

18-643 Lecture 11: Memory Bound Designs

James C. Hoe

Department of ECE

Carnegie Mellon University

Housekeeping

- Your goal today: see examples of customizing memory paths to algorithms, and vice versa
- Notices
 - Handout #5: lab 2, **due noon, 10/10**
 - Project status report due each Friday
- Readings (see lecture schedule online)
 - Kung, “Memory requirements for balanced computer architectures,” ISCA 1986.
 - Williams, et al., “Roofline: an insightful . . .,” 2008

Topic 1: Arithmetic Intensity



Arithmetic Intensity

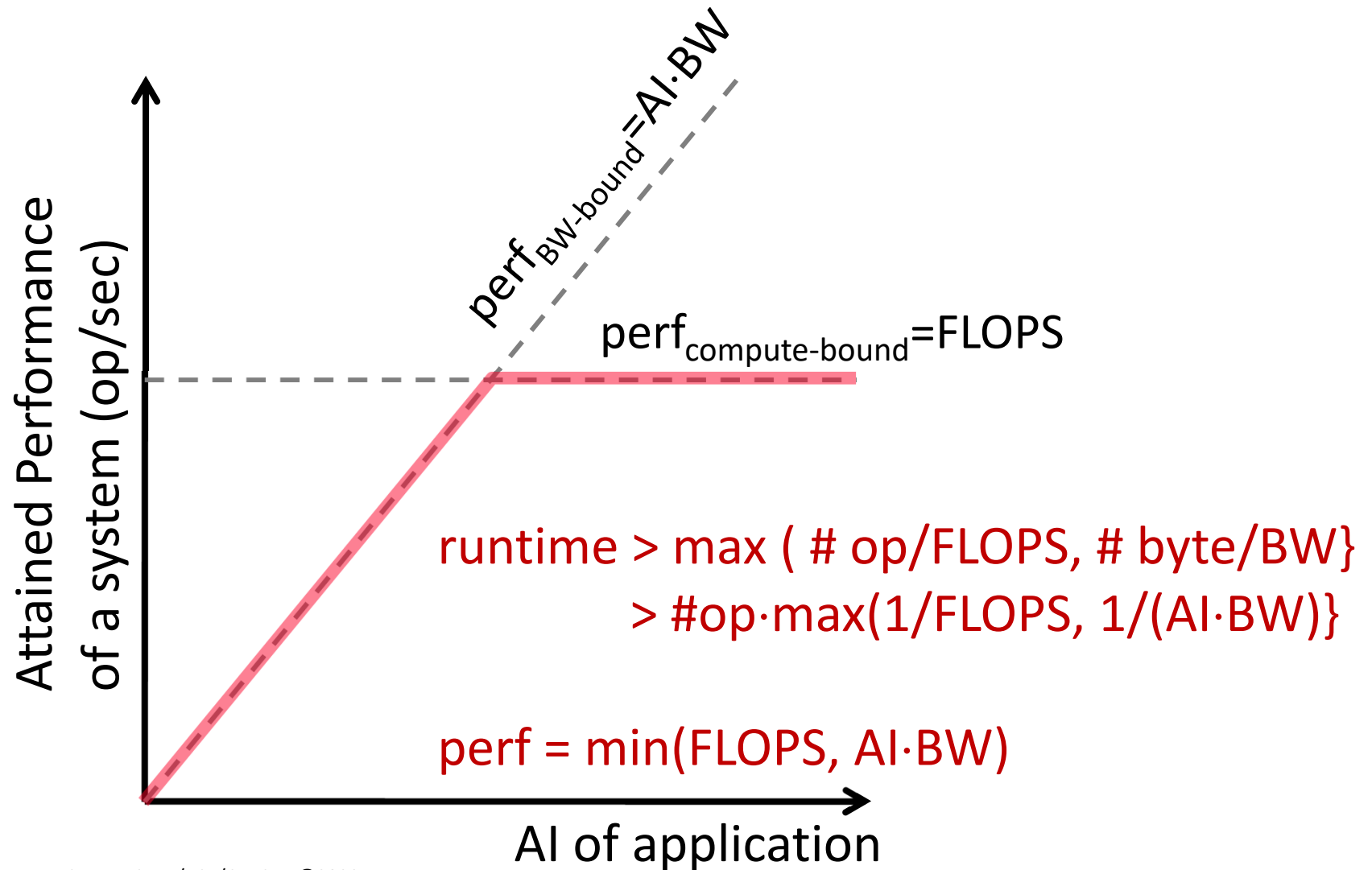
- An algorithm has a cost in terms of operation count
 - $\text{runtime}_{\text{compute-bound}} = \# \text{ operations} / \text{FLOPS}$
- An algorithm also has a cost in terms of number of bytes communicated (ld/st or send/receive)
 - $\text{runtime}_{\text{BW-bound}} = \# \text{ bytes} / \text{BW}$
- Which one dominates depends on
 - ratio of FLOPS and BW of platform
 - ratio of ops and bytes of algorithm
- Average **Arithmetic Intensity (AI)**
 - how many ops performed per byte accessed
 - $\# \text{ operations} / \# \text{ bytes}$

Last Time



Roofline Performance Model

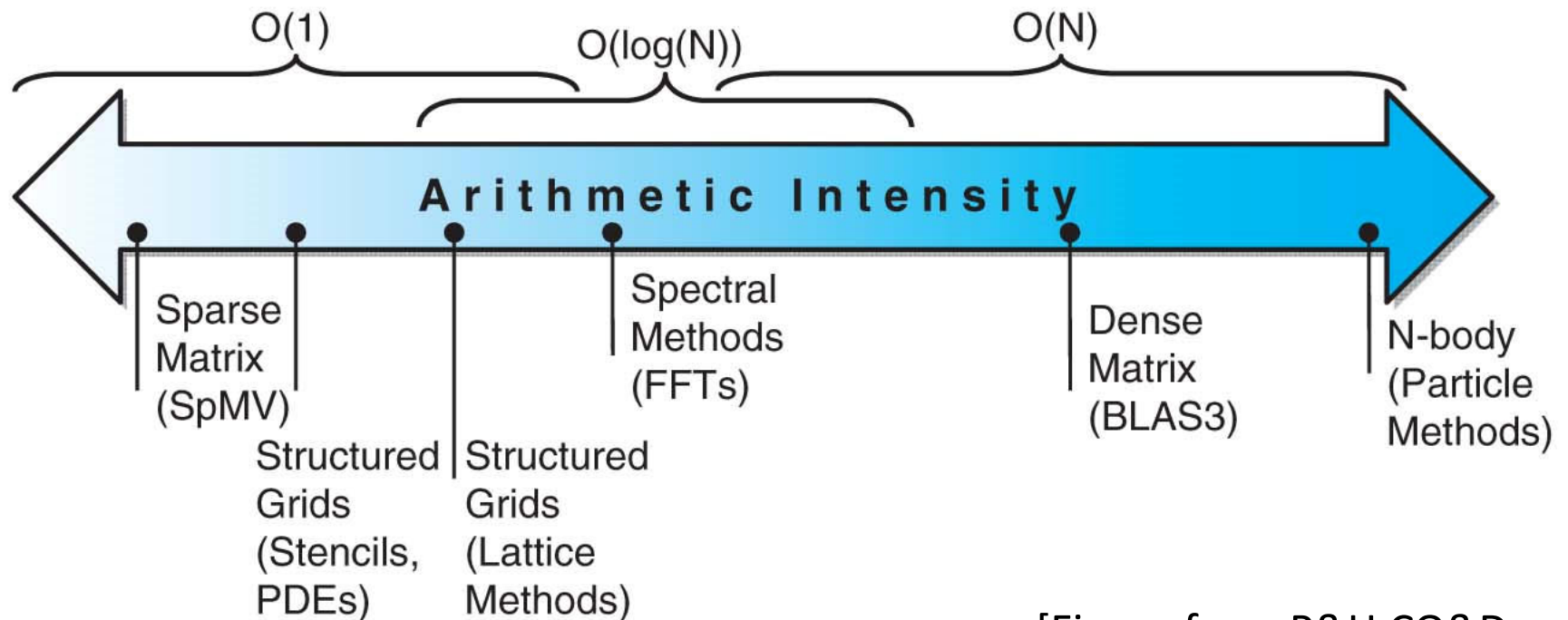
[Williams&Patterson, 2006]



AI and Algorithms

harder to speed up
& harder to scale up

easier



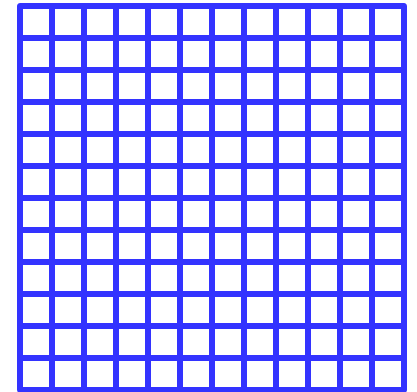
[Figure from P&H CO&D,
COPYRIGHT 2009 Elsevier.
ALL RIGHTS RESERVED.]

Simple AI Example: MMM

```

for (i=0; i<N; i++)
  for (j=0; j<N; j++)
    for (k=0; k<N; k++)
      C[i][j] += A[i][k] * B[k][j];

```



- N^2 data-parallel dot-product's
 - operation count: N^3 float-mult and N^3 float-add
- External memory access (assume 4-byte floats)
 - assume N is large s.t. 1 row/col too large for on-chip
 - $2N^3$ 4-byte reads (of A and B) from DRAM
 - ... N^2 4-byte writes (of C) to DRAM ...
- Arithmetic Intensity $\approx 2N^3 / (4 \cdot 2N^3) = 1/4$

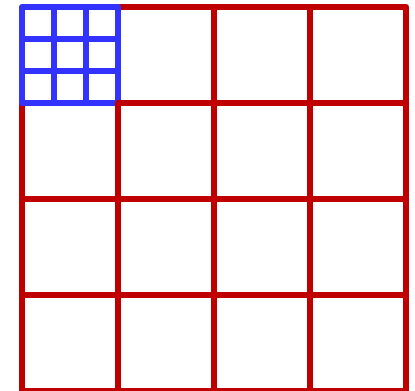
GTX1080: 8 TFLOPS vs 320GByte/sec

Less Simple AI Example: MMM

```

for (i0=0; i0<N; i0+=Nb)
  for (j0=0; j0<N; j0+=Nb)
    for (k0=0; k0<N; k0+=Nb) {
      for (i=i0; i<i0+Nb; i++)
        for (j=j0; j<j0+Nb; j++)
          for (k=k0; k<k0+Nb; k++)
            C[i][j] += A[i][k] * B[k][j];
    }

```



- Imagine a ' N/N_b ' x ' N/N_b ' **MATRIX** of $N_b \times N_b$ matrices
 - inner-triple is straightforward **matrix-matrix** mult
 - outer-triple is **MATRIX-MATRIX** mult
- To improve AI, hold $N_b \times N_b$ sub-matrices on-chip for data-reuse

AI of blocked MMM Kernel ($N_b \times N_b$)

```

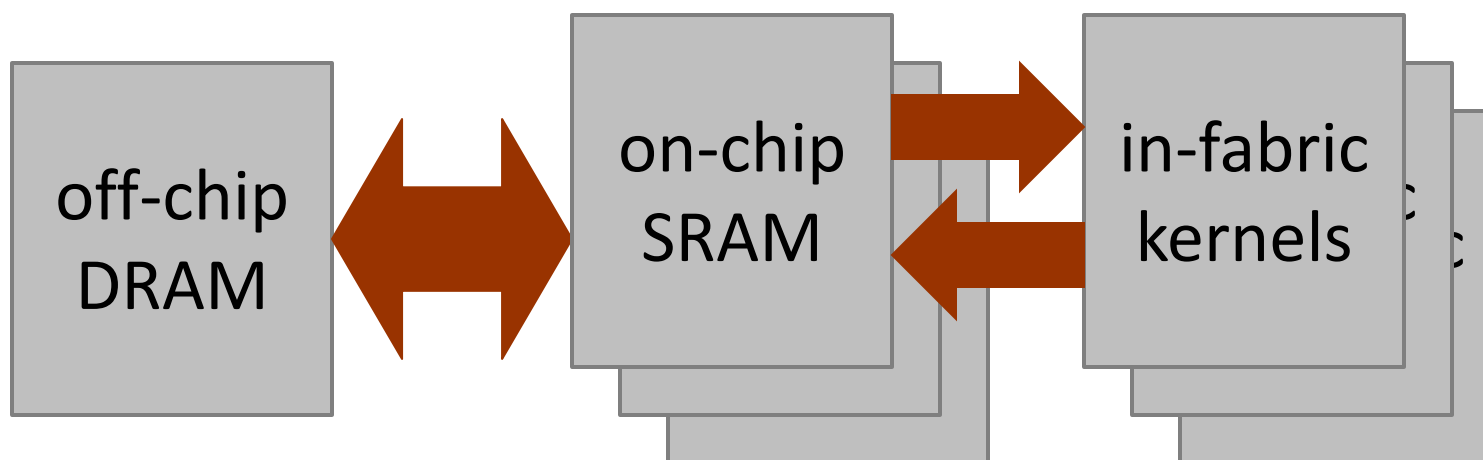
for (i=i0; i<i0+Nb; i++)
  for (j=j0; j<j0+Nb; j++) {
    t=C[i][j];
    for (k=k0; k<k0+Nb; k++)
      t+=A[i][k]*B[k][j];
    C[i][j]=t;
  }

```

- Operation count: N_b^3 float-mult and N_b^3 float-add
- When **A**, **B** fit in scratchpad ($2 \times N_b^2 \times 4$ bytes)
 - $2 \times N_b^3$ 4-byte on-chip reads (**A**, **B**) (fast)
 - $2 \times N_b^2$ 4-byte off-chip DRAM read **A**, **B** (slow)
 - $2 \times N_b^2$ 4-byte off-chip DRAM read/write of **C** (slow)
- Arithmetic Intensity = $2N_b^3 / (4 \cdot 4N_b^2) = N_b/8$



The Performance Balancing Act



1. Kernels' **op/sec** requires some **byte/sec** — a function of **algorithm** and **kernel size**
2. On-chip SRAM “filters” kernel **byte/sec** down to DRAM **byte/sec** — a function of SRAM **capacity**
3. DRAM system offers some aggregate **byte/sec** — a function of **access pattern (algorithm)**

Some Hints on Lab 3

- Lab 3 kernel's op/sec just need to be fast enough to match memory-bound ($\text{op/byte} \times \text{byte/sec}$)
- Lab 3 emphasis on improving memory-bound
 - size tiles and order outer loops for data reuse
(don't forget the batch loop!)
 - Mindful of buffer sizes and degree reuse*
 - use memory resources efficiently *(fit bigger tiles)*
 - layout data in DRAM for sequential read *(don't forget to widen the read path)*
- Use DFX to tune the 2 layers differently

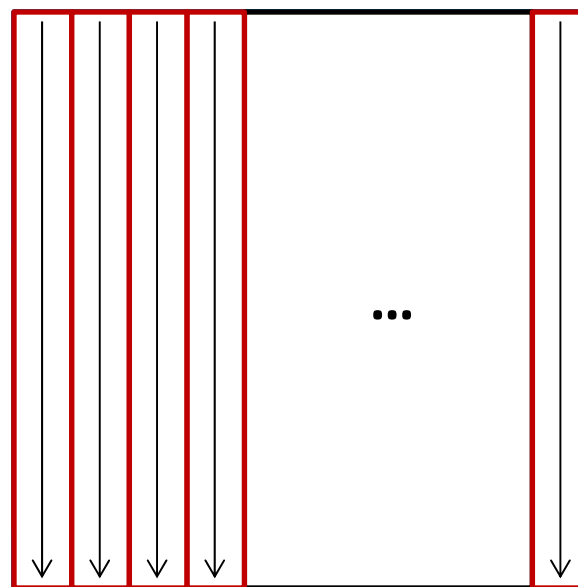
Topic 2: Data Layout and Access Pattern

Data Layout and Access Pattern: 2D-FFT

- Row-column algorithm:

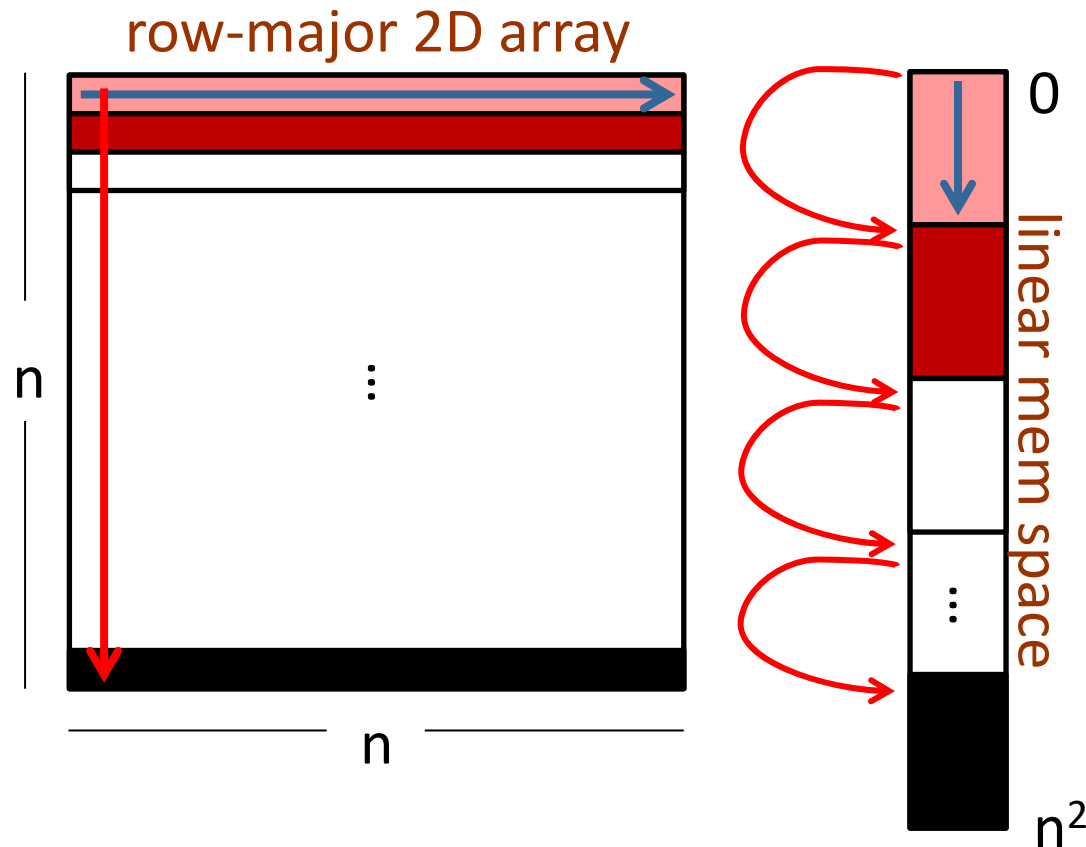
$$2\text{D-DFT}_{n \times n} = \underbrace{(\text{DFT}_n \otimes \text{I}_n)}_{\text{Column Stage}} \underbrace{(\text{I}_n \otimes \text{DFT}_n)}_{\text{Row Stage}}$$

Dataset:
(Logical abstraction
of the 2D dataset)

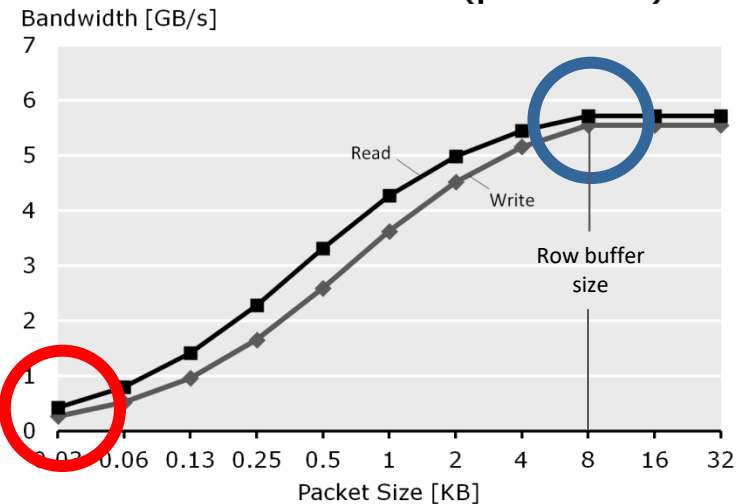


Inefficient DRAM Access Patterns

- Row-wise traversal -> Sequential accesses
- Column-wise traversal -> Large strided accesses

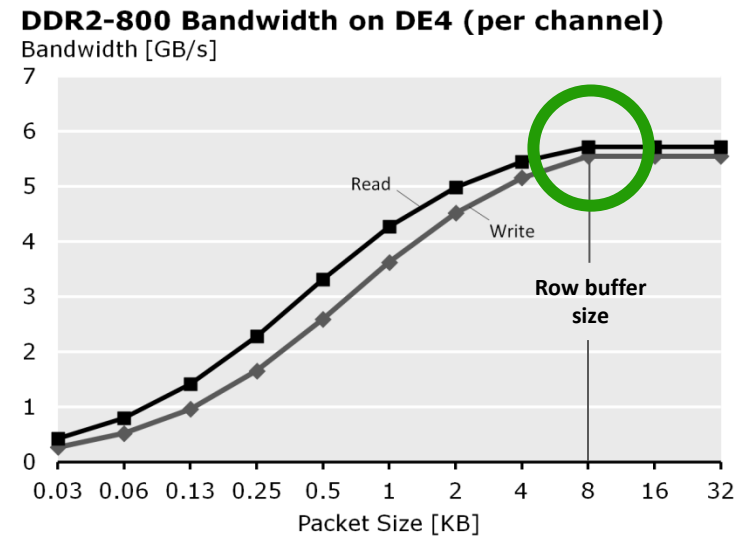
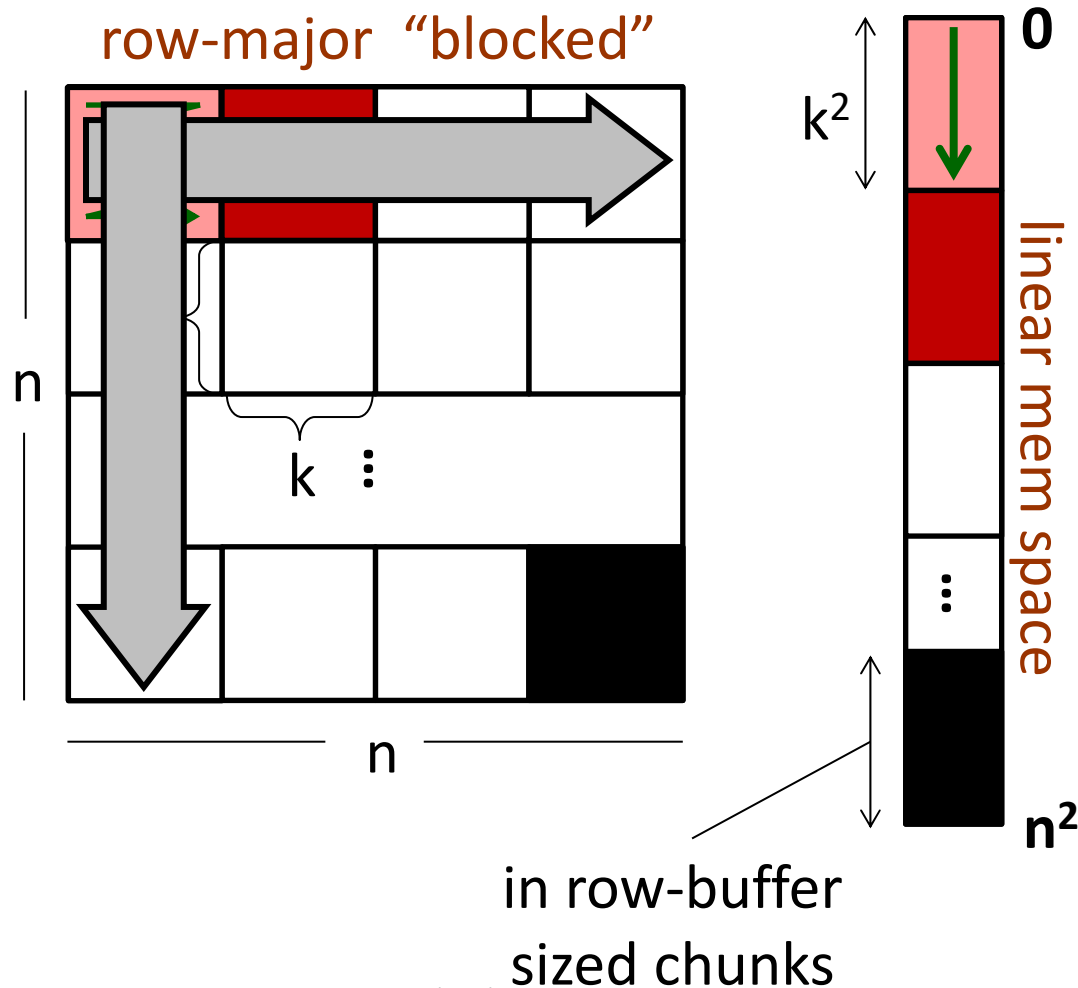


DDR2-800 Bandwidth on DE4 (per channel)



Gather-Scatter

Tiled Layout and Access Patterns



What if you only have $(k/2) \cdot n$ on-chip buffer?

Design Generator w/ Tensor Formalism

$$2\text{D-DFT}_{n \times n} = \overbrace{(\text{DFT}_n \otimes \text{I}_n)}^{\text{column stage}} \overbrace{(\text{I}_n \otimes \text{DFT}_n)}^{\text{row stage}} \quad \text{row-column algorithm}$$

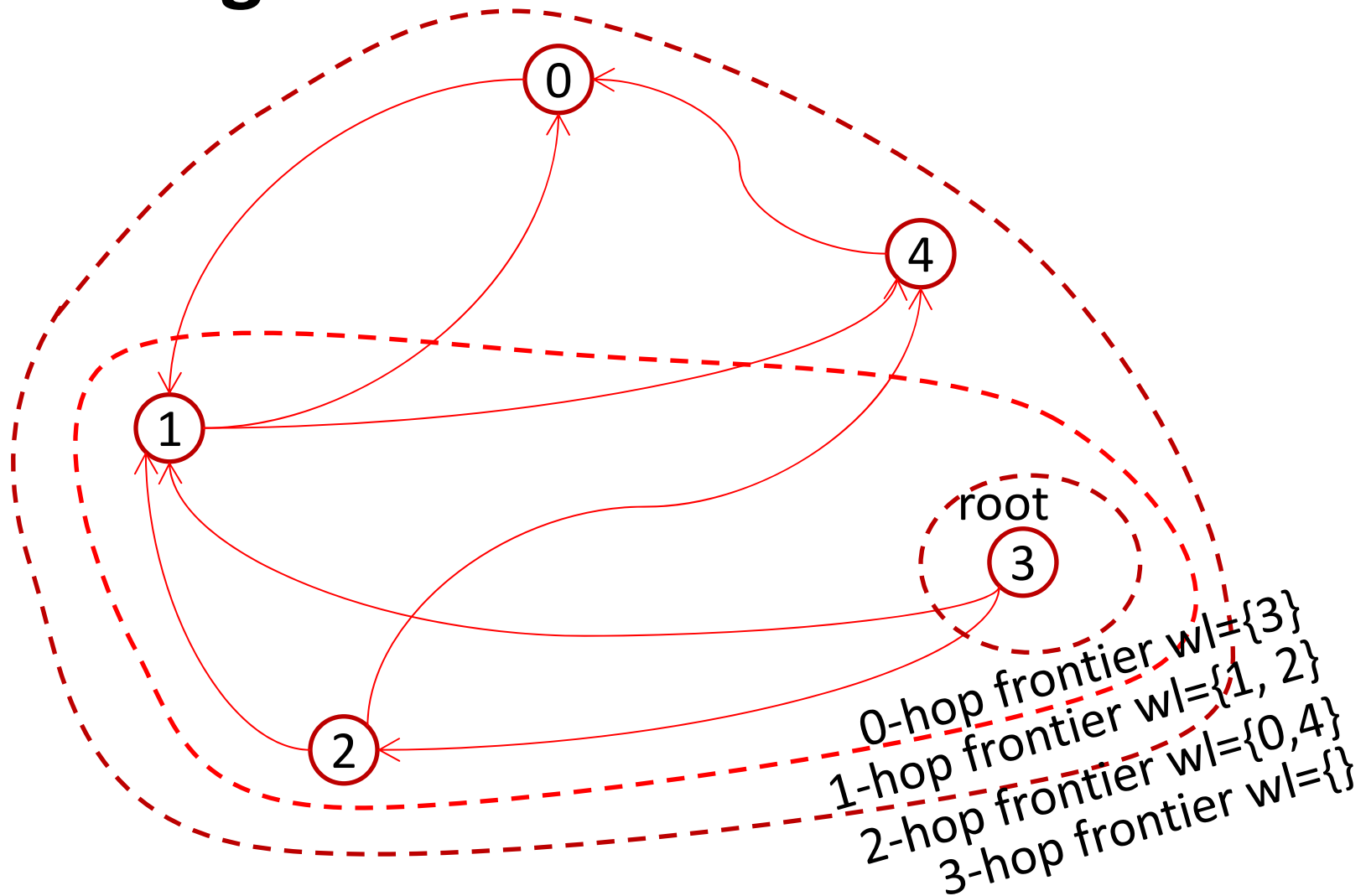
$$= \prod_{i=0}^1 (\text{L}_n^{n^2} (\text{I}_n \otimes \text{DFT}_n) \text{I}_{n^2}) \quad \text{symmetric algorithm}$$

$$= \prod_{i=0}^1 \left(\underbrace{(\text{L}_n^{n^2/k} \otimes \text{I}_k)}_{\text{write tiles column-wise}} \underbrace{(\text{I}_{n/k} \otimes \text{L}_n^{nk})}_{\text{transpose and re-tile on-chip}} \underbrace{(\text{I}_{n/k} \otimes (\text{I}_k \otimes \text{DFT}_n))}_{\text{FFT processing}} \underbrace{(\text{I}_{n/k} \otimes (\text{L}_k^n \otimes \text{I}_k))}_{\text{linearize on-chip}} \underbrace{(\text{I}_{n/k} \otimes (\text{L}_{n/k}^n \otimes \text{I}_k))}_{\text{read tiles row-wise}} \right) \quad \text{symmetric algorithm with tiling}$$

not on midterm

Topic 3: Irregular

Irregular: Breadth First Search



Large graph has more than millions of nodes
with may be handful edges per node

Breadth-First Search (Pseudo Code)

```
foreach (node n in graph) n.dist= $\infty$ ;
```

```
worklist = {root}; root.dist=0;
```

```
foreach (node n in worklist) {  
  foreach (neighbor of n) {  
    if (n.dist + 1 < neighbor.dist) {  
      neighbor.dist = n.dist + 1;  
      add neighbor to worklist;  
    }  
  }  
}
```

*Has Parallelism?
Yes, not perfect,
input-dependent*

(see http://iss.ices.utexas.edu/?p=projects/galois/benchmarks/bread_first_search)

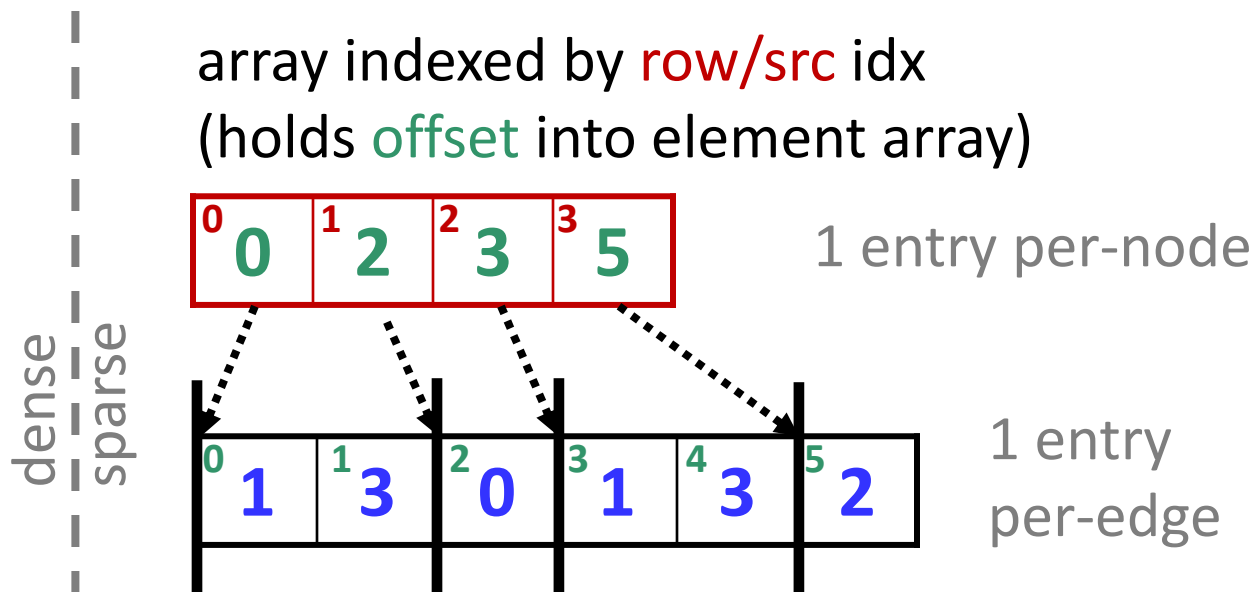
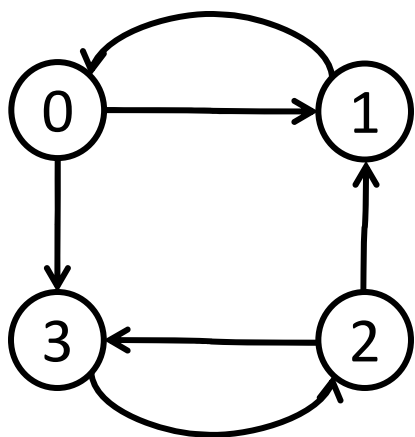
Real Code with CSR Memory Accesses

```
while(wl.mHowmany) { // worklist not empty
    // repeat for each node on frontier
    int curr=wl.mList[wl.mDeq]; // S0
    int myDist=graph->mPerNode[curr].dist; // S1
    int numEdges=graph->mPerNode[curr].fanout; // S1
    int scan=graph->mPerNode[curr].edges; // S1
    { ... dequeue from worklist ...}
    while (numEdges--) {
        // repeat for each neighbor
        int dest=graph->mPerEdge[scan].dest; // S2
        int destDist=graph->mPerNode[dest].dist; // S3
        if ((myDist+1)<destDist) { // S4
            graph->mPerNode[dest].dist=myDist+1; // S4
            { ...enqueue dest to worklist...} // S5
        }
        scan++;
    }
}
```

Compressed Sparse Row (CSR)

Adjacency Matrix

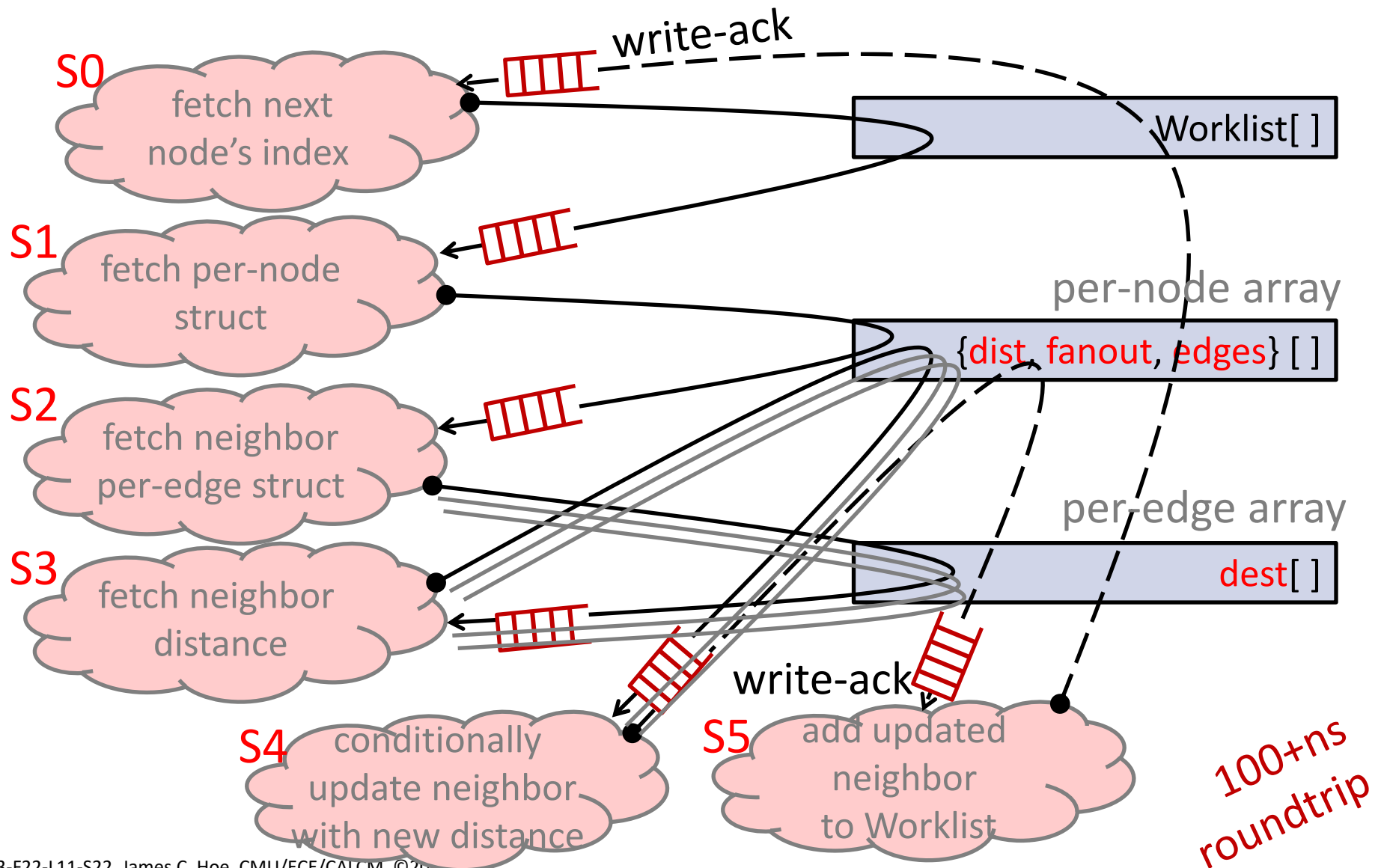
src ↓	dest →	0	1	2	3
0	0	1	0	1	
1	1	0	0	0	
2	0	1	0	1	
3	0	0	1	0	



array of all non-0 elements
in row-order
(holds **col/dest** index)

Large graph has millions or more nodes
each with may be handful edges per node

Elastic HW Processing Pipeline

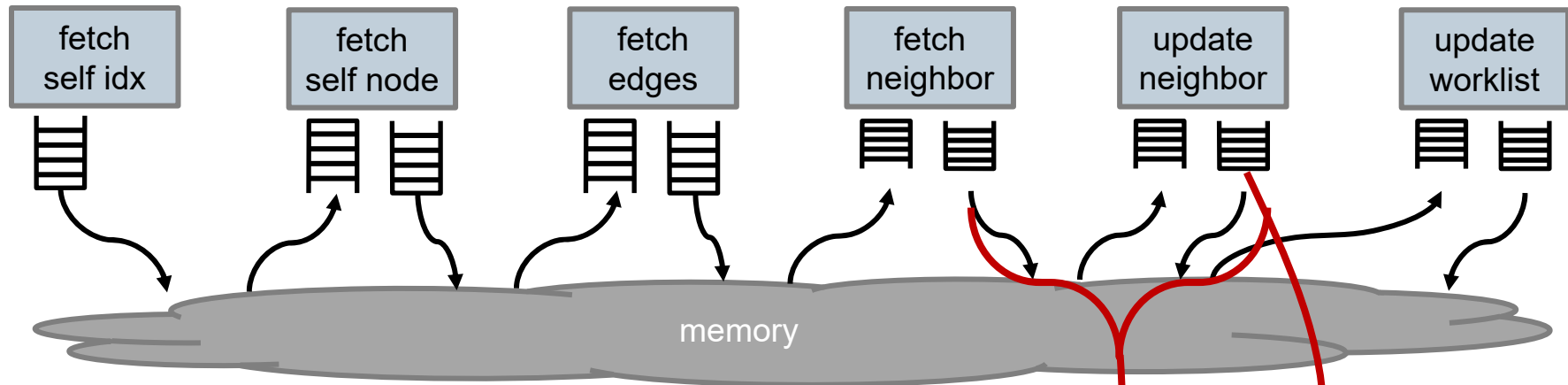


BFS Irregular Access Pattern

- Irregular and graph dependent
 - **S0 read worklist**: spatial locality, non-temporal
 - **S1 read node array (self)**: no locality
 - **S2 read edge array**: some spatial locality, non-temporal
 - **S3 read node array (neighbor)**: no locality
 - **S4 write node array (neighbor)**: temporal with S3
 - **S5 write worklist**: spatial locality, non-temporal
- **S3** most problematic of all
 - **S1** and **S3** lack locality but **S3** repeated per neighbor
 - same number of **S2** and **S3** but **S2** has spatial locality
 - BTW, **S3** and **S4** could have RAW hazard
 - BTW, all read/write granularity is multi-word

Can
caching
help?

How can “Caching” Help?

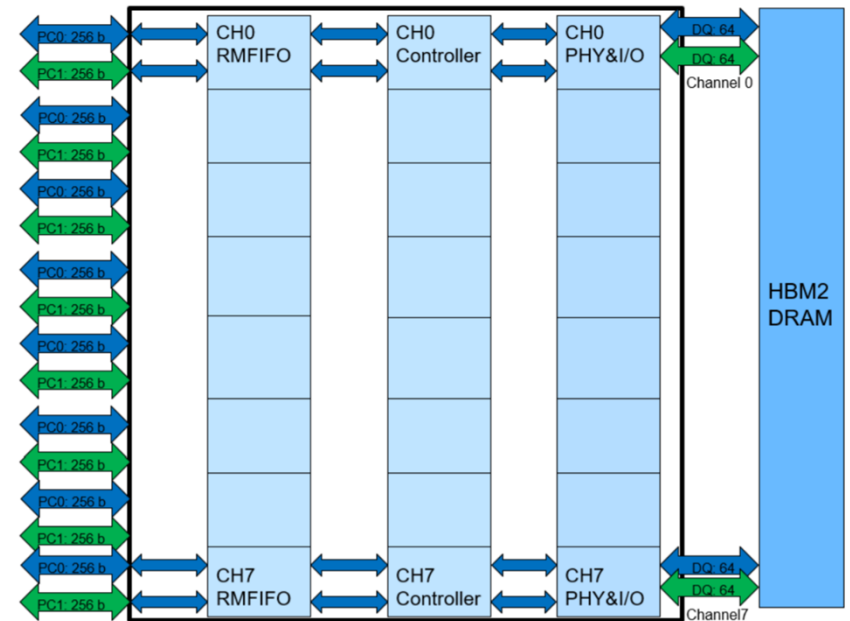


- Custom cache for only neighbor distance in node array (read in **S3** written in **S4**)
- RAW hazard interlock when nodes in pipeline have same neighbor (stall **S3** until conflict-free)
- Coalesces neighbor updates (collect partial writes to multi-word DRAM block)

Organized and operated unlike an ABC cache!!

How can HBM BW help?

- HBM gives you 512GB/sec
 - data partitioned 32 ways
 - 100s nsec latency
 - 32 byte per fetch



- Partition graph data into 32 banks s.t. maximizing non-conflict concurrent accesses across channel
- Per channel, prefetch many 10s of outstanding read requests to cover latency (*Little's Law*)
- Re-index nodes in graph s.t. maximizing spatial locality of each 32B fetch

Complexity of Sparse Algorithms

- Graph processing expressible using linear algebra primitives: SpMV, SpMSpV, SpMM, . . .

Simplicity of form belies performance difficulties

- Basic challenges in large data set, low arithmetic intensity, and irregular access pattern
- graph-dependent behavior requires multiple implementations of same primitive depending on:
 - size and sparsity
 - structured?
 - compressed format: CSR, CSC, COO, . . .

Each combination a different optimal design

Parting Thoughts

- When scaling data size and performance, memory design quickly become the PROBLEM
 - capacity, bandwidth, latency
- FPGAs specialization is an asset
 - balance memory throughput and compute throughput
 - have data to the right place at the right time
 - alter algorithm to memory constraints
- Designing “memorypath” as important as designing “datapath”