Sparse Matrix Performance

CHAI CAPILI

THOMAS FISHER

PRANEETH MARI

SARA ROMERO

The Hypothesis

Optimal performance of sparse matrices is bounded by O(n) non-zero values in the matrix.

Performance will be analyzed through communication costs, storage overhead, and operation runtime (Algorithms)

Our Dataset

20 Matrices used by all team-member implementations

10 matrices of 100x100 [10% dense – 90% dense]

10 matrices of 1000x1000[10% dense – 90% dense]

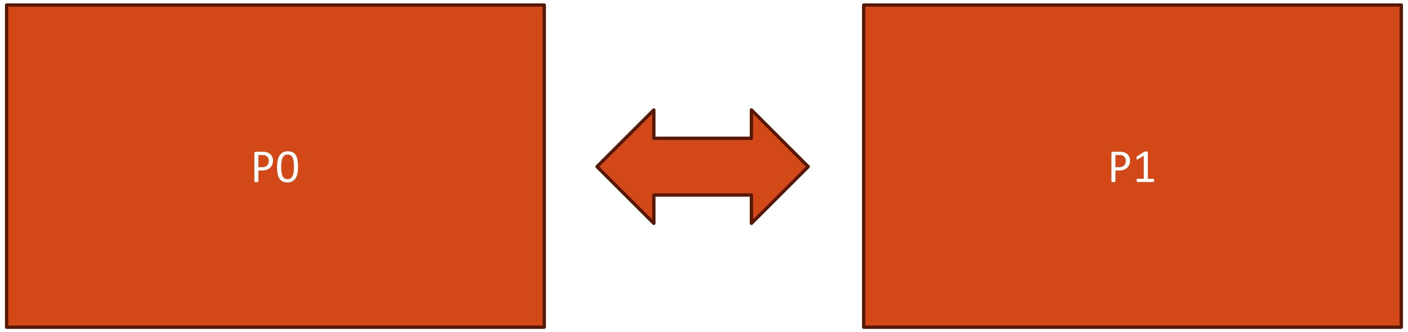
Our smallest Density is the Diagonal Matrix

Sparse Matrix Communication

COO, CSR, CSC Format

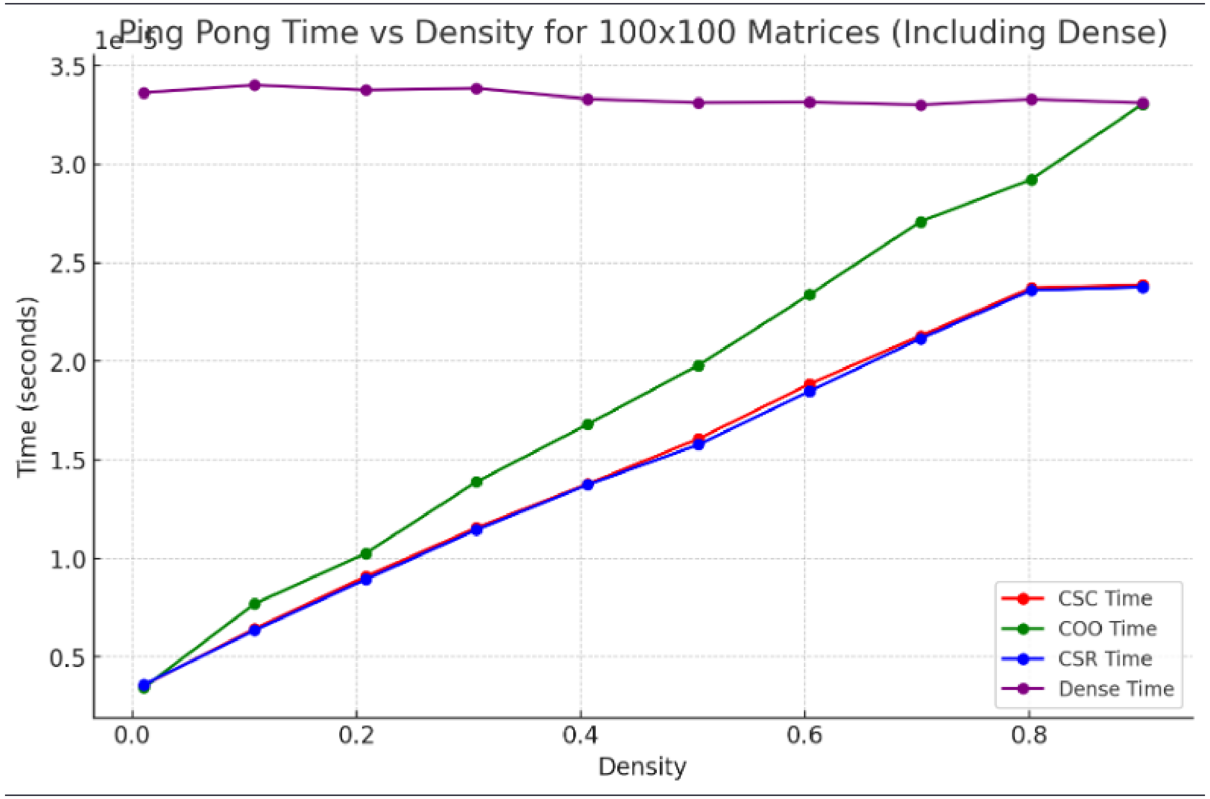
Dense	COO	CSR	CSC
[1,0,2] [0,3,0] [4,0,5]	Values: [1, 2, 3, 4, 5] Cols: [0, 2, 1, 0, 2] Rows: [0, 0, 1, 2, 2]	Values: [1, 2, 3, 4, 5] Cols: [0, 2, 1, 0, 2] Rowptrs: [0, 2, 3, 5]	Values: [1, 4, 3, 2, 5] Rows: [0, 2, 1, 0, 2] Colptrs: [0, 2, 3, 5]

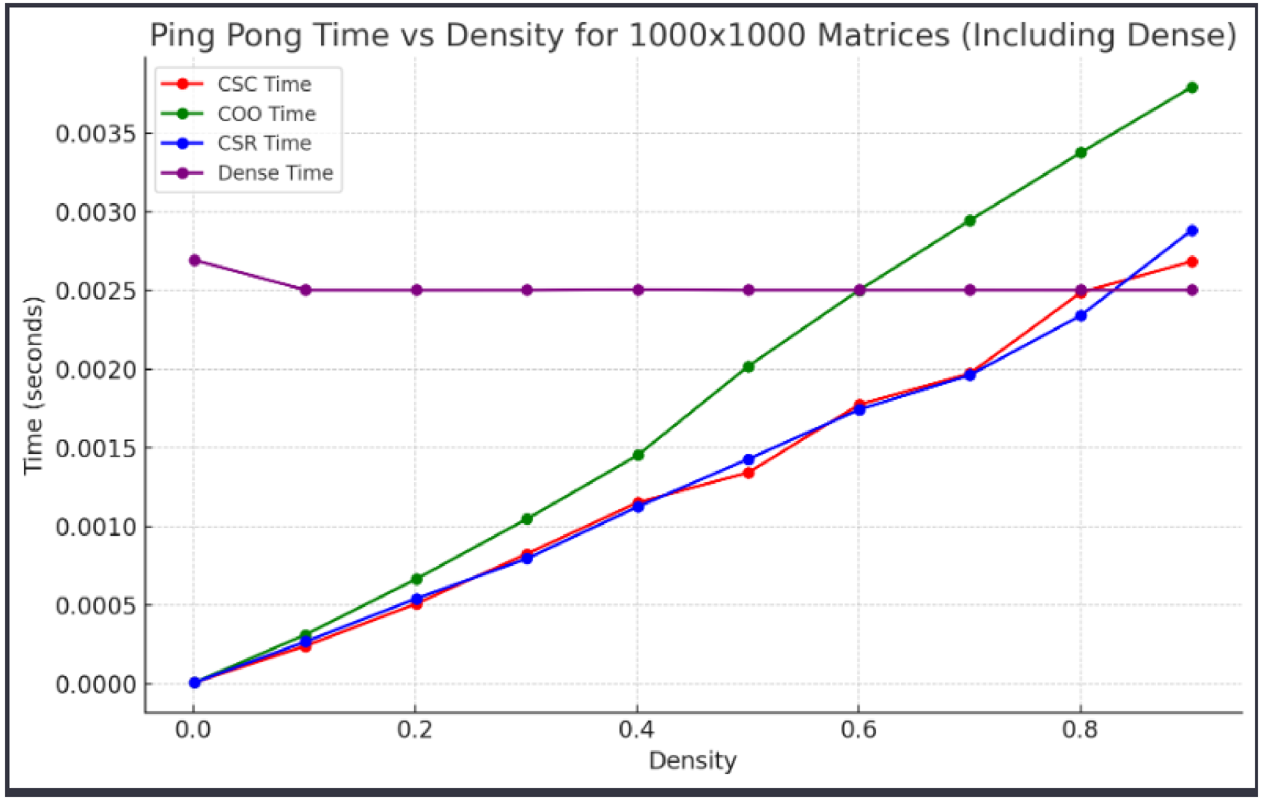
Ping Pong Tests (Communication Costs)



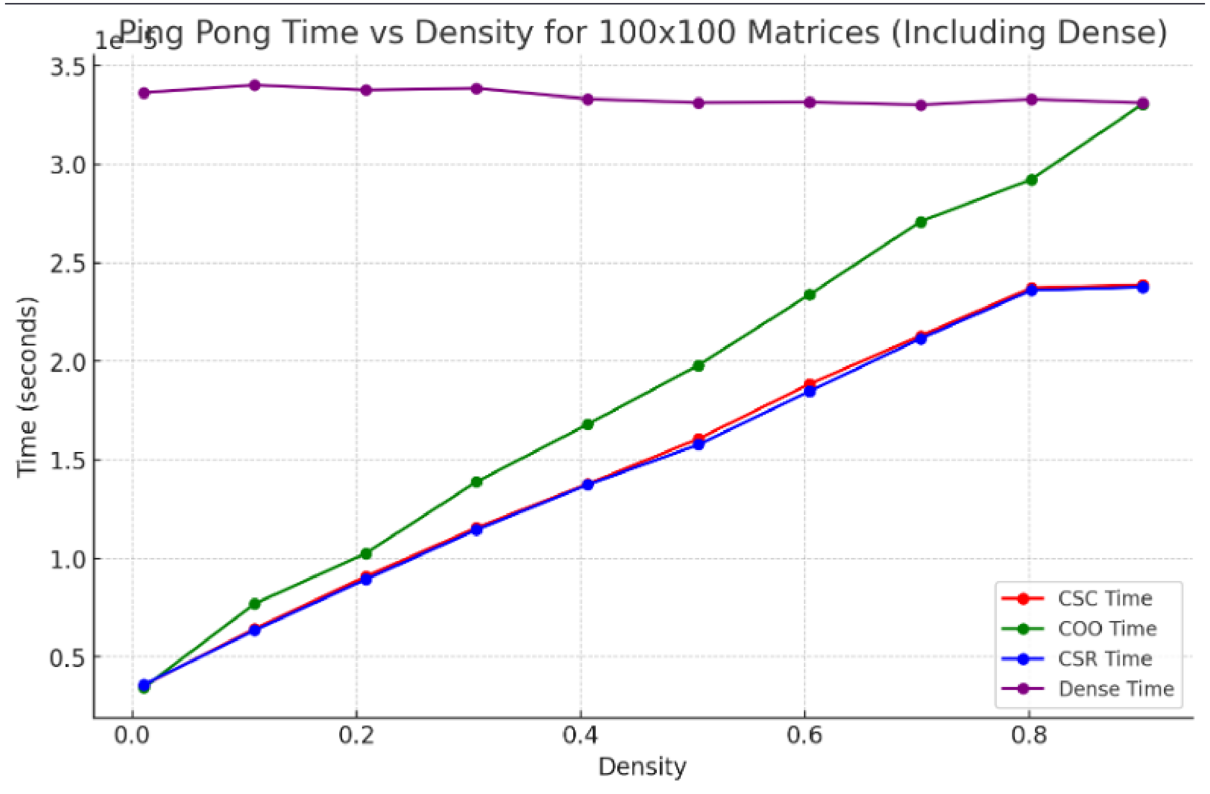
- 5000 iterations of ping-pong per matrix
- MPI_Pack and MPI_Unpack are used with MPI_Send/MPI_Recv on each of the COO,CSR, CSC
- The dense format just uses a basic MPI_Send/MPI_Recv

Ping Pong Timings



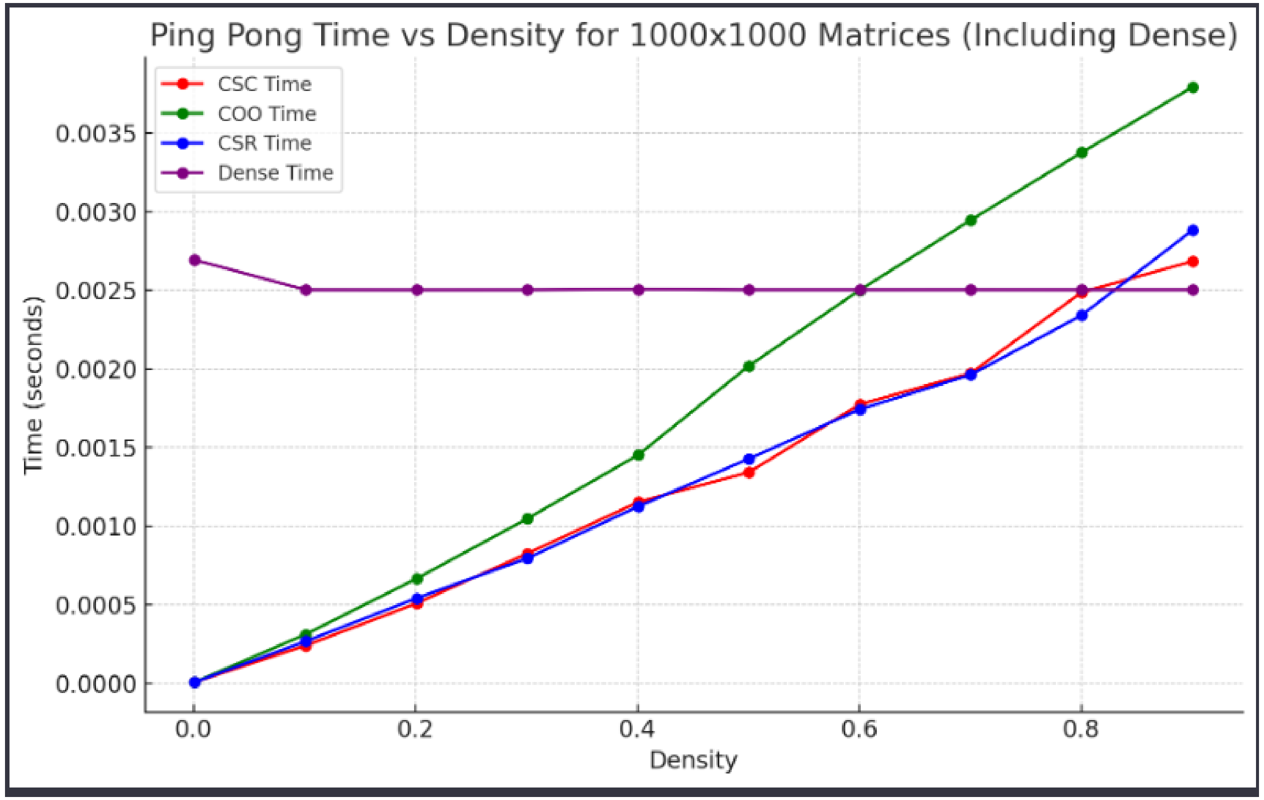


100 x 100 Analysis



- These ping pong timings show the general trend for communication performance between the different formats
- I was expecting to see dense outperform the other formats, but this occurs at 1000x1000
- We see that the best communication performance is at O(n) sparsity

1000 x 1000 Analysis



- At about 0.8 density, the cost of communication for the dense format outperforms the other formats
- Again, We see that the best communication performance is at O(n) sparsity

Sparse Matrix Storage Analysis

Theoretical Analysis

- Mathematical modeling of matrix formats
  - Dense

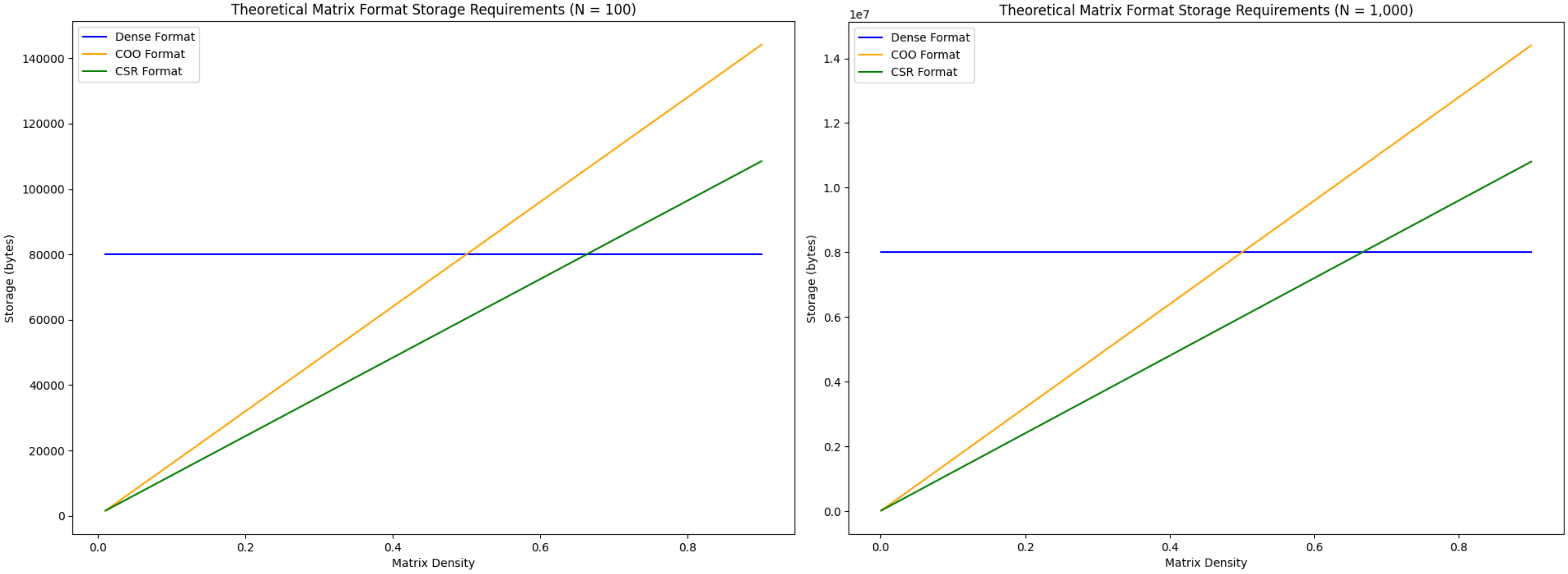
    n^2 elements of the type used in the matrix
  - COO

    $2d \cdot n^2$ integers $+\, d \cdot n^2$ elements of the type used by the matrix
  - CSR

    $(n + 1) + d \cdot n^2$ integers $+\, d \cdot n^2$ elements of the type used by the matrix
  - CSC

    $(n + 1) + d \cdot n^2$ integers $+\, d \cdot n^2$ elements of the type used by the matrix

Theoretical Results



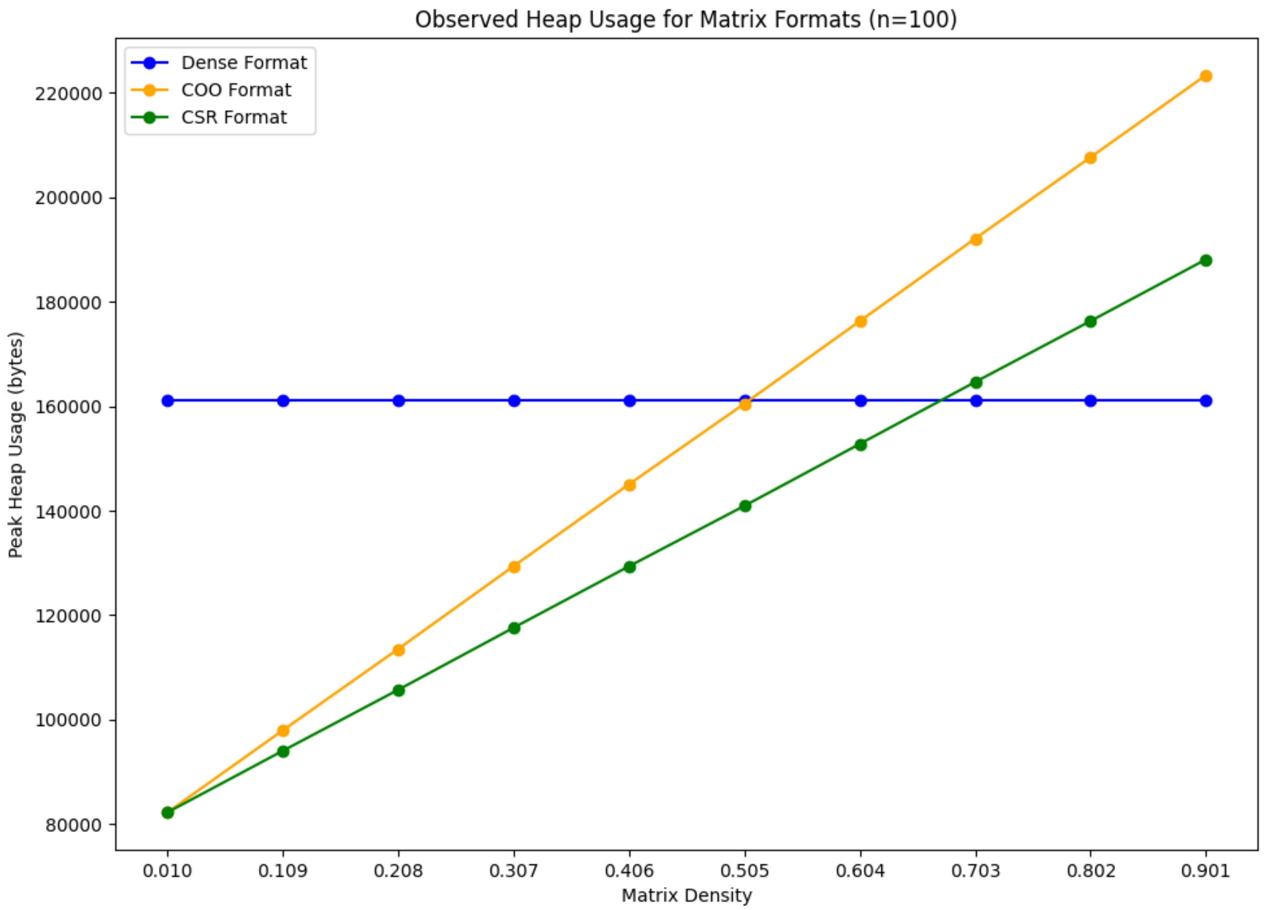
Experimental Analysis

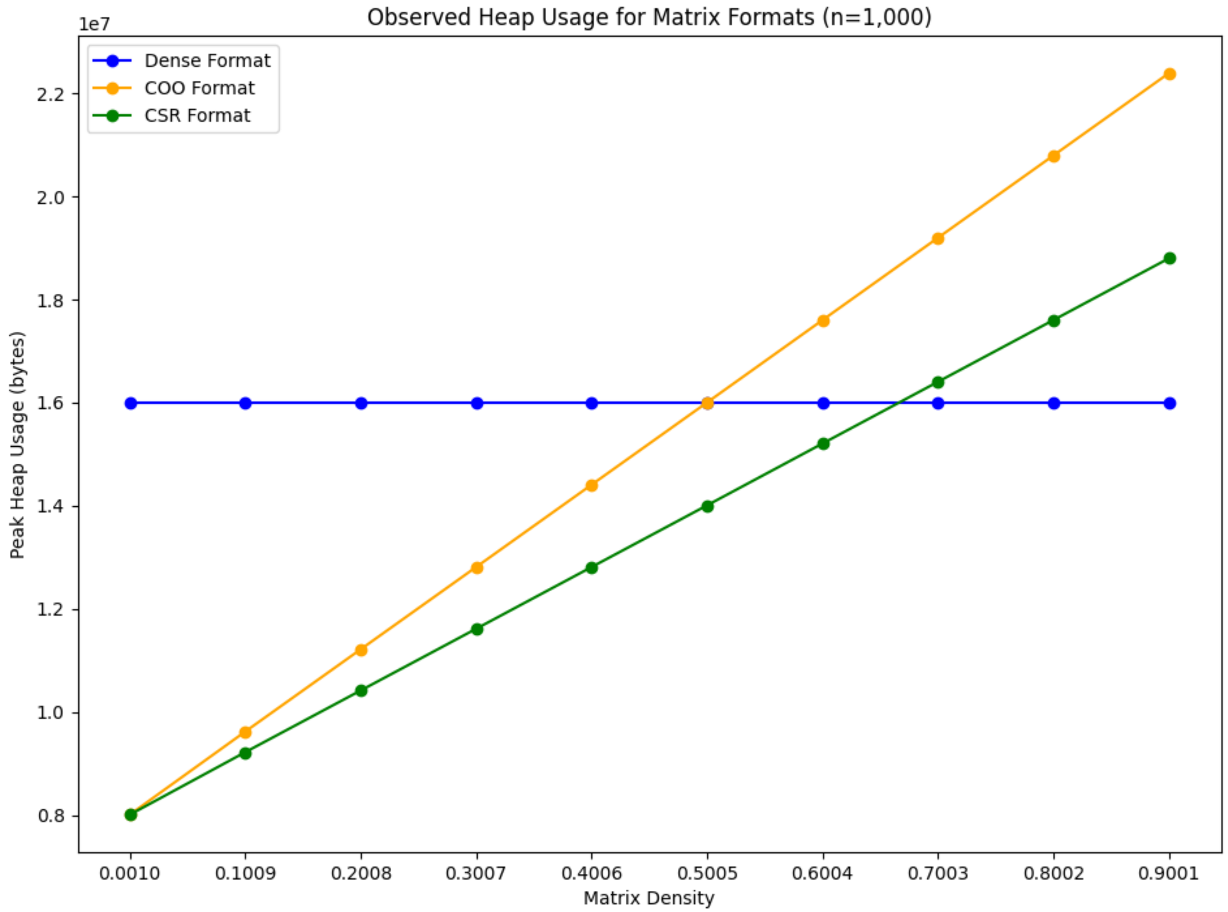
- Used Valgrind's massif tool to measure heap usage.

- Generated 10 data files each for dimensions 100 and 1,000.


```
desc: --massif-out-
file=analysis_files/COO/coo_dimension_1000_nonzeros_1000_massif_instructi
ons --time-unit=i
cmd: ./profiler_analysis
../../matrices/standardized_matrices/dimension_1000_nonzeros_1000.mtx
time_unit: i
#-----------
snapshot=0
#-----------
time=0
mem_heap_B=0
mem_heap_extra_B=0
mem_stacks_B=0
heap_tree=empty
#-----------
snapshot=1
#-----------
time=123861
mem_heap_B=568
mem_heap_extra_B=16
mem_stacks_B=0
heap_tree=empty
#-----------
snapshot=2
#-----------
time=128463
mem_heap_B=8000568
mem_heap_extra_B=3544
mem_stacks_B=0
heap_tree=empty
#-----------
snapshot=3
#-----------
```

Experimental Results



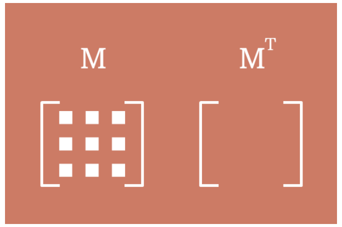


Sparse Matrix Transpose

CSR Transpose

“Transpose of a matrix,” Math Doubts, https://www.mathdoubts.com/matrix/transpose/ (accessed Dec. 3, 2023).

$$\begin{bmatrix} 0 & 1 & 2 \\ 0 & 3 & 0 \\ 4 & 0 & 5 \end{bmatrix}$$



$$\begin{bmatrix} 0 & 0 & 4 \\ 1 & 3 & 0 \\ 2 & 0 & 5 \end{bmatrix}$$

Values: {1, 2, 3, 4, 5}
Row Pointers: {0, 2, 3, 5}
Columns: {1, 2, 1, 0, 2}

Values: {4, 1, 3, 2, 5}
Row Pointers: {0, 1, 3, 5}
Columns: {2, 0, 1, 0, 2}

Something interesting to observe…

$$\begin{bmatrix} 0 & 1 & 2 \\ 0 & 3 & 0 \\ 4 & 0 & 5 \end{bmatrix} \longrightarrow \begin{bmatrix} 0 & 0 & 4 \\ 1 & 3 & 0 \\ 2 & 0 & 5 \end{bmatrix}$$

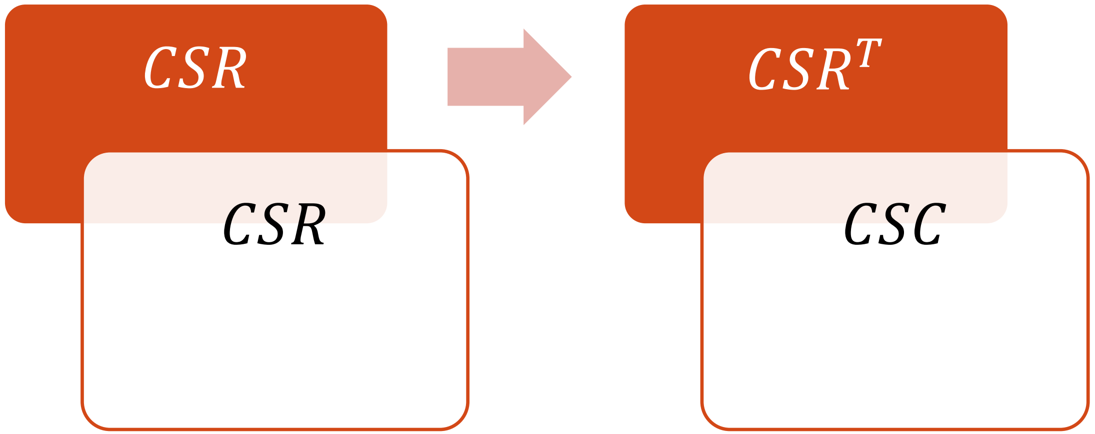
Values: {4, 1, 3, 2, 5}
Column Pointers: {0, 1, 3, 5}
Rows: {2, 0, 1, 0, 2}

Values: {4, 1, 3, 2, 5}
Row Pointer: {0, 1, 3, 5}
Columns: {2, 0, 1, 0, 2}

If we represent the original matrix in CSC, it's equivalent to the transposed CSR matrix

CSR Transpose Algorithm

Same algorithm as converting CSR to CSC, just different variable names

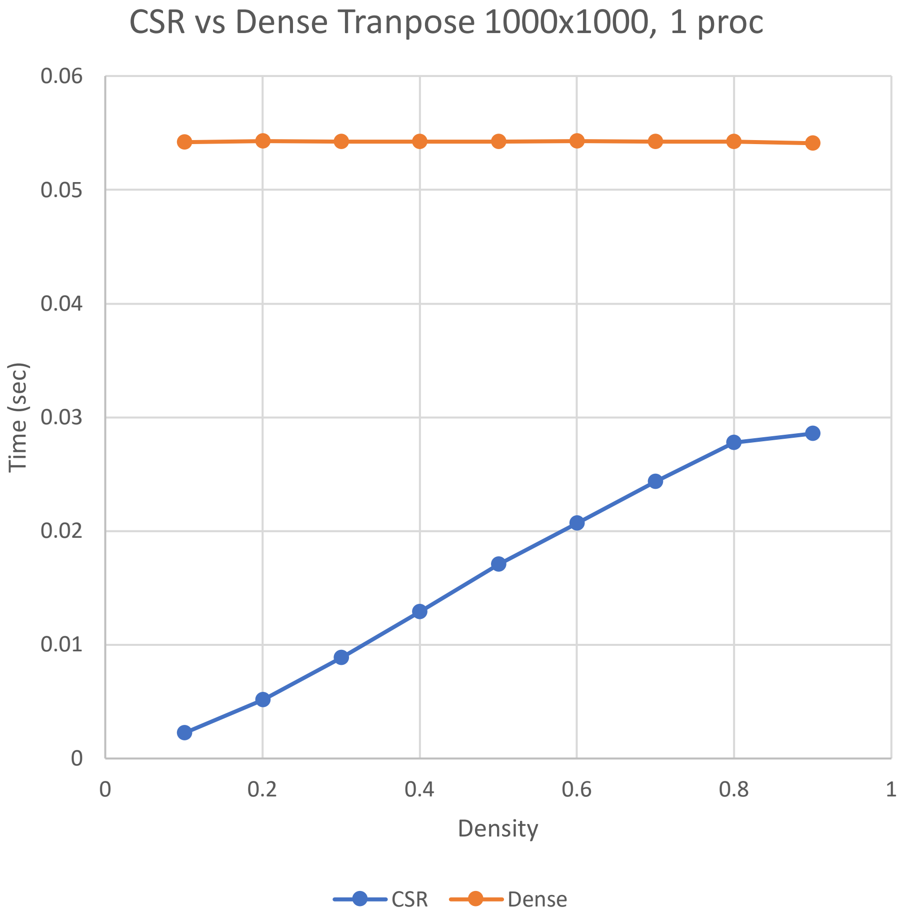


Dense Algorithm

Partition	Transpose	Insert
Partition matrix A into square blocks	Transpose block	Insert block into matrix t_A

Averages of 100 iterations:

CSR vs Dense Transpose



- Not the expected trade-off in runtime at O(n) non-zeros
- CSR has better operational runtime across all densities
- Hard to effectively compare due to extremely different algorithms

Parallelizing the algorithm

Challenging

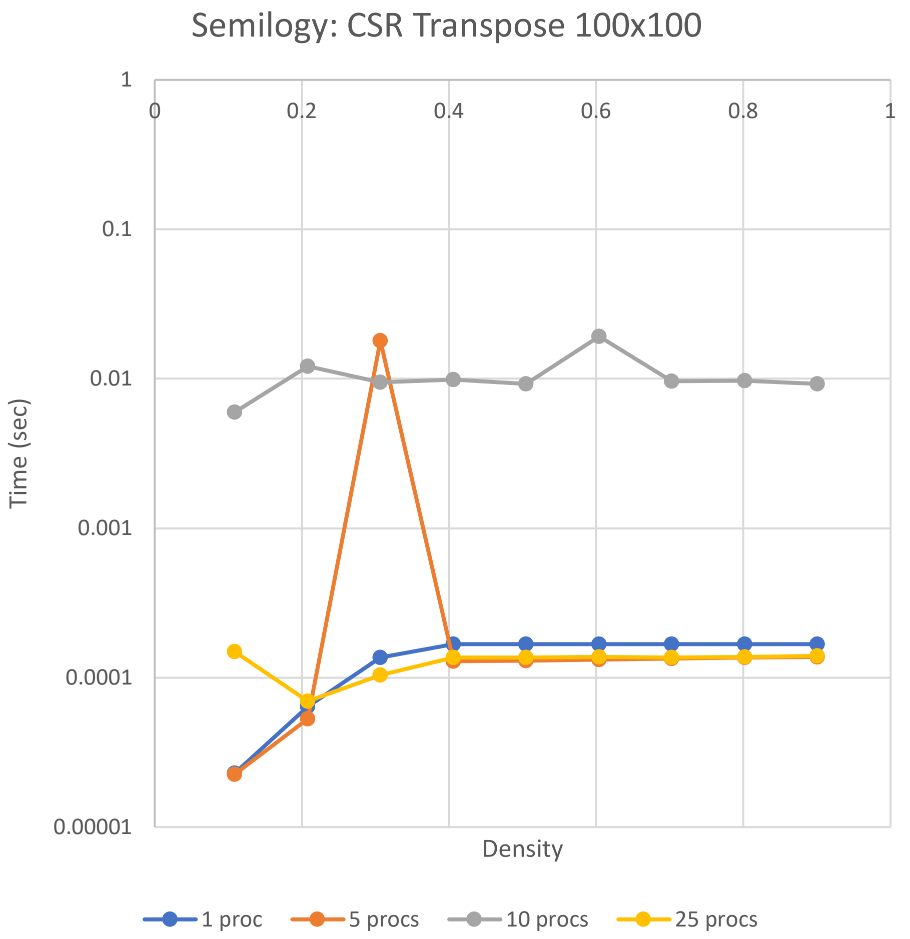
- Calculating each row pointer relies on summing the value of the previous pointer

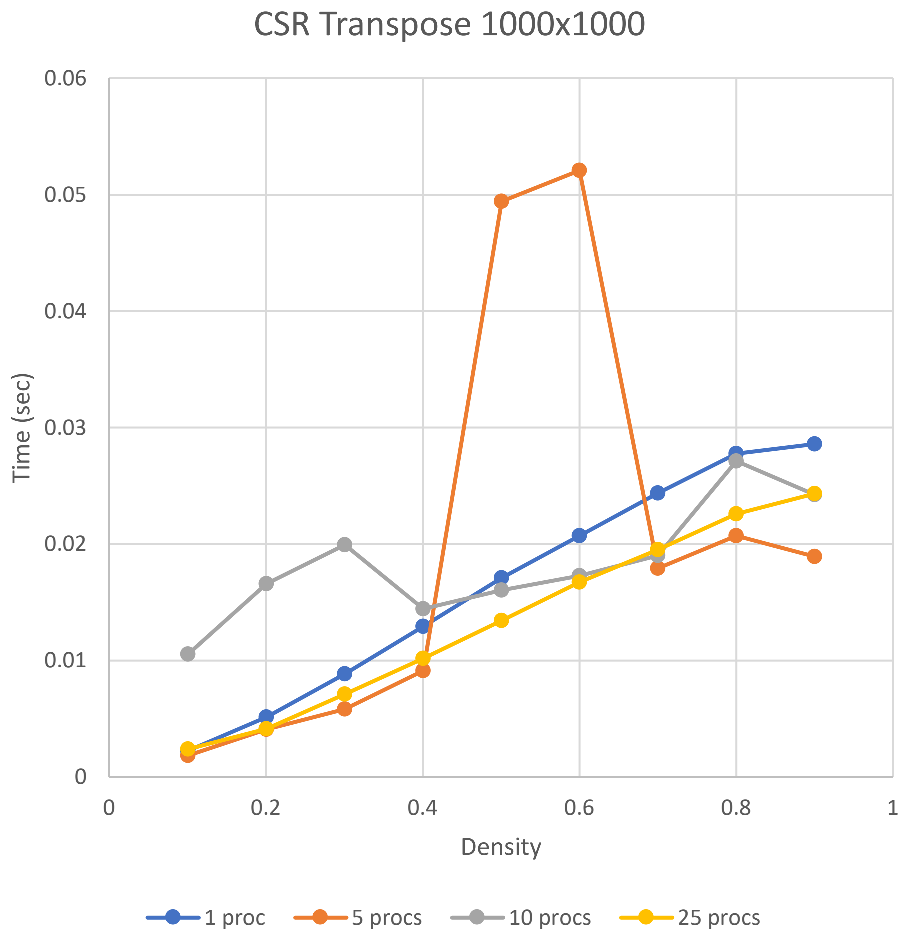
What can be parallelized?

- Counting the number of non-zeros in each column of the original matrix.
  - This is used to later to calculate the transposed row pointer

Averages of 100 iterations:

Varying Communication Overhead





What about CSC and COO Transpose?

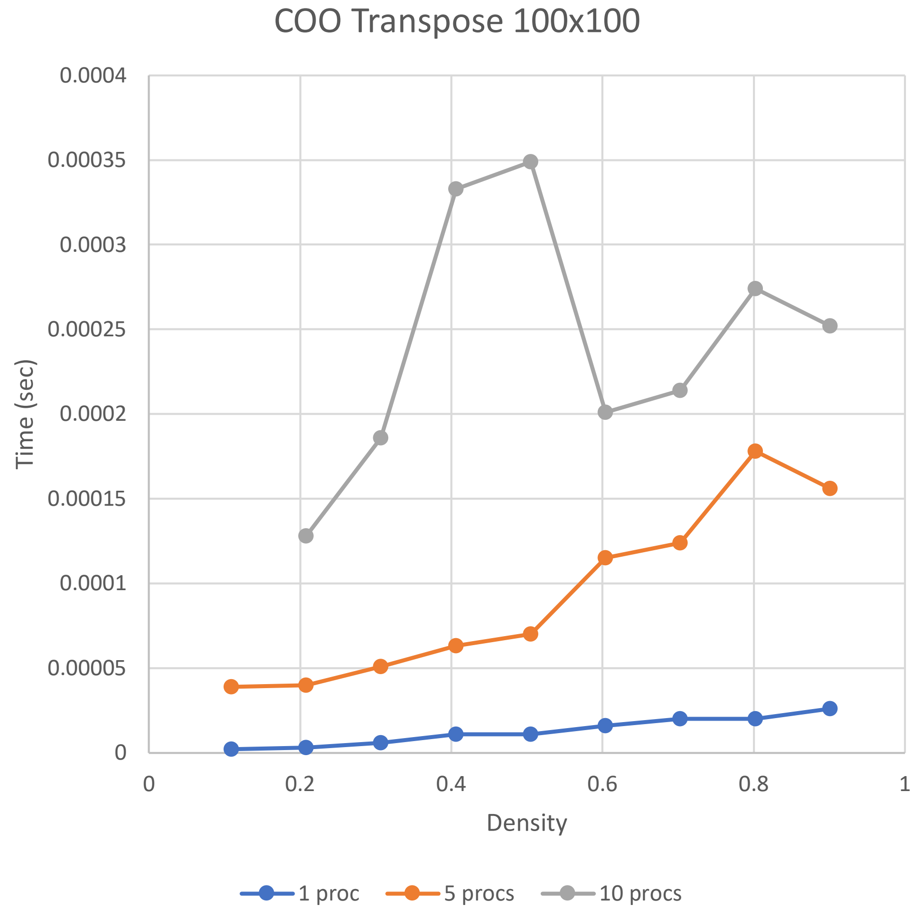
CSC is the same algorithm as CSR, just different variable names

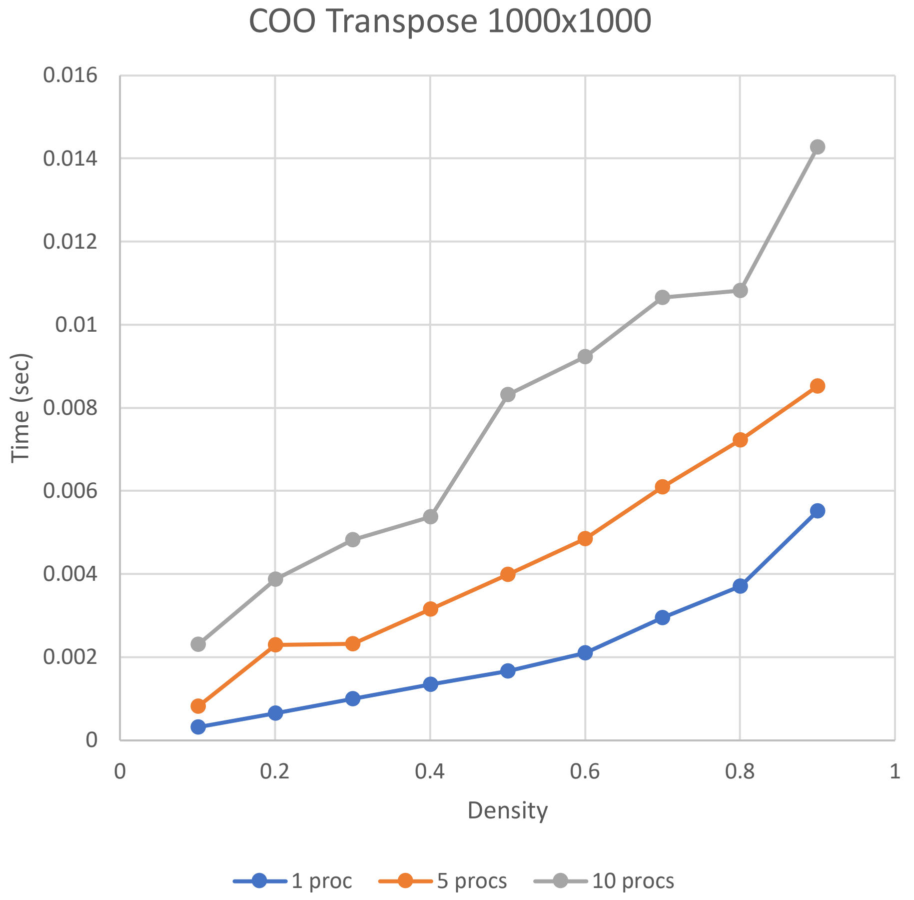
COO is trivial

- Just swap the row and column arrays

Averages of 100 iterations:

Major communication and storage overhead

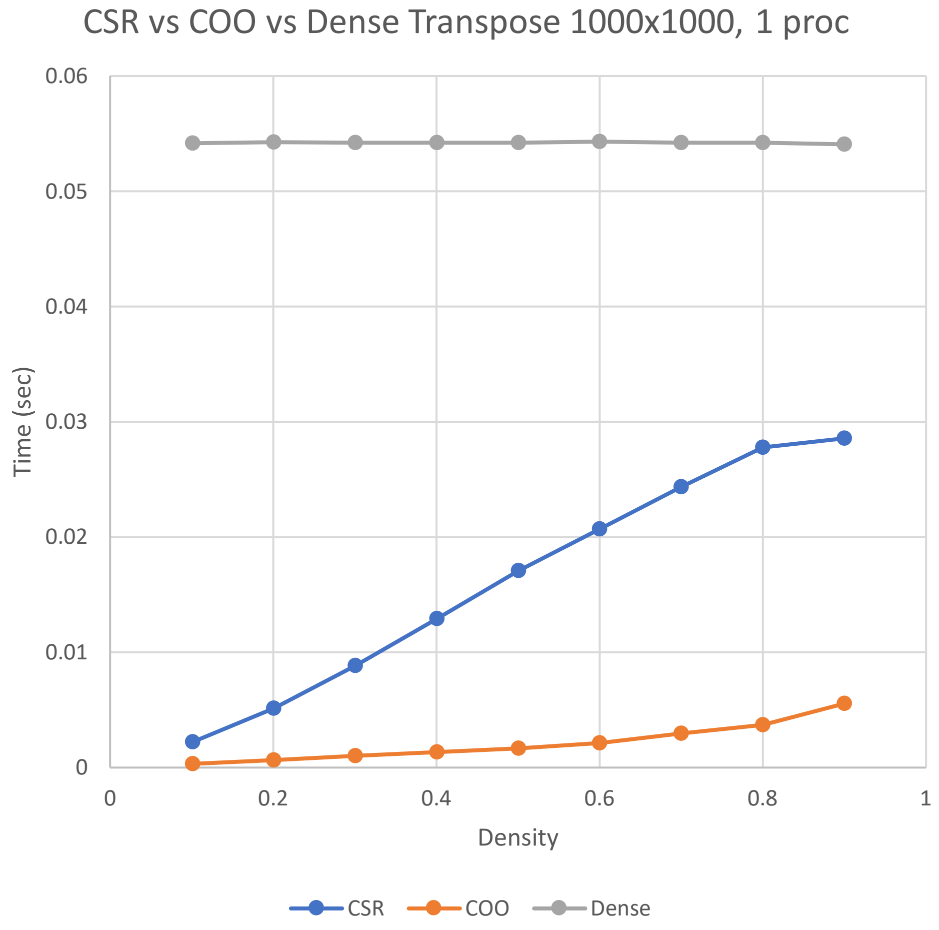




Averages of 100 iterations:

CSR vs COO vs Dense Transpose

- Unexpectedly, COO outperforms CSR in operational runtime
- CSR transpose algorithm has increased complexity due to row pointers

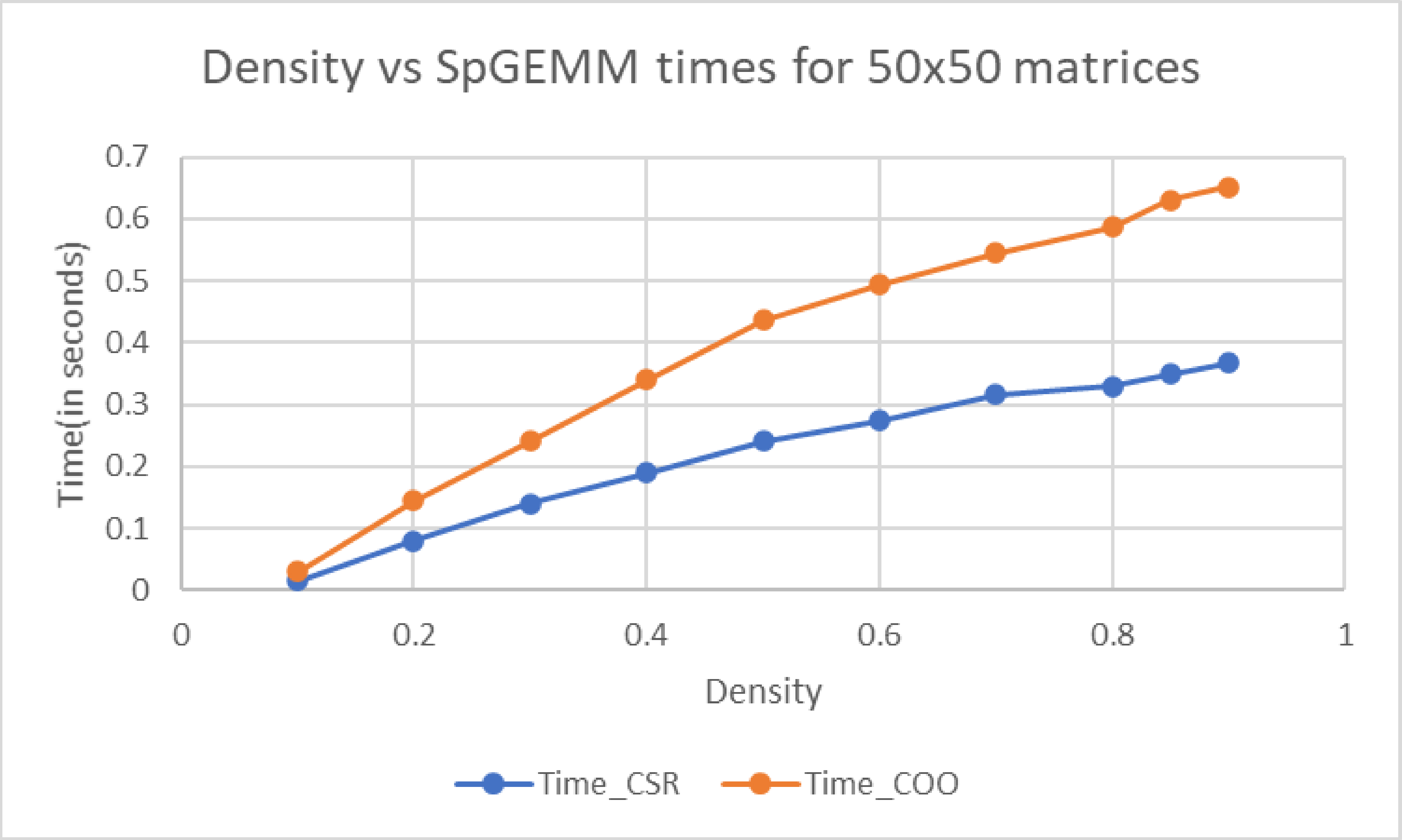


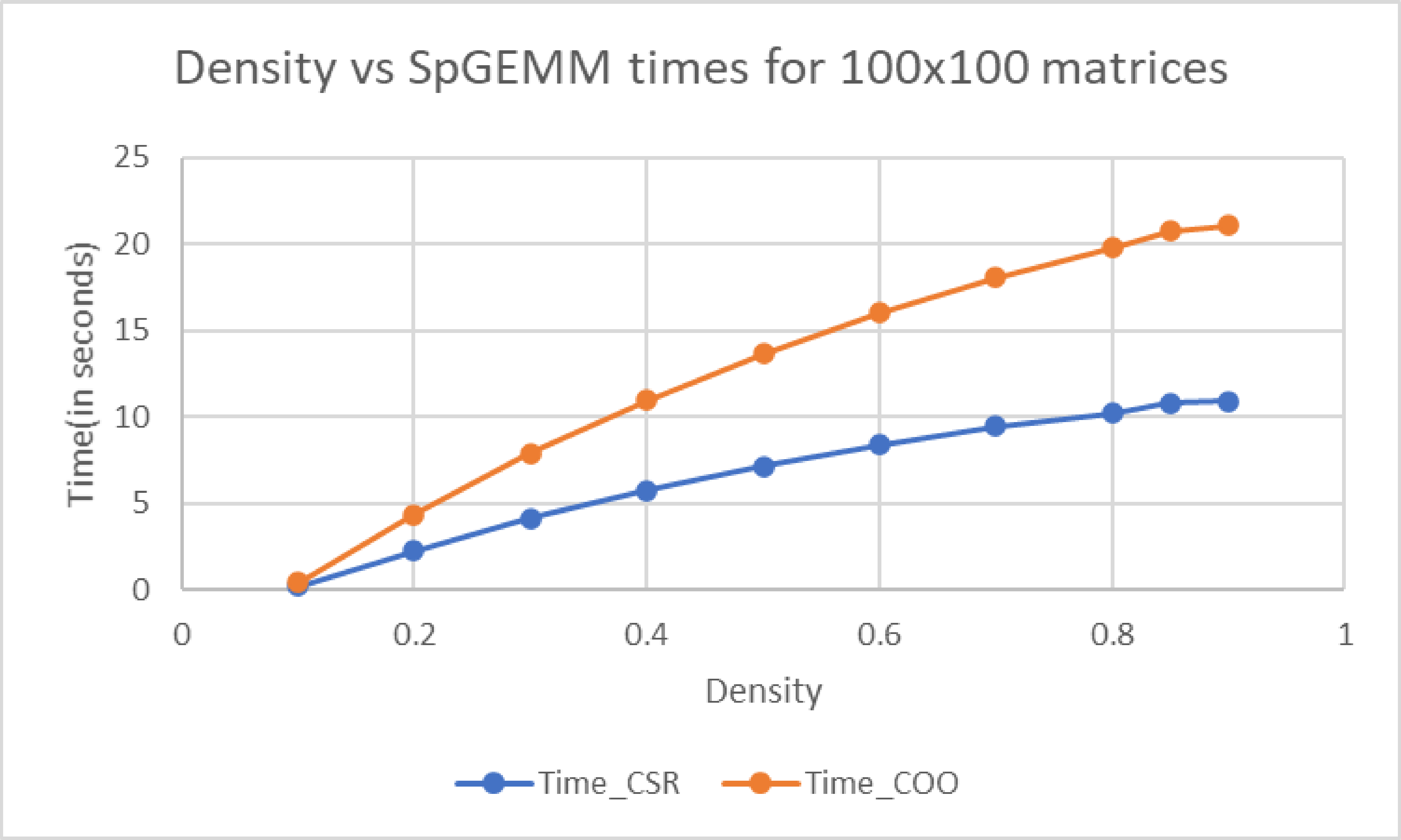
Sparse Matrix-Matrix Multiplication

Experimental Analysis using COO, CSR

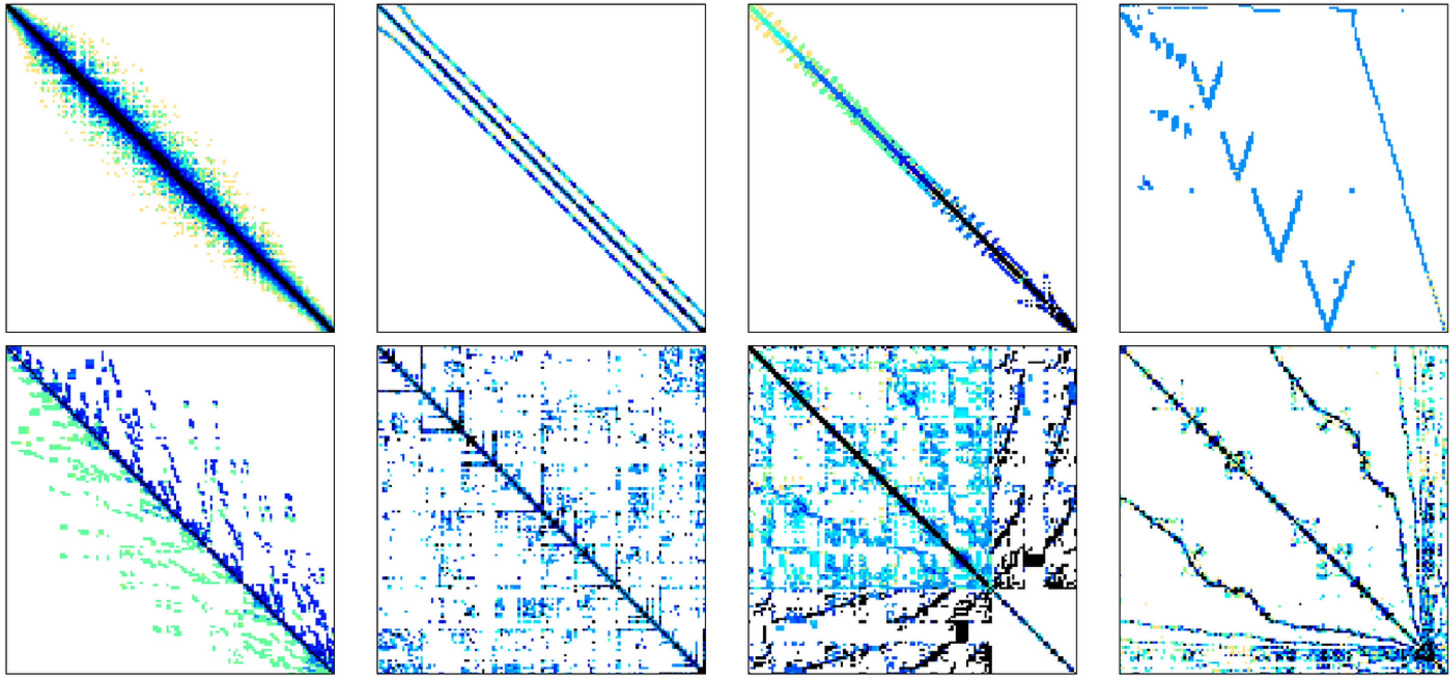
- Performance comparison of different matrix formats (COO, CSR)
- Observed trend: CSR outperformed COO
- Also observed findings that support the hypothesis O(n) non-zero values maximize performance.

- Challenges faced: Implementing parallel version of sparse matrix-matrix multiplication. Ran into segmentation faults.


Density vs SpGEMM times for 50x50 matrices
Time(in seconds)
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.2
0.4
0.6
0.8
1
Density
Time_CSR
Time_COO


Density vs SpGEMM times for 100x100 matrices
25
20
15
10
5
0
Time(in seconds)
0
0.2
0.4
0.6
0.8
1
Density
Time_CSR
Time_COO

What’s next?



“Sparse matrix-vector multiplication with Cuda,” Medium, https://medium.com/analytics-vidhya/sparse-matrix-vector-multiplication-with-cuda-42d191878e8f (accessed Dec. 3, 2023).

- The effects of sparsity patterns
- Increased parallelism and optimization
- CUDA applications

Conclusion

Our team investigated how density and format affect sparse matrix performance across a variety of metrics.

We explored sparse matrix operation cost, communication cost, and storage cost.

Our analysis supports the hypothesis that O(n) non-zero values in a sparse matrix maximizes performance.

Entire code base (utilities, matrices, analysis files, etc.) created from scratch.

Acknowledgements

- The CSR transpose algorithm is based off algorithms developed by SciPy for Python, check out their GitHub repository:
  - https://github.com/scipy/scipy/blob/8a64c938ddf1ae4c02a08d2c5e38daeb8d061d38/scipy/sparse/sparsetools/csr.h#L419
  - For more information visit https://scipy.org/

- We would like to thank the UNM Center for Advanced Research Computing, supported in part by the National Science Foundation, for providing the research computing resources used in this work.