Cole Fettkether
COEN 145

Program 1: Sparse Accumulator Matrix Matrix Multiplication

Introduction

This paper will explore the process of sparse matrix matrix multiplication and how singling out the non zero values in the sparse matrix speeds it up. To do this, the accumulator method is used which stores the values as rowVal, the index within a row as rowInd, and the start of each row within rowVals as rowPtr. The second matrix is then transposed, which allows for the multiplication to be accurate.

Description

To determine which threads would do each work, the following line was used::

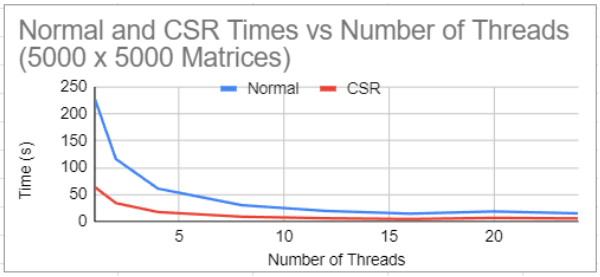
```
#pragma omp parallel for shared(result)
```

This loop uses the open mp interface to parallelize a for loop, where the array result is shared among many threads in order to store the result of each specific row by column multiplication. In order to construct the CSR accumulator algorithm, four for loops were used. The first two iterate through each row of compressed matrix A and compressed transposed matrix B, specifically from the start to end index of the non-zero values in the matrices. The purpose of the transpose is so that the matrix matrix multiplication is easier to do row by row. The two inner for loops iterate through each row of non zero values in A and B, and check if the row indices are the same. If they are, the values are multiplied and added to the corresponding spot in the output matrix. This algorithm is much faster and more efficient than a full matrix matrix multiplication, as it only multiplies the non zero values.

Analysis

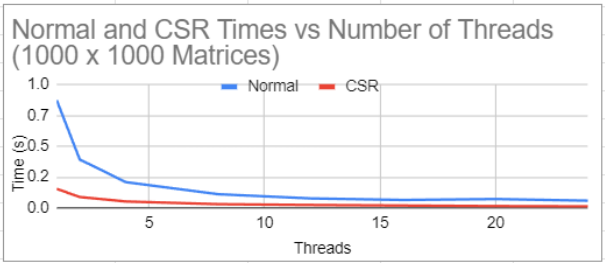
Below are the results for a 5000 x 5000 matrix-matrix multiplication with 1% fill for a varying number of threads. As shown below, the CSR algorithm is significantly faster than the normal matrix-matrix multiplication algorithm.

Size	Threads	Fill	Normal	CSR
5000	1	0.01	227.48	63.74
5000	2	0.01	115.943	34.021
5000	4	0.01	61.0162	17.409
5000	8	0.01	30.1209	8.73715
5000	12	0.01	19.4472	5.85094
5000	16	0.01	14.4111	4.39297
5000	20	0.01	18.4855	6.52888
5000	24	0.01	14.9435	5.46539



Similar results are seen for a 1000 x 1000 matrix-matrix multiplication with 1% fill as expected.

Size	Threads	Fill	Normal	CSR
1000	1	0.01	0.8761441	0.154894
1000	2	0.01	0.3928622	0.088421
1000	4	0.01	0.2090374	0.052146
1000	8	0.01	0.1111818	0.030267
1000	12	0.01	0.07704	0.023829
1000	16	0.01	0.064583	0.019193
1000	20	0.01	0.071961	0.014566
1000	24	0.01	0.058847	0.012311

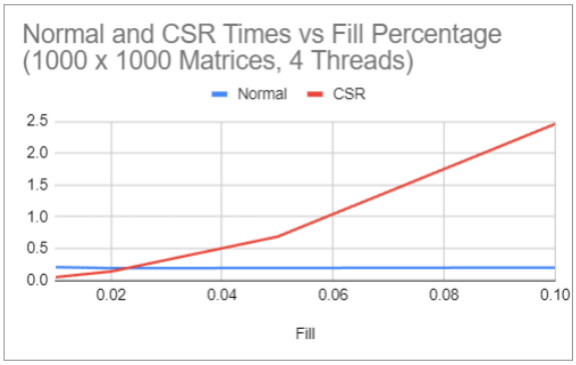


When we scale down the numbers to extremely low fill values, we see drastic improvement in the efficiency of the algorithm. For .01% fill, the CSR algorithm is exponentially faster than normal matrix-matrix multiplication as there are so few values being multiplied.

Size	Threads	Fill	Normal	CSR
5000	28	0.0001	13.2935	0.0123238
10000	28	0.0001	127.915	0.0617976
20000	28	0.0001	-	0.417681
30000	28	0.0001	-	1.39045
40000	28	0.0001	-	3.50447

We can also observe how different fill sizes affect the results of the CSR algorithm. As shown below, the algorithm is much faster for lower fill values, and as fill percentage increases, the speed of the algorithm slows down.

Size	Threads	Fill	Normal	CSR
1000	4	0.01	0.2090374	0.052146
1000	4	0.02	0.194015	0.14091
1000	4	0.05	0.196558	0.6886
1000	4	0.1	0.202374	2.45998



Conclusion

Overall, we see that the CSR matrix-matrix multiplication algorithm is much faster than a traditional matrix-matrix multiplication algorithm. This is extremely important as there are many datasets which have extremely sparse matrices, making this algorithm essential to acceptable performance, which is highly relevant in many datasets and machine learning algorithms. This report demonstrates how the CSR algorithm provides valuable speedup compared to normal matrix-matrix multiplication.