Quantitative analysis

We used CT-scan images from TCIA Archive.

Data Citation

`Albertina, B., Watson, M., Holback, C., Jarosz, R., Kirk, S., Lee, Y., Rieger-Christ, K., & Lemmerman, J. (2016). The Ca`

TCIA Citation

`Clark, K., Vendt, B., Smith, K., Freymann, J., Kirby, J., Koppel, P., Moore, S., Phillips, S., Maffitt, D., Pringle, M.,`

Preprocessing

- First 32bit .tiff images were converted to 1bit .tiff images as for our research we are interested only in binary type of matrices. Conversion happened with command `magick mogrify -format tif -depth 1 *.tif` in the correct folder.
- Next we generated matrices from these tiff images with python. In these matrices zero represents black pixel and one white pixel.

Independent and dependent variables

- With python we also generated preliminary matrix conversion to Block Storage Format
- In our research our independent variables are the matrices generated from CT-scan images and dependent variables are the sizes of the matrices in the Block Storage Row format.
    - As the CT-scan images have same amount of pixels 512x512, only property here are the black and white pixels.
    - For the matrix storage format we are interested in the pixels that are important to store and therefore interesting properties for dependent variable are the space and time the storage format requires.

Exploratory analysis

- With exploratory data anaylsis we were investigating if the images and generated sparse matrices contain dense sub-matrices. This was done with python and if CT-scan contains dense submatrix, program will just simply print 'True' to stdout. The check was done by first creating new matrix of sum of every neighboring (3 elements every direction) elements and this "neighbor_sum_matrix" was then transformed to array by summing every element in each row. Finally it was checked that if in this array there is at least one row with only zeroes (the sum is 0) and at least one row with sum value higher than 512 (this was chosen by comparing the results).

Statistical tools

Since our project work aims at improving the sparse matrix multiplication performance, and do not need to analyze the semantical analysis for data, we will need standard and tools to measure the performance of sparse matrix multiplication based on our future proposed storage method. The measurement will be applied via C++

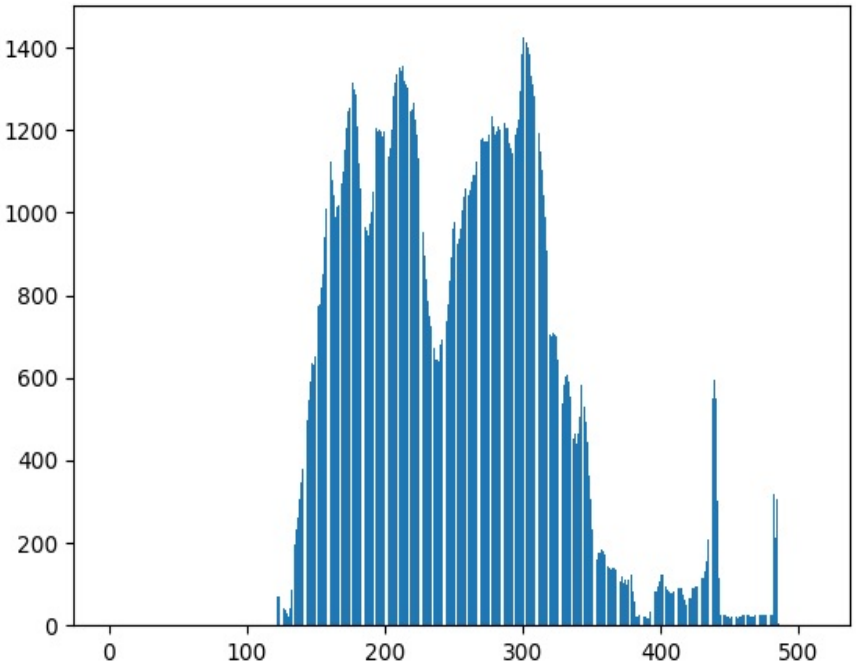
The Timer will be defined as follows, and the multiplication required timing will be at the middle of timer.start() and timer.stop(): ``` anonymouslib_timer CSR5Spmv_timer; CSR5Spmv_timer.start();

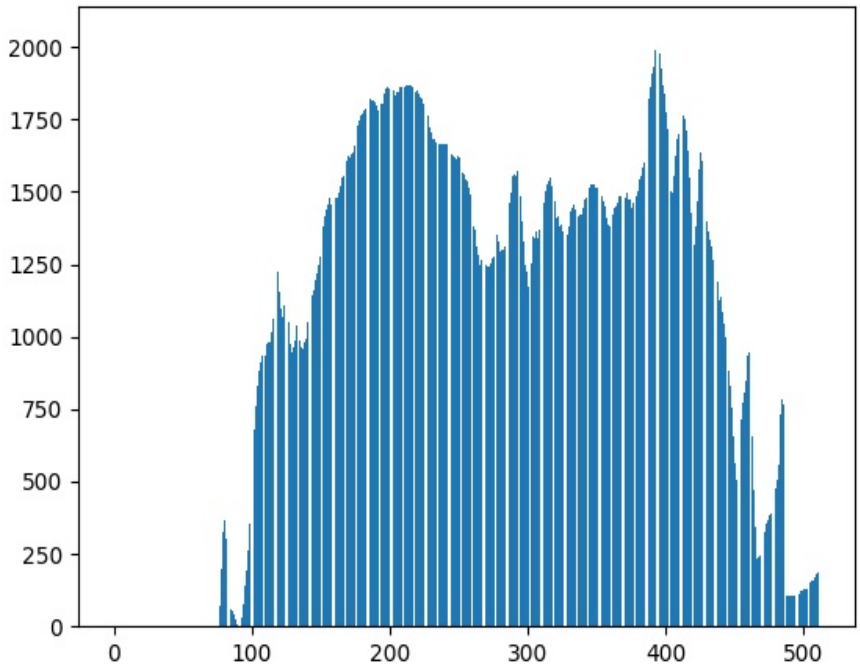
```
err = A.spmv(alpha, d_y);
//cout << "spmv err = " << err << endl;
checkCudaErrors(cudaMemcpy(y, d_y, m * sizeof(VALUE_TYPE), cudaMemcpyDeviceToHost));

double CSR5Spmv_time = CSR5Spmv_timer.stop() / (double)NUM_RUN;



We will get GFlops, Bandwidth, and computation time we want via:


if (NUM_RUN)
    cout << "CSR5-based SpMV time = " << CSR5Spmv_time
         << " ms. Bandwidth = " << gb/(1.0e+6 * CSR5Spmv_time)
         << " GB/s. GFlops = " << gflop/(1.0e+6 * CSR5Spmv_time)  << " GFlops." << endl;


```

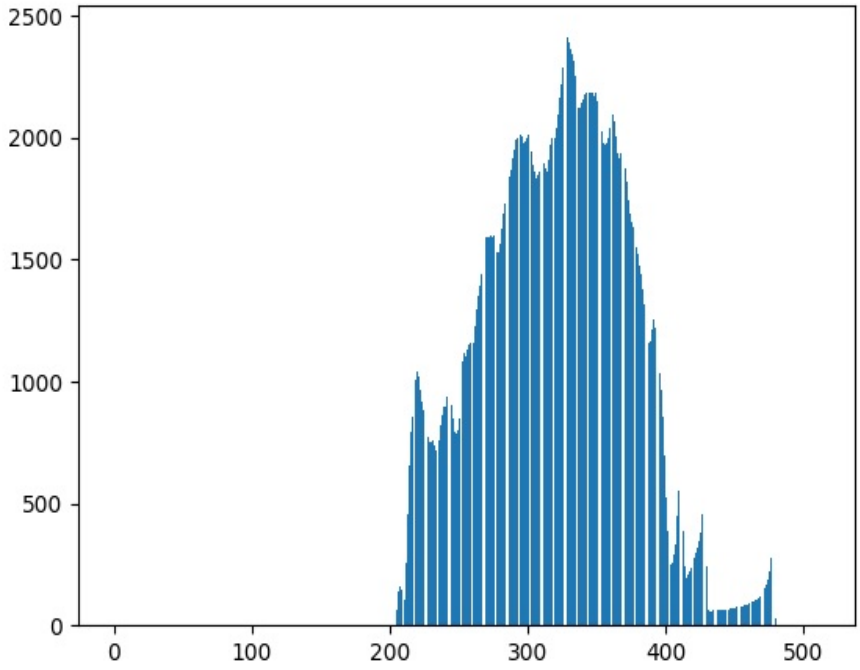
To measure and analysis the performance of the algorithm. We will also track the memory occupation condition for shared memory and global memory. And estimate the theoretical data exchange frequency for the proposed algorithm

Figures

We generated couple bar charts confirming that there is indeed dense submatrices in our CT-scan matricies.







Summary

We used Python to perform some basic data analysis on the medical picture dataset, then imported the results into BSR to get a better understanding. Binary pixels give the images the same size of (512,512). Additionally, we conducted a few quick checks to see if the dataset contained dense sub-matrices, a crucial characteristic for our upcoming work. And we describe the C++-based performance measurement method we used for our project. The dataset's basic feature function graphics are then drawn. The result of the data analysis indicated that our project topic are meaningful and feasible.

In the future we will design the storage method based on BSR, and apply it on CPU and GPU based on C++. And move it to python if possible, to combine with practical sparse matrix multiplication applications. For our research we will be measuring the performance(space and time) of our proposed storage method when doing matrix multiplication.