. IEEE, 2008.

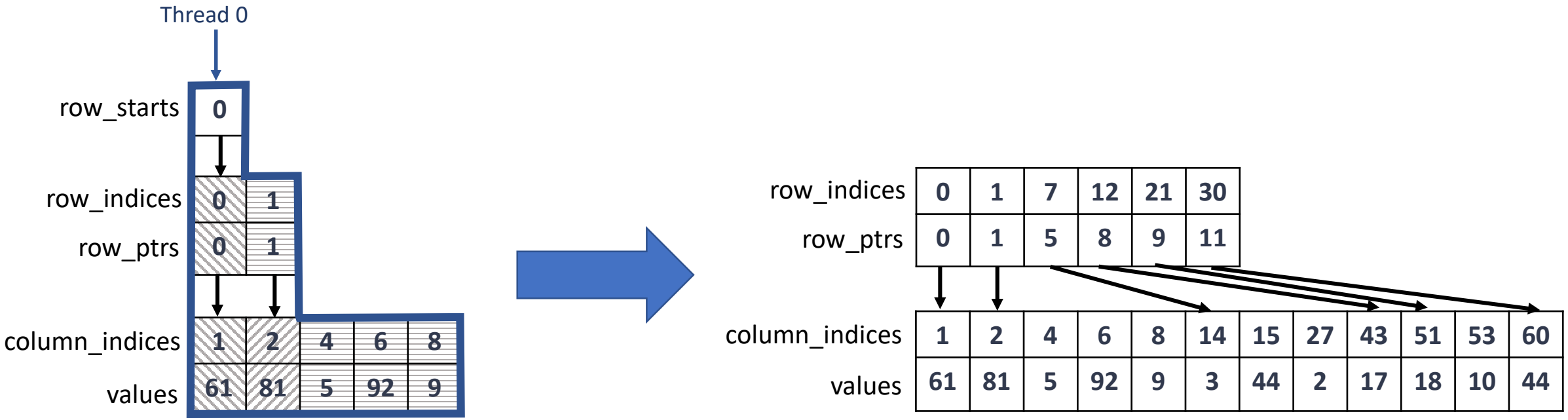
Nested Parallelism in DCSR/DCSC

- A DCSR representation is composed of multiple CSR representation
- 2 Explicit parallelism levels:
  - Level 1 – Task/Thread level parallelism across the external indirection array
  - Level 2 – Data-level parallelism within each sub-CSR representation



Inner CSR Processing

- Each thread processes a small unit of a CSR unit
- For demonstration purposes, let's make the sub-CSR larger

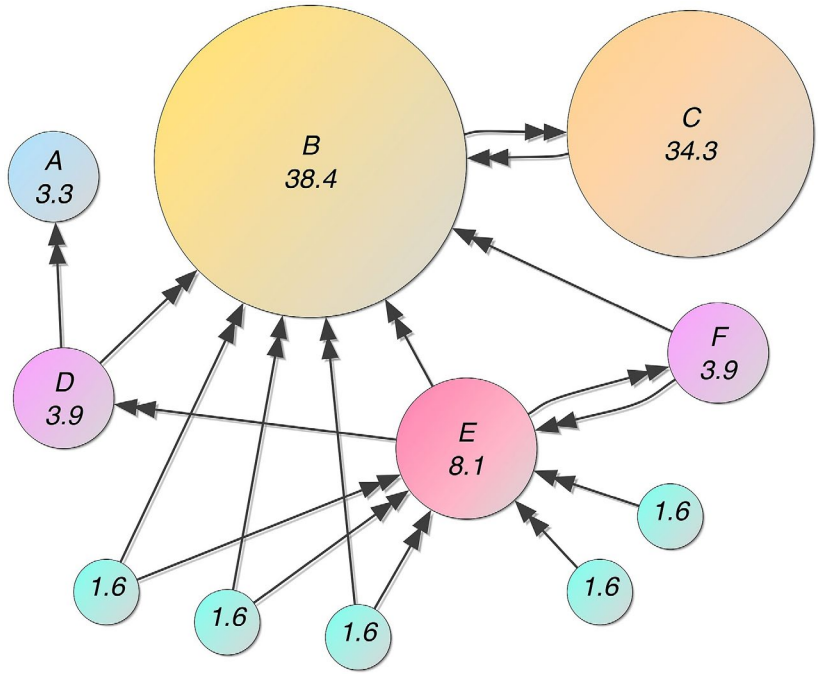


Sidenote : PageRank

- Measure of importance of nodes in a directed graph
- Represents a random walk
- Can be implemented as an iterative SpMV
- Common iterative graph processing benchmark

$$P = A * \frac{1}{N}$$

$$y^{(k)} = d * P * y^{(k-1)} + \frac{(1-d)}{|V|}$$

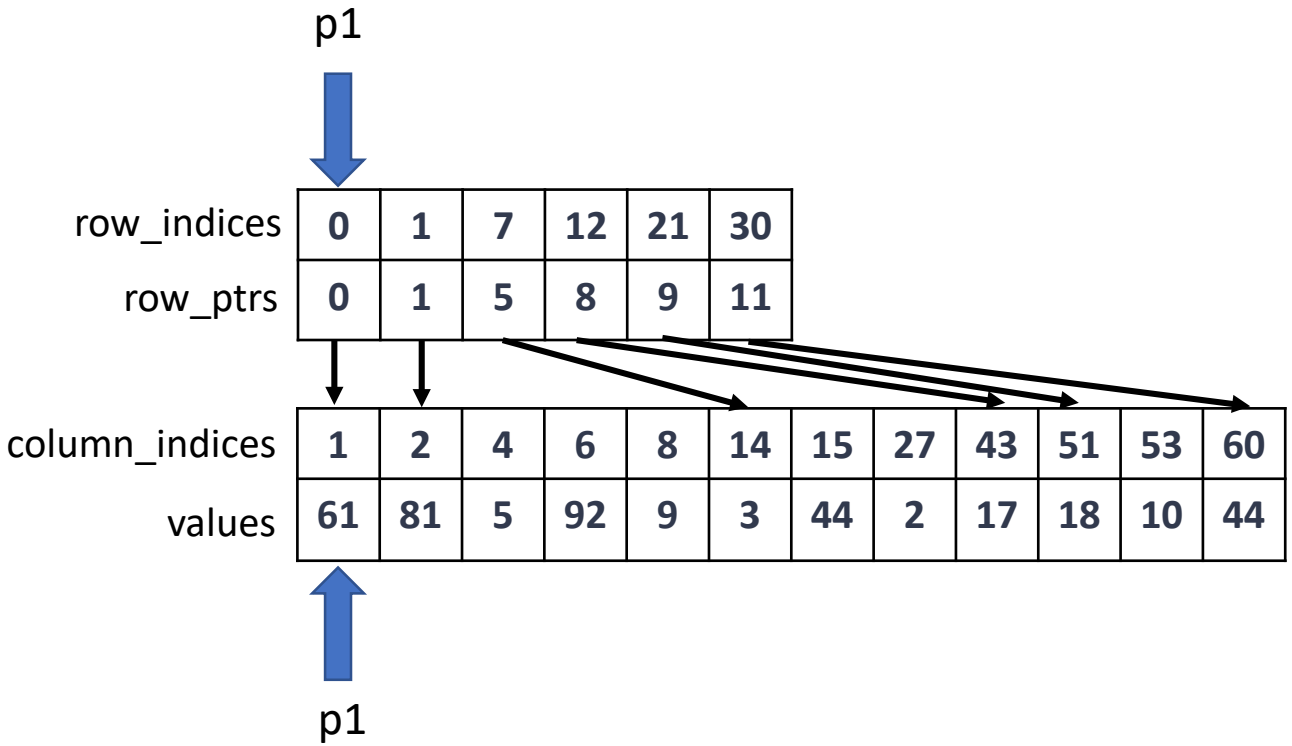


$$PR(u) = (1-d) + d \sum_{v \in B_u} \frac{PR(v)}{N_v}$$

Images: https://en.wikipedia.org/wiki/File:PageRanks-Example.jpg

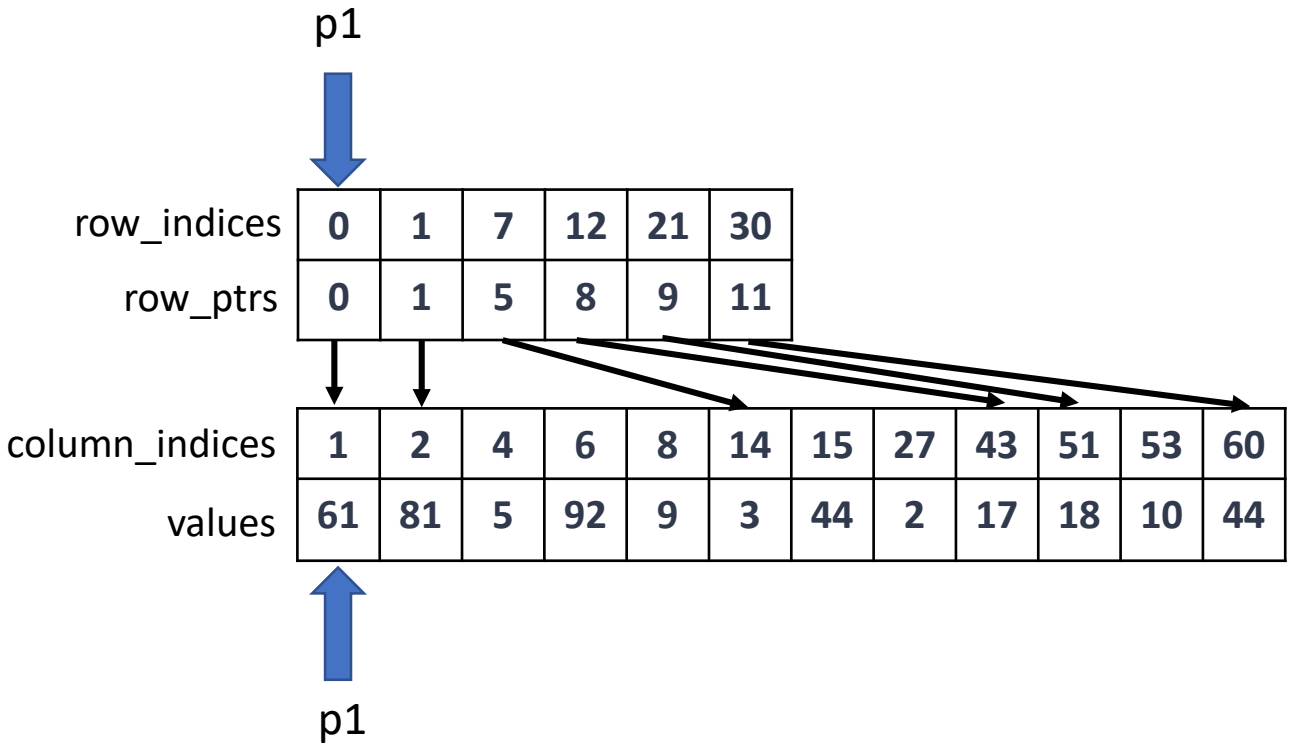
Simple Scalar Sparse Matrix Traversal

- Process the internal CSR in a simple scalar loop
- Traverse the pointers array
- Follow the pointer to the values array
- Perform the required operation (multiplication and accumulation for SpMV)



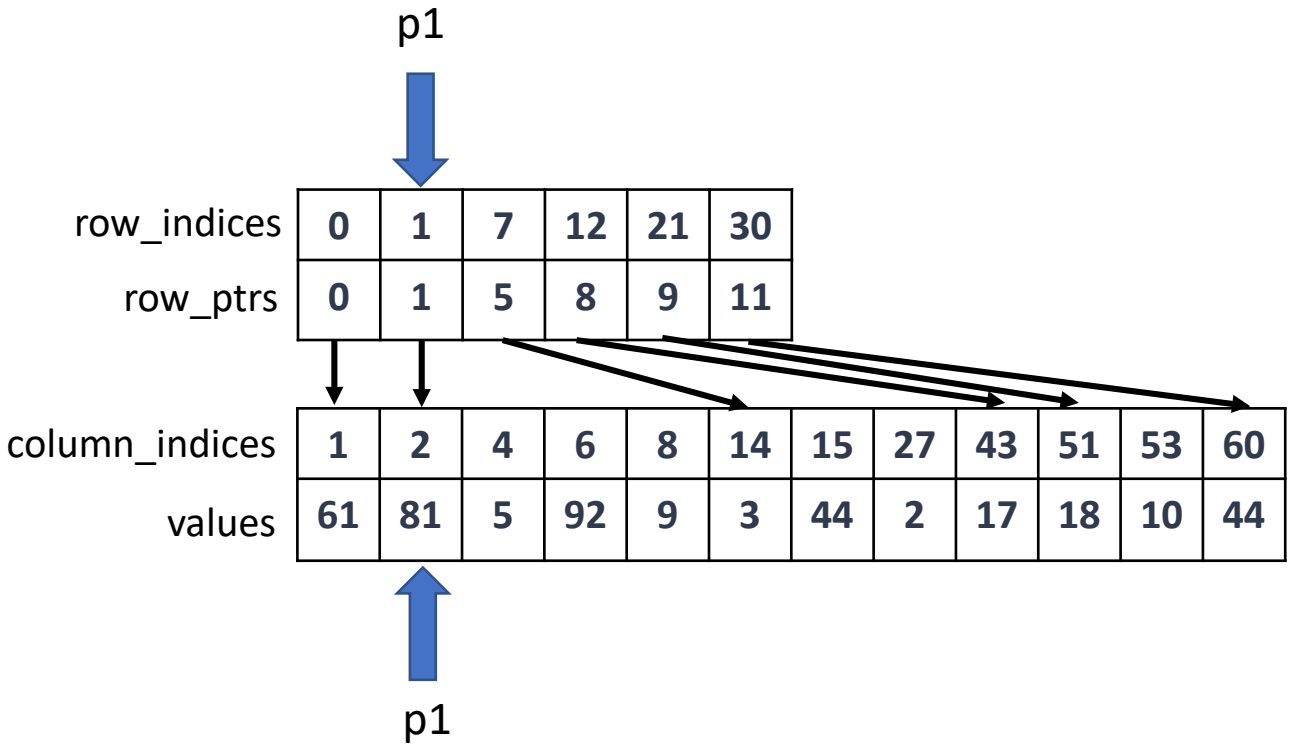
Simple Scalar Sparse Matrix Traversal

- Process the internal CSR in a simple scalar loop
- Traverse the pointers array
- Follow the pointer to the values array
- Perform the required operation (multiplication and accumulation for SpMV)



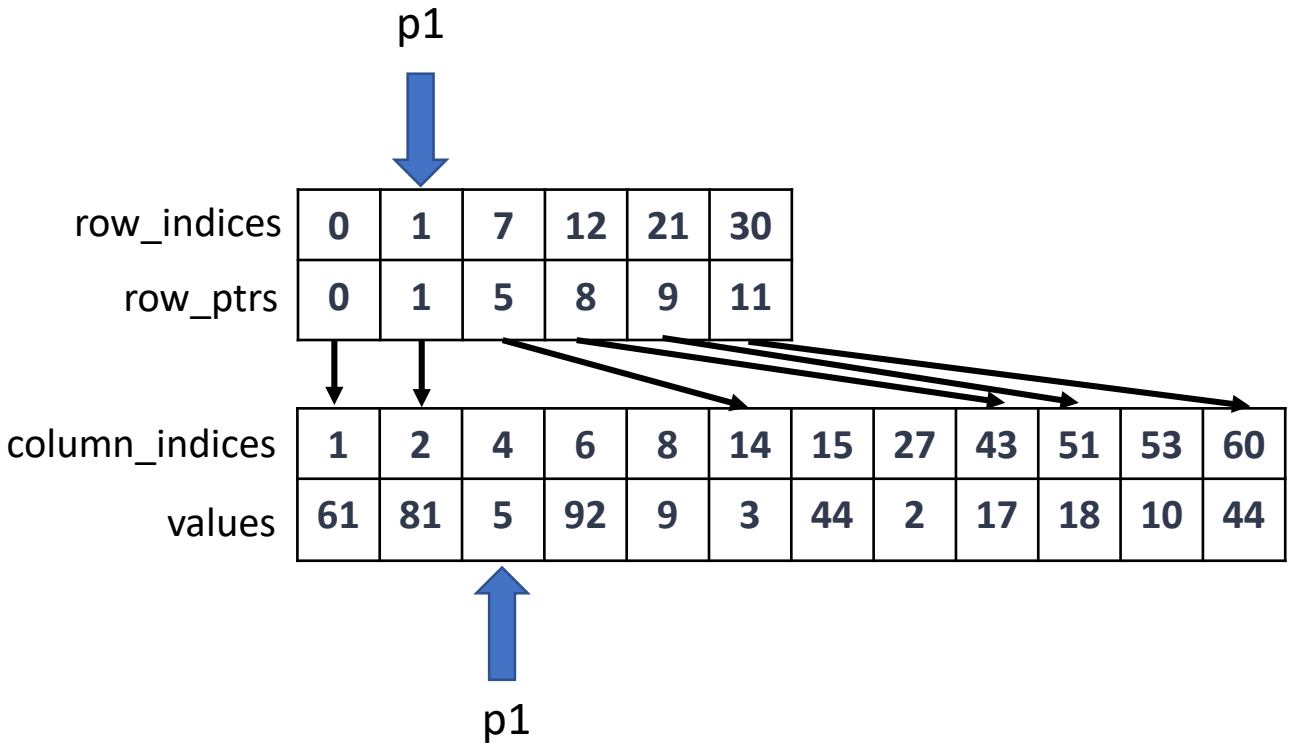
Simple Scalar Sparse Matrix Traversal

- Process the internal CSR in a simple scalar loop
- Traverse the pointers array
- Follow the pointer to the values array
- Perform the required operation (multiplication and accumulation for SpMV)



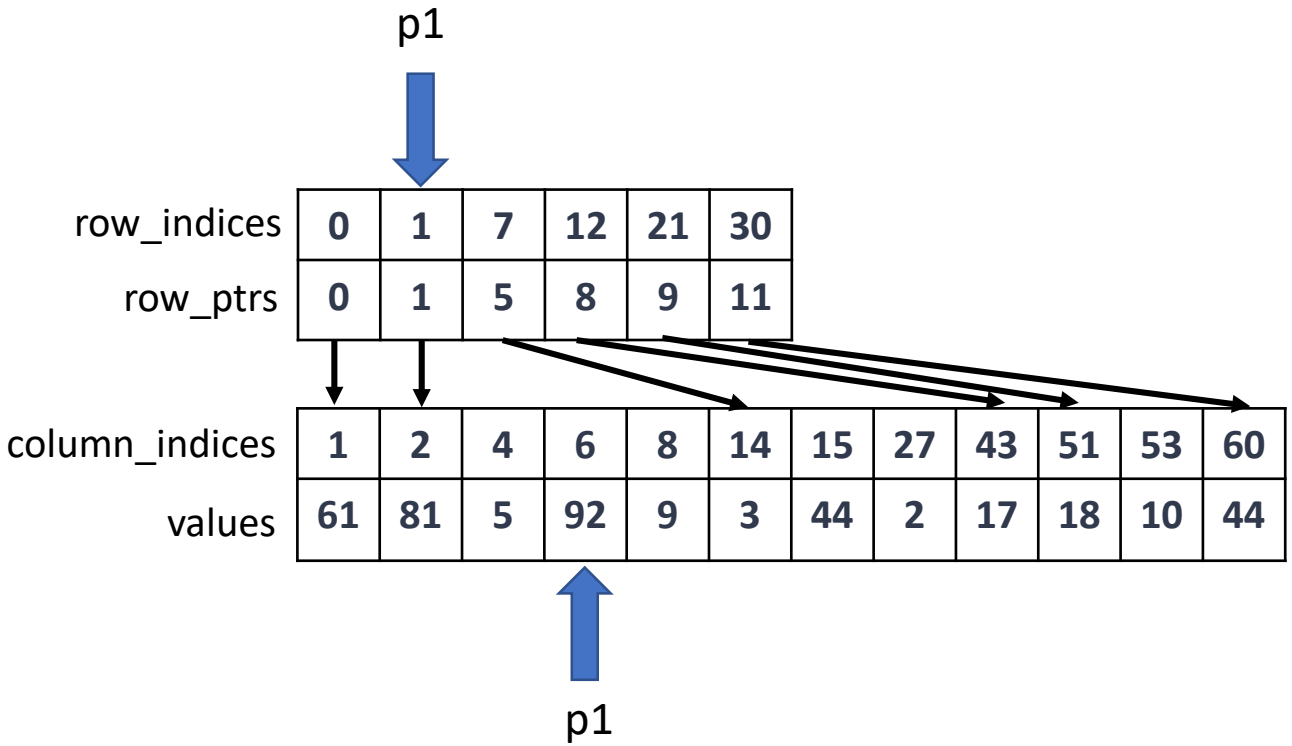
Simple Scalar Sparse Matrix Traversal

- Process the internal CSR in a simple scalar loop
- Traverse the pointers array
- Follow the pointer to the values array
- Perform the required operation (multiplication and accumulation for SpMV)



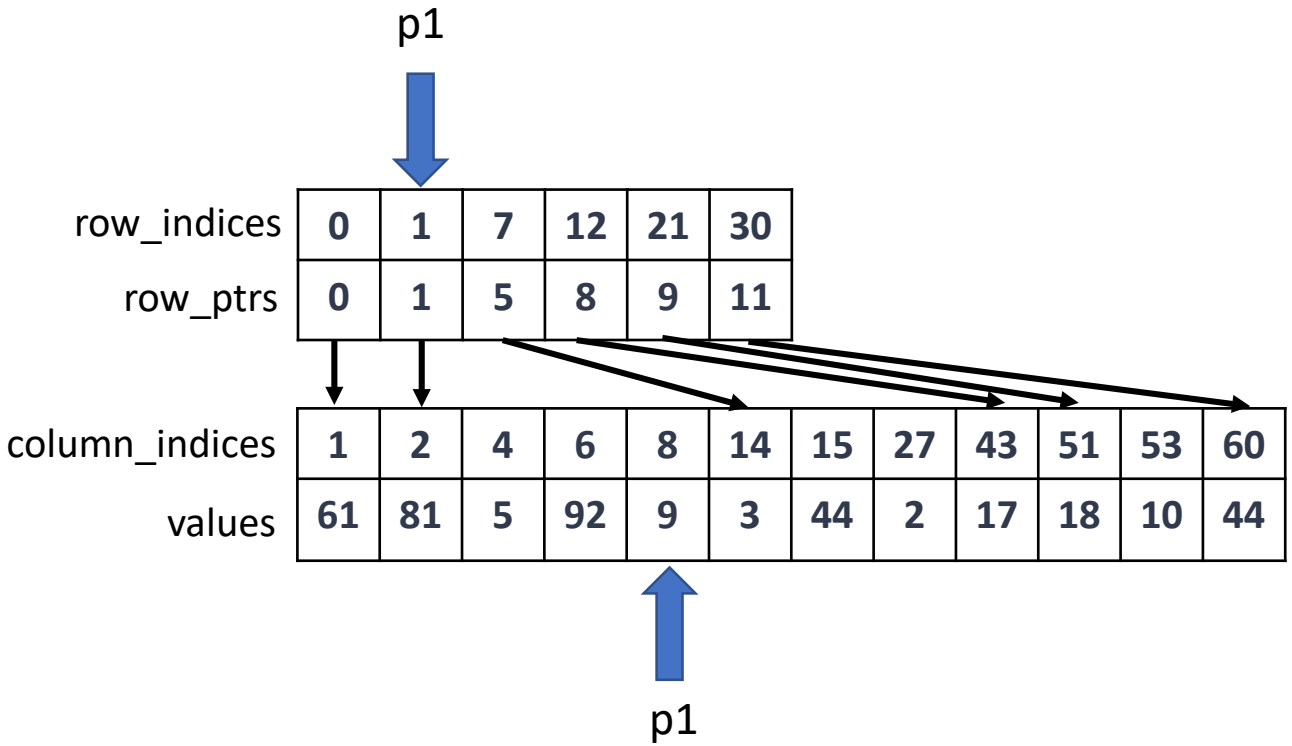
Simple Scalar Sparse Matrix Traversal

- Process the internal CSR in a simple scalar loop
- Traverse the pointers array
- Follow the pointer to the values array
- Perform the required operation (multiplication and accumulation for SpMV)



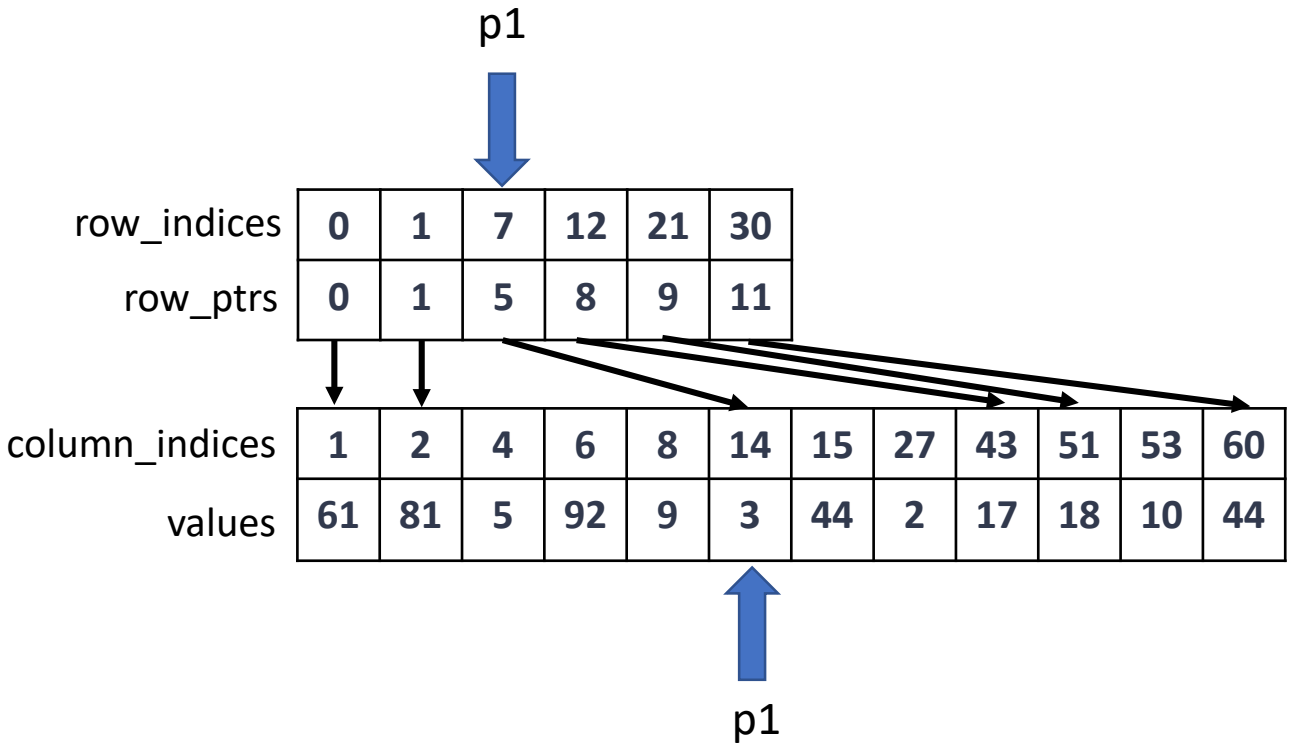
Simple Scalar Sparse Matrix Traversal

- Process the internal CSR in a simple scalar loop
- Traverse the pointers array
- Follow the pointer to the values array
- Perform the required operation (multiplication and accumulation for SpMV)



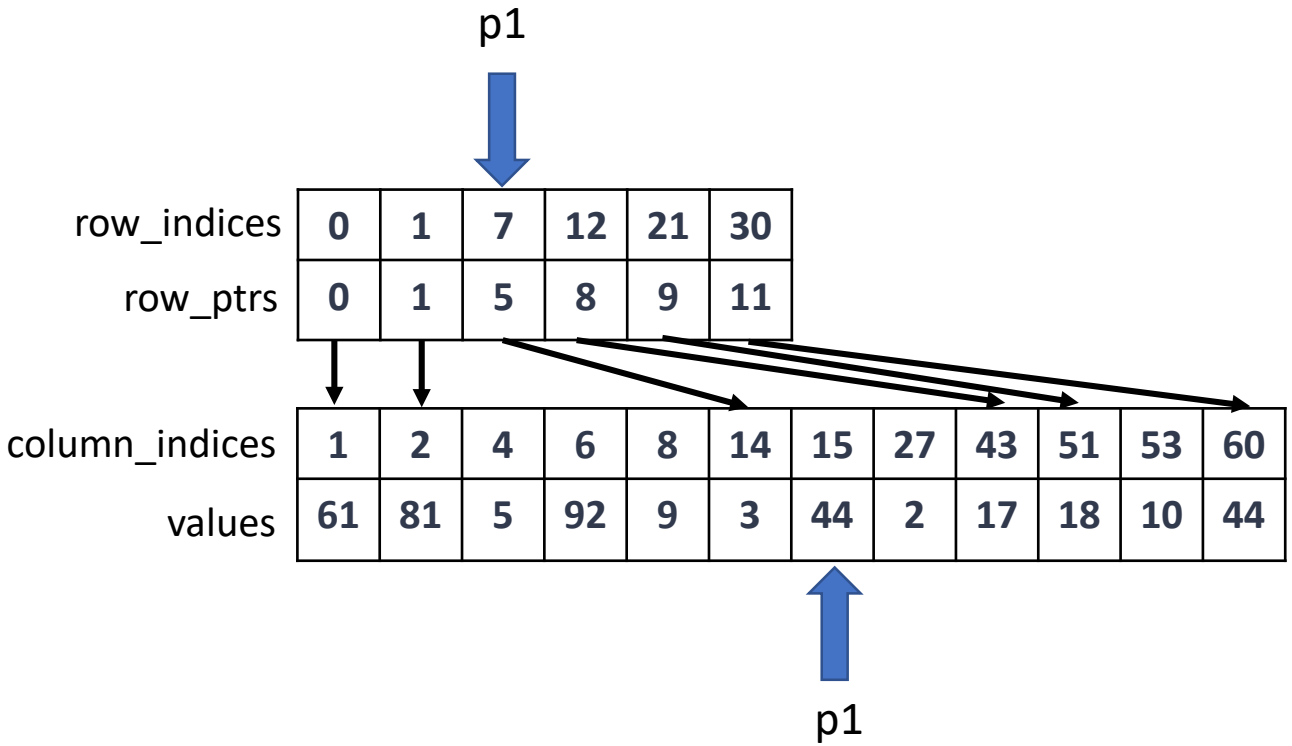
Simple Scalar Sparse Matrix Traversal

- Process the internal CSR in a simple scalar loop
- Traverse the pointers array
- Follow the pointer to the values array
- Perform the required operation (multiplication and accumulation for SpMV)



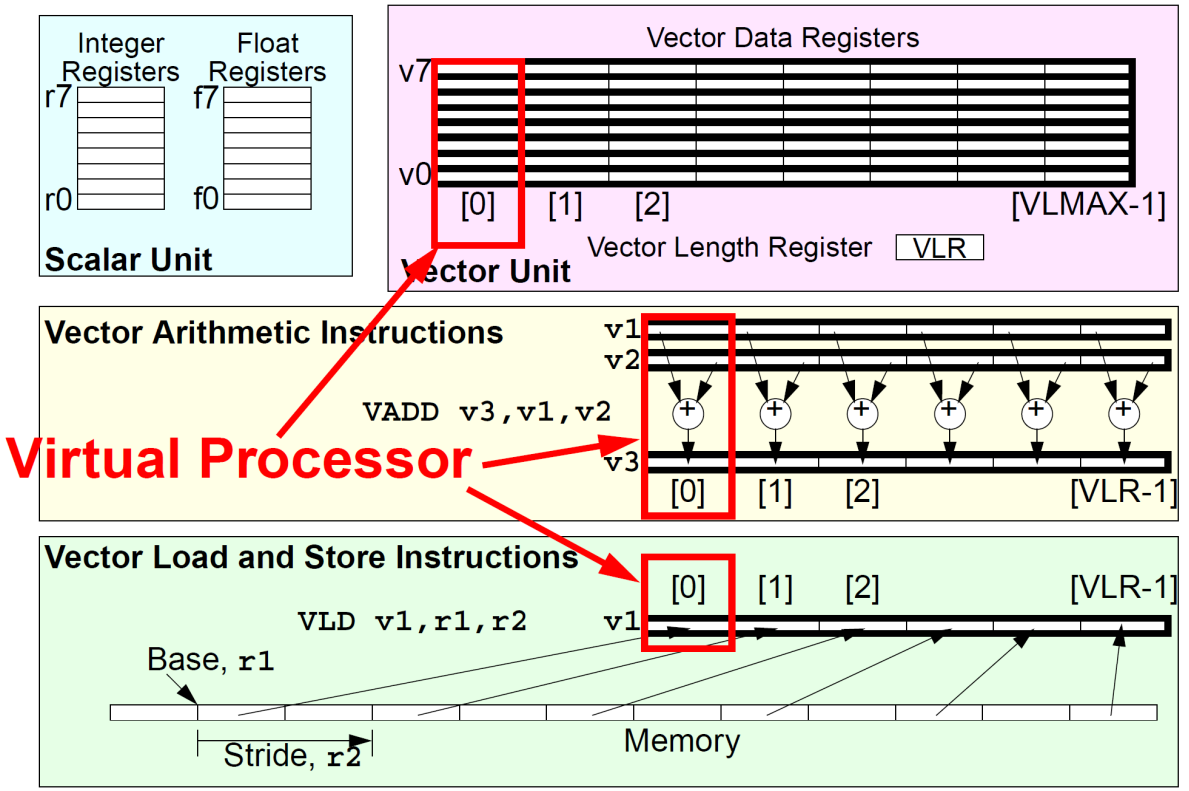
Simple Scalar Sparse Matrix Traversal

- Process the internal CSR in a simple scalar loop
- Traverse the pointers array
- Follow the pointer to the values array
- Perform the required operation (multiplication and accumulation for SpMV)



Virtual Processors View

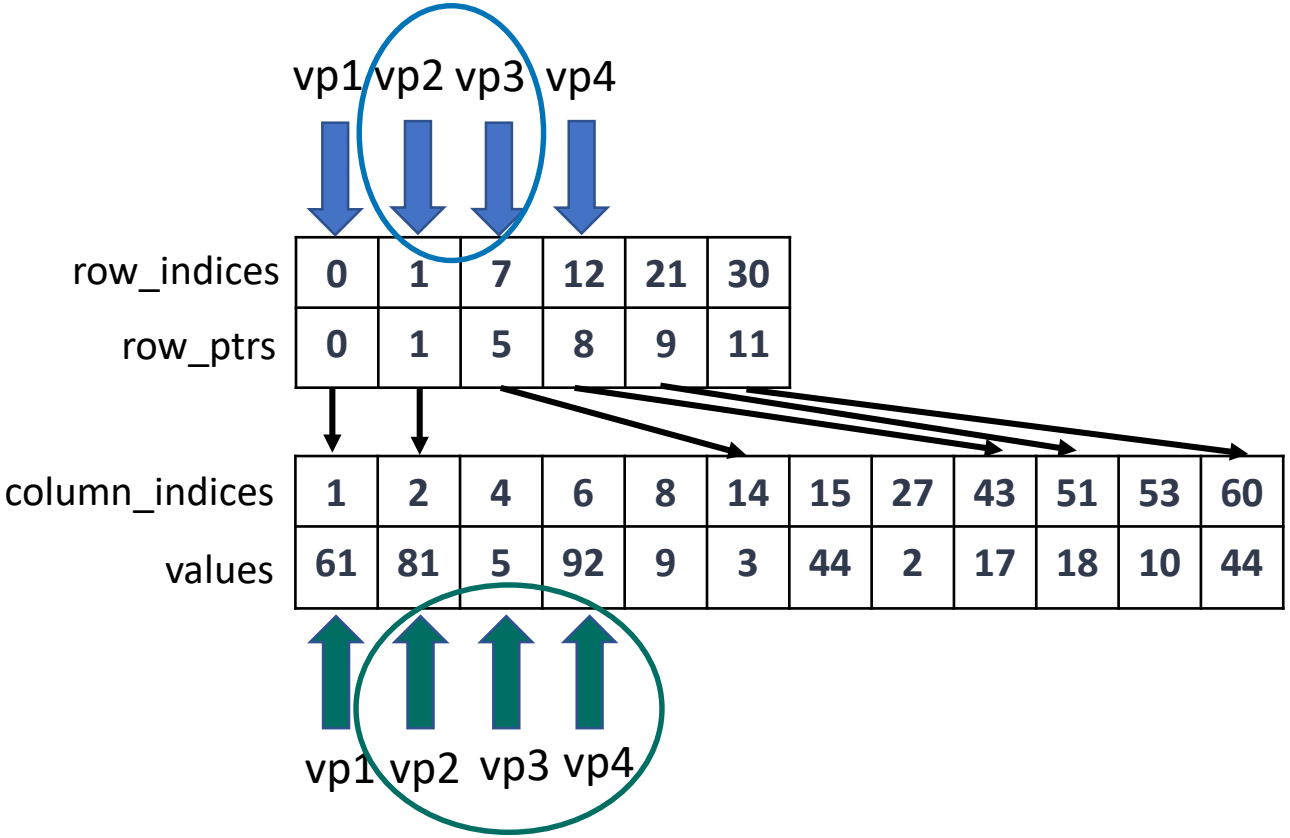
- View of data parallel accelerators as lock-step execution engines
  - No need to dive into μarch
- Number of virtual processors proportional to vector length
- Example: vector lengths of 4 => 4 virtual processors
  - Not necessarily implemented as 4 functional units.



Virtual Processors View, Figure 2.3, from Vector Microprocessors, PhD dissertation by Krste Asanovic

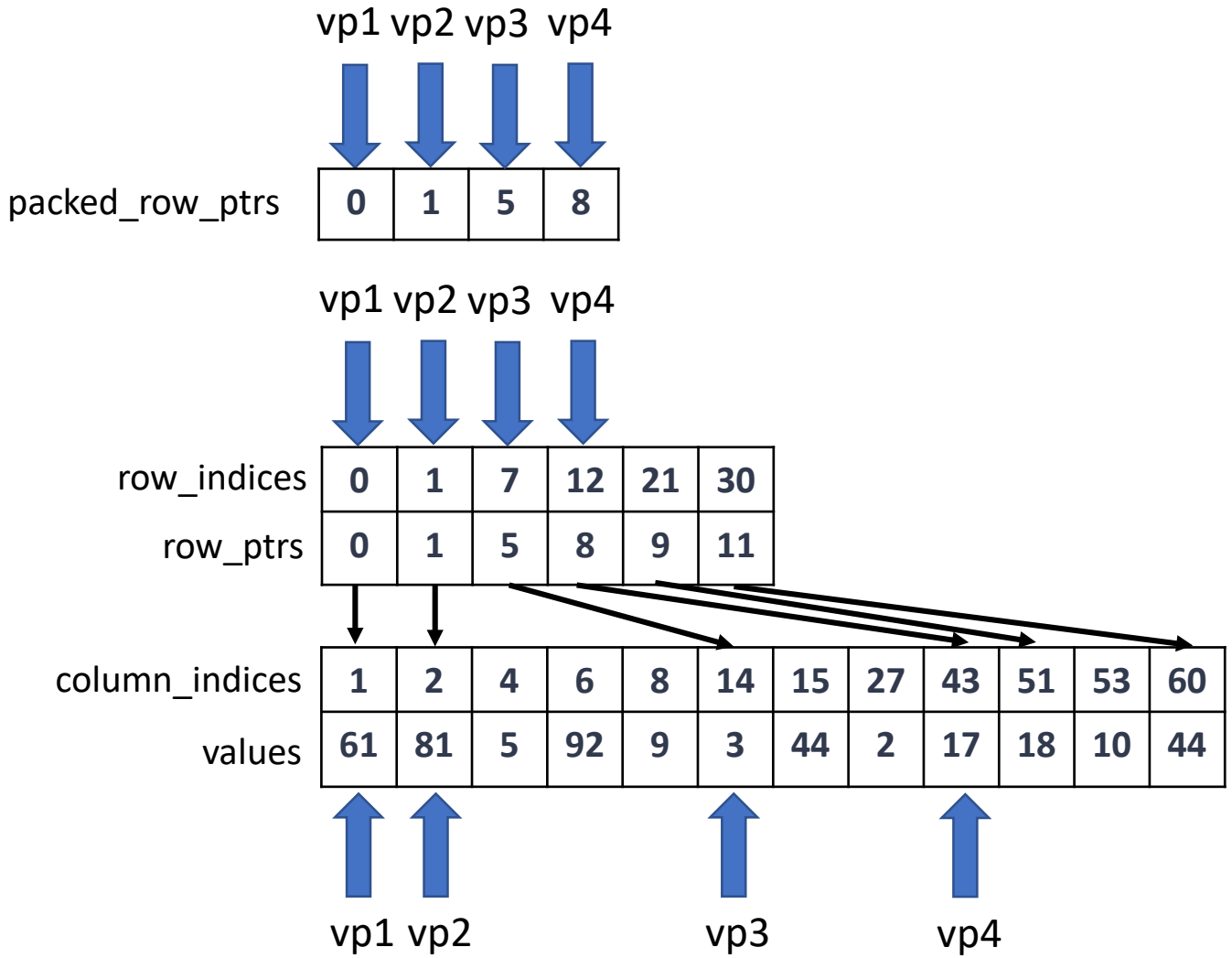
Stripmining

- Stripmining - the most common technique for loop vectorization
- Operate over strips of data based on the vector-length
- Why does simple stripmining not work for CSR/CSC SpMV?
  - Pointer arrays: load imbalance – different pointers point to rows of different lengths
  - Values array: serialization on AMOs – need to accumulate all the values of the strip



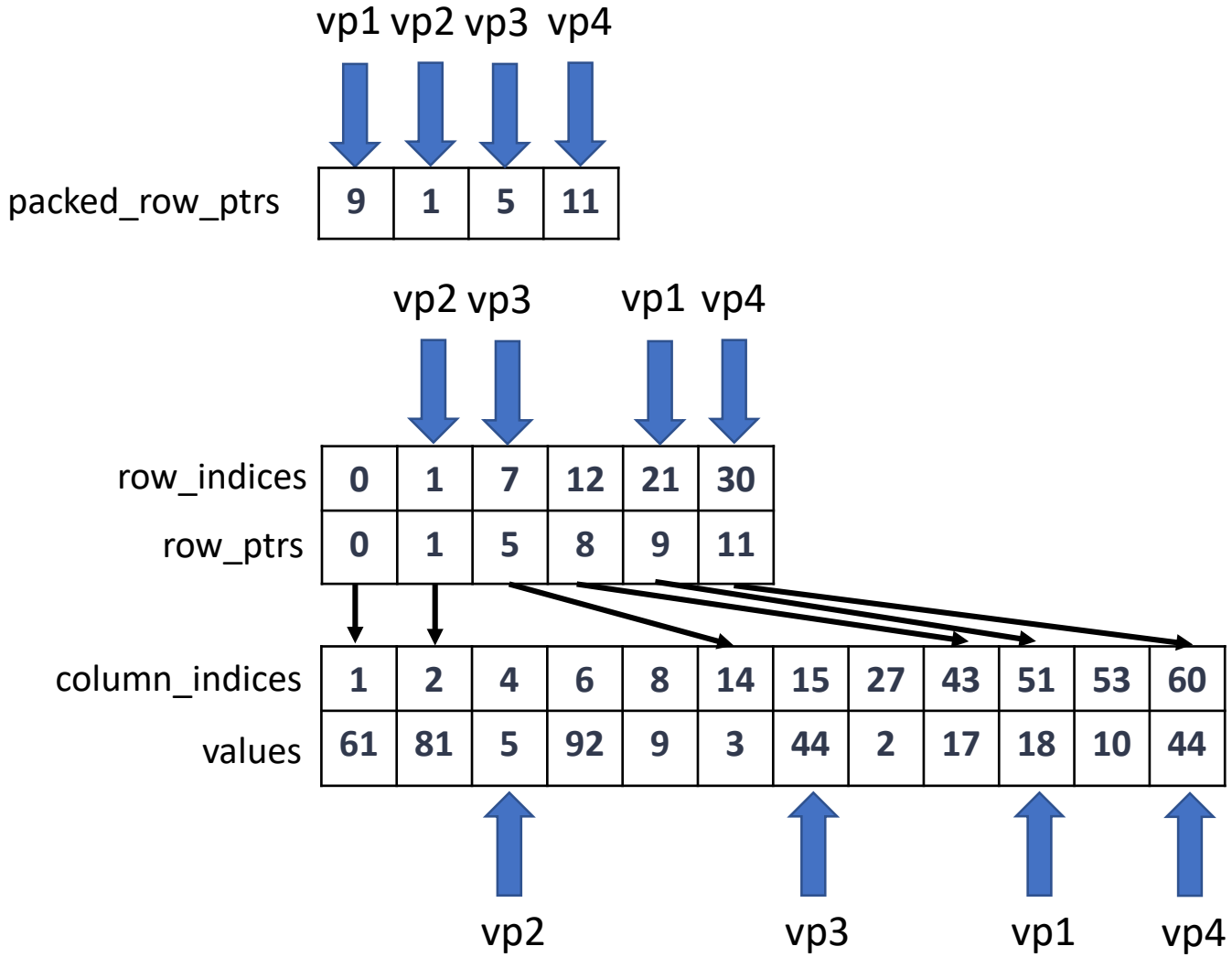
Packed Stripmining

- Parallel processing of the pointer array (node-centric)
- Problem: Simple stripmining has low utilization of virtual processors due to load-balancing and non-uniform vertex degree distribution
- Solution: Pack the row pointers (vertices) to maintain high utilization of virtual processors
  - Scalar re-packing after every stripmining iteration



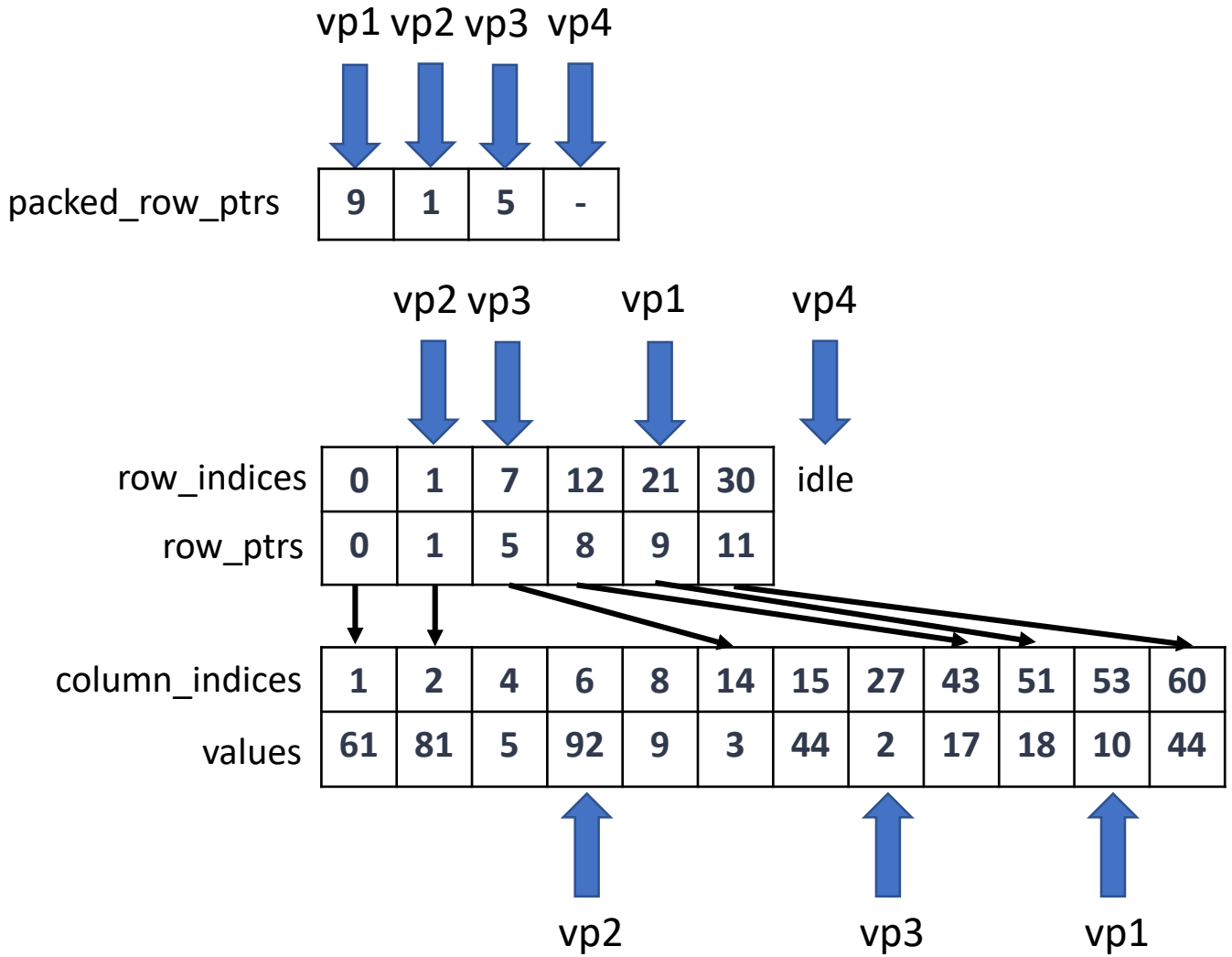
Packed Stripmining

- Parallel processing of the pointer array (node-centric)
- Problem: Simple stripmining has low utilization of virtual processors due to load-balancing and non-uniform vertex degree distribution
- Solution: Pack the row pointers (vertices) to maintain high utilization of virtual processors
  - Scalar re-packing after every stripmining iteration



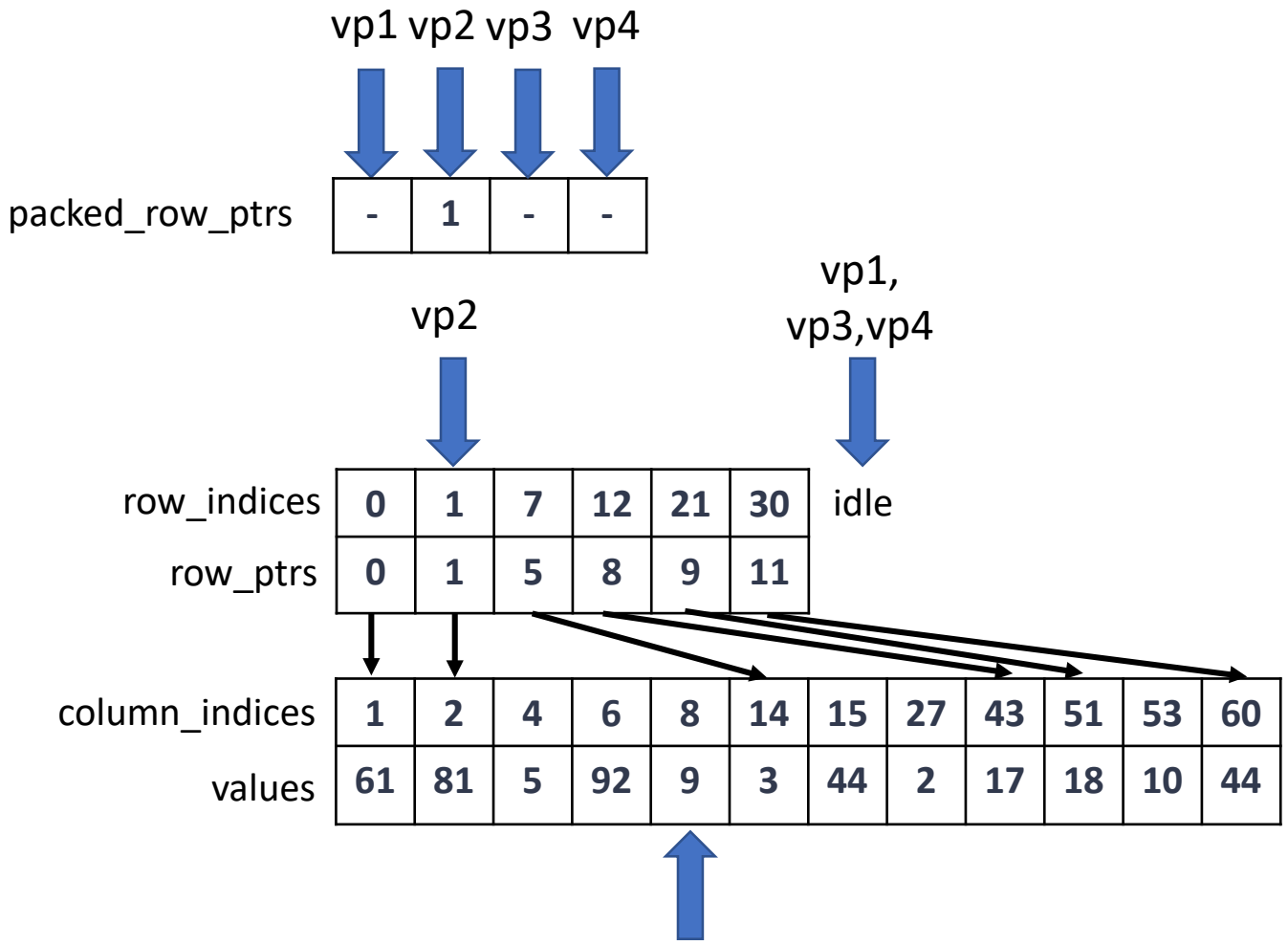
Packed Stripmining

- Parallel processing of the pointer array (node-centric)
- Problem: Simple stripmining has low utilization of virtual processors due to load-balancing and non-uniform vertex degree distribution
- Solution: Pack the row pointers (vertices) to maintain high utilization of virtual processors
  - Scalar re-packing after every stripmining iteration



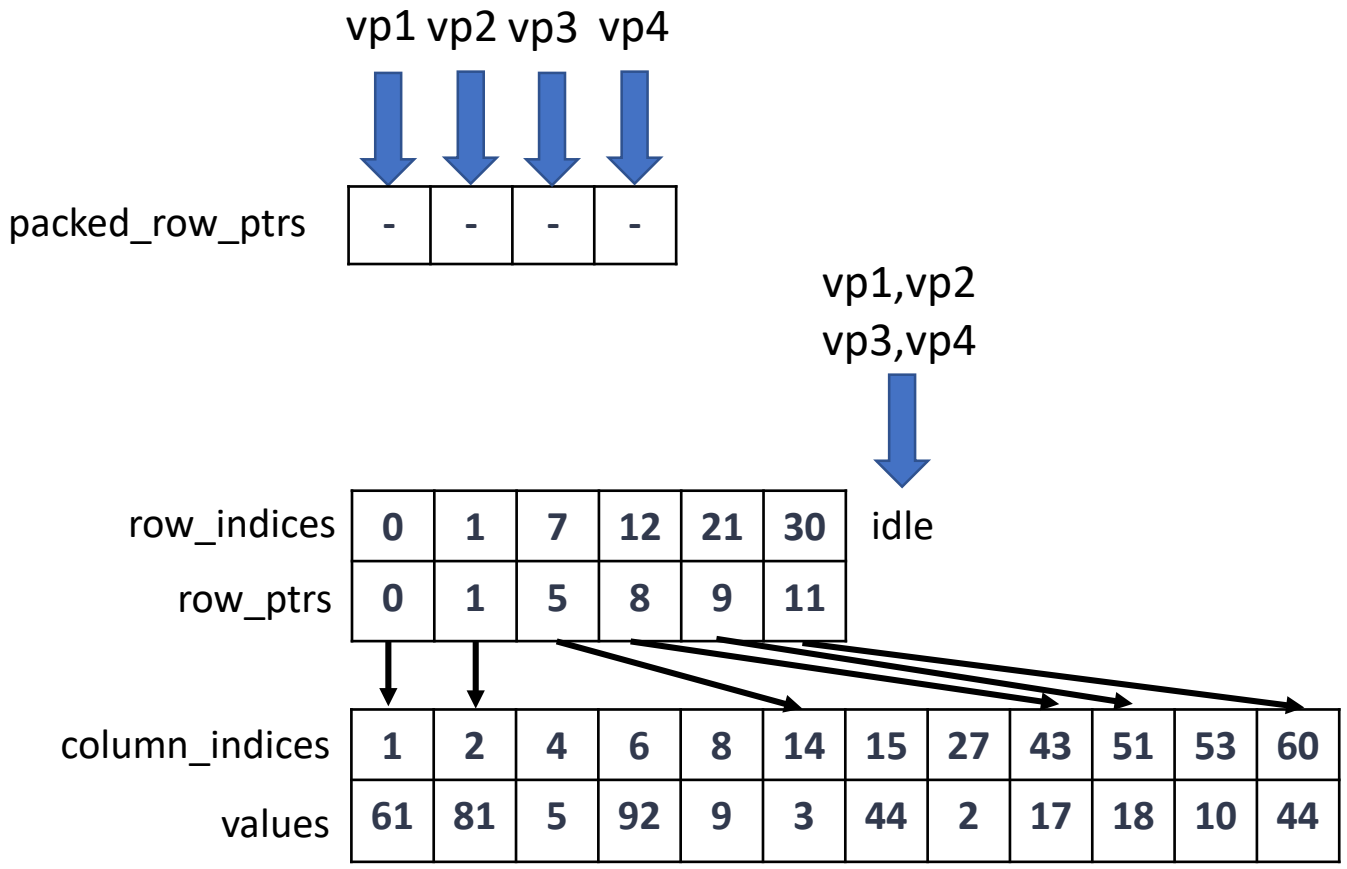
Packed Stripmining

- Parallel processing of the pointer array (node-centric)
- Problem: Simple stripmining has low utilization of virtual processors due to load-balancing and non-uniform vertex degree distribution
- Solution: Pack the row pointers (vertices) to maintain high utilization of virtual processors
  - Scalar re-packing after every stripmining iteration



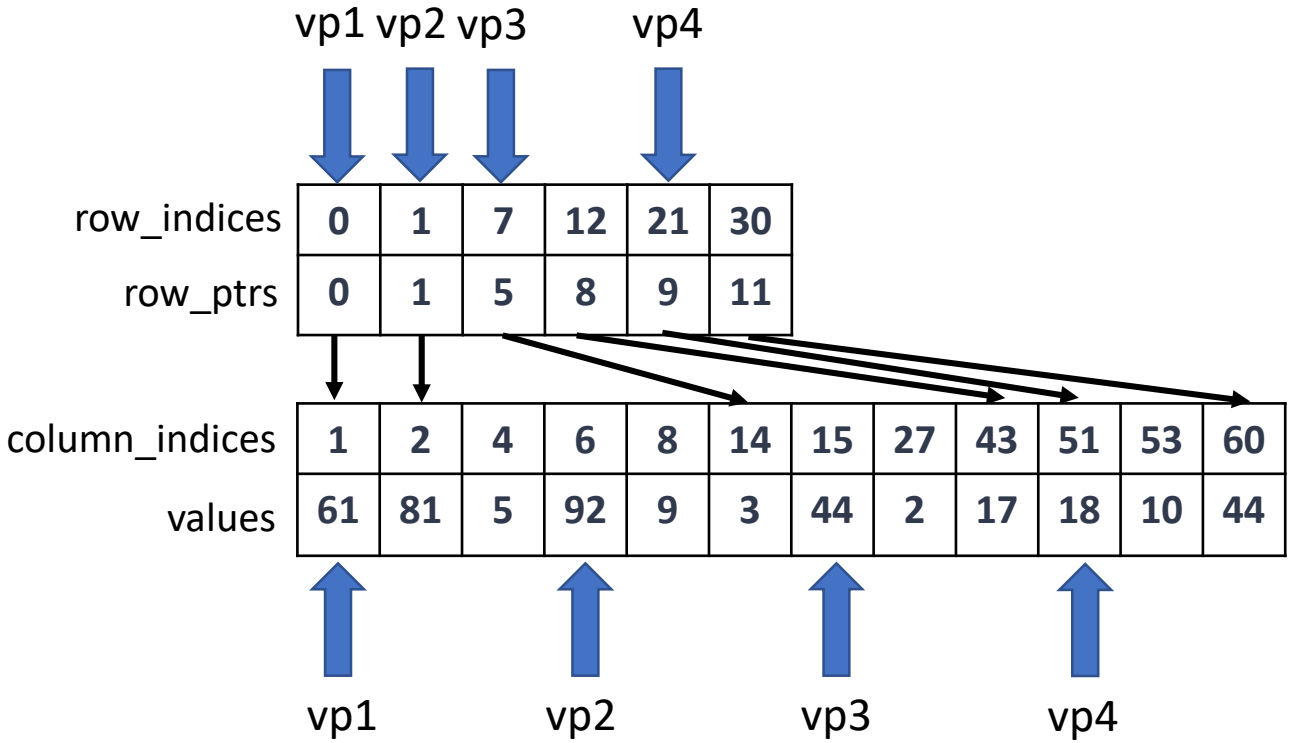
Packed Stripmining

- Parallel processing of the pointer array (node-centric)
- Problem: Simple stripmining has low utilization of virtual processors due to load-balancing and non-uniform vertex degree distribution
- Solution: Pack the row pointers (vertices) to maintain high utilization of virtual processors
  - Scalar re-packing after every stripmining iteration



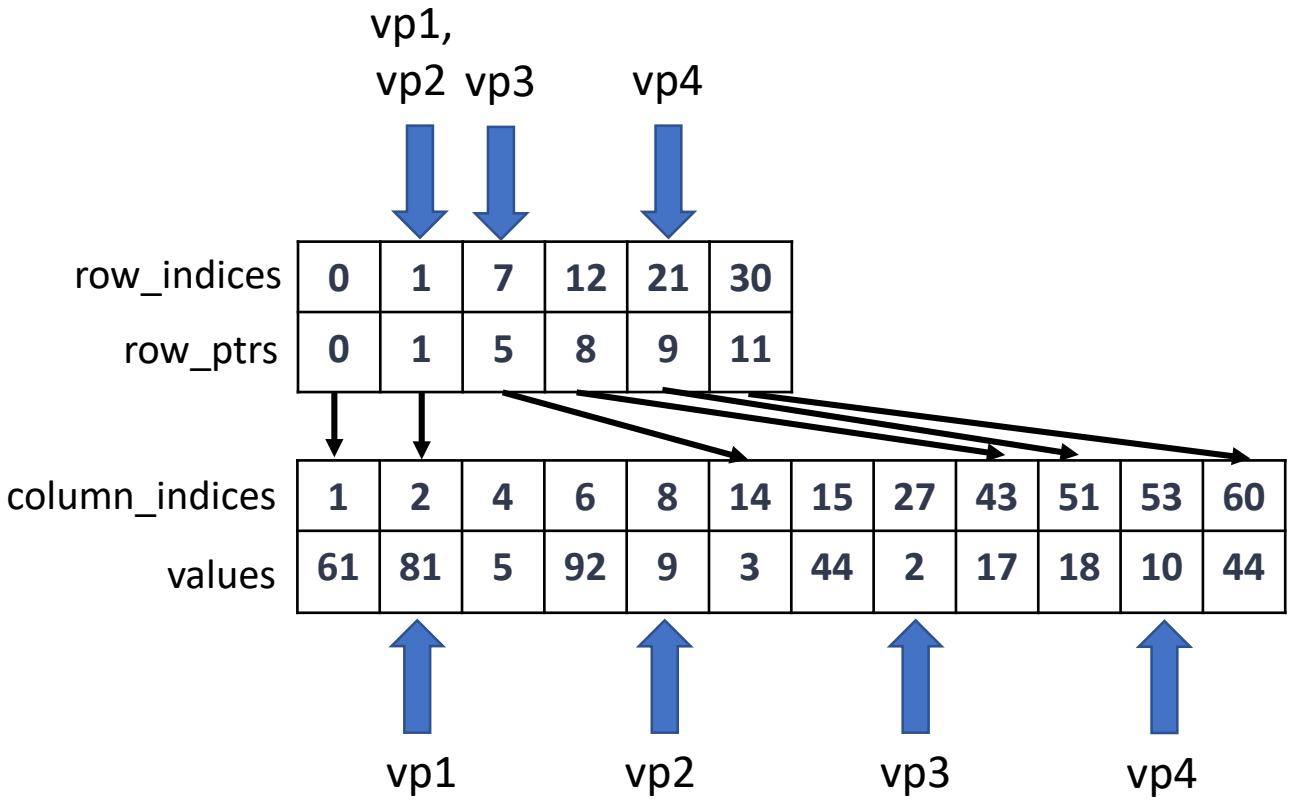
Loop Raking

- Parallel processing of the values array (edge-centric)
- Problem: Accumulation serialization within single vertex
- Solution: Distribute accumulation across different vertices by processing values array in constant intervals (rake)
  - Allows for trivial load-balancing and high virtual processor utilization without repacking
  - Requires predicated tracking of row transitions



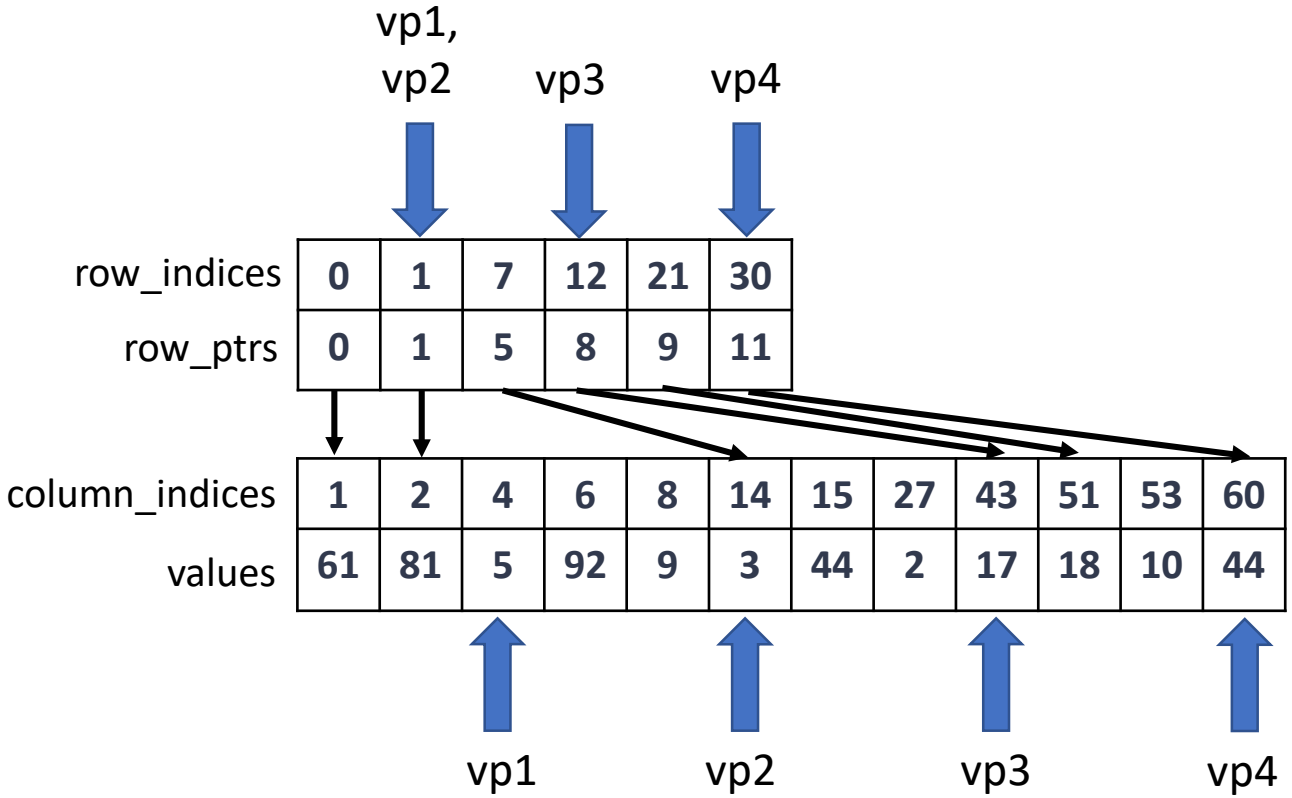
Loop Raking

- Parallel processing of the values array (edge-centric)
- Problem: Accumulation serialization within single vertex
- Solution: Distribute accumulation across different vertices by processing values array in constant intervals (rake)
  - Allows for trivial load-balancing and high virtual processor utilization without repacking
  - Requires predicated tracking of row transitions



Loop Raking

- Parallel processing of the values array (edge-centric)
- Problem: Accumulation serialization within single vertex
- Solution: Distribute accumulation across different vertices by processing values array in constant intervals (rake)
  - Allows for trivial load-balancing and high virtual processor utilization without repacking
  - Requires predicated tracking of row transitions



Evaluation Method – Software Stack

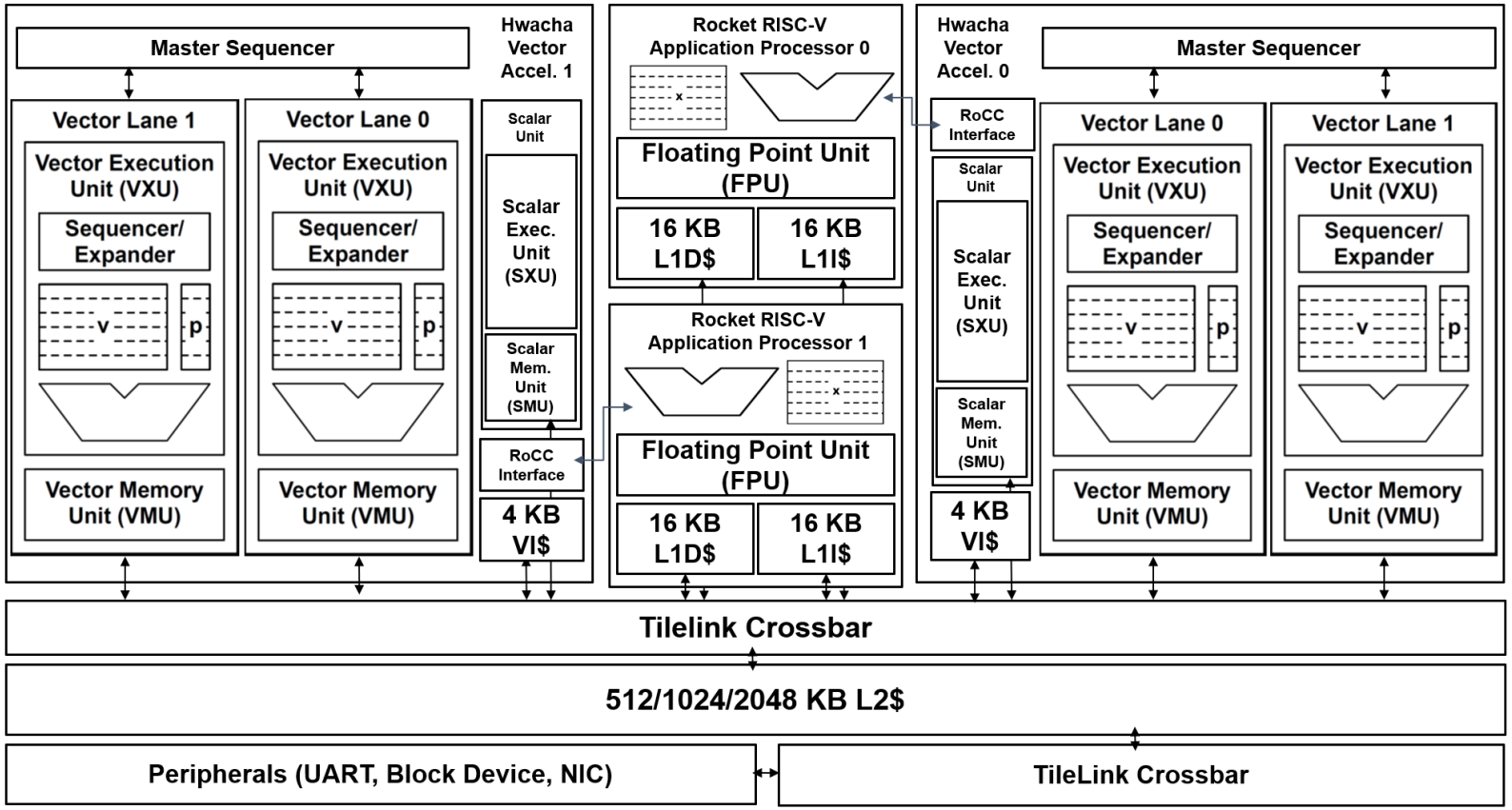
- GraphMat
  - High-performance parallel graph processing framework
  - Vertex-programming front-end interface mapped to linear algebra backend
  - Uses DCSC/DCSR data-structures
  - Parallelism using OpenMP and MPI
  - Used in other architecture graph processing evaluations
- OpenMP
  - Common shared-memory parallel programming multi-threading model
  - Scalable programming model for multi-processors
  - Compile-time and run-time features
  - Used for outer-level thread parallelism

Evaluation Method – Hardware Stack

- Rocket Chip SoC generator
  - Configurable SoC parameters such as L2 caches size and processor tiles
  - Real RTL – conclusions directly reflect on test chips and real silicon
- FireSim – cycle-exact FPGA-accelerated simulation on the public cloud
- Why FireSim and Rocket Chip?
  - Full OpenMP and Linux software stack
  - Vector architectures require detailed μarch
  - DDR Memory models – important for sparse data-structures
  - Real RTL – conclusions directly reflect on test chips and real silicon

Design Space Exploration

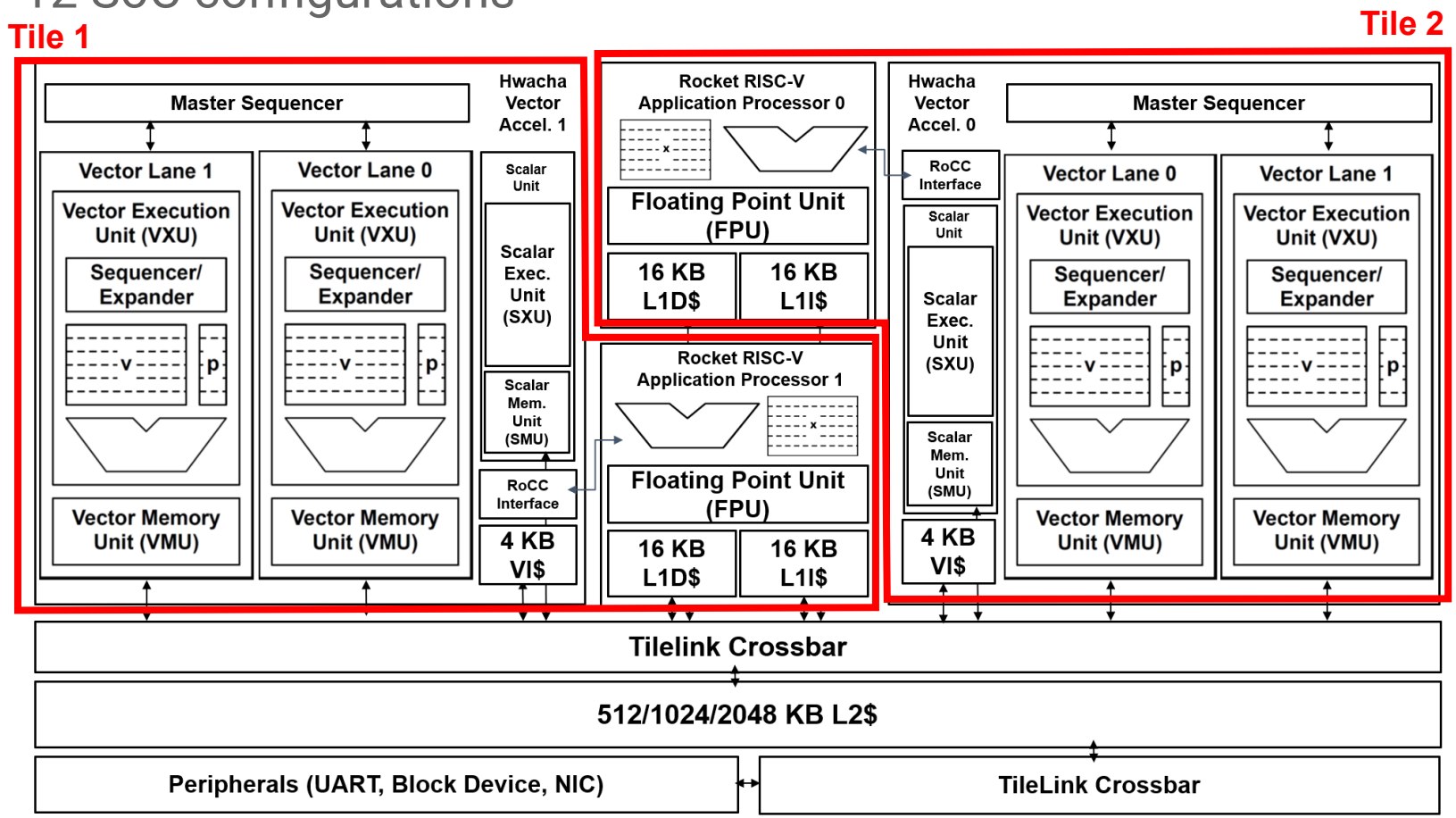
- 12 SoC configurations



Name	Tiles	Vector Lanes / Tile	L2 Cache Size
T1L1C512			
T1L1C1024			
T1L1C2048			
T1L2C512			
T1L2C1024			
T1L2C2048			
T2L1C512			
T2L1C1024			
T2L1C2048			
T2L2C512			
T2L2C1024			
T2L2C2048			

Design Space Exploration

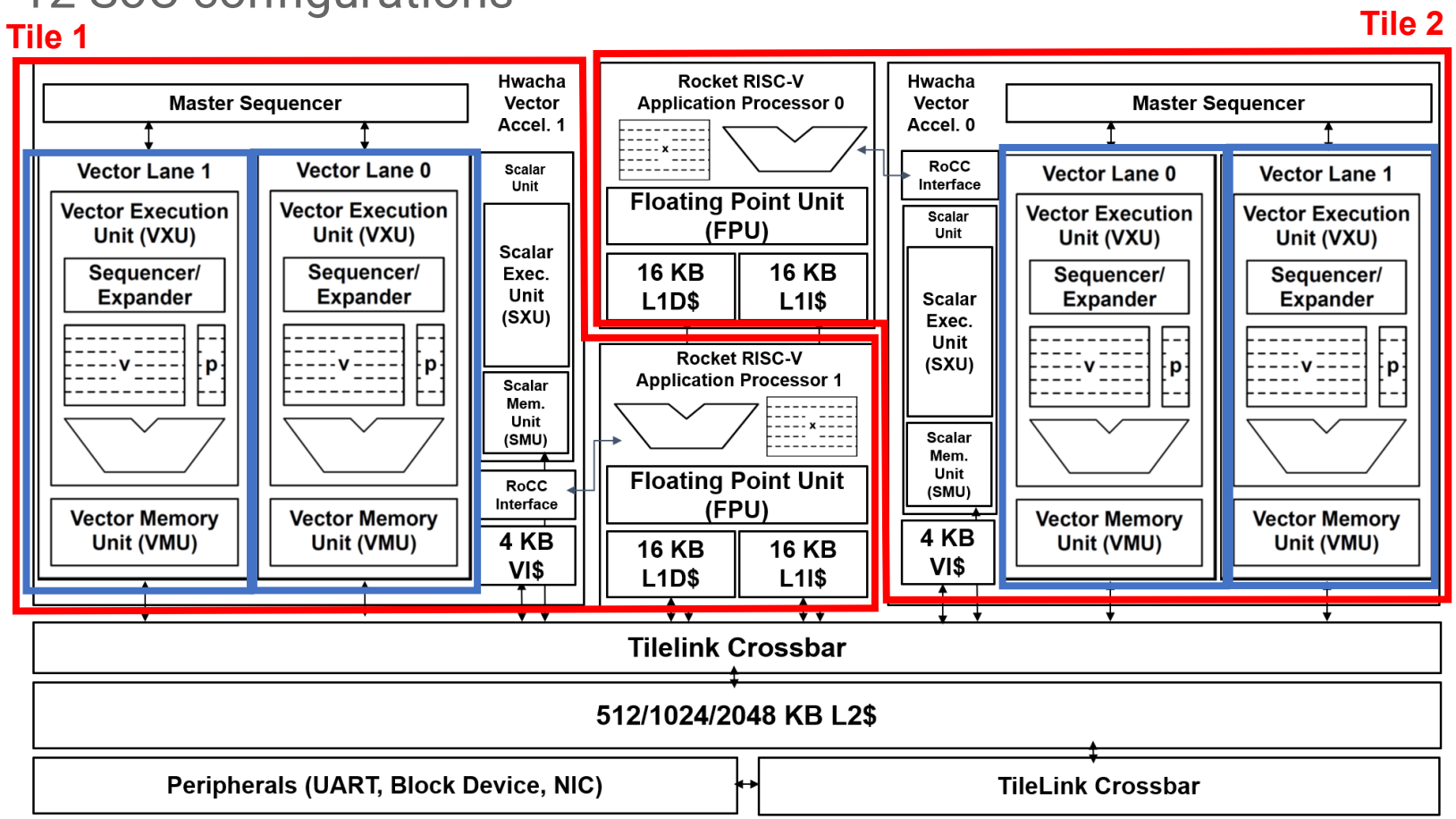
- 12 SoC configurations



Name	Tiles	Vector Lanes / Tile	L2 Cache Size
T1L1C512	1		
T1L1C1024	1		
T1L1C2048	1		
T1L2C512	1		
T1L2C1024	1		
T1L2C2048	1		
T2L1C512	2		
T2L1C1024	2		
T2L1C2048	2		
T2L2C512	2		
T2L2C1024	2		
T2L2C2048	2		

Design Space Exploration

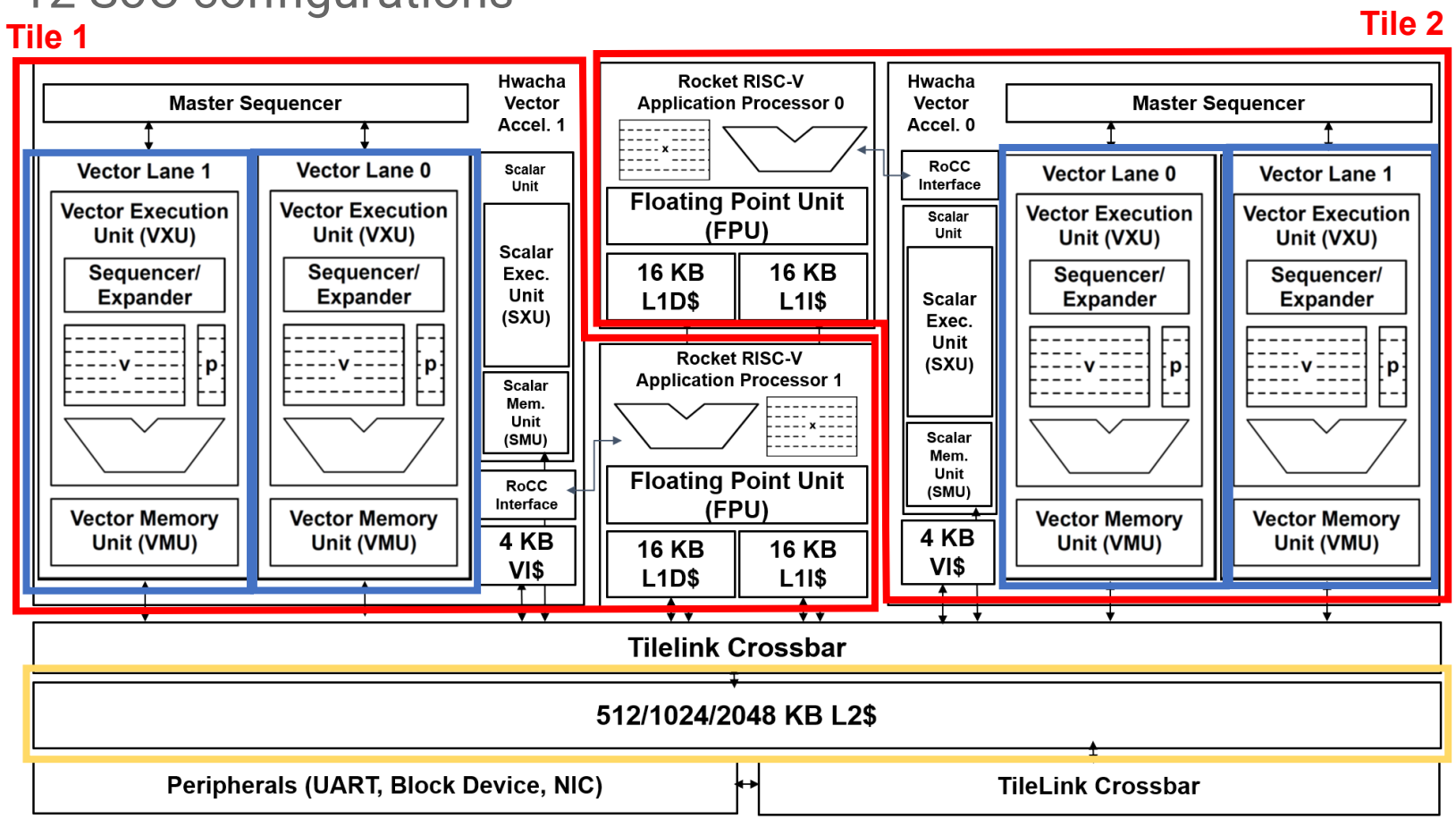
- 12 SoC configurations



Name	Tiles	Vector Lanes / Tile	L2 Cache Size
T1L1C512	1	1	
T1L1C1024	1	1	
T1L1C2048	1	1	
T1L2C512	1	2	
T1L2C1024	1	2	
T1L2C2048	1	2	
T2L1C512	2	1	
T2L1C1024	2	1	
T2L1C2048	2	1	
T2L2C512	2	2	
T2L2C1024	2	2	
T2L2C2048	2	2	

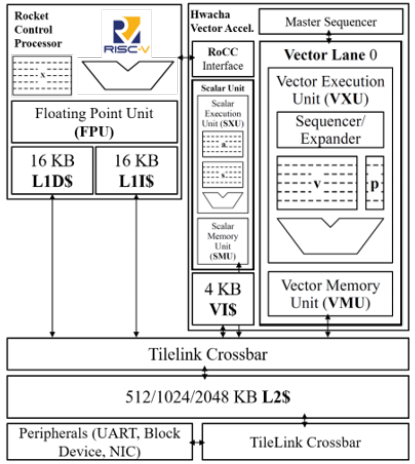
Design Space Exploration

- 12 SoC configurations

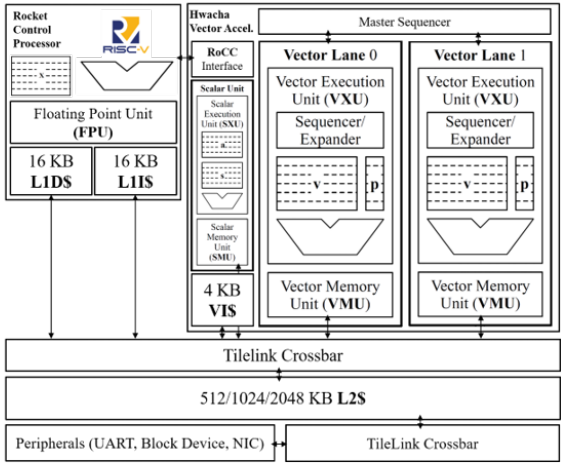


Name	Tiles	Vector Lanes / Tile	L2 Cache Size
T1L1C512	1	1	512
T1L1C1024	1	1	1024
T1L1C2048	1	1	2048
T1L2C512	1	2	512
T1L2C1024	1	2	1024
T1L2C2048	1	2	2048
T2L1C512	2	1	512
T2L1C1024	2	1	1024
T2L1C2048	2	1	2048
T2L2C512	2	2	512
T2L2C1024	2	2	1024
T2L2C2048	2	2	2048

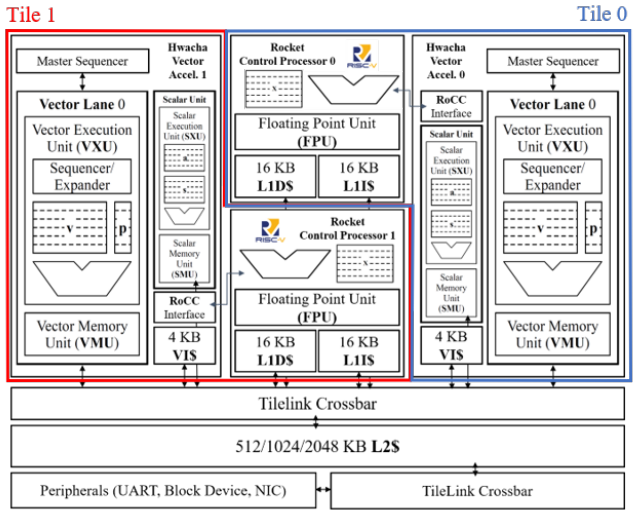
Design Space Exploration



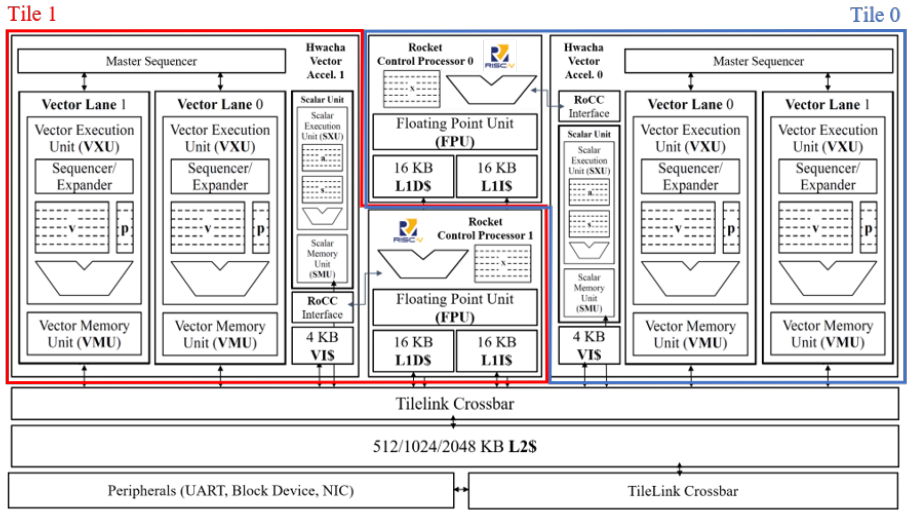
(a) Single Tile, Single Vector Lane



(b) Single Tile, Dual Vector Lane



(c) Dual Tile, Single Vector Lane



(d) Dual Tile, Dual Vector Lane

Software parameters

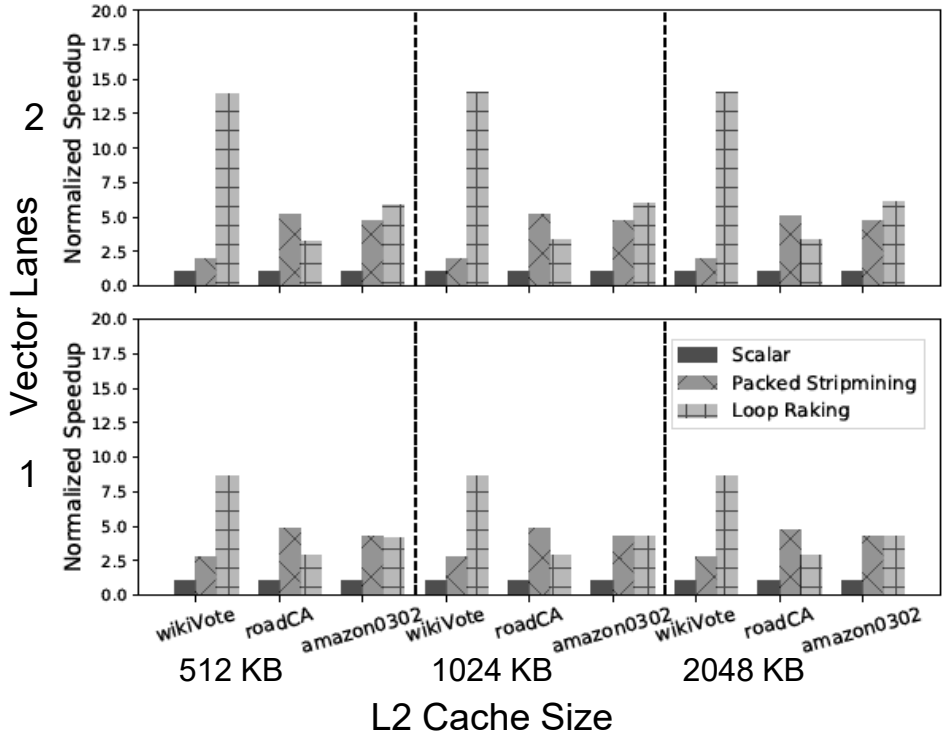
- DCSR Partition Factor
  - Affects granularity of tasks-level parallelism
  - Many tasks/partitions can results in shorter vector length for the inner parallelism level
  - *num_DCSR_partitions = num_hardware_threads x DCSR_partition_factor*
- Graphs
  - Three graphs from the Stanford Network Analysis Project (SNAP)

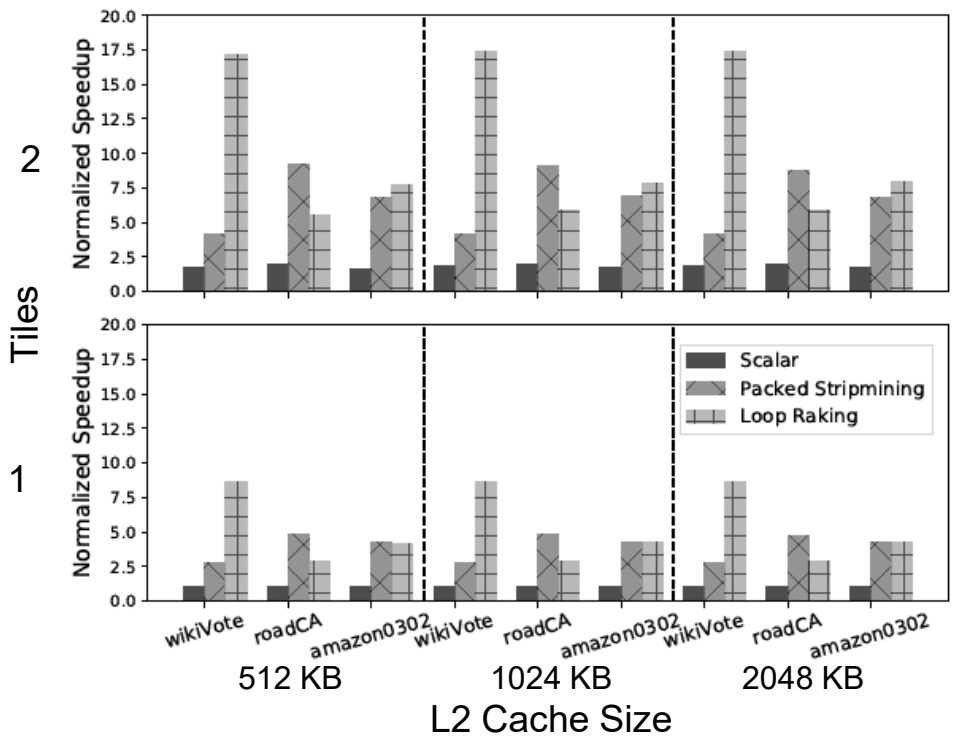
DCSR Partition Factor	DCSR Partitions Single-Tile	DCSR Partitions Dual-Tile
1	1	2
2	2	4
4	4	8
8	8	16
16	16	32

Name	Vertices	Edges	Size
wikiVote	7115	103689	433 KB
roadNet-CA	1965206	2766607	18 MB
amazon0302	262111	1234877	5.7 MB

L2 Cache Size

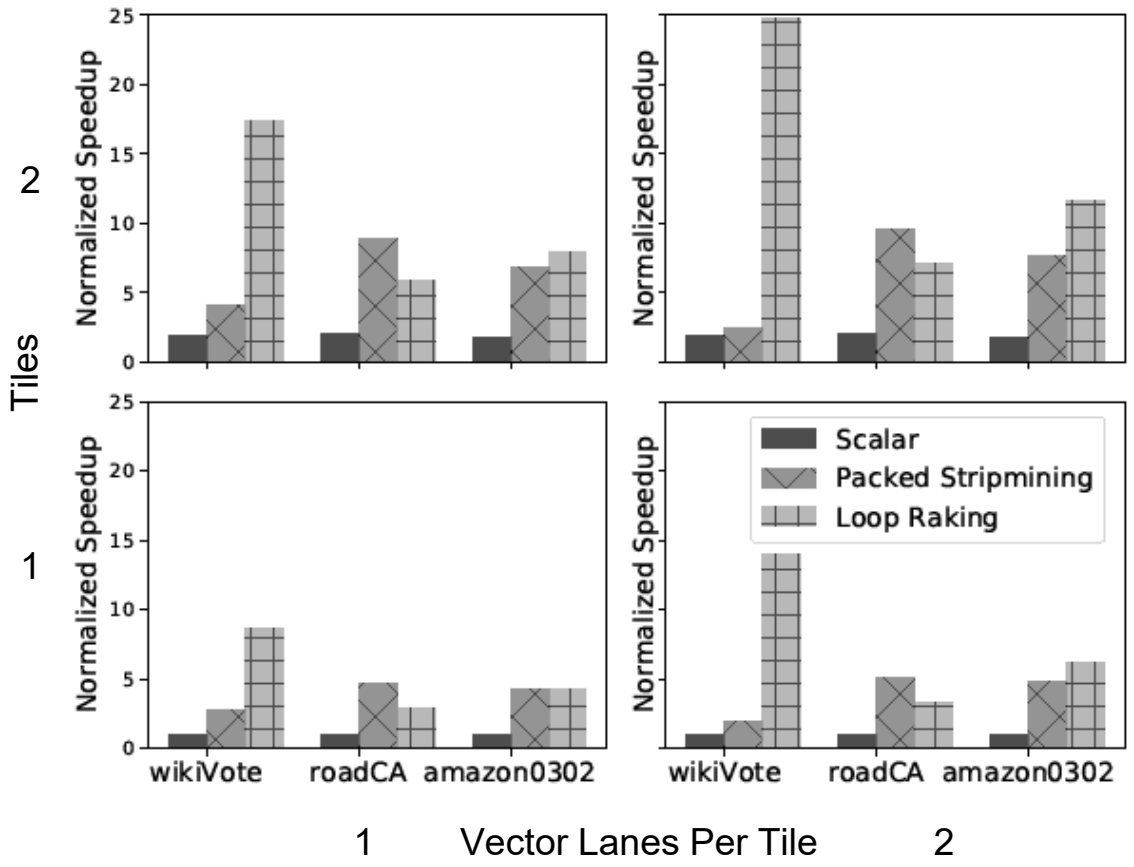
- L2 Cache size does not have an impact
  - Typical of graph workloads with irregular memory accesses
  - Exception: fitting completely within the cache. wikiVote graph fits in L2 size, so demonstrates significantly higher speedup





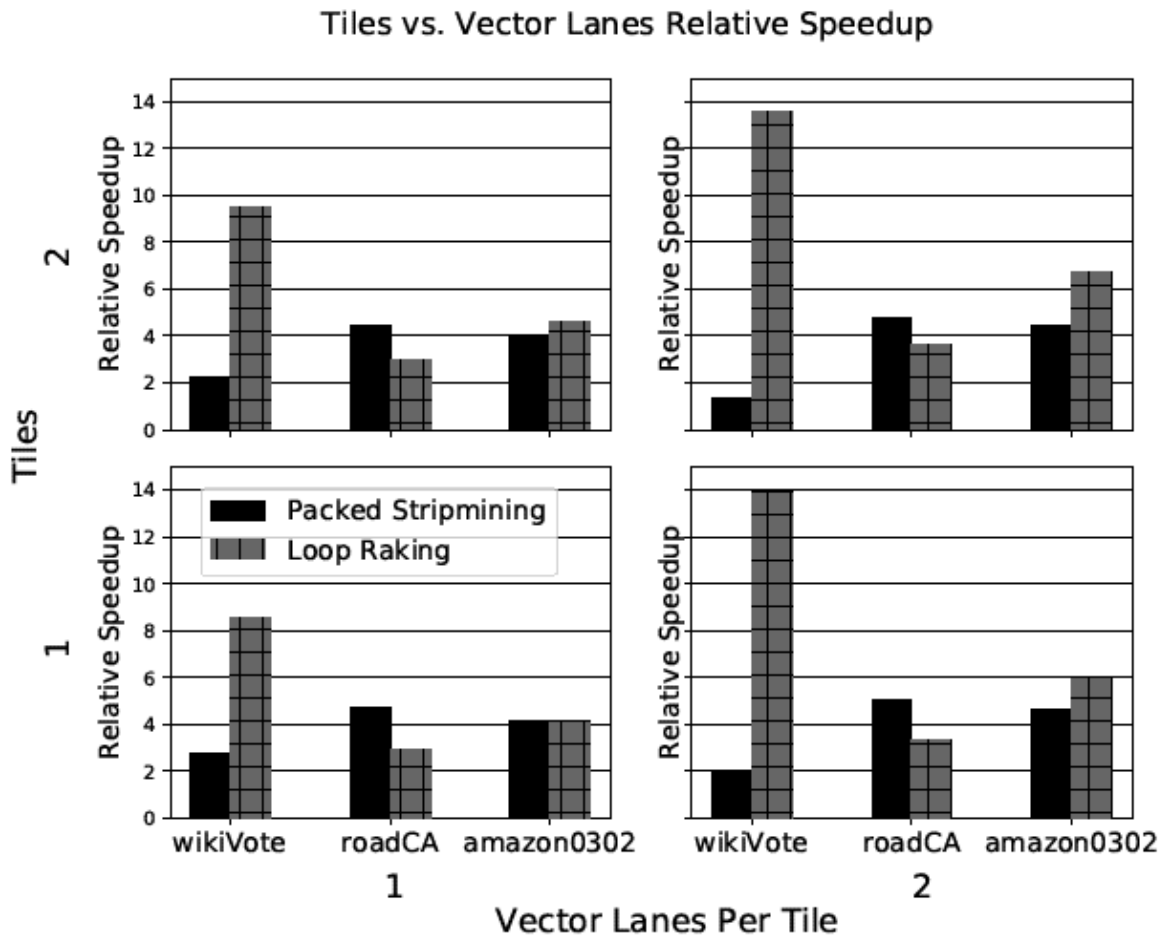
Scaling and Absolute Speedup

- Absolute speedup compared to minimal scalar hardware config
- Multiple tiles present near linear scaling
- Multi-Tile, Single Lane as an efficient design point
  - Single vector lane provides significant speedup (greater than the additional 4 ops/cycles)
  - Additional vector lanes (>1) demonstrate smaller overall absolute speedups



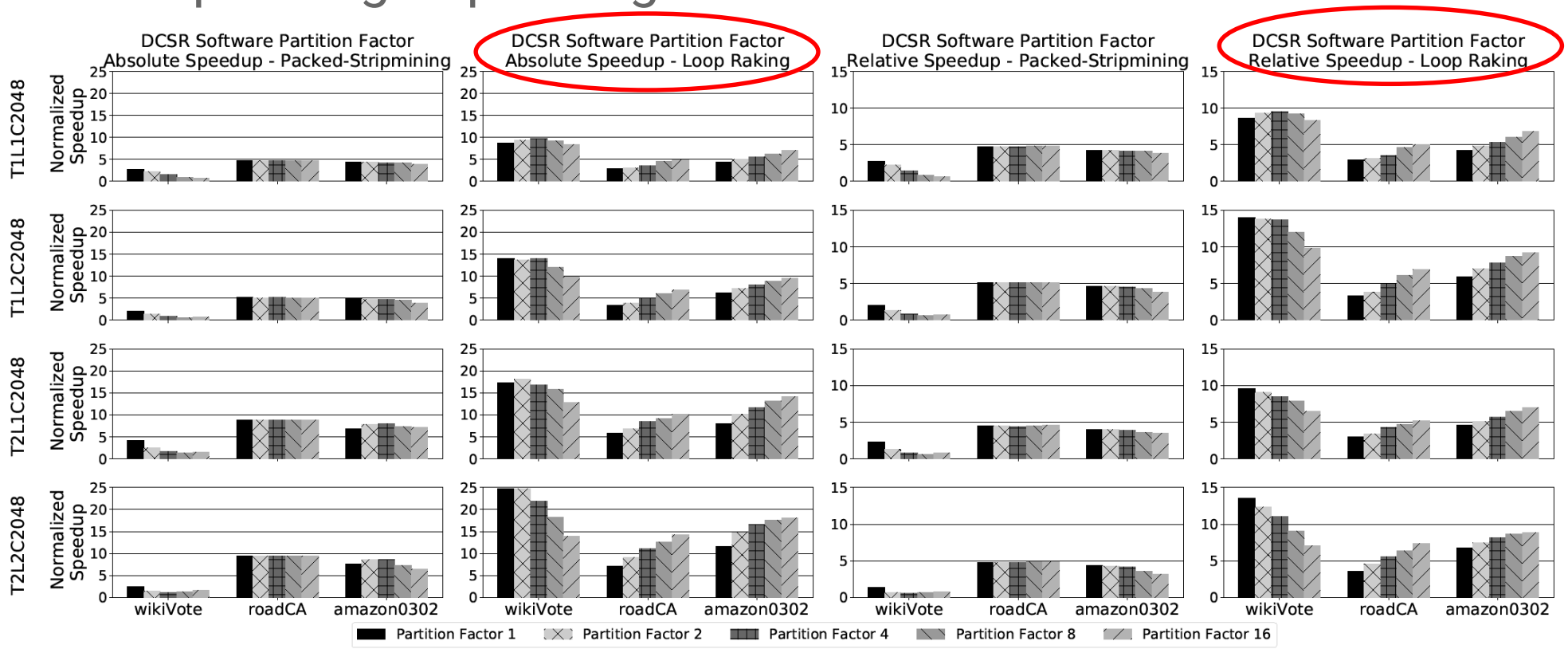
Tiles vs. Vector Lanes

- Relative Speedup compared to the parallel-scalar implementation on the same hardware configuration
- Single-tile-Dual-lane configuration presents higher relative speedup compared to dual-tile-single-lane, even though they have the same overall number of lanes
  - Multi-lane designs have an added benefit in conjunction with multi-core designs



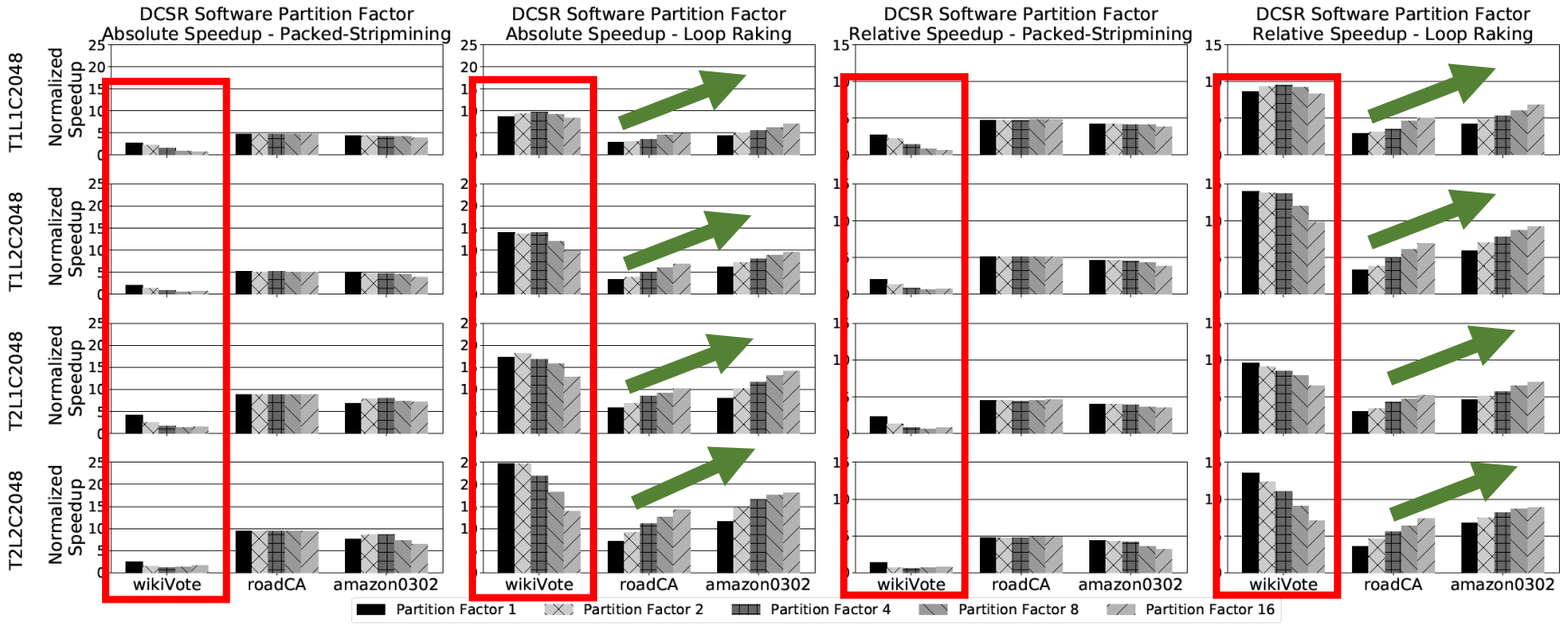
Loop Raking vs. Packed - Stripmining

- Loop-raking can out-perform in all tested hardware configurations, depending on software parameter configuration
- Packed-stripmining re-packing overhead



Software DCSR Partition Factor

- Better performance with higher DCSR partition factors
  - Finer grained load-balancing
  - Exception: small wikiVote graph, due to shorter vector lengths and overhead



Graph Properties

- Bigger graphs present smaller absolute speedups
  - wikiVote > amazon0302 > roadCA
- Small graph effects (wikiVote)
  - Fitting fully in L2 cache can more than double the speedup
  - Vector unit utilization in PageRank depends on the number of vertices **with outgoing edges**
    - As opposed to overall graph size
    - wikiVote has 8000 vertices (enough to keep the vector unit utilized with a high partition factor), but only 2300 vertices with outgoing edges.
- Tested graphs were not significantly scale-free
  - No observed power-law graph effects

Conclusions

- Software/Hardware design space exploration
  - Full Linux-based parallel programming software stack
  - Open-source, silicon-proven hardware
- 4x-25x absolute speedup, 2x-14x vectorized relative speedup
- Loop raking is a better technique than packed-stripmining
- Higher DCSR partitions => better load-balancing
  - Assuming the graph is big enough
- Multi-tile, single vector lane configuration as an efficient design point

Acknowledgments

- Colin Schmidt
- The information, data, or work presented herein was funded in part by the Advanced Research Projects Agency-Energy (ARPA-E), U.S. Department of Energy, under Award Number DE-AR0000849. Research was partially funded by ADEPT Lab industrial sponsor Intel, under the Agile ISTC, and ADEPT Lab affiliates Google, Siemens, and SK Hynix. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof

Questions/Comments