LAB2 | 2023

Matrix Multiplication (Multi-Threading)

مروان احمد ربيع احمد **19016605**

1 Introduction

You are required to implement a multi-threaded matrix multiplication program.

The input to the program is two matrices A(xy) and B(yz) that are read from corresponding text files. The output is a matrix C(x*z) that is written to an output text file.

A parallelized version of matrix multiplication can be done using one of these three methods:

1. A thread computes the output C matrix i.e. without multi-threading. (A thread per matrix).

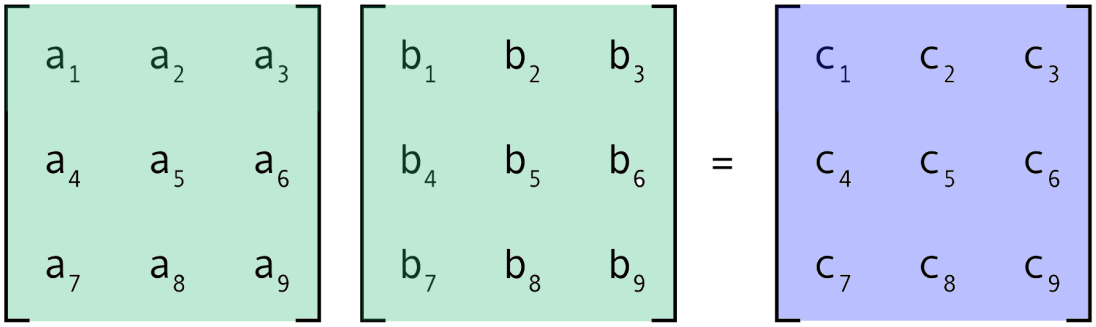

Figure 1: Thread Per Matrix

2. A thread computes each row in the output C matrix. (A thread per row).

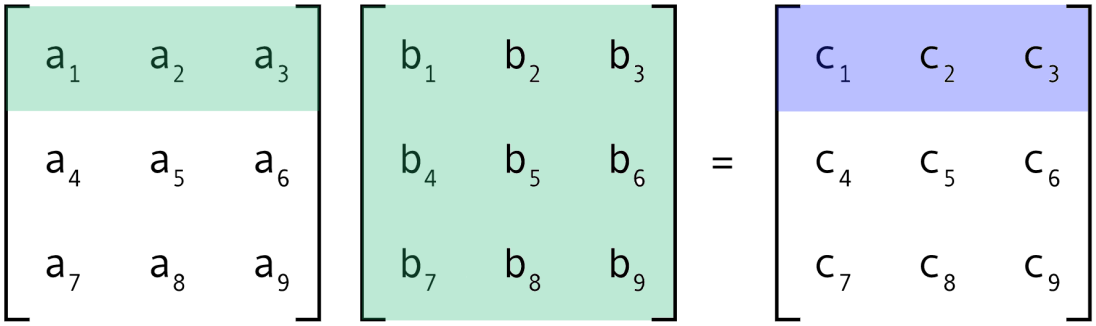

Figure 2: Thread Per Row

3. A thread computes each element in the output C matrix. (A thread per element).

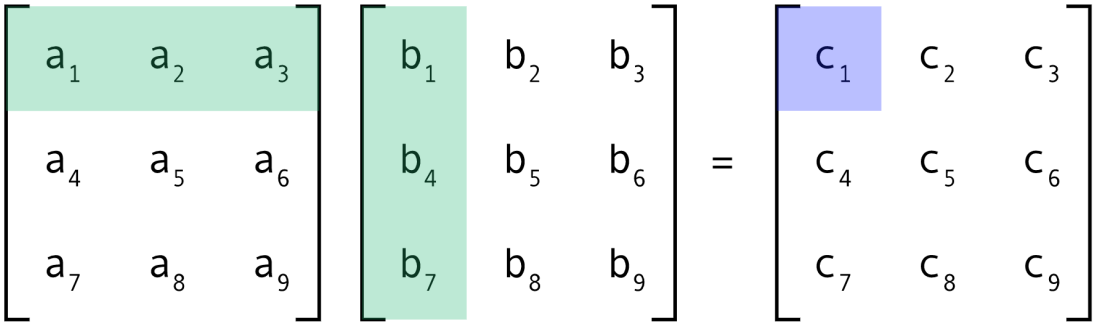

Figure 3: Thread Per Element

2 Test Cases

2.1 Test Case 1

```
Method: A thread per element
row=10 col=10
415     430     445     460     475     490     505     520     535     550
940     980     1020    1060    1100    1140    1180    1220    1260    1300
1465    1530    1595    1660    1725    1790    1855    1920    1985    2050
1990    2080    2170    2260    2350    2440    2530    2620    2710    2800
2515    2630    2745    2860    2975    3090    3205    3320    3435    3550
3040    3180    3320    3460    3600    3740    3880    4020    4160    4300
3565    3730    3895    4060    4225    4390    4555    4720    4885    5050
4090    4280    4470    4660    4850    5040    5230    5420    5610    5800
4615    4830    5045    5260    5475    5690    5905    6120    6335    6550
5140    5380    5620    5860    6100    6340    6580    6820    7060    7300
```

Figure 4: Output of Test Case 1

```
Microseconds taken by method 1: 144
Microseconds taken by method 2: 1192
Microseconds taken by method 3: 4629
```

Figure 5: Time of Test Case 1

2.2 Test Case 2

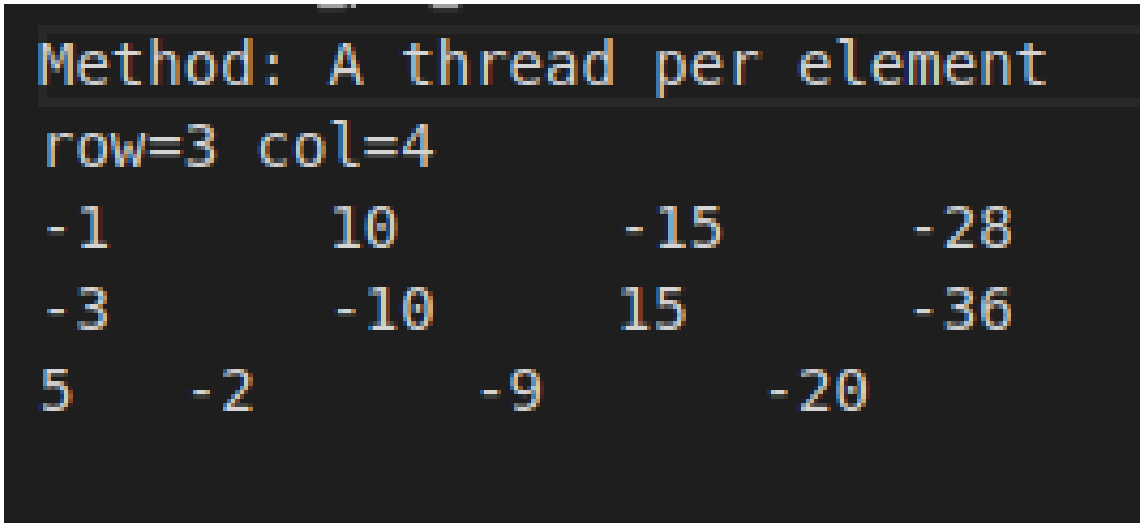

Figure 6: Output of Test Case 2

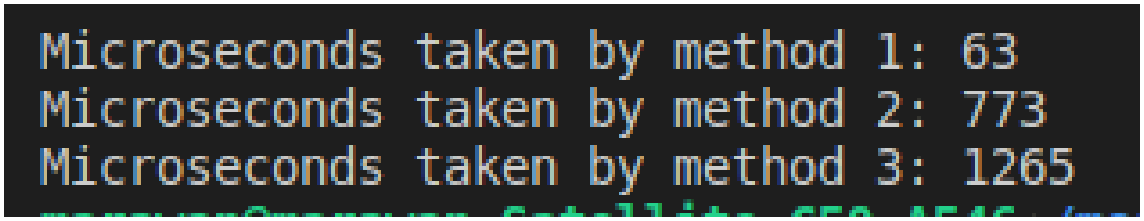

Figure 7: Time of Test Case 2

2.3 Test Case 3

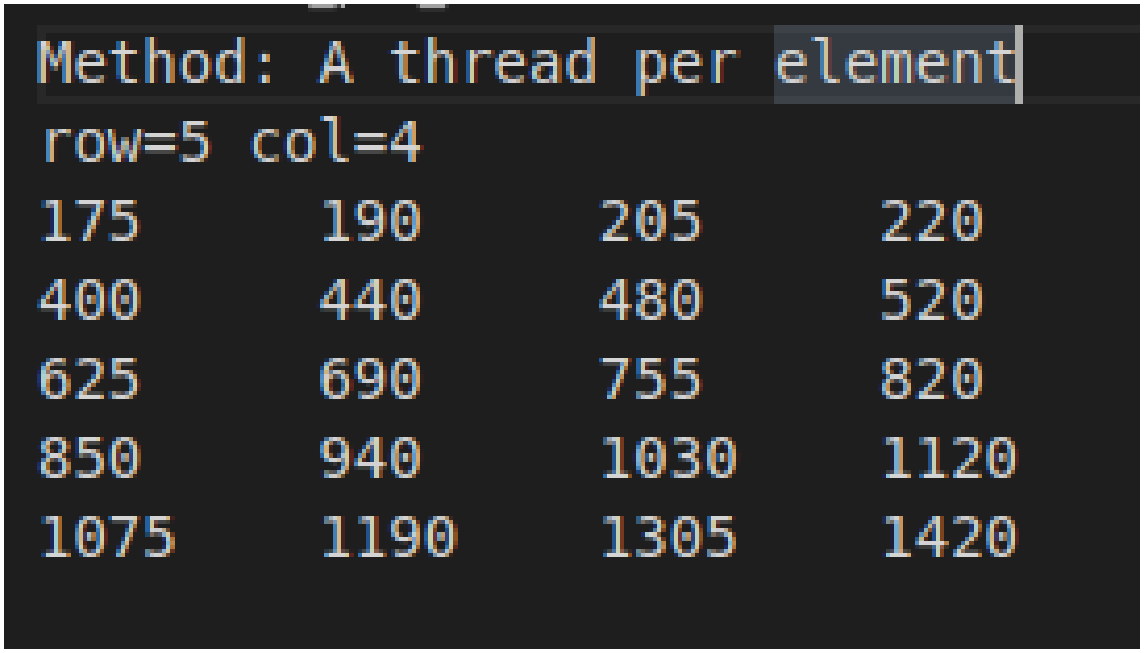

Figure 8: Output of Test Case 3

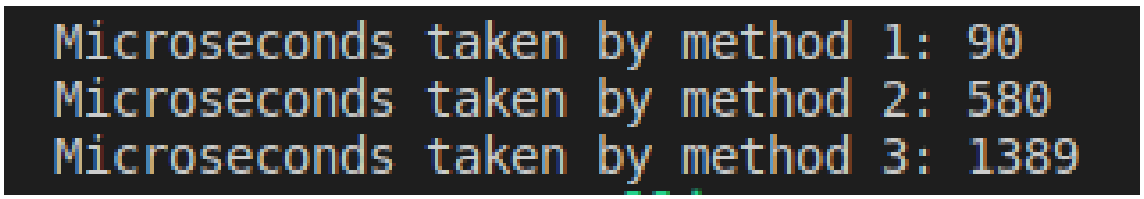

Figure 9: Time of Test Case 3

3 Code

3.1 Includes and type definitions

```c
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <sys/time.h>
// #define MAX_STR_SIZE 100
// int *glbl_matrix_1;
// int *glbl_matrix_2;
int *glbl_matrix_3_r_th;
int *glbl_matrix_3_e_th;

// int glbl_row_1,glbl_row_2,glbl_column_1,glbl_column_2;
char *glbl_output_filename;



typedef struct
{
    int row_1,row_2,col_1,col_2;
    int* matrix_1, *matrix_2;
    int* matrix_3_tpm, *matrix_3_tpr, *matrix_3_tpe;
    int current_row,current_column;
    struct timeval stop_tpm, start_tpm;
    struct timeval stop_tpr, start_tpr;
    struct timeval stop_tpe, start_tpe;
}combined_matrix_struct;
combined_matrix_struct glbl_matrices;
```

3.2 Read the size of matrix

```c

void read_matrix_size (int *row_par, int *column_par, char *file_name){
    char* extension = ".txt";
    char fileSpec[strlen(file_name)+strlen(extension)+1];
    FILE *fp;

    snprintf( fileSpec, sizeof( fileSpec ), "%s%s", file_name,
    extension );

    fp = fopen( fileSpec, "r" );
    fscanf(fp, "row=%d col=%d", row_par,column_par);
    // printf("%d %d", *row_par, *column_par)
    fclose(fp);
}
```

3.3 Read the matrices from file

```
41
42 // void populate_matrix (int *row_par, int *column_par, char *file_name
   ,int matrix[*row_par][*column_par]){
43 void populate_matrix (int row_par, int column_par, char *file_name, int
    *matrix){
44     char* extension = ".txt";
45     char fileSpec[strlen(file_name)+strlen(extension)+1];
46     FILE *fp;
47     // printf("**\n%d, %d\n**\n", row_par,column_par);
48     snprintf( fileSpec, sizeof( fileSpec ), "%s%s", file_name,
   extension );
49     fp = fopen( fileSpec, "r" );
50     fscanf(fp, "%*[^\n]\n");
51
52     for(int i=0;i<row_par;i++)
53     {
54         for(int j=0;j<column_par;j++)
55         {
56             // printf("%d, %d\n", i,j);
57             // int temp;
58             fscanf(fp, "%d ",(matrix+i*column_par+j));
59             // printf("%d ",temp);
60         }
61     }
62 }
```

3.4 Thread per matrix

```
63
64 void write_matrix_to_file(int *matrix,int row,int column, char *
   filename, char* extension, char* message){
65     char fileSpec[strlen(filename)+strlen(extension)+1];
66     FILE *fp;
67     snprintf( fileSpec, sizeof( fileSpec ), "%s%s", filename, extension
       );
68     if((fp=fopen(fileSpec, "w"))==NULL) {
69         printf("Cannot open file.\n");
70     }
71     fwrite(message, sizeof(char), strlen(message), fp);
72     fprintf(fp, "\n");
73     fprintf(fp, "row=%d col=%d", row,column);
74     fprintf(fp, "\n");
75   for(int i=0; i<row; i++){
76     for(int j=0; j<column; j++){
77       fprintf(fp, "%d  \t", *(matrix + i * column+ j));
78     }
79     fprintf(fp, "\n");
80   }
81   fclose(fp);
82 }
83
84 void thread_per_matrix(combined_matrix_struct* lcl_matrices, char *
   file_name){
85     gettimeofday(&glbl_matrices.start_tpm, NULL); //start checking time
86     // int matrix_3[*row_par_1][*column_par_2];
87     int out_row = lcl_matrices->row_1;
88     int out_col = lcl_matrices->col_2;
89     int inner_dim = lcl_matrices->col_1;
90     int *lcl_matrix_1 = lcl_matrices->matrix_1;
```

```
91      int *lcl_matrix_2 = lcl_matrices->matrix_2;
92      int *matrix_3 = lcl_matrices->matrix_3_tpm;
93
94      for (int i = 0; i < out_row; i++) {
95          for (int j = 0; j < out_col; j++) {
96              *(matrix_3 + i * out_col+ j) = 0;
97              for (int k = 0; k < inner_dim; k++) {
98                  *(matrix_3 + i * out_col+ j) += *(lcl_matrix_1+i*
    inner_dim+k) * *(lcl_matrix_2+k*out_col+j);
99              }
100         }
101     }
102     printf("\nThe matrix using thread per matrix is : \n");
103     for (int i = 0; i < out_row; i++) {
104         printf("\n");
105         for (int j = 0; j < out_col; j++) {
106             printf("%d\t",*(matrix_3 + i * out_col+ j));
107         }
108     }
109     printf("\n\n");
110     gettimeofday(&glbl_matrices.stop_tpm, NULL); //end checking time
111 }
```

3.5 Thread per row

```
112
113 //Each thread computes single element in the resultant matrix
114 void *multiply_row(void* arg){
115     combined_matrix_struct *lcl_matrices;
116     lcl_matrices = (combined_matrix_struct *) arg;
117     // int current_row = (int *)arg;
118     int current_row = lcl_matrices->current_row;
119     int out_row = lcl_matrices->row_1;
120     int out_col = lcl_matrices->col_2;
121     int inner_dim = lcl_matrices->col_1;
122     int *lcl_matrix_1 = lcl_matrices->matrix_1;
123     int *lcl_matrix_2 = lcl_matrices->matrix_2;
124     int *matrix_3 = glbl_matrices.matrix_3_tpr;
125     // printf("%d ", out_col);
126     // printf("%d %d %d %d\n", out_row,out_col,inner_dim, current_row);
127     for (int j = 0; j < out_col; j++) {
128         // printf("%d",j);
129         *(matrix_3+current_row*out_col+j) = 0;
130         for (int k = 0; k < inner_dim; k++) {
131             *(matrix_3+current_row*out_col+j) += *(lcl_matrix_1+
    current_row*inner_dim+k) * *(lcl_matrix_2+k*out_col+j);
132         }
133         // printf("%d", *(glbl_matrix_3_r_th+current_row*out_col+j));
134     }
135     pthread_exit(1);
136 }
137
138 void thread_per_row(void){
139     gettimeofday(&glbl_matrices.start_tpr, NULL); //start checking time
140     pthread_t threads[glbl_matrices.row_1];
141
142     for (int i = 0; i < glbl_matrices.row_1; i++) {
143         combined_matrix_struct *lcl_matrices;
144         lcl_matrices=(combined_matrix_struct *)malloc(sizeof(
    combined_matrix_struct));
145         memcpy(lcl_matrices, &glbl_matrices, sizeof(
```

```
    combined_matrix_struct));
146         lcl_matrices->current_row = i;
147         pthread_create(&threads[i], NULL, multiply_row, (void*)(
    lcl_matrices));
148     }
149
150     // joining and waiting for all threads to complete
151     for (int i = 0; i < glbl_matrices.row_1; i++)
152         pthread_join(threads[i], NULL);
153     printf("\nThe matrix using thrad per row is : \n");
154     for(int i=0;i<glbl_matrices.row_1;i++)
155     {
156         printf("\n");
157         for(int j=0;j<glbl_matrices.col_2;j++)
158         {
159             printf("%d\t",*(glbl_matrices.matrix_3_tpr + i *
    glbl_matrices.col_2 + j));
160         }
161     }
162     printf("\n\n");
163     gettimeofday(&glbl_matrices.stop_tpr, NULL); //end checking time
164 }
```

3.6 Thread per element

```
166
167 void *multiply_element(void* arg){
168     combined_matrix_struct *lcl_matrices;
169     lcl_matrices = (combined_matrix_struct *) arg;
170     int current_row = lcl_matrices->current_row;
171     int current_column = lcl_matrices->current_column;
172     int out_row = lcl_matrices->row_1;
173     int out_col = lcl_matrices->col_2;
174     int inner_dim = lcl_matrices->col_1;
175     int *lcl_matrix_1 = lcl_matrices->matrix_1;
176     int *lcl_matrix_2 = lcl_matrices->matrix_2;
177     int *matrix_3 = glbl_matrices.matrix_3_tpe;
178
179    *(matrix_3+current_row*out_col+current_column) = 0;
180
181     for (int k = 0; k < inner_dim; k++) {
182         *(matrix_3+current_row*out_col+current_column) += *(
    lcl_matrix_1+current_row*inner_dim+k) * *(lcl_matrix_2+k*out_col+
    current_column);
183     }
184     pthread_exit(1);
185 }
186
187 void thread_per_element(void){
188     gettimeofday(&glbl_matrices.start_tpe, NULL); //start checking time
189     pthread_t threads[glbl_matrices.row_1][glbl_matrices.col_2];
190
191     for (int i = 0; i < glbl_matrices.row_1; i++) {
192         for (int j = 0; j < glbl_matrices.col_2; j++) {
193             combined_matrix_struct *lcl_matrices;
194             lcl_matrices=(combined_matrix_struct *)malloc(sizeof(
    combined_matrix_struct));
195             memcpy(lcl_matrices, &glbl_matrices, sizeof(
    combined_matrix_struct));
196             lcl_matrices->current_row = i;
197             lcl_matrices->current_column = j;
```

```
198             pthread_create(&threads[i][j], NULL, multiply_element, (
    void*)(lcl_matrices));
199         }
200     }
201 
202     // joining and waiting for all threads to complete
203     for (int i = 0; i < glbl_matrices.row_1; i++) {
204         for (int j = 0; j < glbl_matrices.col_2; j++) {
205             pthread_join(threads[i][j], NULL);
206         }
207     }
208      printf("\nThe matrix using thread per element is : \n");
209     for(int i=0;i<glbl_matrices.row_1;i++)
210     {
211         printf("\n");
212         for(int j=0;j<glbl_matrices.col_2;j++)
213         {
214             printf("%d\t",*(glbl_matrices.matrix_3_tpe + i *
    glbl_matrices.col_2 + j));
215         }
216     }
217     printf("\n\n");
218     gettimeofday(&glbl_matrices.stop_tpe, NULL); //end checking time
219 }
220 void main (int argc, char **argv){
```

3.7 Main

```
220 void main (int argc, char **argv){
221     // printf("You have entered %d arguments:\n", argc);
222     char *input_filename_1;
223     char *input_filename_2;
224     char *out_filename;
225 
226     if (argc == 4){
227         // char *input_filename_1,*input_filename_2;
228         // char *out_filename;
229         input_filename_1 = (char*) malloc((strlen(argv[1]) + 1) *
    sizeof(char));
230         strcpy(input_filename_1,argv[1]);
231         input_filename_2 = (char*) malloc((strlen(argv[2]) + 1) *
    sizeof(char));
232         strcpy(input_filename_2,argv[2]);
233         out_filename = (char*) malloc((strlen(argv[3]) + 1) * sizeof(
    char));
234         strcpy(out_filename,argv[3]);
235     }
236     else{
237         input_filename_1 = (char*) malloc((strlen("a") + 1) * sizeof(
    char));
238         strcpy(input_filename_1,"a");
239         input_filename_2 = (char*) malloc((strlen("b") + 1) * sizeof(
    char));
240         strcpy(input_filename_2,"b");
241         out_filename = (char*) malloc((strlen("c") + 1) * sizeof(char))
    ;
242         strcpy(out_filename,"c");
243     }
244 
245     // printf("%s\n", input_filename_1);
246     // printf("%s\n", input_filename_2);
```

```
247     // printf("%s\n", out_filename);
248     read_matrix_size (&glbl_matrices.row_1, &glbl_matrices.col_1,
    input_filename_1);
249     read_matrix_size (&glbl_matrices.row_2, &glbl_matrices.col_2,
    input_filename_2);
250 
251     glbl_matrices.matrix_1 = (int*) malloc(glbl_matrices.row_1 *
    glbl_matrices.col_1 * sizeof(int));
252     glbl_matrices.matrix_2 = (int*) malloc(glbl_matrices.row_2 *
    glbl_matrices.col_2 * sizeof(int));
253 
254     populate_matrix (glbl_matrices.row_1, glbl_matrices.col_1,
    input_filename_1, glbl_matrices.matrix_1);
255     populate_matrix (glbl_matrices.row_2, glbl_matrices.col_2,
    input_filename_2, glbl_matrices.matrix_2);
256 
257     glbl_matrices.matrix_3_tpm = (int*) malloc(glbl_matrices.row_1 *
    glbl_matrices.col_2 * sizeof(int));
258     glbl_matrices.matrix_3_tpr = (int*) malloc(glbl_matrices.row_1 *
    glbl_matrices.col_2 * sizeof(int));
259     glbl_matrices.matrix_3_tpe = (int*) malloc(glbl_matrices.row_1 *
    glbl_matrices.col_2 * sizeof(int));
260 
261     thread_per_matrix(&glbl_matrices, "c");
262     thread_per_row();
263     thread_per_element();
264 
265     // printf("Seconds taken by method 1%lu\n", stop.tv_sec - start.
    tv_sec);
266     printf("Microseconds taken by method 1: %lu\n", glbl_matrices.
    stop_tpm.tv_usec - glbl_matrices.start_tpm.tv_usec);
267     printf("Microseconds taken by method 2: %lu\n", glbl_matrices.
    stop_tpr.tv_usec - glbl_matrices.start_tpr.tv_usec);
268     printf("Microseconds taken by method 3: %lu\n", glbl_matrices.
    stop_tpe.tv_usec - glbl_matrices.start_tpr.tv_usec);
269 
270     write_matrix_to_file(glbl_matrices.matrix_3_tpm, glbl_matrices.
    row_1, glbl_matrices.col_2, out_filename, "_per_matrix.txt", "
    Method: A thread per matrix");
271     write_matrix_to_file(glbl_matrices.matrix_3_tpr, glbl_matrices.
    row_1, glbl_matrices.col_2, out_filename, "_per_row.txt", "Method:
    A thread per row");
272     write_matrix_to_file(glbl_matrices.matrix_3_tpe, glbl_matrices.
    row_1, glbl_matrices.col_2, out_filename, "_per_element.txt", "
    Method: A thread per element");
273 
274     free(glbl_matrices.matrix_1);
275     free(glbl_matrices.matrix_2);
276     free(glbl_matrices.matrix_3_tpm);
277     free(glbl_matrices.matrix_3_tpr);
278     free(glbl_matrices.matrix_3_tpe);
279 }
```