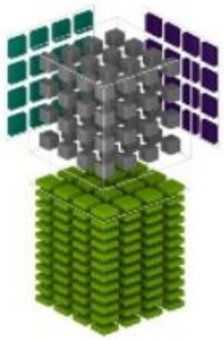
Reordering sparse matrices
for fast multiplication on tensor architectures

Github repository:

https://github.com/LACSFUB/SPARTA

Paolo Sylos Labini
Faculty of Computer Science,
Free University of Bozen-Bolzano

Motivation



Tensor cores are good at multiplying dense arrays

But what about **sparse matrices** multiplication**?**

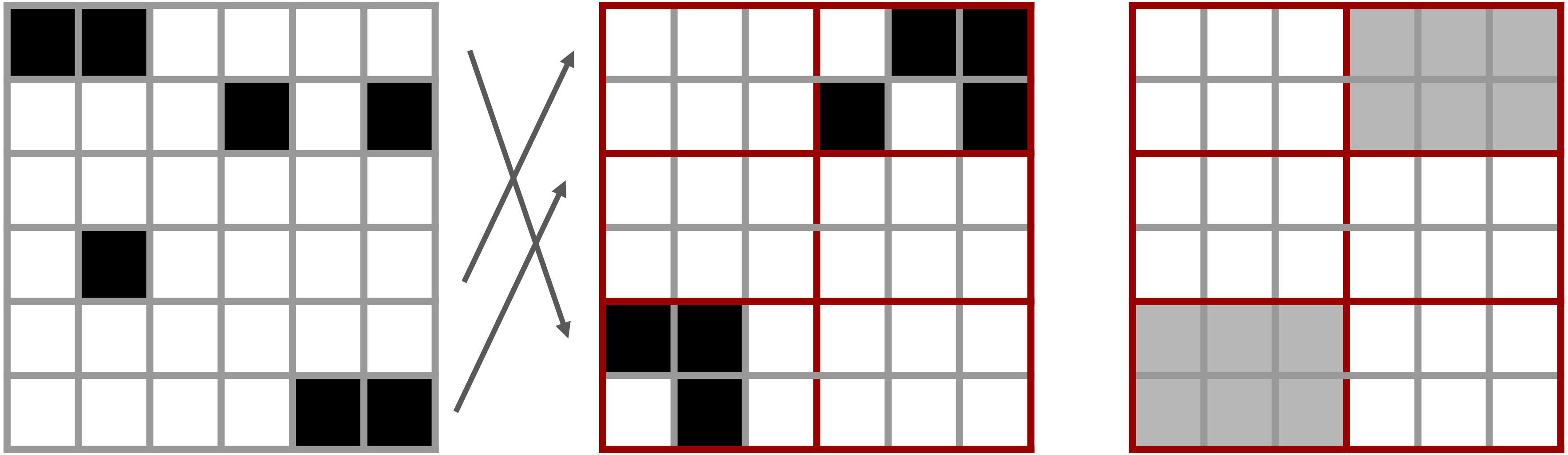
Should we use sparse-specific data structures (CSR) and algorithms (cusparse)?

We can’t benefit of the tensor architecture

Should we treat the sparse matrix as a dense one and use tensor architectures?

We end up multiplying a lot of zeros

Our solution



ORIGINAL

REORDERED

BLOCKED

Applications

Reordering takes time

If the same matrix is multiplied repeatedly, we can absorb the cost

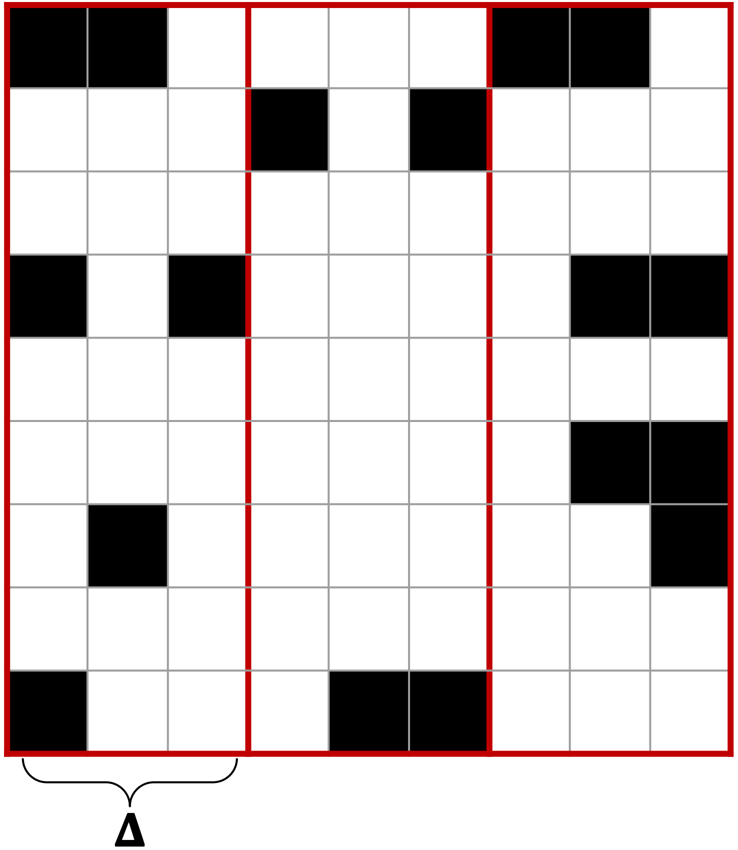
- graphs
- neural networks
- matrix collections

The reordering algorithm

Find similar rows and group them together in blocks

1) Partitioning the columns
2) Grouping by hash
3) Grouping by similarity

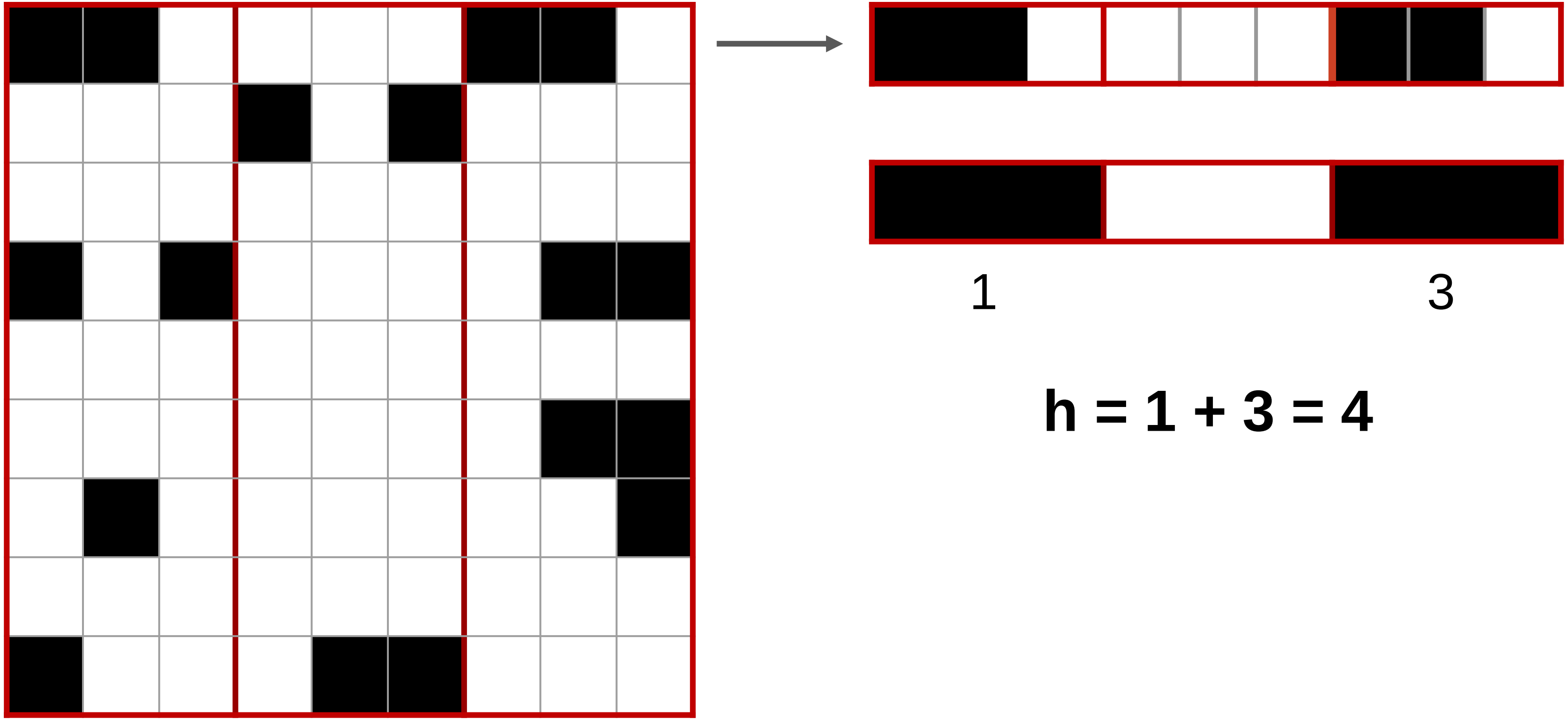
The reordering algorithm – partitioning the columns



Choice of **Δ** may depend on

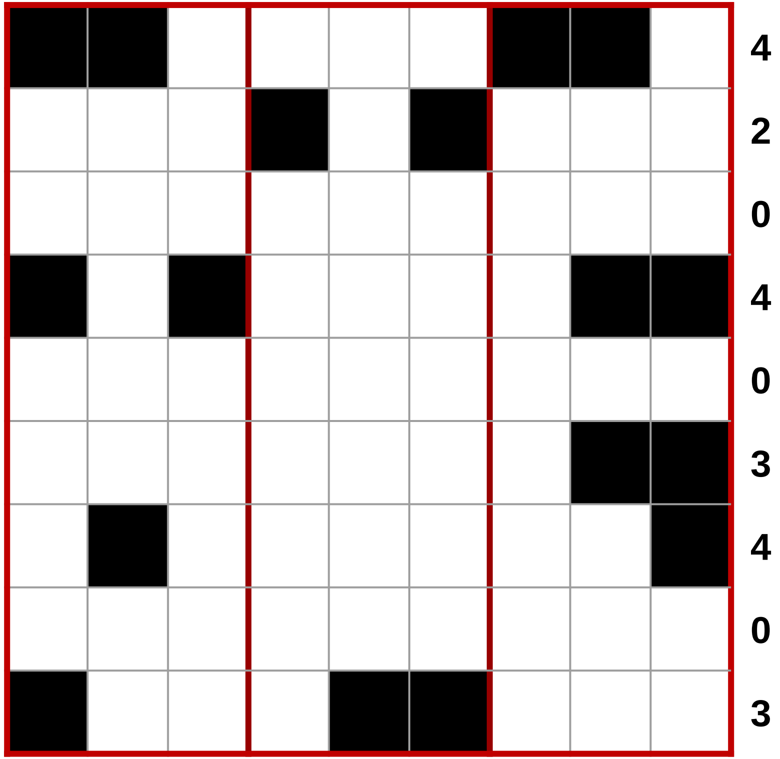
- architecture
- previous knowledge of the matrix
- ...

The reordering algorithm - hashing the rows

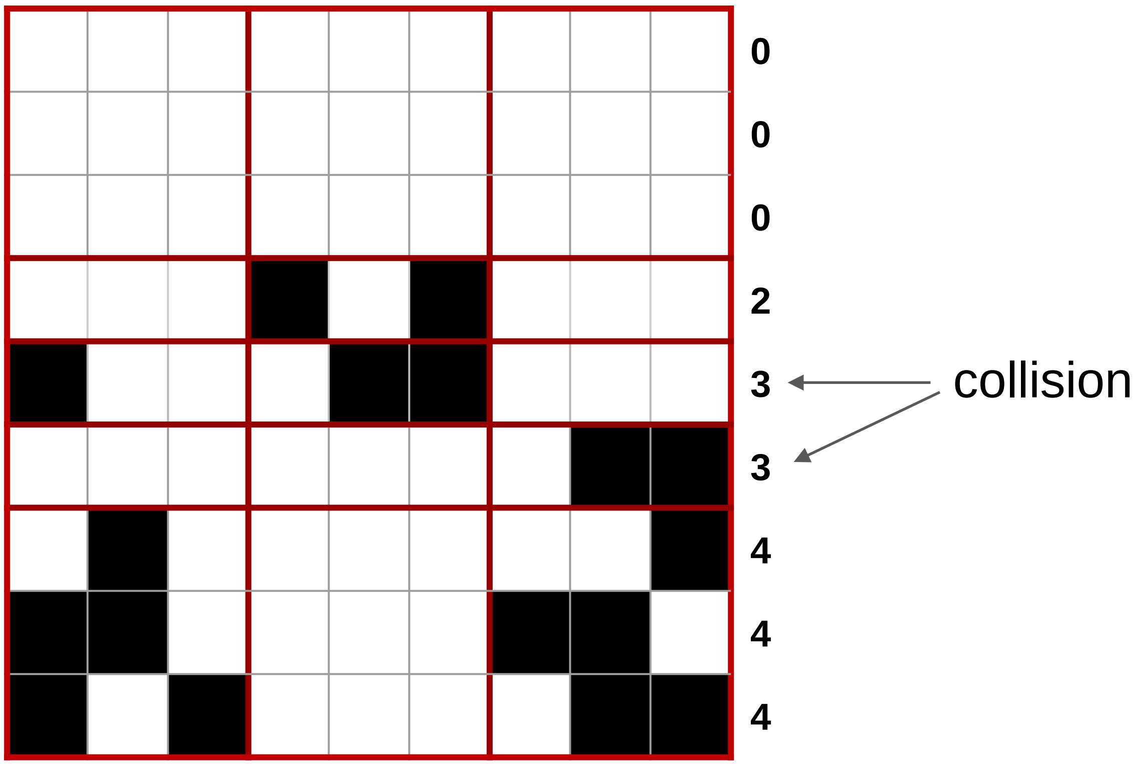


$$h = 1 + 3 = 4$$

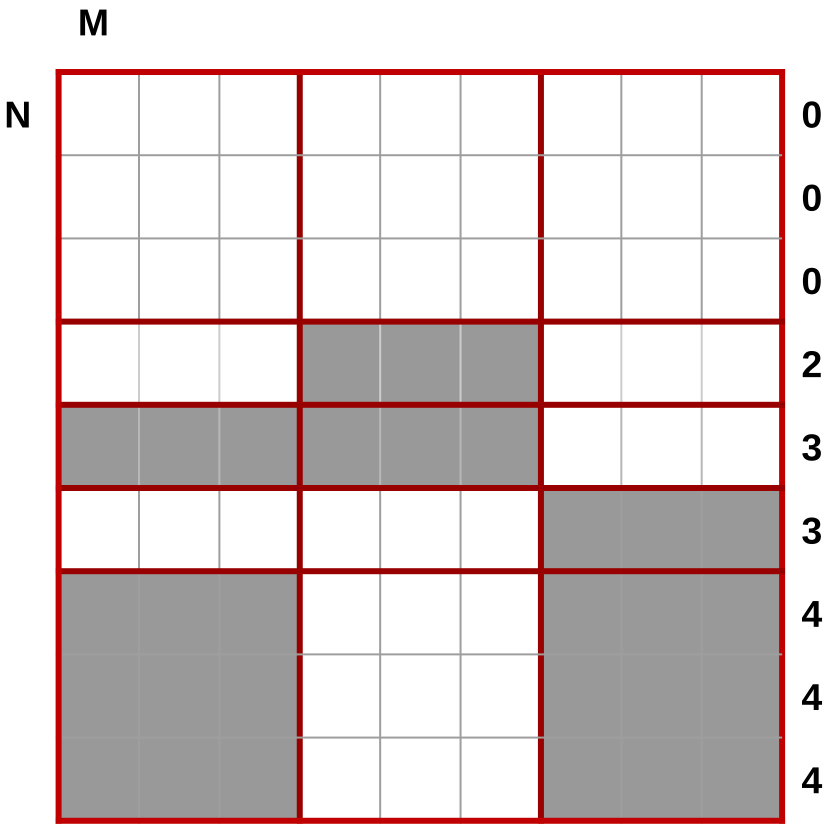
The reordering algorithm - hashing the rows



The reordering algorithm – grouping by hash



The reordering algorithm - grouping by hash

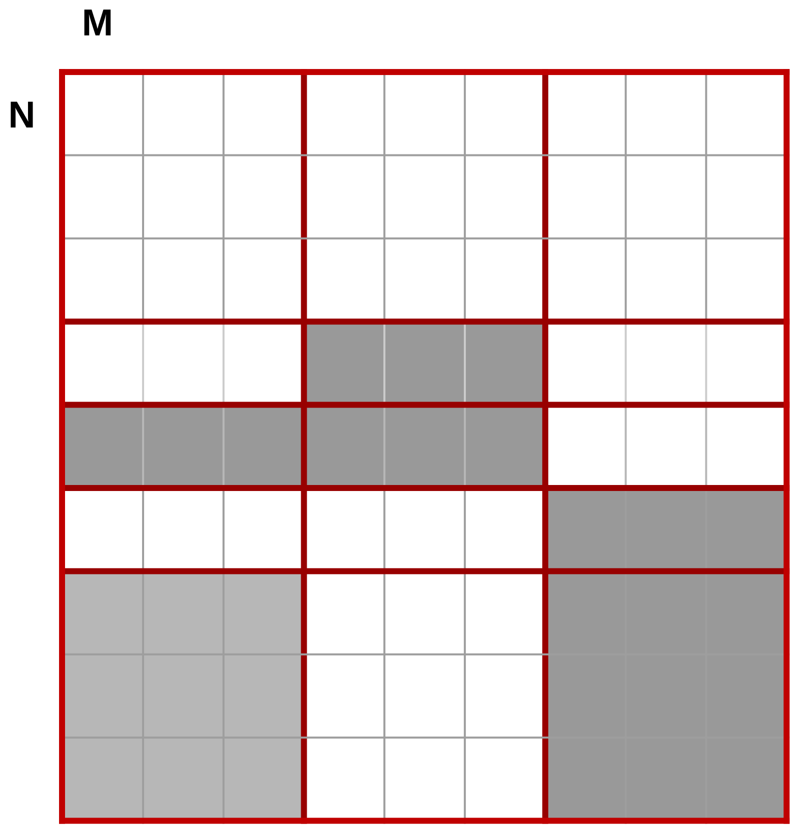


1) **hashing**: **O(NxM)**
   for CSR matrix with K entries: O(K)

1) **sorting: O(N log N)**

1) **grouping:** approx. NxM

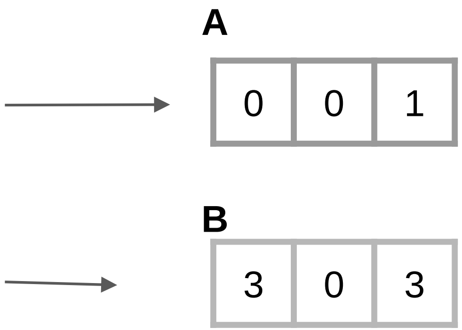
The reordering algorithm - grouping by cosine



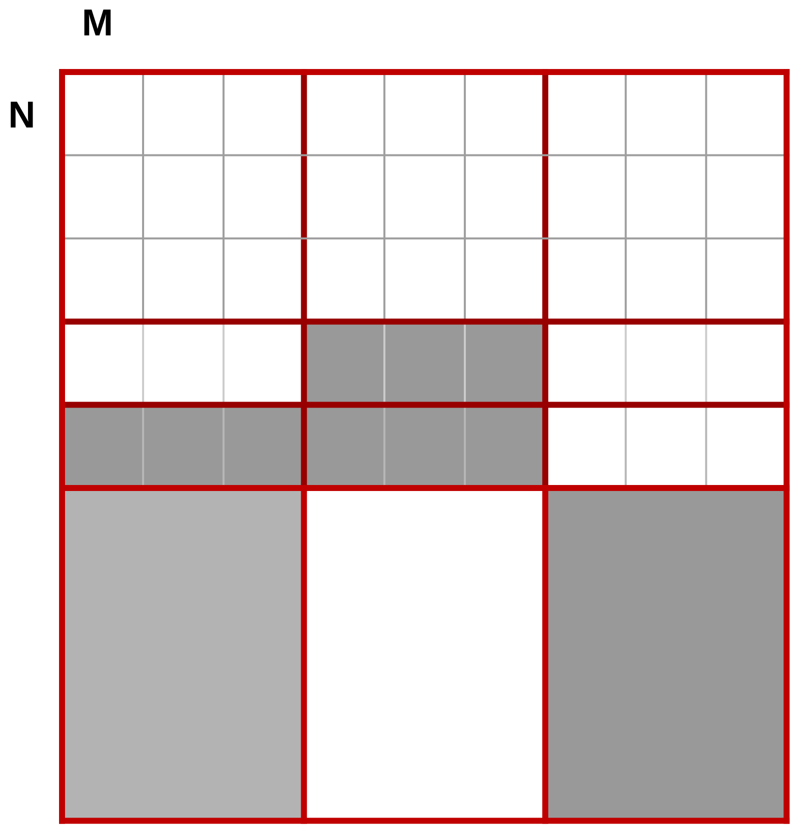
For each pair of groups A, B

if $\cos \theta(A,B) \geq \varepsilon$

merge A and B



The reordering algorithm - grouping by cosine

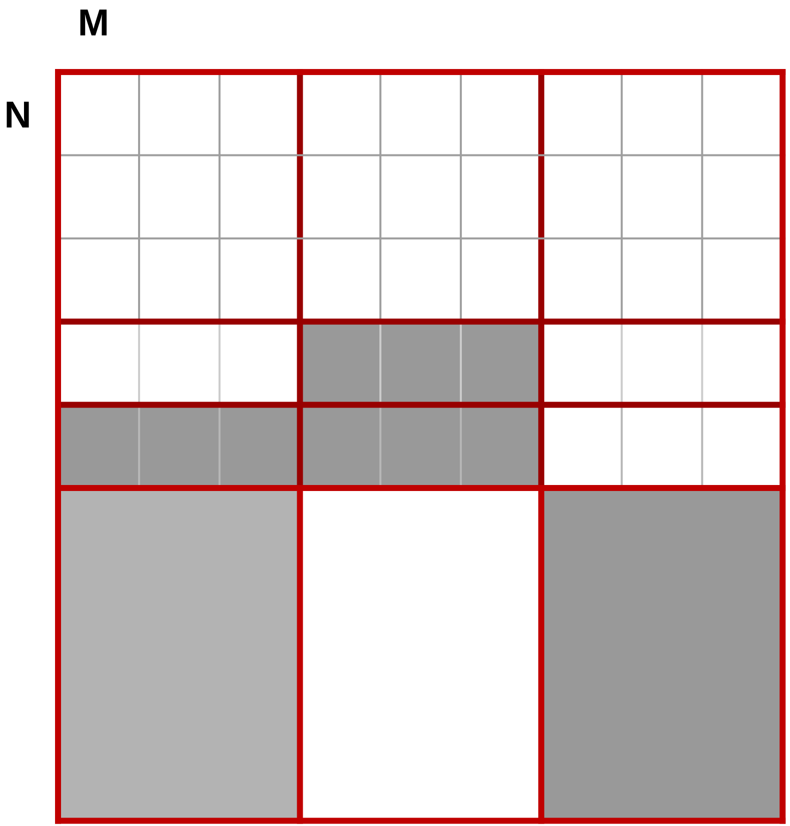


For each pair of groups A, B

if $\cos \theta(A,B) \geq \varepsilon$

merge A and B

The reordering algorithm - grouping by cosine



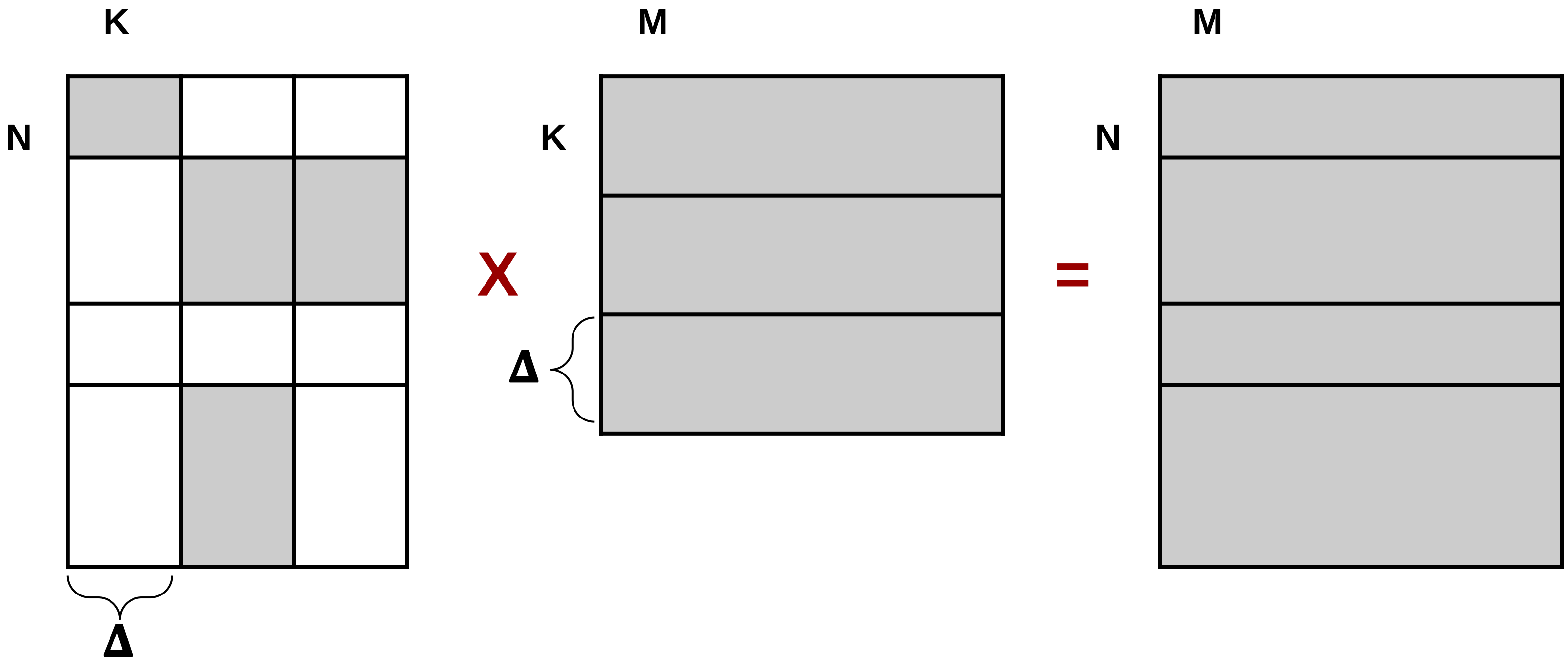
O(N*N*M)

But with some optimization:

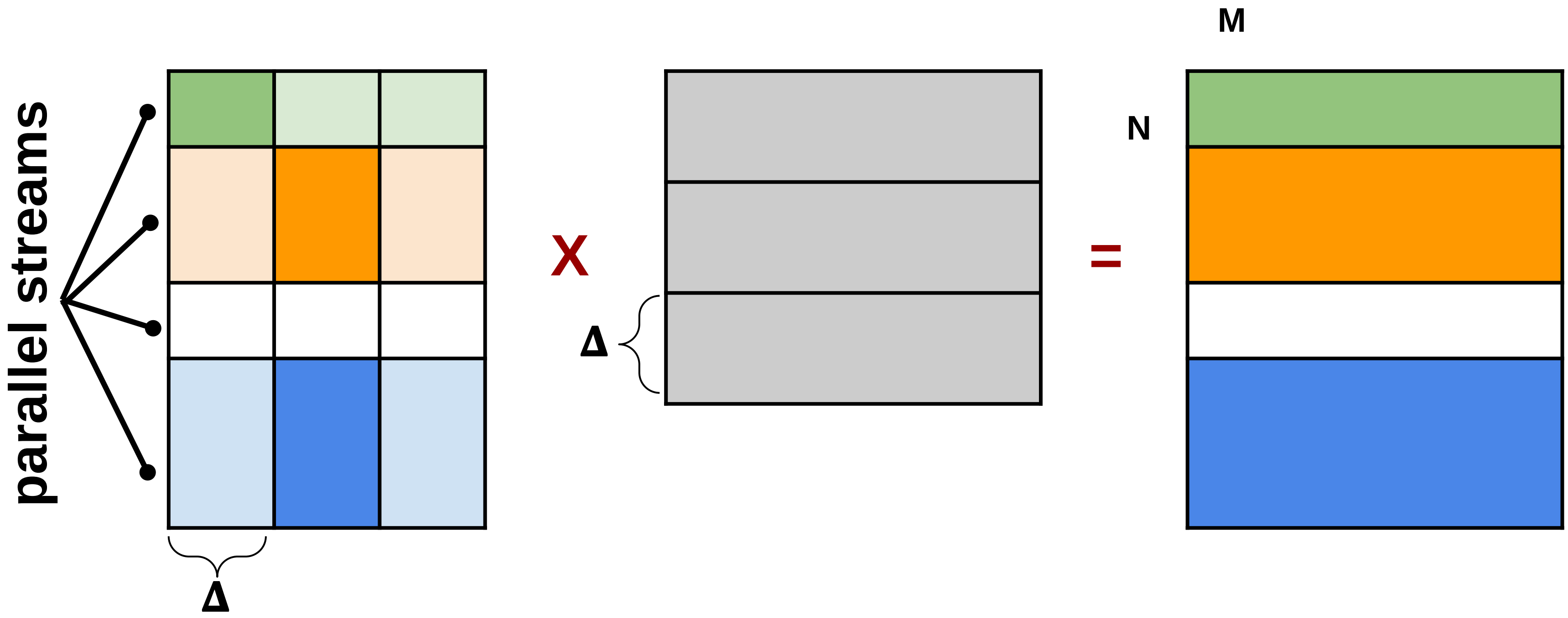
- Only groups are compared, not rows
- Only patterns are compared, not entries

Still, too expensive!

Parallel multiplication



Parallel multiplication



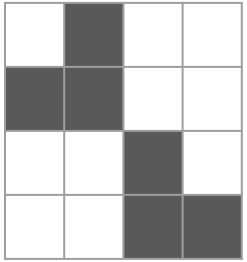
Reorder - experiments with synthetic matrices

Synthetic block matrix

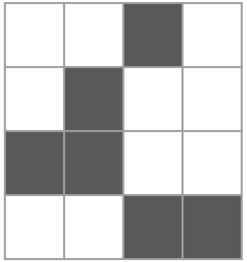
$\Delta = 2$

$\mu = 75\%$

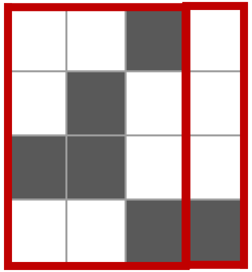
$\eta = 50\%$



Scrambled



Partitioned



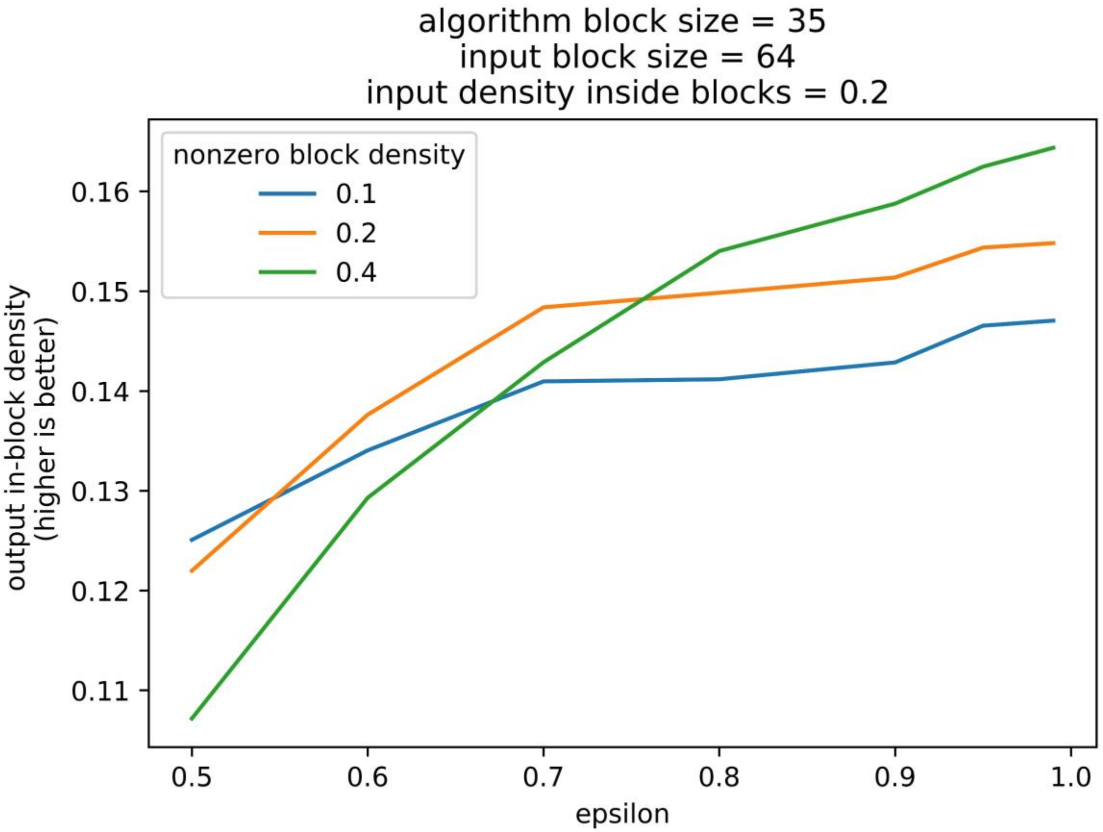
$\Delta' = 3$

Δ = original block size

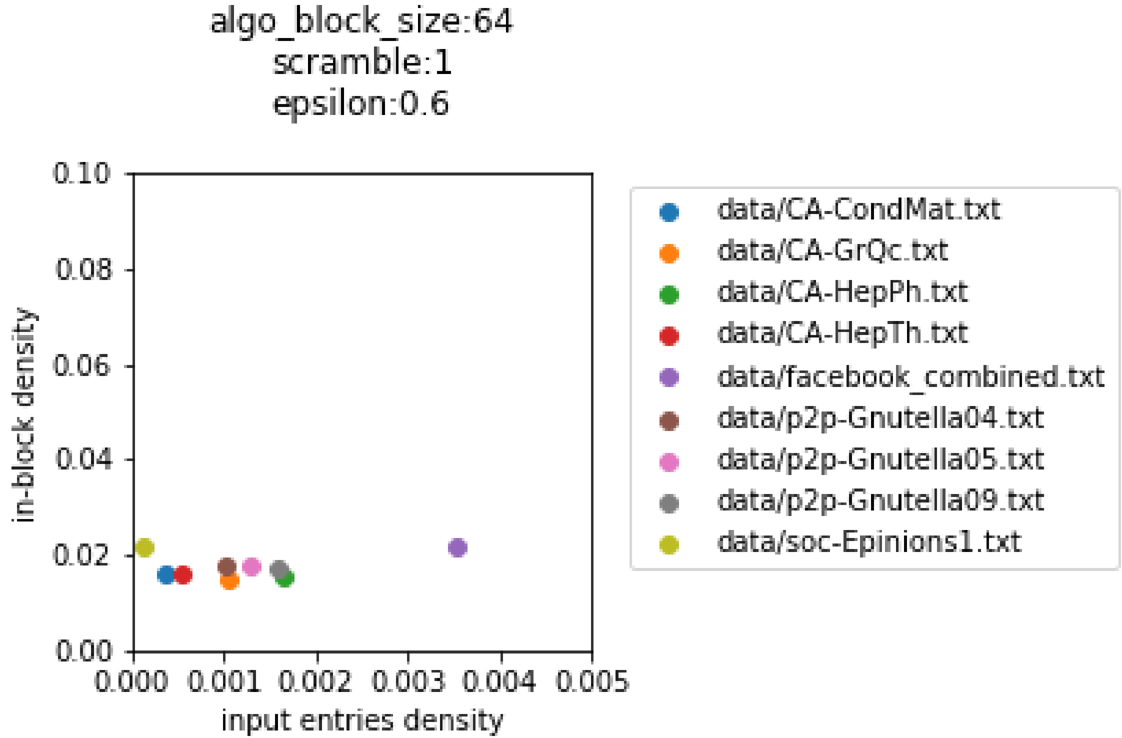
μ = original in-block density

η = original nonzero block density

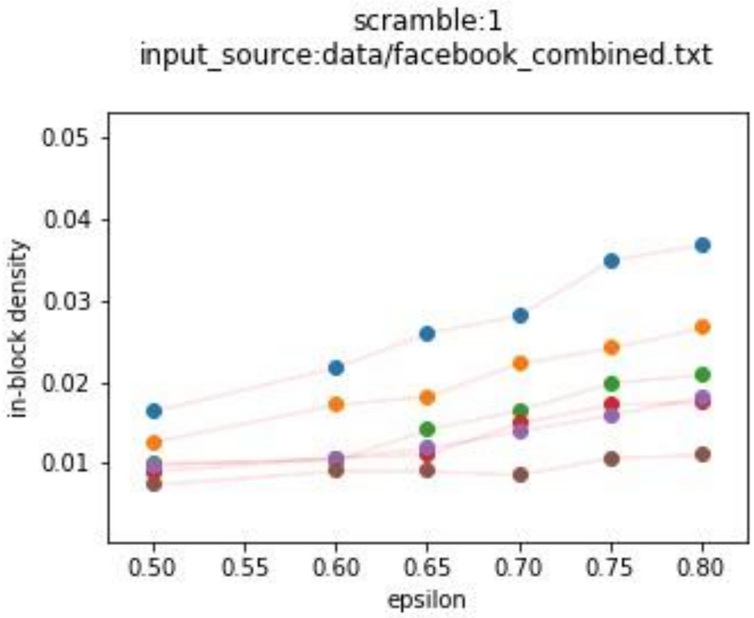
Reorder - experiments with synthetic matrices

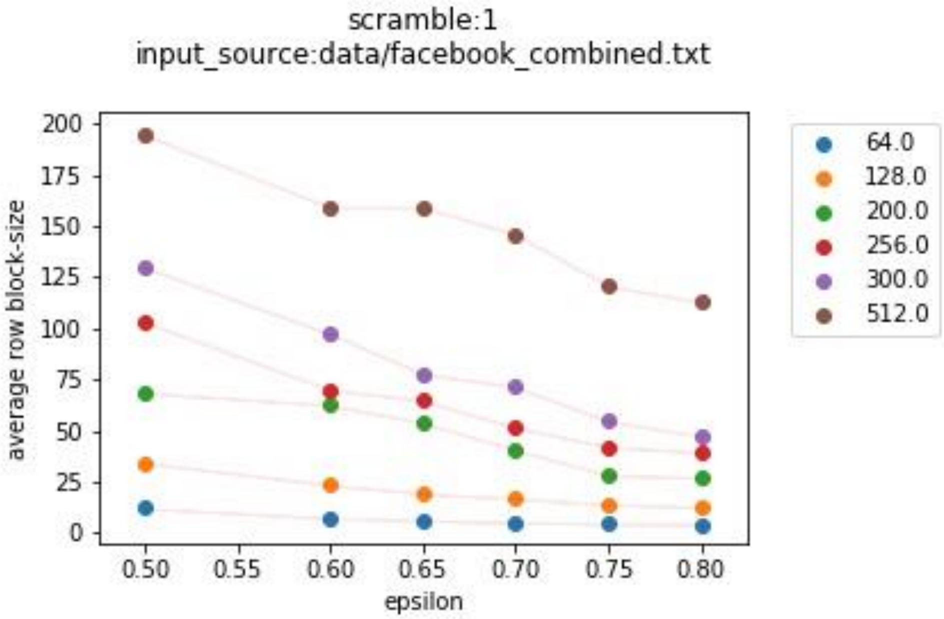


Reorder - experiments with real matrices

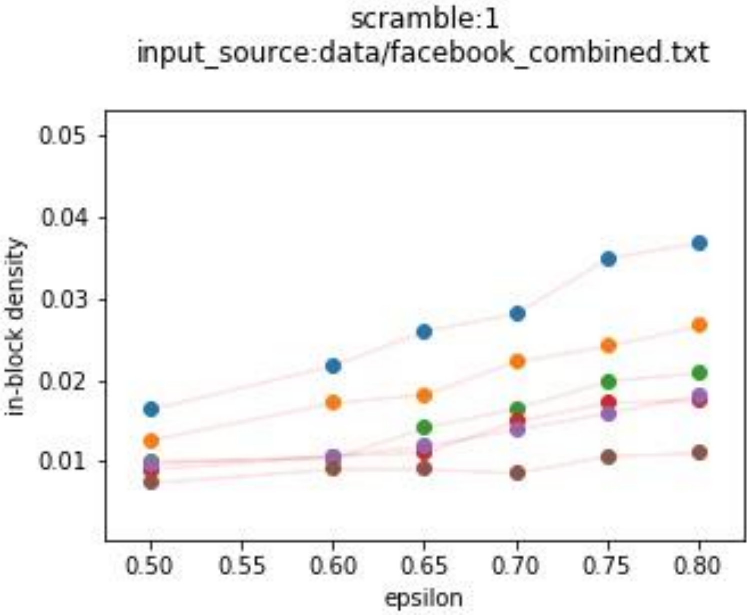


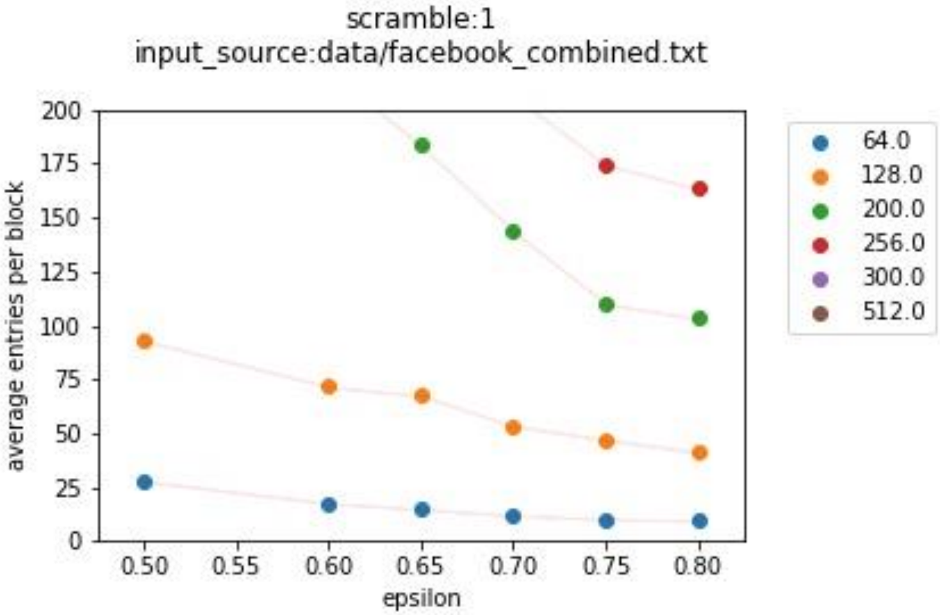
Reorder - experiments with real matrices



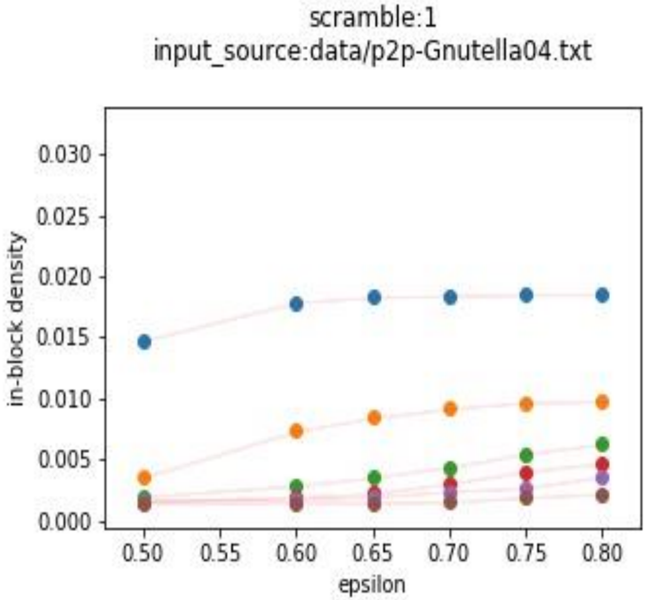


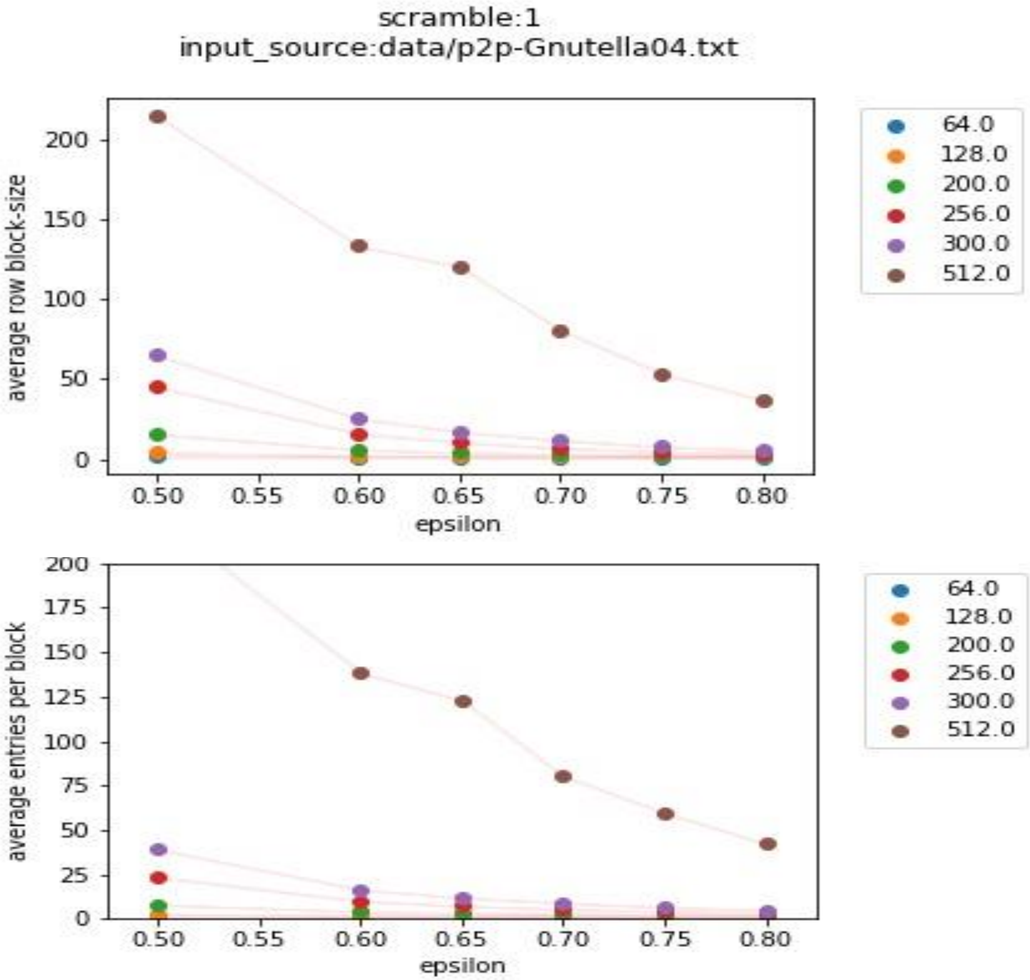
Reorder - experiments with real matrices



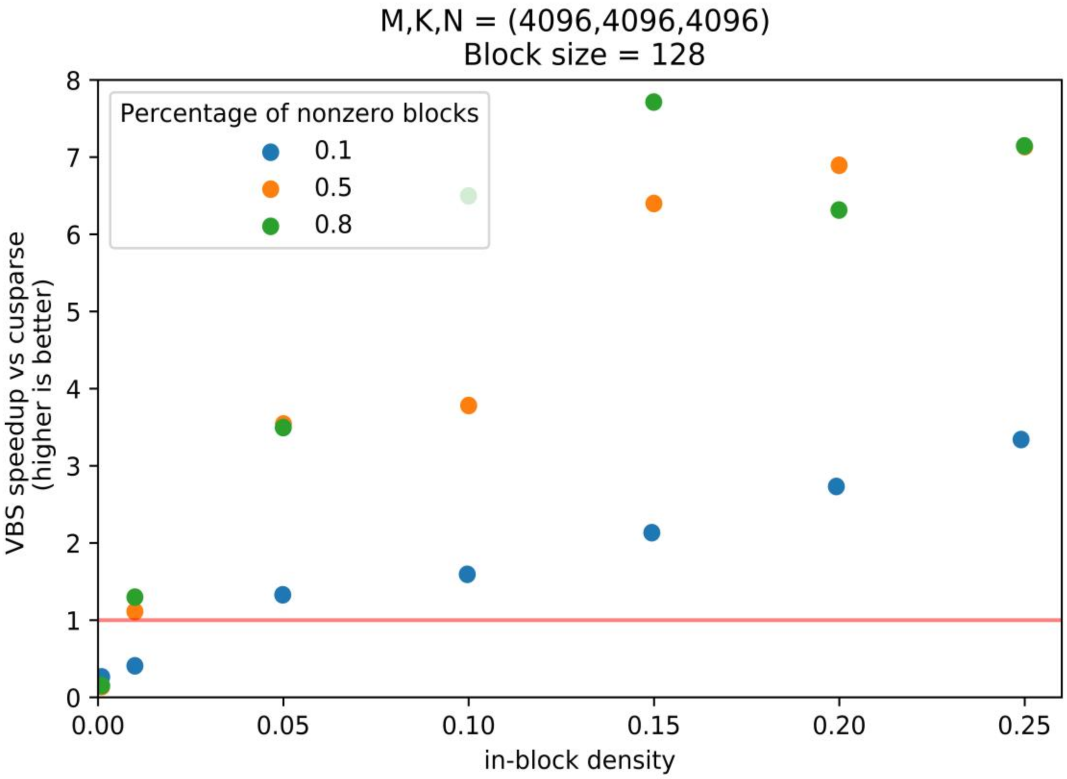


Reorder

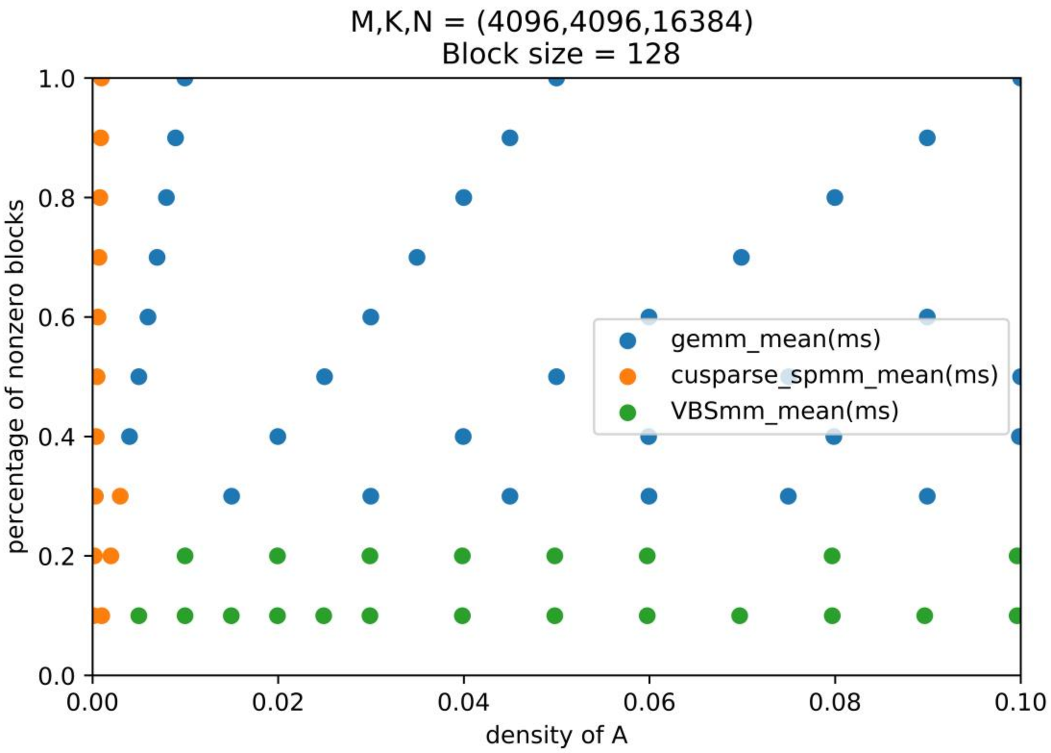

scramble:1
input_source:data/p2p-Gnutella04.txt
in-block density
epsilon
0.000
0.005
0.010
0.015
0.020
0.025
0.030
0.50
0.55
0.60
0.65
0.70
0.75
0.80


scramble:1
input_source:data/p2p-Gnutella04.txt
average row block-size
epsilon
0
50
100
150
200
0.50
0.55
0.60
0.65
0.70
0.75
0.80
64.0
128.0
200.0
256.0
300.0
512.0
average entries per block
epsilon
0
25
50
75
100
125
150
175
200
64.0
128.0
200.0
256.0
300.0
512.0

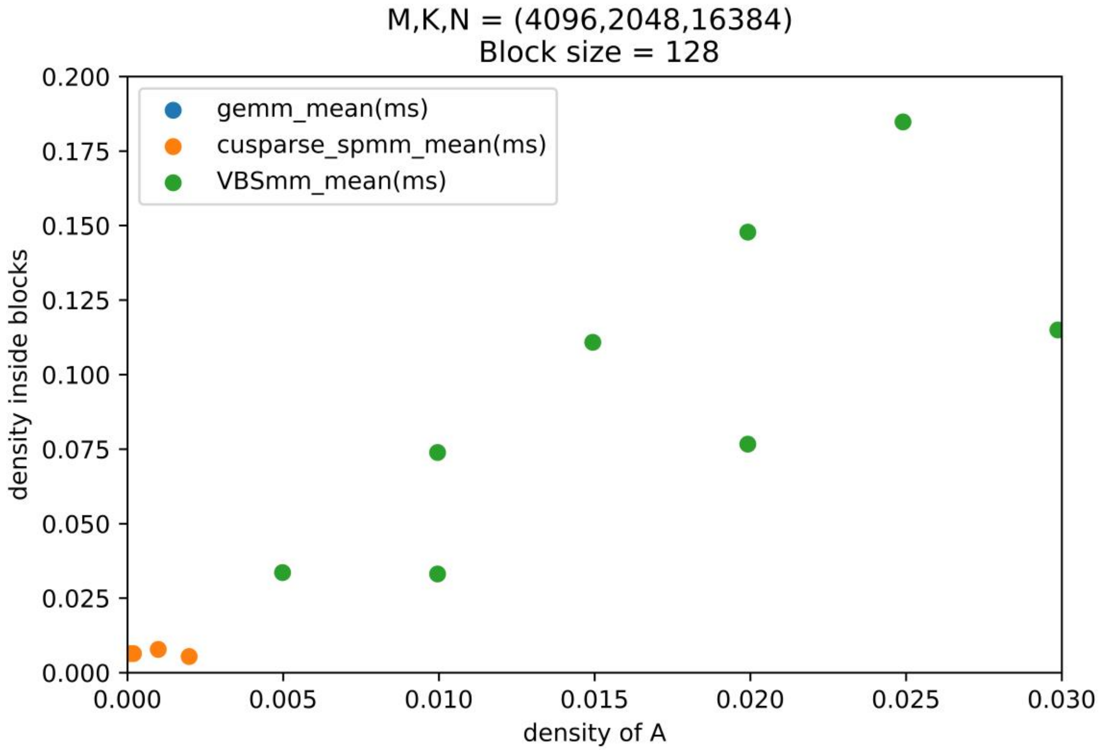
Multiplication - experiments



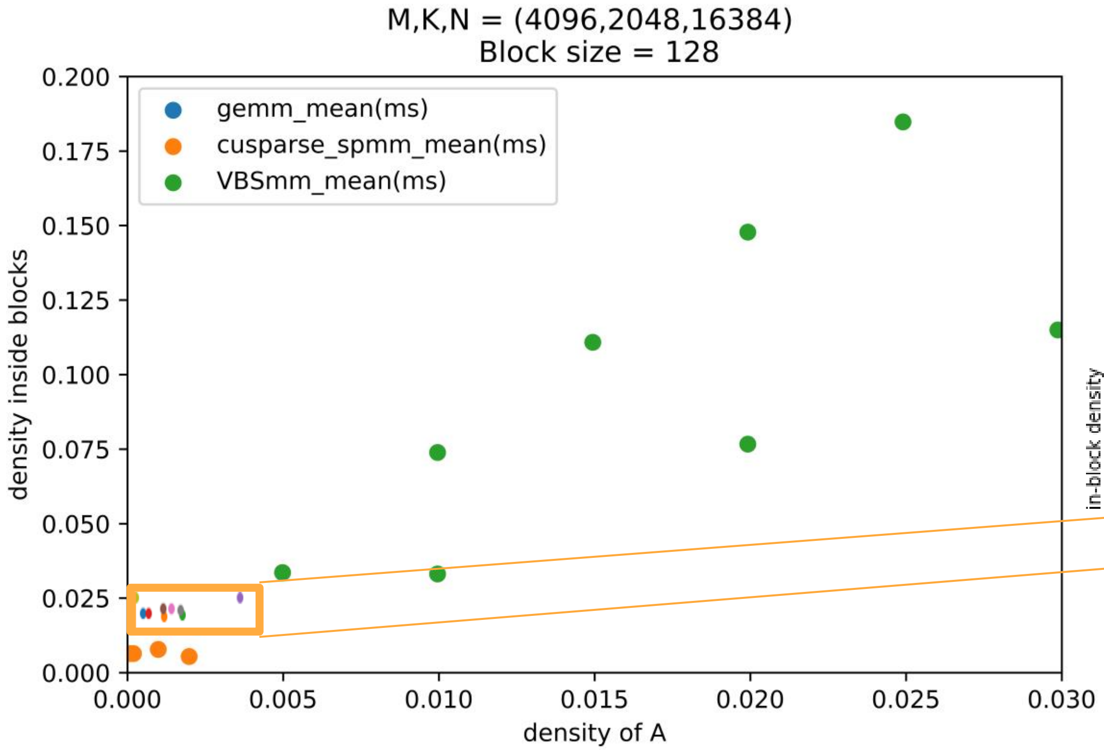
Multiplication - experiments

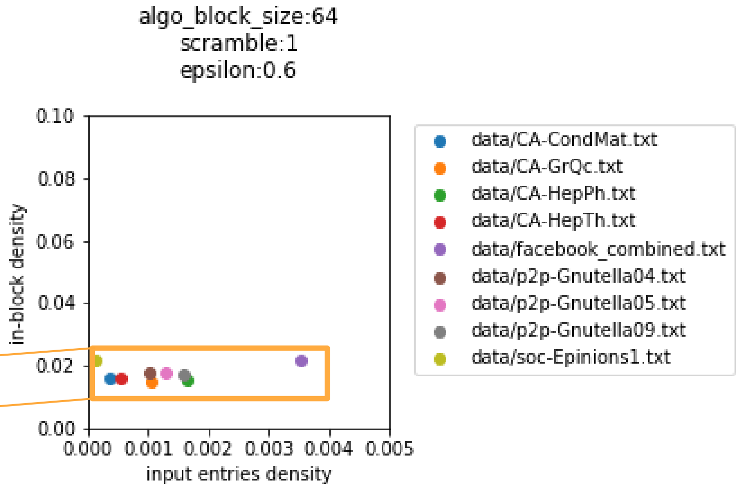


Multiplication - experiments



Multiplication - experiments





Future directions – Better, faster reordering

- Parallel implementation
- Local Sensitive Hashing

Future directions – LSH

Faster reordering algorithm through local sensitive hashing and nearest neighbours search

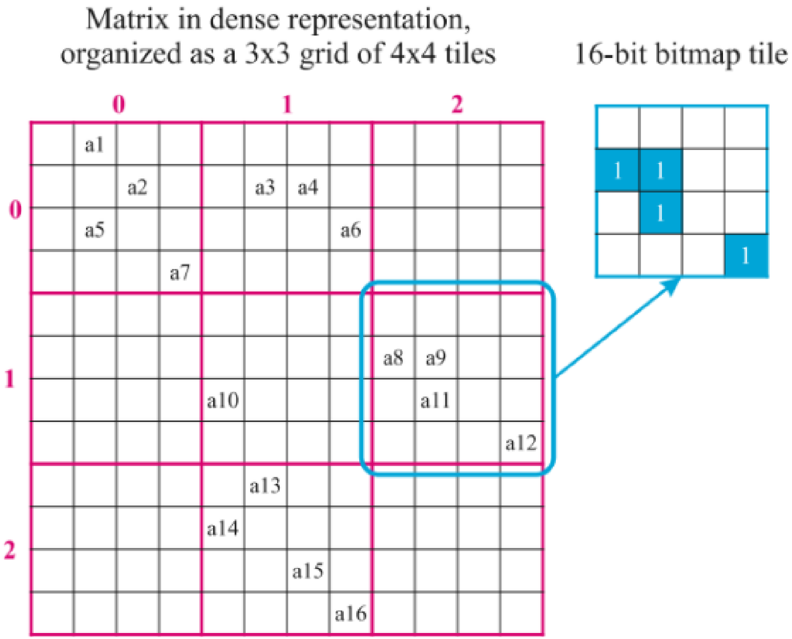
MinHash (random permutations): approximate Jaccard similarity

SimHash (random projections): approximate cosine similarity

Future directions - Compare to related work

Accelerating sparse matrix–matrix multiplication with GPU Tensor Cores☆

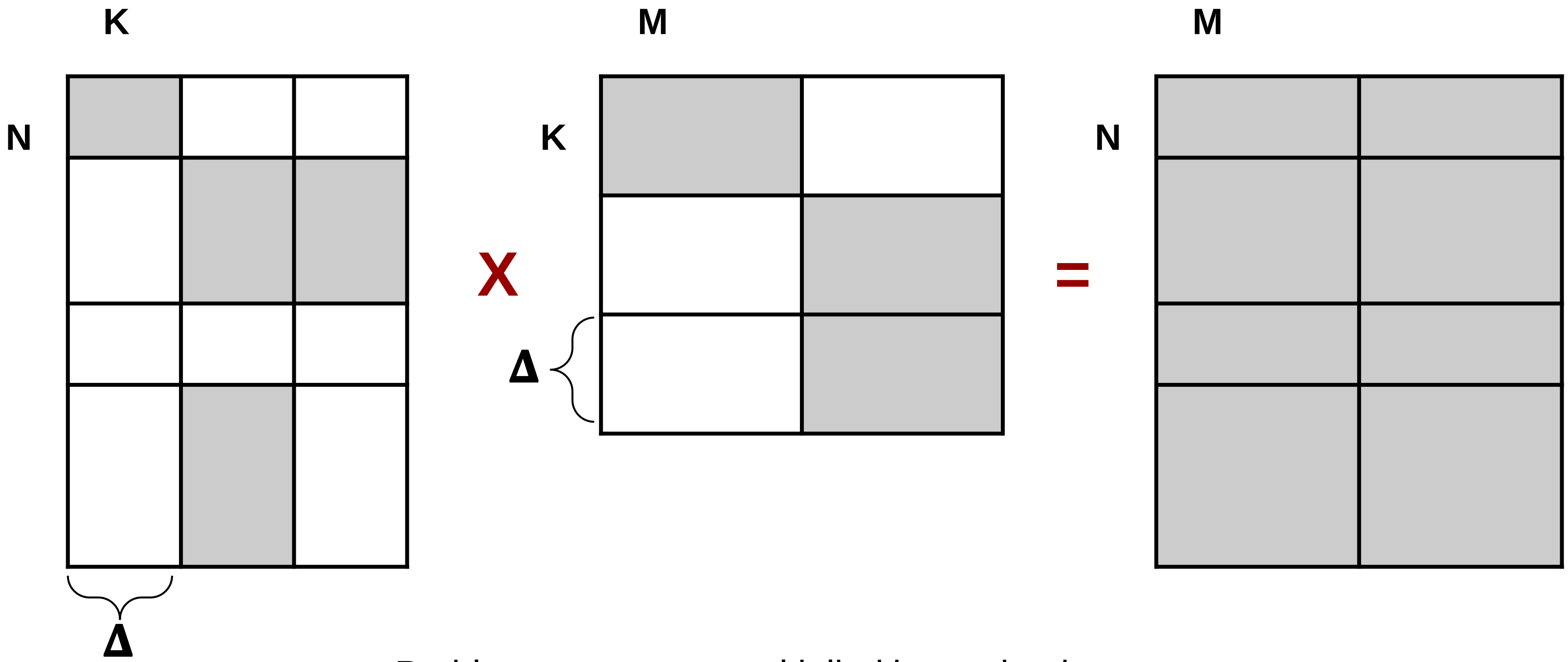
Orestis Zachariadis [a,*], Nitin Satpute [a], Juan Gómez-Luna [b], Joaquíı



Future directions - Tensor-core implementation

- Optimal size of block partition?
- Best way to parallelize?
- Focused experiments

Future directions - Sparse-Sparse multiplication



Problem: zeros gets multiplied by each other
Number of unnecessary operations is squared