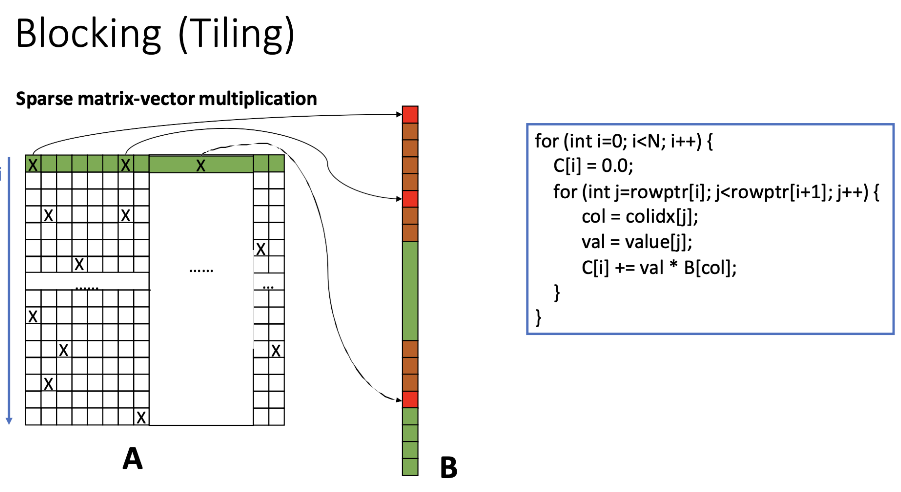
7585-A High Performance Computing problem 2

Group Name: **Ramesh Adhikari** (ID Number: 912147172) and **Maksuda Rabeya** (ID Number: 912154290)

February 2023

Task: Write a program for sparse matrix-vector multiplication, and try to optimize the performance with tiling and unrolling ideas discussed in the context of dense matrices.
Input data: Sparse matrix: copy the file from https://sparse.tamu.edu/SNAP/higgs-twitter. The first row of the input file is the number of rows, the number of columns, number of nonzeros of the sparse matrix Each of the following row represents a nonzero as row index, column index, value. A dense vector assumes all elements in the dense vector are 1.



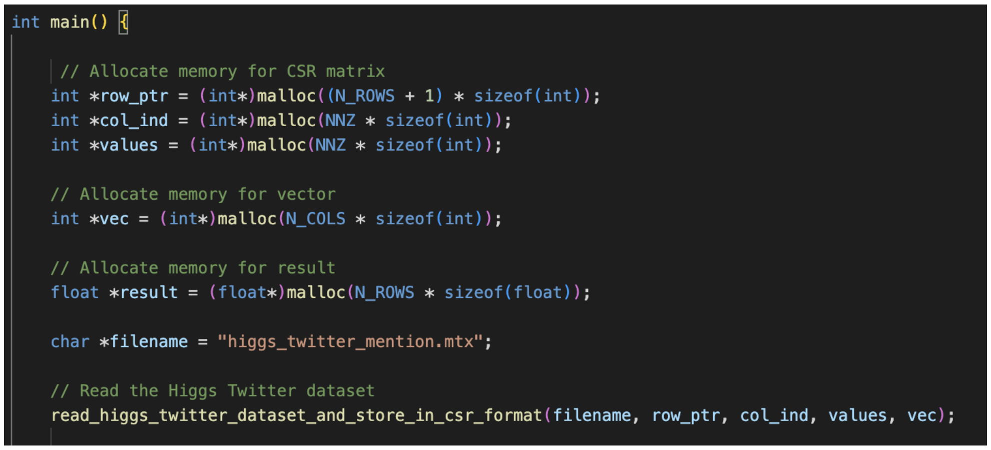
```
for (int i=0; i<N; i++) {
   C[i] = 0.0;
   for (int j=rowptr[i]; j<rowptr[i+1]; j++) {
      col = colidx[j];
      val = value[j];
      C[i] += val * B[col];
   }
}
```

Requirements:

1. Store the sparse matrix in CSR or COO, see for more details.

At first, we read the matrix data from the file Higgs-Twitter-mention.mtx consisting of the Number of Row Index: 456626, Number of Column Index 456626, and the total number of nonzero elements in matrix 150818. Each of the following row consists of a nonzero as row index, column index, and value.

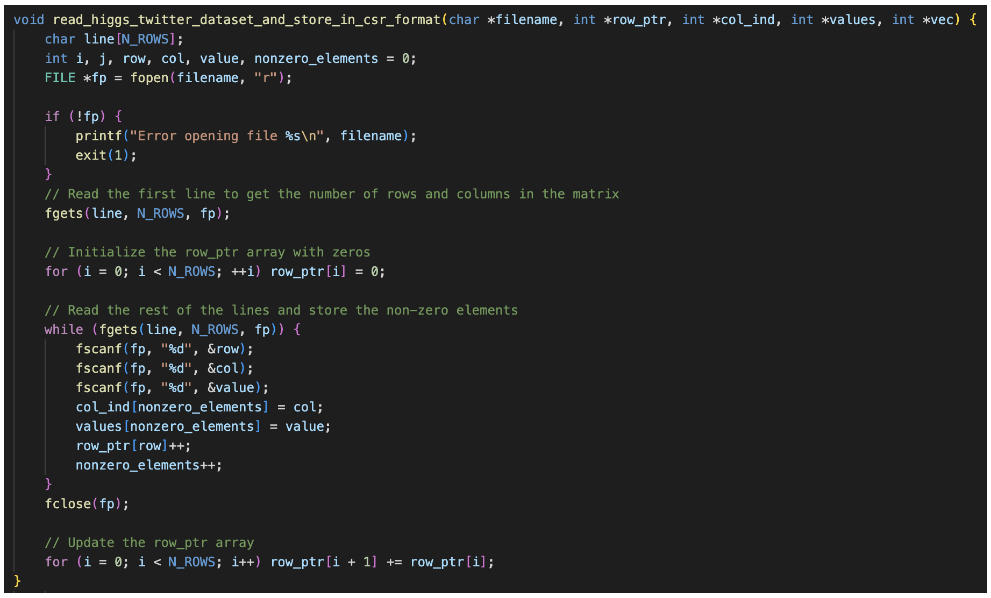
We allocate the memory for the CSR matrix, vector, and result as below. After that, we read the file and store the sparse matrix in CSR format.



```c
int main() {

    // Allocate memory for CSR matrix
    int *row_ptr = (int*)malloc((N_ROWS + 1) * sizeof(int));
    int *col_ind = (int*)malloc(NNZ * sizeof(int));
    int *values = (int*)malloc(NNZ * sizeof(int));

    // Allocate memory for vector
    int *vec = (int*)malloc(N_COLS * sizeof(int));

    // Allocate memory for result
    float *result = (float*)malloc(N_ROWS * sizeof(float));

    char *filename = "higgs_twitter_mention.mtx";

    // Read the Higgs Twitter dataset
    read_higgs_twitter_dataset_and_store_in_csr_format(filename, row_ptr, col_ind, values, vec);
```

Figure 1: Resource allocation

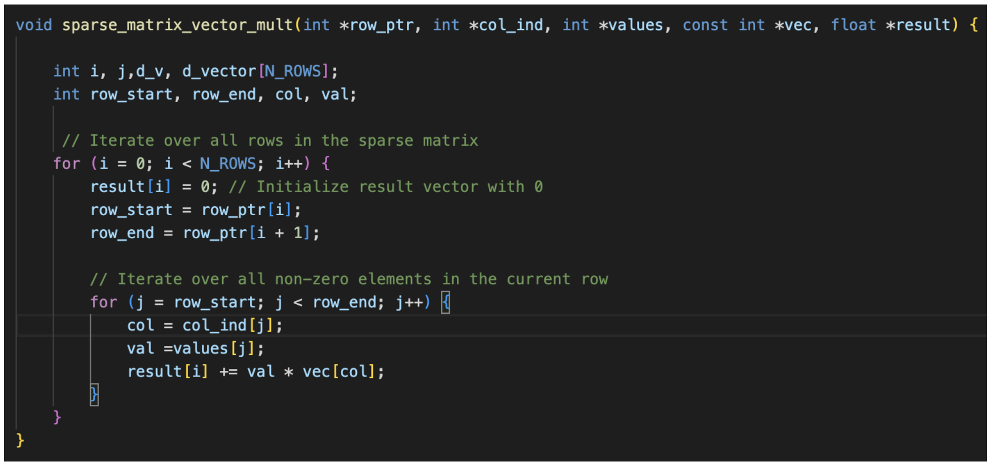
2. Write the program in C or C++, compile the program with g++



```c
void read_higgs_twitter_dataset_and_store_in_csr_format(char *filename, int *row_ptr, int *col_ind, int *values, int *vec) {
    char line[N_ROWS];
    int i, j, row, col, value, nonzero_elements = 0;
    FILE *fp = fopen(filename, "r");

    if (!fp) {
        printf("Error opening file %s\n", filename);
        exit(1);
    }
    // Read the first line to get the number of rows and columns in the matrix
    fgets(line, N_ROWS, fp);

    // Initialize the row_ptr array with zeros
    for (i = 0; i < N_ROWS; ++i) row_ptr[i] = 0;

    // Read the rest of the lines and store the non-zero elements
    while (fgets(line, N_ROWS, fp)) {
        fscanf(fp, "%d", &row);
        fscanf(fp, "%d", &col);
        fscanf(fp, "%d", &value);
        col_ind[nonzero_elements] = col;
        values[nonzero_elements] = value;
        row_ptr[row]++;
        nonzero_elements++;
    }
    fclose(fp);

    // Update the row_ptr array
    for (i = 0; i < N_ROWS; i++) row_ptr[i + 1] += row_ptr[i];
}
```

Figure 2: Store data in CSR format

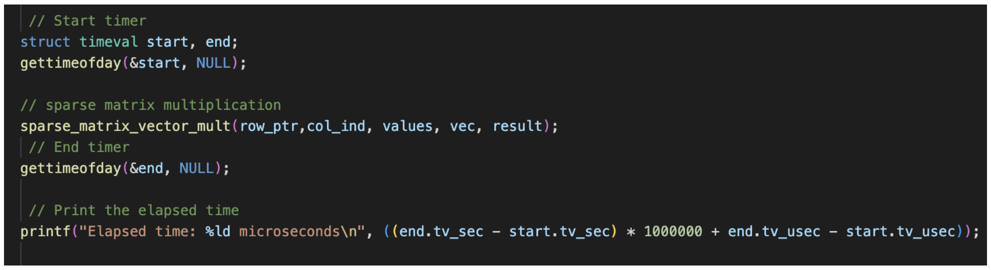
We create the function for sparse matrix-vector multiplication, the screenshot of the code is as below:



```c
void sparse_matrix_vector_mult(int *row_ptr, int *col_ind, int *values, const int *vec, float *result) {

    int i, j,d_v, d_vector[N_ROWS];
    int row_start, row_end, col, val;

     // Iterate over all rows in the sparse matrix
    for (i = 0; i < N_ROWS; i++) {
        result[i] = 0; // Initialize result vector with 0
        row_start = row_ptr[i];
        row_end = row_ptr[i + 1];

        // Iterate over all non-zero elements in the current row
        for (j = row_start; j < row_end; j++) {
            col = col_ind[j];
            val =values[j];
            result[i] += val * vec[col];
        }
    }
}
```

Figure 3: Sparse matrix-vector multiplication without using tiling and unrolling

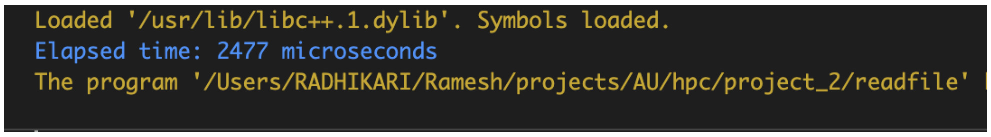
3. Use ‘gettimeofday’ to measure the execution time of the loop.
We use gettimeofday to measure the execution time of the loop. The execution



```c
 // Start timer
struct timeval start, end;
gettimeofday(&start, NULL);

// sparse matrix multiplication
sparse_matrix_vector_mult(row_ptr,col_ind, values, vec, result);
 // End timer
gettimeofday(&end, NULL);

 // Print the elapsed time
printf("Elapsed time: %ld microseconds\n", ((end.tv_sec - start.tv_sec) * 1000000 + end.tv_usec - start.tv_usec));
```

Figure 4: Used gettimeofday to measure the loop elapsed time

time of the sparse matrix-vector multiplication without blocking is found as:

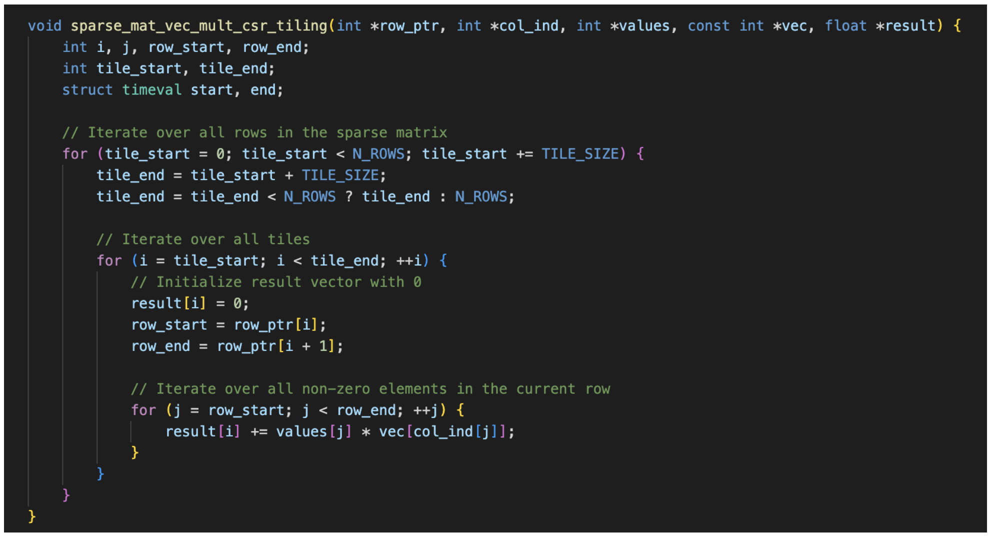


```
Loaded '/usr/lib/libc++.1.dylib'. Symbols loaded.
Elapsed time: 2477 microseconds
The program '/Users/RADHIKARI/Ramesh/projects/AU/hpc/project_2/readfile'
```

Figure 5: Elasped time of sparse matrix-vector multiplication without tiling and unrolling

4. Try to divide the sparse matrix into column blocks, and see if the performance is improved.

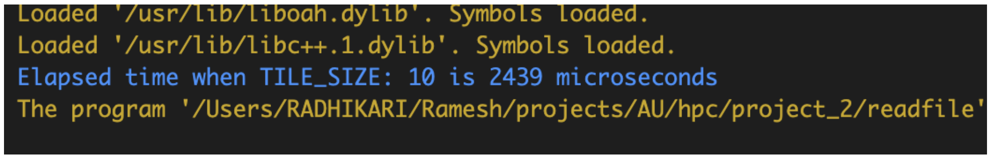
We divide the matrix into blocks (tiles) and improve the performance. The screenshot of the code of the sparse matrix-vector multiplication with blocking (tiling) is as below:



```
void sparse_mat_vec_mult_csr_tiling(int *row_ptr, int *col_ind, int *values, const int *vec, float *result) {
    int i, j, row_start, row_end;
    int tile_start, tile_end;
    struct timeval start, end;

    // Iterate over all rows in the sparse matrix
    for (tile_start = 0; tile_start < N_ROWS; tile_start += TILE_SIZE) {
        tile_end = tile_start + TILE_SIZE;
        tile_end = tile_end < N_ROWS ? tile_end : N_ROWS;

        // Iterate over all tiles
        for (i = tile_start; i < tile_end; ++i) {
            // Initialize result vector with 0
            result[i] = 0;
            row_start = row_ptr[i];
            row_end = row_ptr[i + 1];

            // Iterate over all non-zero elements in the current row
            for (j = row_start; j < row_end; ++j) {
                result[i] += values[j] * vec[col_ind[j]];
            }
        }
    }
}
```

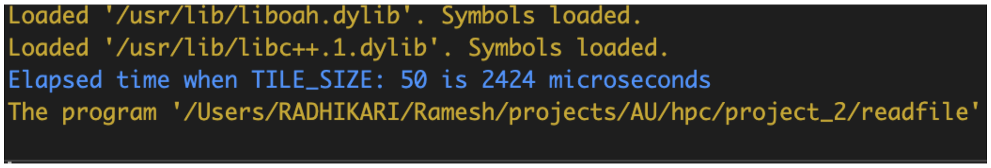
Figure 6: Sparse Matrix vector multiplication with tiling

We capture the execution time of the loop with respect to the tile size. We present the result of using tile sizes 10,50,100,500,10000. We observed that using the tile improves the performance but we have to care about the tile size. Increasing the tile size may not always increase the performance.



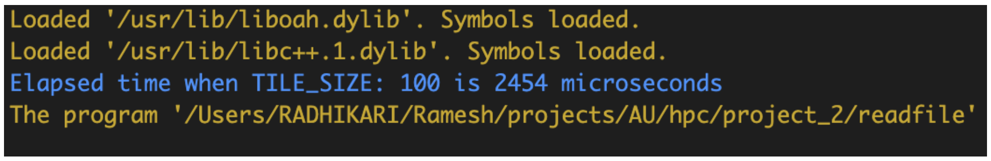
```
Loaded '/usr/lib/liboah.dylib'. Symbols loaded.
Loaded '/usr/lib/libc++.1.dylib'. Symbols loaded.
Elapsed time when TILE_SIZE: 10 is 2439 microseconds
The program '/Users/RADHIKARI/Ramesh/projects/AU/hpc/project_2/readfile'
```

Figure 7: Loop Elapsed time when TILE SIZE is 10



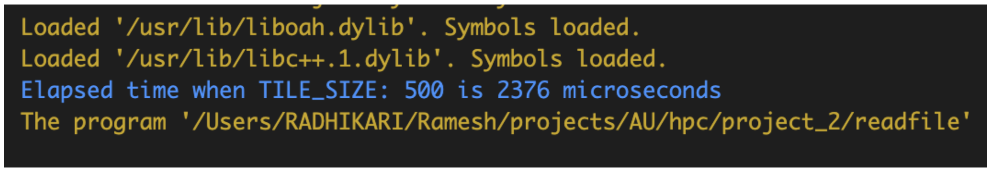
```
Loaded '/usr/lib/liboah.dylib'. Symbols loaded.
Loaded '/usr/lib/libc++.1.dylib'. Symbols loaded.
Elapsed time when TILE_SIZE: 50 is 2424 microseconds
The program '/Users/RADHIKARI/Ramesh/projects/AU/hpc/project_2/readfile'
```

Figure 8: Loop Elapsed time when TILE SIZE is 50



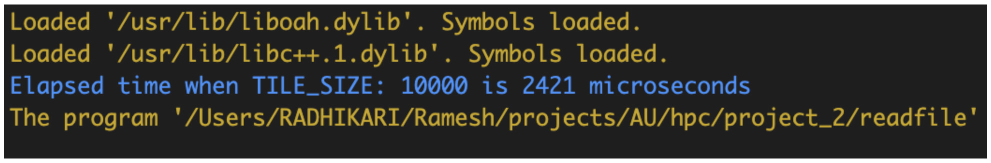
```
Loaded '/usr/lib/liboah.dylib'. Symbols loaded.
Loaded '/usr/lib/libc++.1.dylib'. Symbols loaded.
Elapsed time when TILE_SIZE: 100 is 2454 microseconds
The program '/Users/RADHIKARI/Ramesh/projects/AU/hpc/project_2/readfile'
```

Figure 9: Loop Elapsed time when TILE SIZE is 100



```
Loaded '/usr/lib/liboah.dylib'. Symbols loaded.
Loaded '/usr/lib/libc++.1.dylib'. Symbols loaded.
Elapsed time when TILE_SIZE: 500 is 2376 microseconds
The program '/Users/RADHIKARI/Ramesh/projects/AU/hpc/project_2/readfile'
```

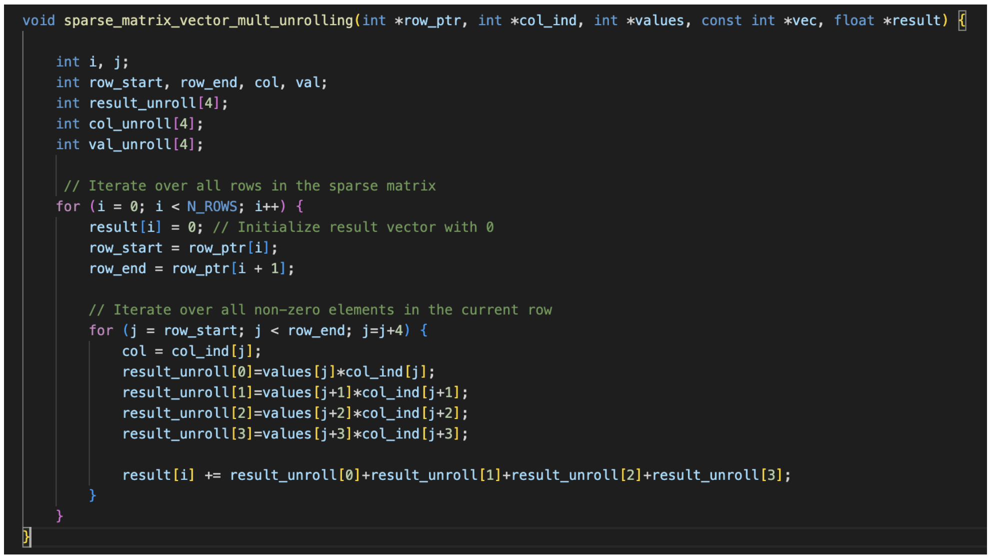
Figure 10: Loop Elapsed time when TILE SIZE is 500



```
Loaded '/usr/lib/liboah.dylib'. Symbols loaded.
Loaded '/usr/lib/libc++.1.dylib'. Symbols loaded.
Elapsed time when TILE_SIZE: 10000 is 2421 microseconds
The program '/Users/RADHIKARI/Ramesh/projects/AU/hpc/project_2/readfile'
```

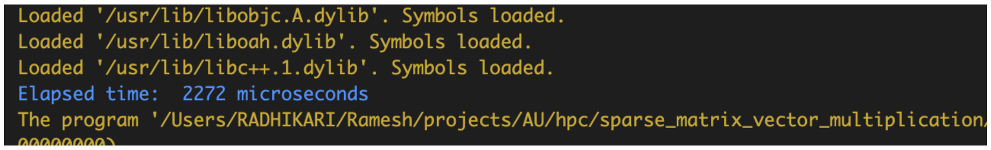
Figure 11: Loop Elapsed time when TILE SIZE is 10000

We tried to optimize the performance of the loop by unrolling and we get some improvement after applying unrolling. At the end of the program execution, we free (released) the allocated memory.



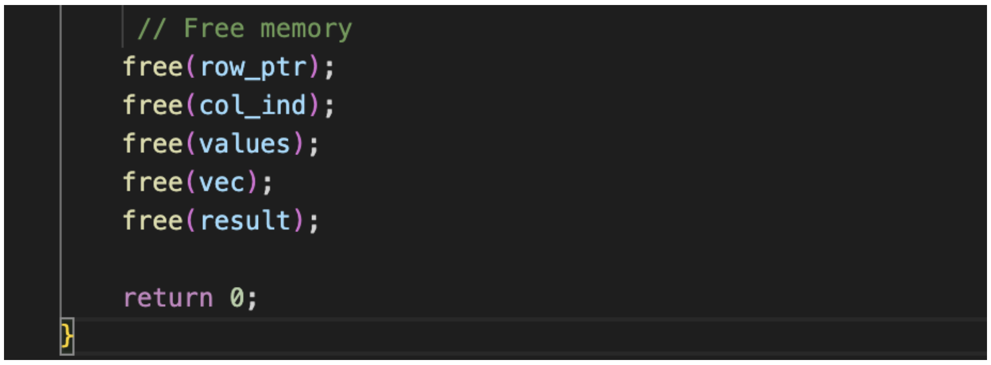
```c
void sparse_matrix_vector_mult_unrolling(int *row_ptr, int *col_ind, int *values, const int *vec, float *result) {

    int i, j;
    int row_start, row_end, col, val;
    int result_unroll[4];
    int col_unroll[4];
    int val_unroll[4];

     // Iterate over all rows in the sparse matrix
    for (i = 0; i < N_ROWS; i++) {
        result[i] = 0; // Initialize result vector with 0
        row_start = row_ptr[i];
        row_end = row_ptr[i + 1];

        // Iterate over all non-zero elements in the current row
        for (j = row_start; j < row_end; j=j+4) {
            col = col_ind[j];
            result_unroll[0]=values[j]*col_ind[j];
            result_unroll[1]=values[j+1]*col_ind[j+1];
            result_unroll[2]=values[j+2]*col_ind[j+2];
            result_unroll[3]=values[j+3]*col_ind[j+3];

            result[i] += result_unroll[0]+result_unroll[1]+result_unroll[2]+result_unroll[3];
        }
    }
}
```

Figure 12: Sparse Matrix vector multiplication with unrolling



```
Loaded '/usr/lib/libobjc.A.dylib'. Symbols loaded.
Loaded '/usr/lib/liboah.dylib'. Symbols loaded.
Loaded '/usr/lib/libc++.1.dylib'. Symbols loaded.
Elapsed time:  2272 microseconds
The program '/Users/RADHIKARI/Ramesh/projects/AU/hpc/sparse_matrix_vector_multiplication/
00000000).
```

Figure 13: Loop Elapsed time after applying unrolling



```
    // Free memory
    free(row_ptr);
    free(col_ind);
    free(values);
    free(vec);
    free(result);

    return 0;
}
```

Figure 14: Release (free) allocate memory

Summary: From this exercise, we learn about sparse matrix and sparse matrix-vector multiplication. We also use tiling and unrolling for performance improvement. And observed that dividing the sparse matrix into blocks (tiles) can help improve performance by reducing the number of cache misses. Moreover, increasing the tile size may not always increase the performance, we have to care about the tile size. Finally, we apply the unrolling technique to improve the performance, and the loop time Elapsed is decreased from 2477 to 2272 microseconds.