Improving Locality in Consecutive Sparse and Dense Matrix Multiplications

Mohammad Mahdi Salehi, Kazem Cheshmi

Electrical and Computer Engineering

McMaster University

Outline

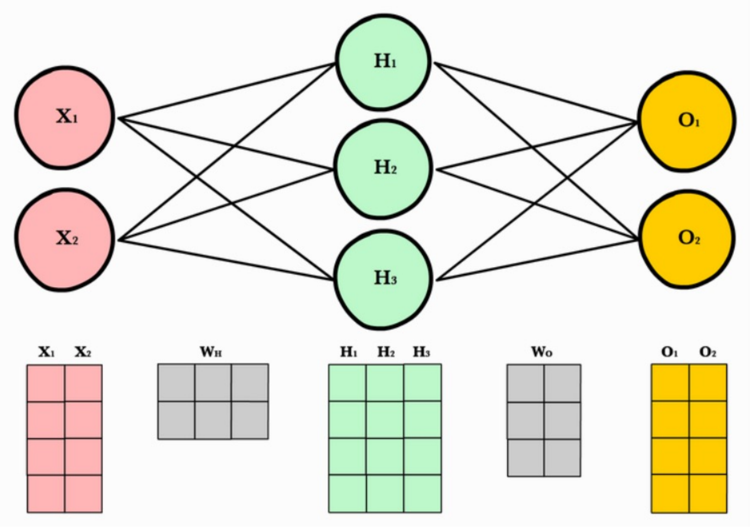
- Motivation
- Prior Work
- Methodology
- Experimental Results

Motivation

Improving Locality in Consecutive Sparse and Dense Matrix Multiplications

Consecutive Matrix Multiplications

- Common Pairs:
  - GeMM-SpMM
  - SpMM-SpMM
- Machine Learning
  - Sparse Matrices:
    - Graph neural networks(graph adjacency matrix)
    - Sparse neural networks(pruned weights)
- Linear Solvers
- Power Methods

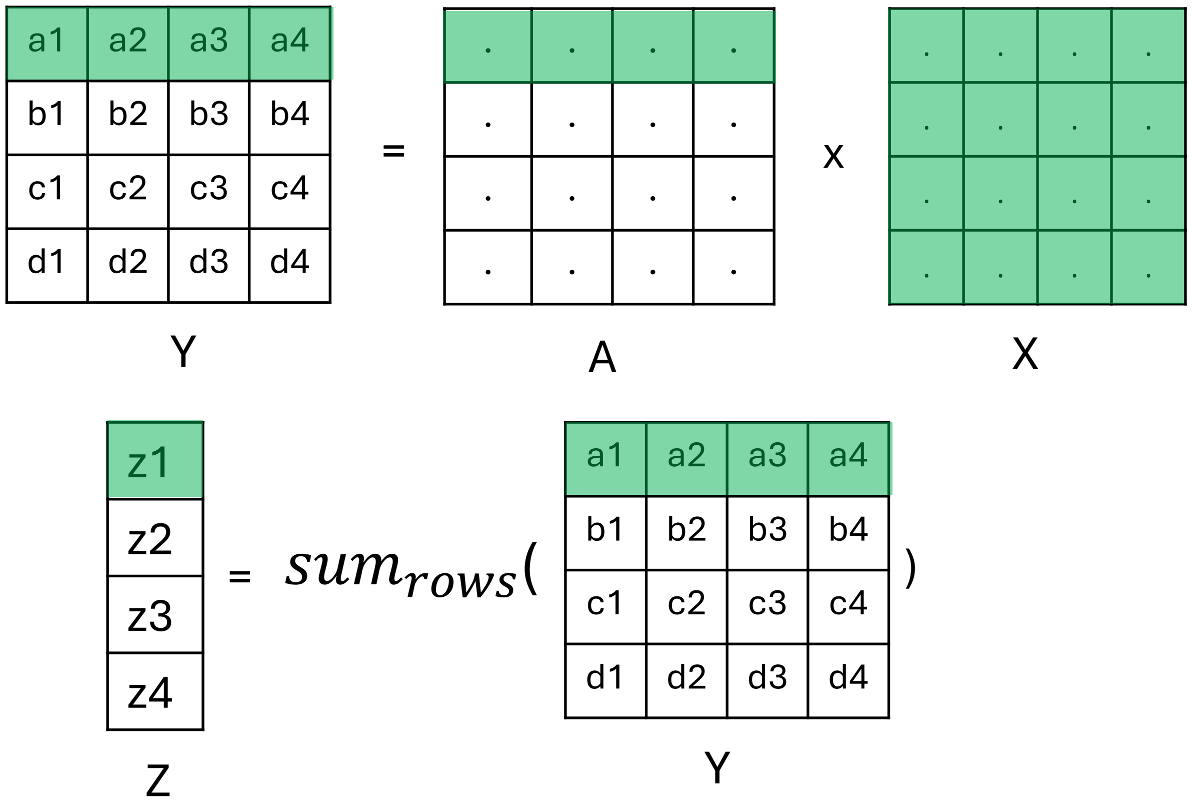


GeMM-VecOp: Fusion Opportunities

- Example:
  - Y = A * X
  - Z = $VecOp_{rows}$(Y)

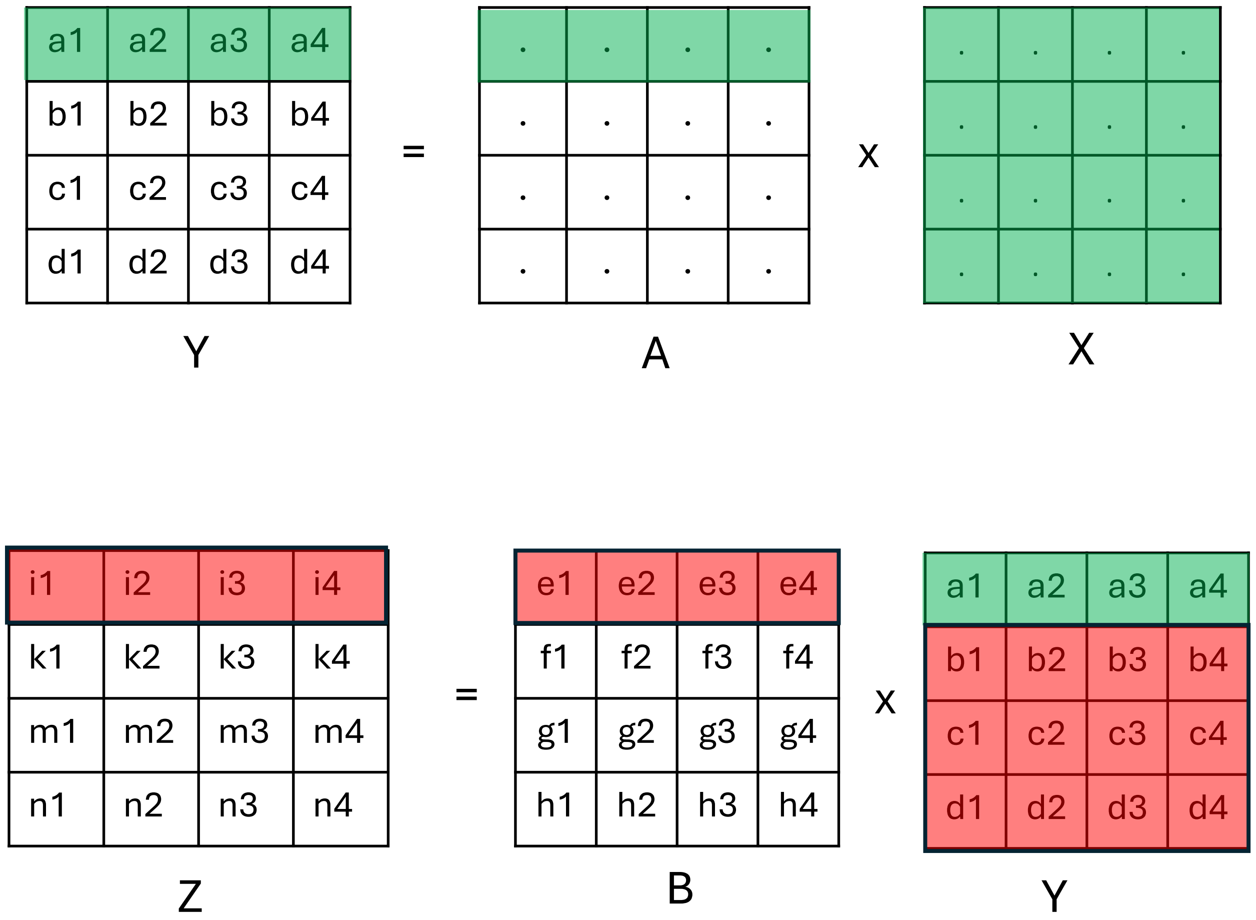
    z1 = sum(a1,..,a4)
    z2 = sum(b1,...,b4)
    ...

- Why should we perform fusion?
  - Enables Reuse of each row.
  - Small fast memory.



GeMM-GeMM: Fusion Opportunities

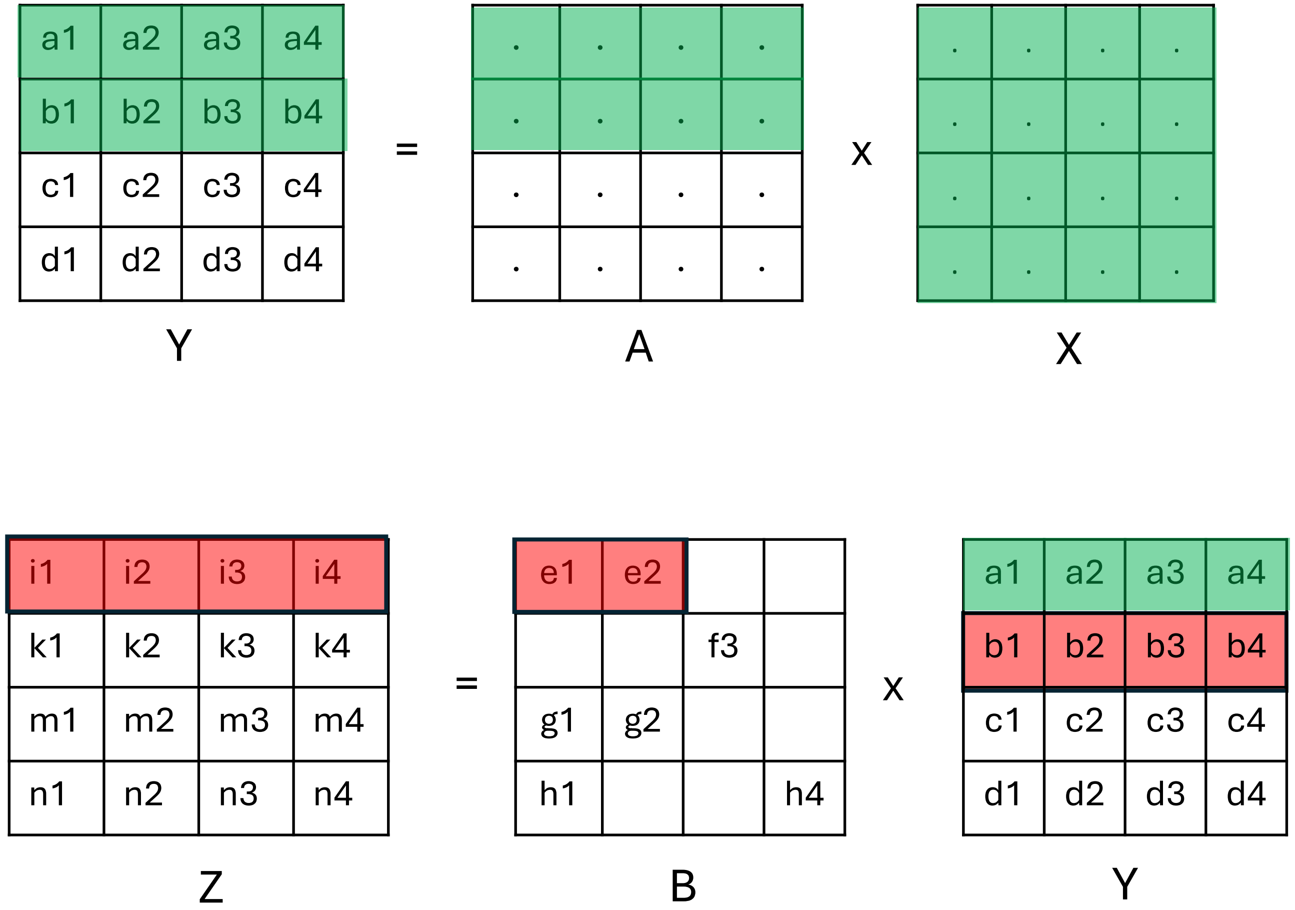
- Example:
  - Y = A * X
  - Z = B * Y
- [i1,...,i4] = [e1,...,e4] * Y
- Need to read all data in the Y matrix -> not able to use fast memory.



GeMM-SpMM: Fusion Opportunities

- Example:
  - Y = A * X
  - Z = B * Y

- Sparsity removes need to some parts of intermediate data.
- Intermediate data has reuse potential.
- Sparsity need to be analyzed before the operations.

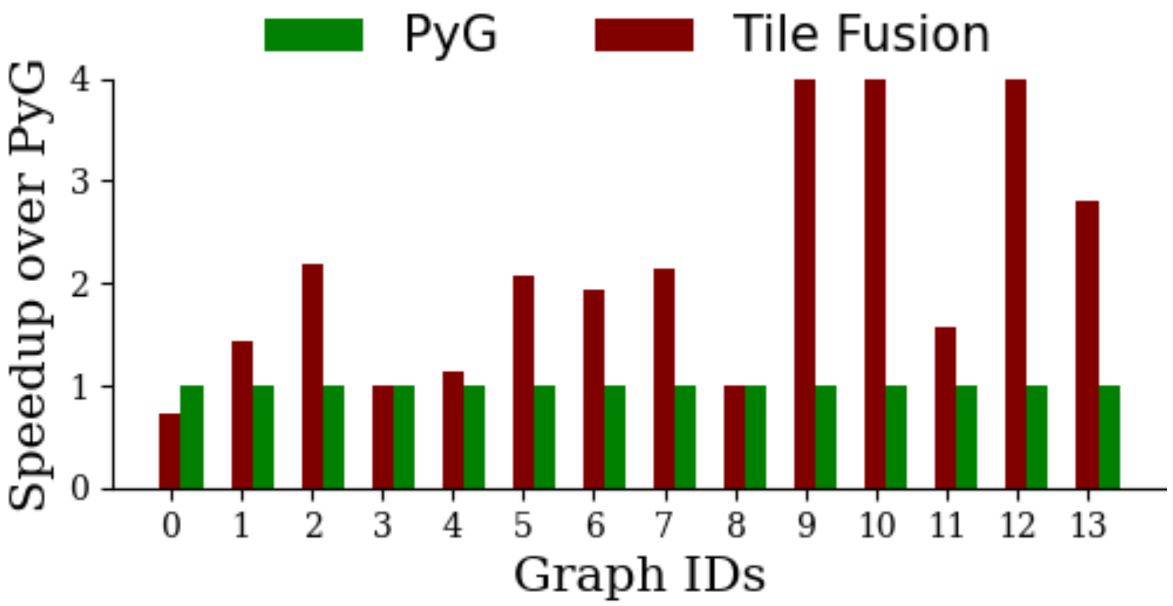


Static Sparsity: Amortizing Cost

- Sparsity analysis cost. Ex.: O(nnz) for GeMM-SpMM
- Scheduling cost
- Amortizing the cost when we have repetitive executions.

End-to-end Results

- Result of applying our methodology to full-batch GCN training(Fusing GeMM-SpMM)
- GeMM: linear transformation
- SpMM: graph aggregation



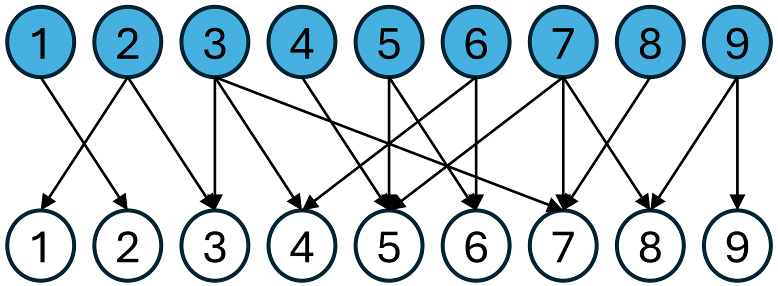
Tile fusion has achieved 2.33 average speedup over PyTorch Geometric (PyG).

Prior Work

Improving Locality in Consecutive Sparse and Dense Matrix Multiplications

GeMM-SpMM DAG

- We create a DAG for representing data dependences to schedule operations.
- Y = B * C
- Z = A * Y



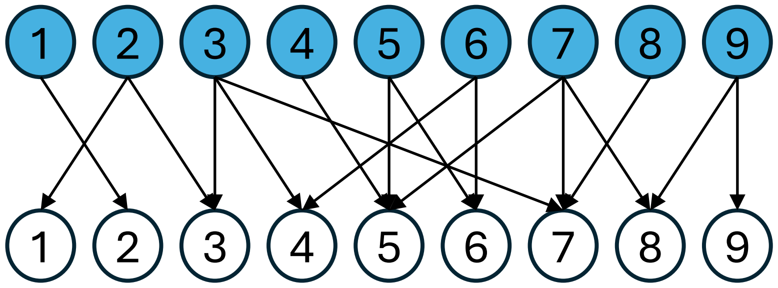
DAG *G*

1		X							
2	X								
3		X	X						
4			X			X			
5				X	X		X		
6					X	X			
7			X				X	X	
8							X		X
9									X

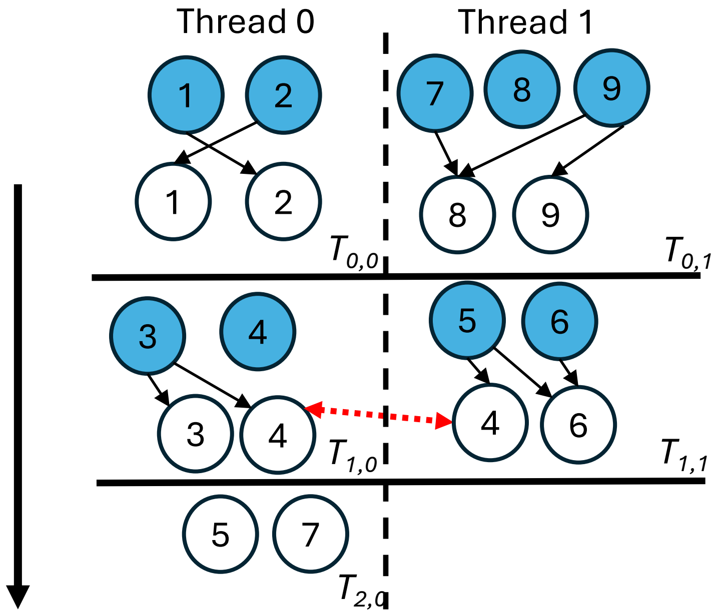
A

Run-time Schedulers: Atomic Tiling

- Fine-grained load balanced tiles
- Atomic instructions
- Idle threads



DAG *G*



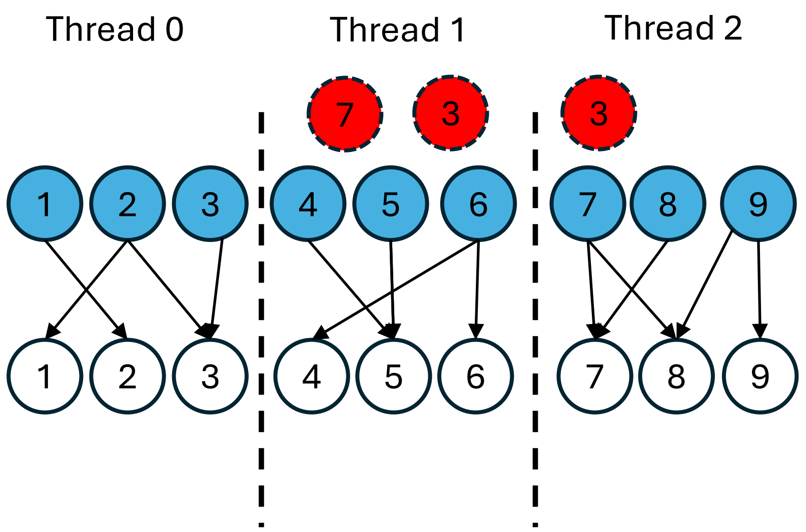
Atomic instruction: Needed
Synchronization barriers: 2
Overlapped computations: 0

Driven by sparse tiling:

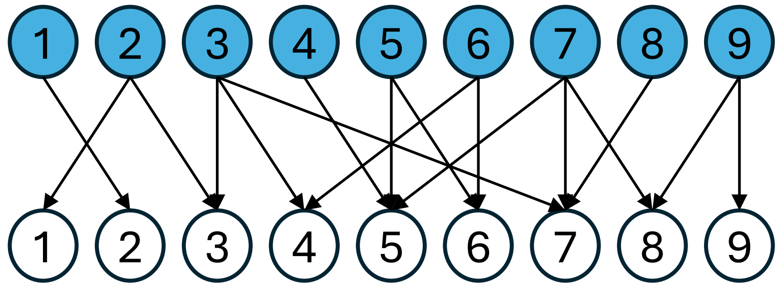
C. D. Krieger *et al.*, "Loop Chaining: A Programming Abstraction for Balancing Locality and Parallelism," *2013 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum*, Cambridge, MA, USA, 2013, pp. 375-384, doi: 10.1109/IPDPSW.2013.68.

Run-time Schedulers: Overlapped Tiling

- No synchronization barrier
- Redundant computations



Atomic instruction: Not Needed
Synchronization barriers: 0
Overlapped computations: 3



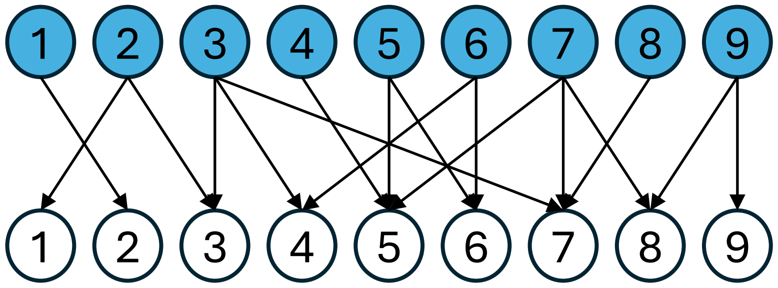
DAG *G*

Driven by communication avoiding:

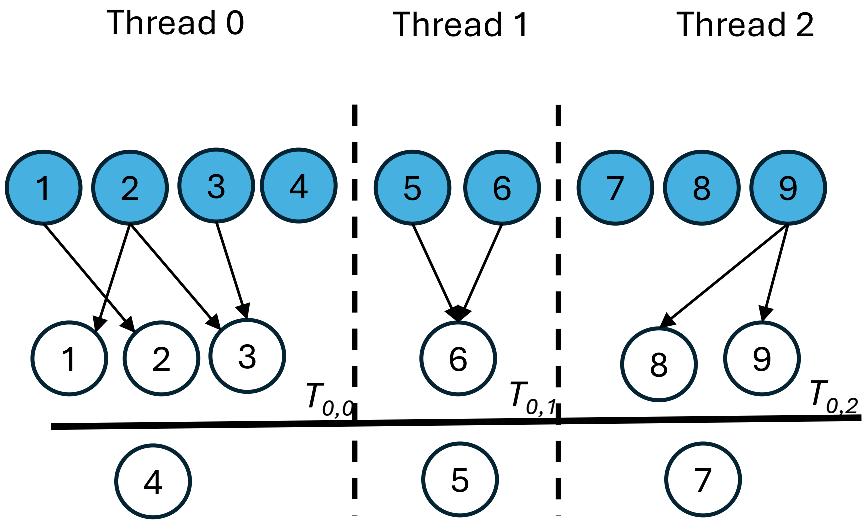
JamesDemmel,MarkHoemmen,MarghoobMohiyuddin,andKather- ine Yelick. 2008. Avoiding communication in sparse matrix computa- tions. In 2008 IEEE International Symposium on Parallel and Distributed Processing. IEEE, 1–12.

Run-time Schedulers: Tile Fusion

- No atomic Instruction
- No redundant operations
- Load balanced variable tile sizes
- Synchronization barrier



DAG *G*

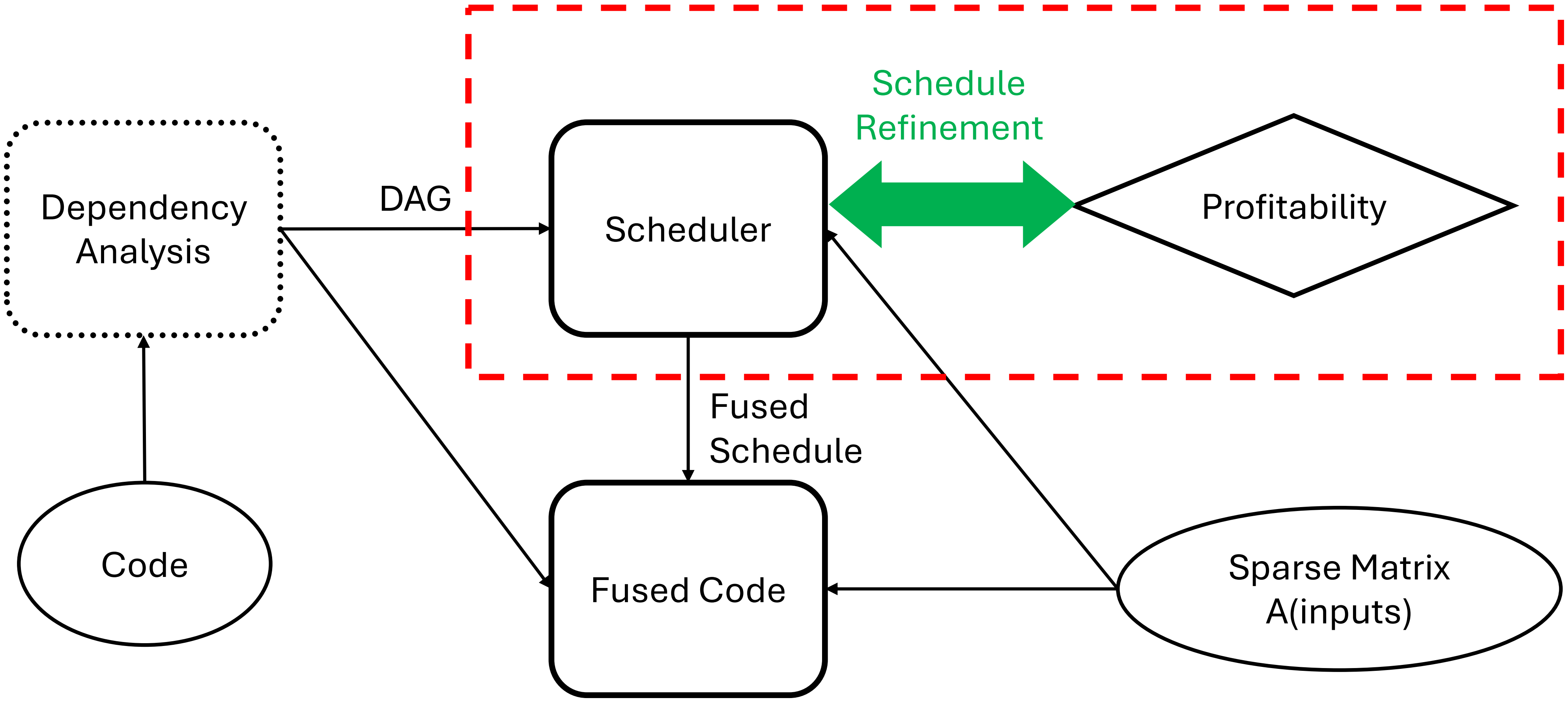


Atomic instruction: Not Needed
Synchronization barriers: 1
Overlapped computations: 0

Methodology

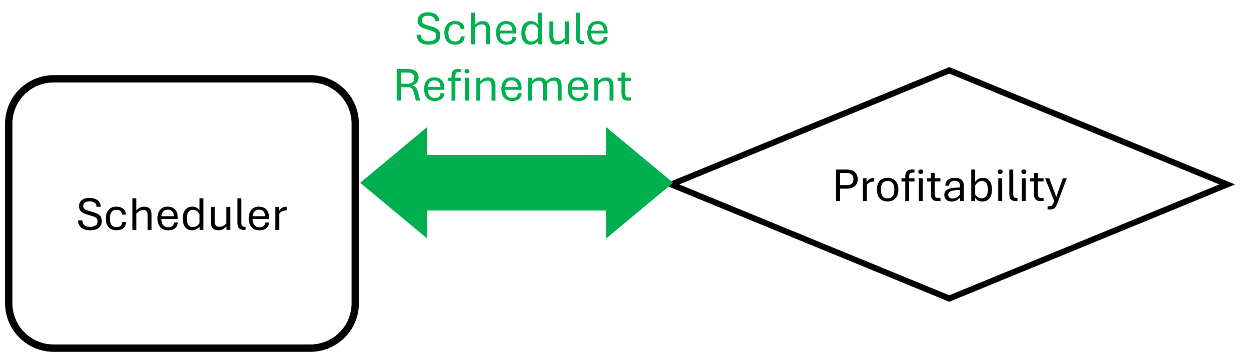
Improving Locality in Consecutive Sparse and Dense Matrix Multiplications

Tile Fusion

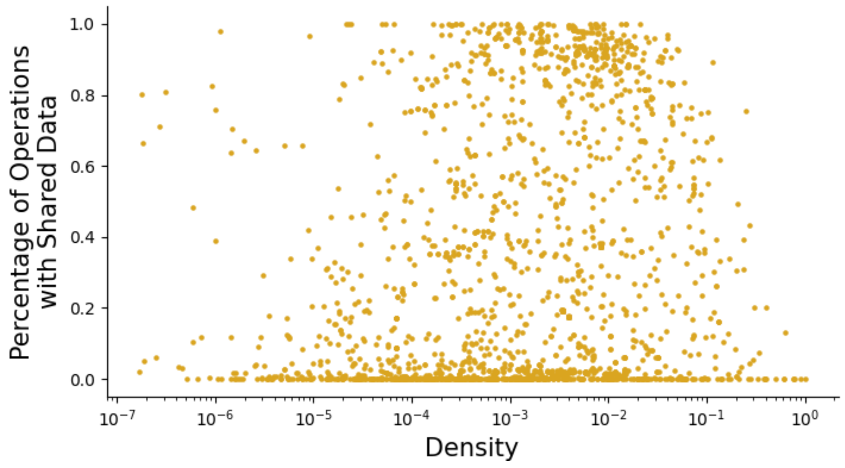


Schedule
Refinement
Dependency
Analysis
DAG
Scheduler
Profitability
Fused
Schedule
Code
Fused Code
Sparse Matrix
A(inputs)

Tile Fusion

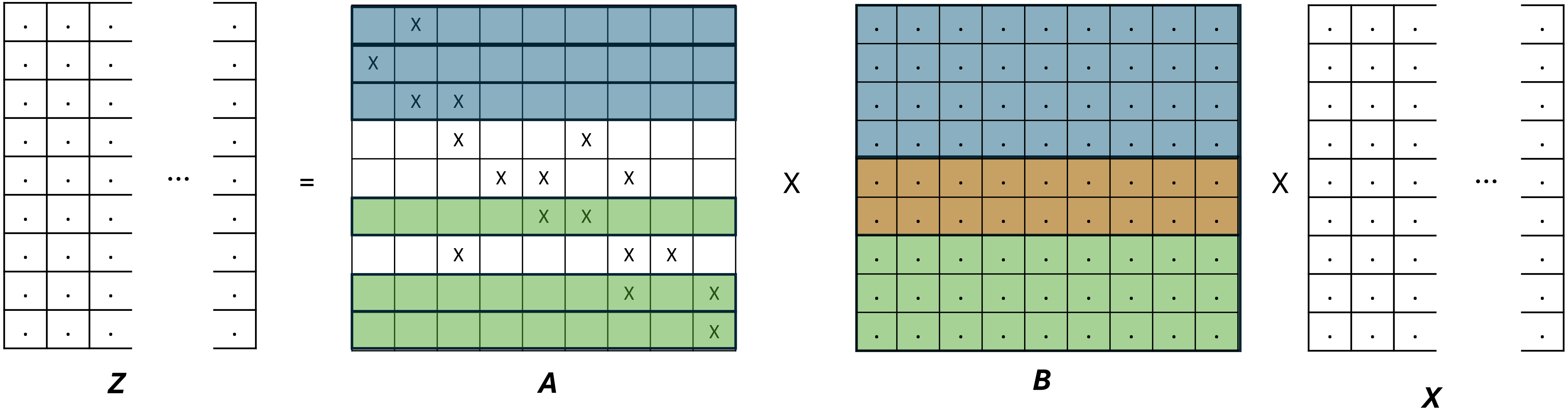


- Coarse-grained tiles
- Fused ratio
- 2893 suit sparse matrices
  - 34% fused ratio on average for coarse tiles.
  - Coarse tile: tile size = 2048



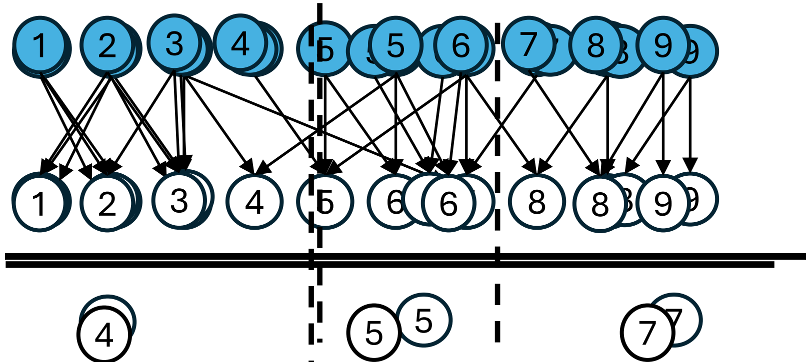
$$fused\ ratio = \frac{Number\ of\ fused\ computations}{Number\ of\ all\ computations}$$

Scheduler Example: GeMM-SpMM



Step 1: DAG

Step 2: Coarse-grain Tile Fusion



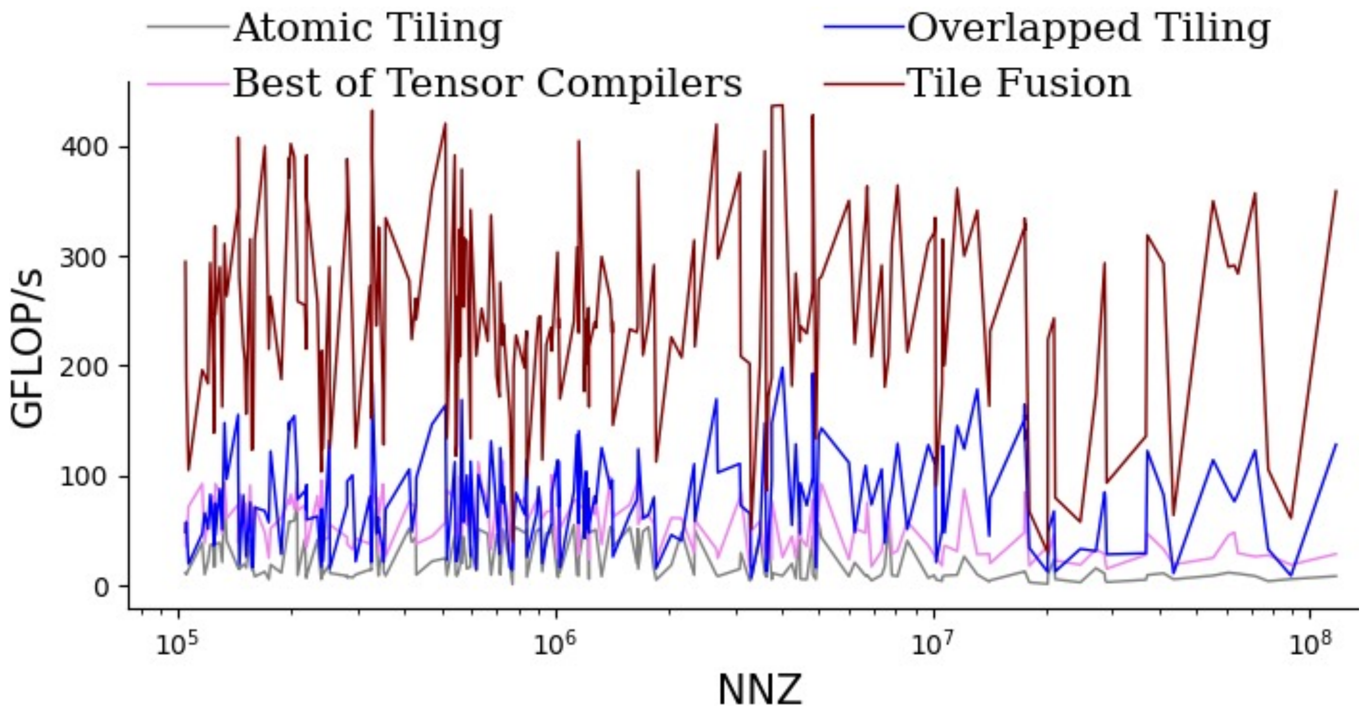
Experimental Results

Improving Locality in Consecutive Sparse and Dense Matrix Multiplications

Experiment Setup

- Intel Icelake architecture with 40 cores
- Single operation experiments(GeMM-SpMM, SpMM-SpMM)
  - 230 matrices from suitsparse collection
  - Compared with prior works and best of tensor compilers(LNR, TACO)
  - Compared with Intel Math Kernel Library (MKL)
- End-to-end experiment
  - 2 Layer GCN full batch training with 100 epochs
  - For chosen GNN benchmark graphs
  - Compared with pytorch_geometric

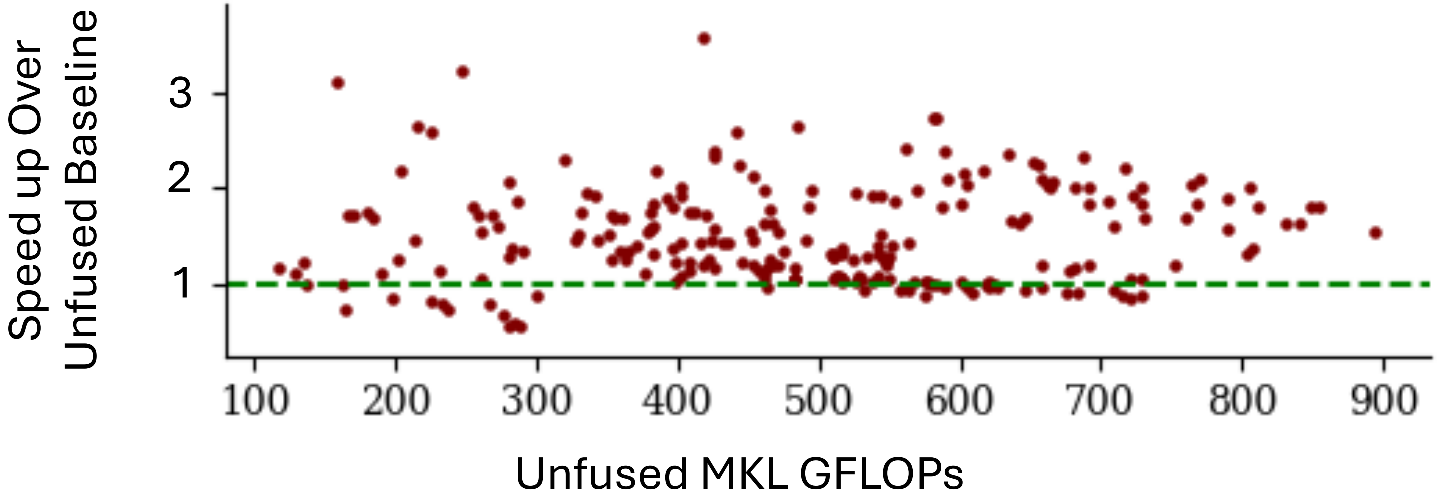
Results: Single Operation vs Fused Implementations



Tile fusion has achieved 3.5 average speedup over best of fused implementations.

Results: Single Operation vs Unfused MKL

- GeMM-SpMM



Tile fusion has achieved 1.42 average speedup over unfused MKL.

Application: GCN training

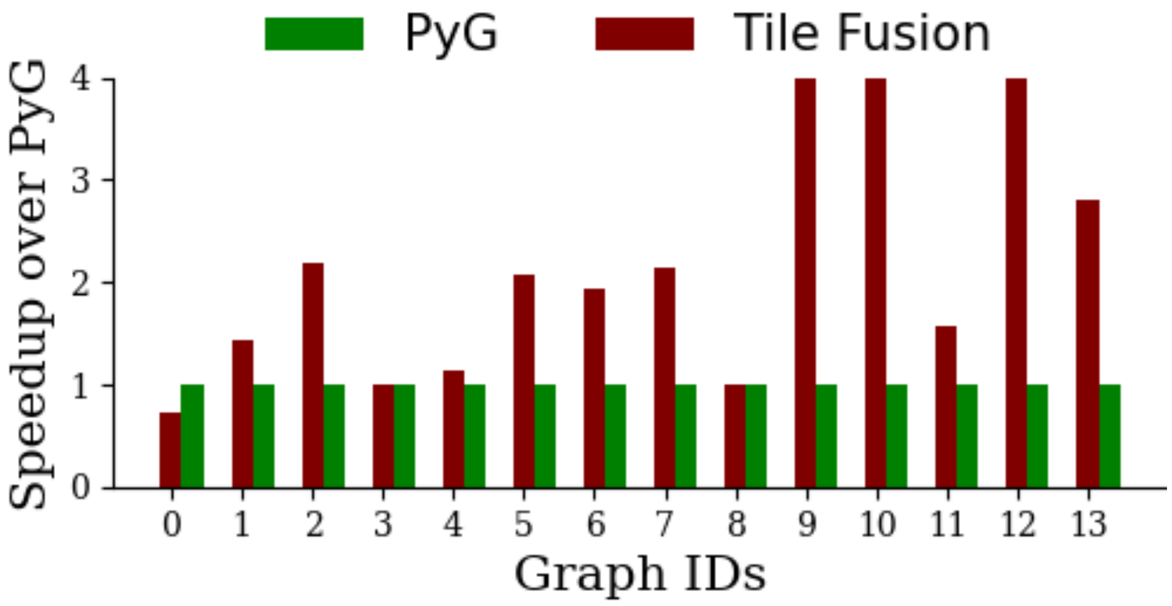
- Aimed for Sparse matrix multiplications when sparsity is static for several executions of a kernel.
- Example: Graph Convolutional Networks Training

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right)$$

- When training a graph this can be seen as Â * (H * W) which can be interpreted as two consecutive tensor contraction kernels:

  GeMM-SpMM

End-to-end Results



Id	Name	Vertices	Edges
0	Amazon2k [27]	303,296	586,902
1	Coauthor CS [33]	18,333	163,788
2	Coauthor Physics [33]	34,493	495,924
3	Cora [5]	19,793	63,421
4	DeezerEurope [32]	28,281	185,504
5	Facebook [31]	22,470	342,004
6	Flickr [41]	89,250	899,756
7	Github [31]	37,700	578,006
8	OGBN Arxiv [17]	232,965	114,615,892
9	OGBN products [17]	2,449,029	123,718,152
10	OGBN proteins [17]	132,534	79,122,504
11	PPI [43]	56,944	818,716
12	Reddit [41]	232,965	23,213,838
13	Yelp [41]	716,847	13,954,819

Tile fusion has achieved 2.33 average speedup over PyTorch Geometric (PyG).