

Universidade de São Paulo
Instituto de Física de São Carlos

**High Performance
Programming-Parallel MPI**

Éverton Luís Mendes da Silva (10728171)

Contents

1	Resume	2
2	Theoretical Introduction	2
3	Description of files	3
3.1	Code Linked List	3
3.2	Code FindEigen - Sequential	7
3.3	Code FindEigen_omp - Parallel	11
3.4	Code FindEigen_mpi - Parallel	16
4	Performance analysis	22
4.1	Small	23
4.2	Medium	24
4.3	Large	26
4.4	Huge	27
4.5	All size files	29
4.6	Comparisons	30
5	Conclusions	33
6	Reference	33

1 Resume

Currently, there are several applications in which, given a matrix A , the eigenvalues and eigenvectors are relevant information to solve the problem. With this in mind, there are several methods to find these values, however, most of them do not take into account the limitation of current computers in dealing with large memories and processing speed of current CPUs (approximately 10^8 iterations per second). As an example, if we have a problem with a large number of variables (N), algorithms that have a complexity of N^2 ($N \rightarrow \infty$, $\Theta(N^2)$) are unfeasible for solution due to the high processing time. Therefore, in this project, the representation of a graph as a weighted adjacency matrix was discarded, thus the multiplication of a matrix by a vector was implemented with an weighted adjacency list (weighted linked list). Finally, the power iteration method for matrix multiplication was implemented in two ways: sequential and parallel.

2 Theoretical Introduction

The multiplication of a matrix by a vector can be done in two ways, one with matrix representation and the other with an adjacency list, as below:

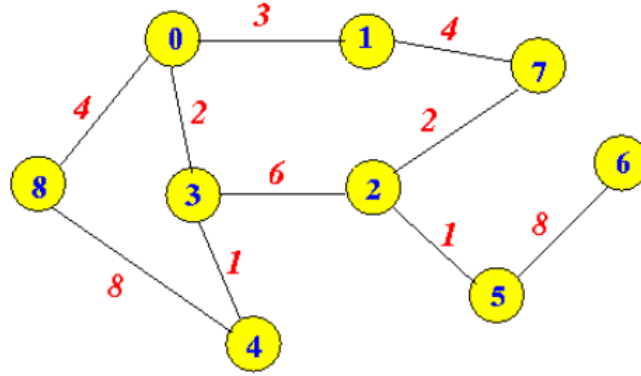


Figure 1: Graph

	0	1	2	3	4	5	6	7	8
0	0	3	0	2	0	0	0	0	4
1	3	0	0	0	0	0	0	4	0
2	0	0	0	6	0	1	0	2	0
3	2	0	6	0	1	0	0	0	0
4	0	0	0	1	0	0	0	0	8
5	0	0	1	0	0	0	8	0	0
6	0	0	0	0	0	8	0	0	0
7	0	4	2	0	0	0	0	0	0
8	4	0	0	0	8	0	0	0	0

Figure 2: Adjacency Matrix

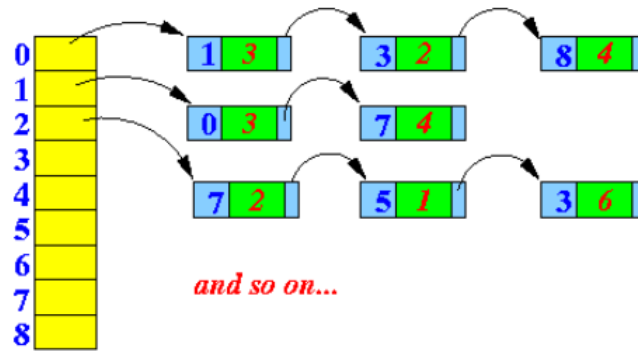


Figure 3: Adjacency List

3 Description of files

This project was divided into four main files that will be presented below. The first consists of a file, named 'AdjList', which reads a file in Pajek format and creates an adjacency list with the respective values. In addition, we have two more files to perform the power iteration method, one sequential and the two other parallels.

3.1 Code Linked List

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 struct Graph
5 {
6     struct Node *head[1];
7 };
8
9 struct Edge
10 {
11     int i, j;
12     double weight;
13 };
14
15 struct Node
16 {
17     int dest;
18     double weight;
19     struct Node *next;
20 };
21
22 struct Graph *createGraph(struct Edge edges[], int n, int
n_vertexs)
23 {
24     struct Graph *graph = (struct Graph *)malloc(sizeof(
struct Graph) + sizeof(struct Node) * (n_vertexs - 1));
25
26     for (int i = 0; i < n_vertexs; i++)
27     {
28         graph->head[i] = NULL;
29     }
30
31     for (int i = 0; i < n; i++)
32     {
33         int src = edges[i].i;
34         int dest = edges[i].j;
35         double weight = edges[i].weight;
36
37         struct Node *newNode = (struct Node *)malloc(sizeof(
struct Node));
38         newNode->dest = dest;
39         newNode->weight = weight;
40
41         newNode->next = graph->head[src];
42
43         graph->head[src] = newNode;
44     }
45     return graph;
46 }

```

```

47
48 void printGraph(struct Graph *graph, int n_vertexs)
49 {
50     int i;
51     for (i = 0; i < n_vertexs; i++)
52     {
53         struct Node *ptr = graph->head[i];
54         while (ptr != NULL)
55         {
56             printf("%d > %d ", i, ptr->dest);
57             ptr = ptr->next;
58         }
59
60         printf("\n");
61     }
62 }
63
64 void returnNumberNeighbors(struct Graph *graph, int n_vertexs
, int *n_neighbors)
65 {
66     for (int i = 0; i < n_vertexs; i++)
67     {
68         struct Node *ptr = graph->head[i];
69         while (ptr != NULL)
70         {
71             ptr = ptr->next;
72             n_neighbors[i]++;
73         }
74     }
75 }
76
77 void neighborsWeight(struct Graph *graph, int vertex, int
n_neighbors, double *weights)
78 {
79
80     struct Node *ptr = graph->head[vertex];
81     for (int i=0; i<n_neighbors; ++i)
82     {
83         weights[i]=ptr->weight;
84         ptr = ptr->next;
85     }
86 }
87
88
89 struct File_data
90 {
91     int n_vertexs;
92     struct Graph *graph;
93 };

```

```

94
95 struct File_data *ReadPajek(char *filename)
96 {
97     FILE *fp;
98     fp = fopen(filename, "r");
99
100     int n_vertexs, n_edges;
101     int i_element, j_element;
102     double weight;
103
104     if (fscanf(fp, "%d", &n_vertexs))
105     {
106     }
107     if (fscanf(fp, "%d ", &n_edges))
108     {
109     }
110
111     struct Edge *Edges = malloc(2 * n_edges * sizeof(struct
Edge));
112
113     int count_equal_ij = 0;
114     for (int edge = 0; edge < n_edges; edge++)
115     {
116         if (fscanf(fp, "%d %d %lf", &i_element, &j_element, &
weight))
117         {
118         }
119
120         if (i_element != j_element)
121         {
122             Edges[2 * edge - count_equal_ij].i = i_element;
123             Edges[2 * edge - count_equal_ij].j = j_element;
124             Edges[2 * edge - count_equal_ij].weight = weight;
125             Edges[2 * edge + 1 - count_equal_ij].i =
j_element;
126             Edges[2 * edge + 1 - count_equal_ij].j =
i_element;
127             Edges[2 * edge + 1 - count_equal_ij].weight =
weight;
128         }
129         else
130         {
131             Edges[2 * edge - count_equal_ij].i = i_element;
132             Edges[2 * edge - count_equal_ij].j = j_element;
133             Edges[2 * edge - count_equal_ij].weight = weight;
134             count_equal_ij += 1;
135         }
136     }
137

```

```

138     fclose(fp);
139
140     struct Graph *graph = createGraph(Edges, 2 * n_edges -
count_equal_ij, n_vertices);
141
142     struct File_data *file_data = malloc(sizeof(int) + sizeof
(struct Graph) + sizeof(struct Node) * (n_vertices - 1));
143     file_data->n_vertices = n_vertices;
144     file_data->graph = graph;
145
146     return file_data;
147 }

```

Listing 1: Read Pajek file and Linked List Representation

3.2 Code FindEigen - Sequential

```

1  #include "AdjListT.c"
2  #include <math.h>
3  #include <string.h>
4  #include <stdio.h>
5  #include <stdlib.h>
6  #include <sys/time.h>
7  #include <unistd.h>
8
9  void read_arguments_or_abort(int argc, char *argv[]);
10 double normalize_vec(int n_vertices, double *vector);
11 void mat_mult_AdjList(struct Graph *graph, double *vector,
double *new_vector, int n_vertices);
12 void printfvector(double *vector, int n_vertices);
13 void cleanVector(double *vector, int n_vertices);
14 double mult_pointers(double num1, double num2);
15 void copy_vec(double *vector, double *new_vector, int
n_vertices);
16
17 int main(int argc, char *argv[])
18 {
19     read_arguments_or_abort(argc, argv);
20     char *input_filename = argv[1];
21
22     double precision;
23     sscanf(argv[2], "%lf", &precision);
24     char *output_filename = argv[3];
25
26     struct File_data *file_data = ReadPajek(input_filename);
27
28     double *vec = (double *)malloc(file_data->n_vertices *
sizeof(double));
29     double *new_vec = (double *)malloc(file_data->n_vertices *

```



```

30     sizeof(double));
31
32     for (int i = 0; i < file_data->n_vertexs; i++)
33     {
34         vec[i] = rand() / (RAND_MAX + 1.0);
35         if (rand() / (RAND_MAX + 1.0) >= 0.5)
36         {
37             vec[i] *= -1;
38         }
39     }
40
41     int stop_iter = 0;
42     double norm_vec, new_norm_vec;
43
44     norm_vec = normalize_vec(file_data->n_vertexs, vec);
45
46     struct timeval t1, t2;
47     gettimeofday(&t1, NULL);
48     while (stop_iter < 3)
49     {
50         cleanVector(new_vec, file_data->n_vertexs);
51         mat_mult_AdjList(file_data->graph, vec, new_vec,
52         file_data->n_vertexs);
53         new_norm_vec = normalize_vec(file_data->n_vertexs,
54         new_vec);
55         copy_vec(vec, new_vec, file_data->n_vertexs);
56
57         if (fabs(new_norm_vec - norm_vec) / new_norm_vec <
58         precision)
59         {
60             stop_iter += 1;
61         }
62         else
63         {
64             stop_iter = 0;
65         }
66
67         norm_vec = new_norm_vec;
68     }
69     gettimeofday(&t2, NULL);
70
71     printf("It took %.17lf milliseconds.\n", (t2.tv_sec - t1.
72     tv_sec) + (t2.tv_usec - t1.tv_usec) / 1e6);
73
74     FILE *output_file;
75     output_file = fopen(output_filename, "w");
76     fprintf(output_file, "%lf\n", new_norm_vec);
77     fprintf(output_file, "%d\n", file_data->n_vertexs);

```

```

74     for (int i = 0; i < file_data->n_vertexs; i++)
75     {
76         fprintf(output_file, "%lf\n", vec[i]);
77     }
78
79     fclose(output_file);
80
81     cleanVector(new_vec, file_data->n_vertexs);
82     mat_mult_AdjList(file_data->graph, vec, new_vec,
83 file_data->n_vertexs);
84     new_norm_vec = normalize_vec(file_data->n_vertexs,
85 new_vec);
86
87     if (fabs(new_norm_vec - norm_vec) / new_norm_vec <
88 precision)
89     {
90         printf("The Method works well\n");
91     }
92     else
93     {
94         printf("The Method don't work so well\n");
95         printf("method: %.17lf precision: %.17lf\n", fabs(
96 new_norm_vec - norm_vec) / new_norm_vec, precision);
97     }
98
99     FILE *time_record_file;
100     char timefilename[100] = "time_";
101     strcat(timefilename, input_filename);
102     printf("%s\n", timefilename);
103     time_record_file = fopen(timefilename, "a+");
104     fprintf(time_record_file, "%.10lf\n", (t2.tv_sec - t1.
105 tv_sec) + (t2.tv_usec - t1.tv_usec) / 1e6);
106     fclose(time_record_file);
107
108     return 0;
109 }
110
111 void read_arguments_or_abort(int argc, char *argv[])
112 {
113     if (argc != 4)
114     {
115         fprintf(stderr, "Usage: %s <number of elements> <
116 number of arrays>\n",
117             argv[0]);
118         exit(505);
119     }
120 }
121
122 double normalize_vec(int n_vertexs, double *vector)

```

```

117 {
118
119     double sum_elements = 0;
120     for (int i = 0; i < n_vertexs; i++)
121     {
122         sum_elements += pow(vector[i], 2);
123     }
124
125     for (int i = 0; i < n_vertexs; i++)
126     {
127         vector[i] /= sqrt(sum_elements);
128     }
129
130     return sqrt(sum_elements);
131 }
132
133 void mat_mult_AdjList(struct Graph *graph, double *vector,
134                     double *new_vector, int n_vertexs)
135 {
136     for (int i = 0; i < n_vertexs; i++)
137     {
138         struct Node *ptr = graph->head[i];
139
140         if (ptr == NULL)
141         {
142         }
143         else
144         {
145             while (ptr != NULL)
146             {
147                 new_vector[i] += mult_pointers(ptr->weight,
148 vector[ptr->dest]);
149                 ptr = ptr->next;
150             }
151         }
152     }
153
154 void printfvector(double *vector, int n_vertexs)
155 {
156     for (int i = 0; i < n_vertexs; i++)
157     {
158         printf("%lf ", vector[i]);
159     }
160     printf("\n");
161 }
162
163 void cleanVector(double *Clean_vector, int n_vertexs)

```

```

164 {
165     double zero = 0;
166     for (int i = 0; i < n_vertexs; i++)
167     {
168         Clean_vector[i] = zero;
169     }
170 }
171
172 double mult_pointers(double num1, double num2)
173 {
174     double aux1 = num1;
175     double aux2 = num2;
176     double mult_value = aux1 * aux2;
177     return mult_value;
178 }
179
180 void copy_vec(double *vector, double *new_vector, int
n_vertexs)
181 {
182     for (int i = 0; i < n_vertexs; i++)
183     {
184         double aux = new_vector[i];
185         vector[i] = aux;
186     }
187 }

```

Listing 2: Power iteratrion - Sequential

3.3 Code FindEigen_omp - Parallel

```

1  #include "AdjListT.c"
2  #include <math.h>
3  #include <string.h>
4  #include <stdio.h>
5  #include <stdlib.h>
6  #include <sys/time.h>
7  #include <unistd.h>
8
9  void read_arguments_or_abort(int argc, char *argv[]);
10 double normalize_vec(int n_vertexs, double *vector);
11 void mat_mult_AdjList(struct Graph *graph, double *vector,
double *new_vector, int n_vertexs);
12 void printfvector(double *vector, int n_vertexs);
13 void cleanVector(double *vector, int n_vertexs);
14 double mult_pointers(double num1, double num2);
15 void copy_vec(double *vector, double *new_vector, int
n_vertexs);
16
17 int main(int argc, char *argv[])

```

```

18 {
19
20     read_arguments_or_abort(argc, argv);
21     char *input_filename = argv[1];
22
23     double precision;
24     sscanf(argv[2], "%lf", &precision);
25
26     char *output_filename = argv[3];
27
28     struct File_data *file_data = ReadPajek(input_filename);
29
30     double *vec = (double *)malloc(file_data->n_vertices *
31     sizeof(double));
32     double *new_vec = (double *)malloc(file_data->n_vertices *
33     sizeof(double));
34
35     for (int i = 0; i < file_data->n_vertices; i++)
36     {
37         vec[i] = rand() / (RAND_MAX + 1.0);
38         if (rand() / (RAND_MAX + 1.0) >= 0.5)
39         {
40             vec[i] *= -1;
41         }
42     }
43
44     int stop_iter = 0;
45     double norm_vec, new_norm_vec;
46
47     norm_vec = normalize_vec(file_data->n_vertices, vec);
48
49     struct timeval t1, t2;
50     gettimeofday(&t1, NULL);
51     while (stop_iter < 3)
52     {
53         cleanVector(new_vec, file_data->n_vertices);
54         mat_mult_AdjList(file_data->graph, vec, new_vec,
55         file_data->n_vertices);
56         new_norm_vec = normalize_vec(file_data->n_vertices,
57         new_vec);
58         copy_vec(vec, new_vec, file_data->n_vertices);
59
60         if (fabs(new_norm_vec - norm_vec) / new_norm_vec <
61         precision)
62         {
63             stop_iter += 1;
64         }
65     }
66     else

```

```

62     {
63         stop_iter = 0;
64     }
65
66     norm_vec = new_norm_vec;
67 }
68 gettimeofday(&t2, NULL);
69
70 printf("It took %.17lf milliseconds.\n", (t2.tv_sec - t1.
tv_sec) + (t2.tv_usec - t1.tv_usec) / 1e6);
71
72 FILE *output_file;
73 output_file = fopen(output_filename, "w");
74 fprintf(output_file, "%lf\n", new_norm_vec);
75 fprintf(output_file, "%d\n", file_data->n_vertexs);
76
77 for (int i = 0; i < file_data->n_vertexs; i++)
78 {
79     fprintf(output_file, "%lf\n", vec[i]);
80 }
81
82 fclose(output_file);
83
84 cleanVector(new_vec, file_data->n_vertexs);
85 mat_mult_AdjList(file_data->graph, vec, new_vec,
file_data->n_vertexs);
86 new_norm_vec = normalize_vec(file_data->n_vertexs,
new_vec);
87
88 if (fabs(new_norm_vec - norm_vec) / new_norm_vec <
precision)
89 {
90     printf("The Method works well\n");
91 }
92 else
93 {
94     printf("The Method don't work so well\n");
95     printf("method: %.17lf precision: %.17lf\n", fabs(
new_norm_vec - norm_vec) / new_norm_vec, precision);
96 }
97
98 FILE *time_record_file;
99 char timefilename[100] = "time_omp_";
100 strcat(timefilename, input_filename);
101 printf("%s\n", timefilename);
102 time_record_file = fopen(timefilename, "a+");
103 fprintf(time_record_file, "%.10lf\n", (t2.tv_sec - t1.
tv_sec) + (t2.tv_usec - t1.tv_usec) / 1e6);
104 fclose(time_record_file);

```

```

105     return 0;
106 }
107
108 void read_arguments_or_abort(int argc, char *argv[])
109 {
110     if (argc != 4)
111     {
112         fprintf(stderr, "Usage: %s <number of elements> <
113             number of arrays>\n",
114             argv[0]);
115         exit(505);
116     }
117
118 double normalize_vec(int n_vertexs, double *vector)
119 {
120
121     double sum_elements = 0;
122 #pragma omp parallel for default(none) shared(vector,
123     n_vertexs) reduction(+ \
124
125         : sum_elements) schedule(static)
126     for (int i = 0; i < n_vertexs; i++)
127     {
128         sum_elements += pow(vector[i], 2);
129     }
130
131 #pragma omp parallel for default(none) shared(vector,
132     n_vertexs, sum_elements) schedule(static)
133     for (int i = 0; i < n_vertexs; i++)
134     {
135         vector[i] /= sqrt(sum_elements);
136     }
137     return sqrt(sum_elements);
138 }
139
140 void mat_mult_AdjList(struct Graph *graph, double *vector,
141     double *new_vector, int n_vertexs)
142 {
143
144 #pragma omp parallel for default(none) shared(new_vector,
145     n_vertexs, vector, graph) schedule(dynamic)
146     for (int i = 0; i < n_vertexs; i++)
147     {
148         struct Node *ptr = graph->head[i];
149         if (ptr == NULL)
150         {
151             }
152         else

```

```

148         {
149             while (ptr != NULL)
150             {
151                 new_vector[i] += mult_pointers(ptr->weight,
vector[ptr->dest]);
152                 ptr = ptr->next;
153             }
154         }
155     }
156 }
157
158 void printfvector(double *vector, int n_vertexs)
159 {
160     for (int i = 0; i < n_vertexs; i++)
161     {
162         printf("%lf ", vector[i]);
163     }
164     printf("\n");
165 }
166
167 void cleanVector(double *Clean_vector, int n_vertexs)
168 {
169     double zero = 0;
170     #pragma omp parallel for default(none) shared(Clean_vector,
n_vertexs, zero) schedule(static)
171     for (int i = 0; i < n_vertexs; i++)
172     {
173         Clean_vector[i] = zero;
174     }
175 }
176
177 double mult_pointers(double num1, double num2)
178 {
179     double aux1 = num1;
180     double aux2 = num2;
181     double mult_value = aux1 * aux2;
182     return mult_value;
183 }
184
185 void copy_vec(double *vector, double *new_vector, int
n_vertexs)
186 {
187     #pragma omp parallel for default(none) shared(new_vector,
vector, n_vertexs) schedule(static)
188     for (int i = 0; i < n_vertexs; i++)
189     {
190         double aux = new_vector[i];
191         vector[i] = aux;
192     }

```


193 }

Listing 3: Power iteration - Parallel OpenMP

3.4 Code FindEigen_mpi - Parallel

```
1  #include "AdjListT.c"
2  #include <math.h>
3  #include <string.h>
4  #include <stdio.h>
5  #include <stdlib.h>
6  #include <sys/time.h>
7  #include <unistd.h>
8  #include <mpi.h>
9
10 void read_arguments_or_abort(int argc, char *argv[]);
11 double normalize_vec(int n_vertexs, double *vector);
12 void mat_mult_AdjList(struct Graph *graph, double *vector,
13     double *new_vector, int *jobs, int size_jobs);
14 void printfvector(double *vector, int n_vertexs);
15 void printfIntVector(int *vector, int n_vertexs);
16 void cleanVector(double *vector, int n_vertexs);
17 double mult_pointers(double num1, double num2);
18 void copy_vec(double *vector, double *new_vector, int
19     n_vertexs);
20
21 int main(int argc, char *argv[])
22 {
23     MPI_Init(&argc, &argv);
24     //numbers of processors and rank
25     int nprocs, rank;
26
27     MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
28     MPI_Comm_rank(MPI_COMM_WORLD, &rank);
29
30     if (argc != 4)
31     {
32         if(rank==0){
33             fprintf(stderr, "Usage: %s <number of elements> <
34             number of arrays>\n",argv[0]);
35         }
36         MPI_Finalize();
37         return EXIT_FAILURE;
38     }
39
40     //Read Pajek File
41     char *input_filename = argv[1];
42     double precision;
43     sscanf(argv[2], "%lf", &precision);
```

```

41     char *output_filename = argv[3];
42     struct File_data *file_data = ReadPajek(input_filename);
43
44     //work division
45     int complete_sections = (file_data->n_vertexs)/nprocs;
46     int rest_sections = file_data->n_vertexs - nprocs*
complete_sections;
47
48     int aux_size;
49     double *auxvec= NULL;
50
51     int *indexJobs=NULL;
52
53     if (rank==0){
54
55         aux_size = complete_sections + rest_sections;
56         auxvec = (double *)malloc(aux_size * sizeof(double));
57
58         indexJobs= (int *)malloc((complete_sections+
rest_sections)*sizeof(int));
59         for (int i=0; i<complete_sections+rest_sections; ++i
){
60             indexJobs[i]= i;
61         }
62     }else{
63         aux_size = complete_sections;
64         auxvec = (double *)malloc(aux_size * sizeof(double));
65
66         indexJobs= (int *)malloc((complete_sections)*sizeof(
int));
67         for (int i=0; i<complete_sections; ++i){
68             indexJobs[i]= complete_sections*rank+
rest_sections + i;
69         }
70     }
71
72     double *vec = (double *)malloc((file_data->n_vertexs) *
sizeof(double));
73     double *new_vec = (double *)malloc((file_data->n_vertexs)
* sizeof(double));
74     double norm_vec, new_norm_vec;
75
76     if (rank == 0){
77         for (int i = 0; i < file_data->n_vertexs; i++)
78         {
79             vec[i] = rand() / (RAND_MAX + 1.0);
80             if (rand() / (RAND_MAX + 1.0) >= 0.5)
81             {
82                 vec[i] *= -1;

```

```

83         }
84     }
85     norm_vec = normalize_vec(file_data->n_vertexs, vec);
86 }
87
88 //Broadcast Random vector
89 MPI_Barrier(MPI_COMM_WORLD);
90 MPI_Bcast(&vec[0], file_data->n_vertexs, MPI_DOUBLE, 0,
MPI_COMM_WORLD);
91 MPI_Bcast(&norm_vec, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);
92
93 //wait all threads arrive here
94 MPI_Barrier(MPI_COMM_WORLD);
95
96 int stop_iter = 0;
97
98
99 struct timeval t1, t2;
100 if (rank==0){
101     gettimeofday(&t1, NULL);
102 }
103
104 while (stop_iter < 3)
105 {
106
107     if (rank == 0){
108         cleanVector(new_vec, file_data->n_vertexs);
109     }
110     MPI_Barrier(MPI_COMM_WORLD);
111     MPI_Bcast(&new_vec[0], file_data->n_vertexs,
MPI_DOUBLE, 0, MPI_COMM_WORLD);
112
113     //multiplication
114     cleanVector(auxvec, aux_size);
115     mat_mult_AdjList(file_data->graph, vec, auxvec,
indexJobs, aux_size);
116     //wait all threads arrive here
117     MPI_Barrier(MPI_COMM_WORLD);
118
119
120     if (rank == 0){
121         MPI_Gather(&auxvec[0], aux_size, MPI_DOUBLE, &
new_vec[0], aux_size, MPI_DOUBLE, 0, MPI_COMM_WORLD);
122     }else{
123         if (rank !=0){
124             MPI_Gather(&auxvec[0], aux_size, MPI_DOUBLE,
NULL, aux_size, MPI_DOUBLE, 0, MPI_COMM_WORLD);
125         }
126

```

```

127     }
128     //wait all threads arrive here
129     MPI_Barrier(MPI_COMM_WORLD);
130     MPI_Bcast(&new_vec[0], file_data->n_vertexs,
131 MPI_DOUBLE, 0, MPI_COMM_WORLD);
132
133     if (rank==0){
134         new_norm_vec = normalize_vec(file_data->n_vertexs
135 , new_vec);
136     }
137     //wait all threads arrive here
138     MPI_Barrier(MPI_COMM_WORLD);
139     MPI_Bcast(&new_vec[0], file_data->n_vertexs,
140 MPI_DOUBLE, 0, MPI_COMM_WORLD);
141     //wait all threads arrive here
142     MPI_Barrier(MPI_COMM_WORLD);
143     MPI_Bcast(&new_norm_vec, 1, MPI_DOUBLE, 0,
144 MPI_COMM_WORLD);
145     copy_vec(vec, new_vec, file_data->n_vertexs);
146
147     if (fabs(new_norm_vec - norm_vec) / new_norm_vec <
148 precision)
149     {
150         stop_iter += 1;
151     }
152     else
153     {
154         stop_iter = 0;
155     }
156
157     norm_vec = new_norm_vec;
158
159     //wait all threads arrive here
160     MPI_Barrier(MPI_COMM_WORLD);
161 }
162
163 if (rank == 0){
164     gettimeofday(&t2, NULL);
165
166     printf("It took %.17lf milliseconds.\n", (t2.tv_sec -
167 t1.tv_sec) + (t2.tv_usec - t1.tv_usec) / 1e6);
168
169     FILE *output_file;
170     output_file = fopen(output_filename, "w");
171     fprintf(output_file, "%lf\n", new_norm_vec);
172     fprintf(output_file, "%d\n", file_data->n_vertexs);
173 }

```

```

170         for (int i = 0; i < file_data->n_vertices; i++)
171         {
172             fprintf(output_file, "%lf\n", vec[i]);
173         }
174         fclose(output_file);
175     }
176
177     MPI_Barrier(MPI_COMM_WORLD);
178
179     cleanVector(new_vec, file_data->n_vertices);
180     cleanVector(auxvec, aux_size);
181     mat_mult_AdjList(file_data->graph, vec, auxvec, indexJobs
182 , aux_size);
183
184     MPI_Barrier(MPI_COMM_WORLD);
185
186     if (rank==0){
187         new_norm_vec = normalize_vec(file_data->n_vertices,
188 new_vec);
189     }
190
191     if (rank == 0){
192         if (fabs(new_norm_vec - norm_vec) / new_norm_vec <
193 precision)
194         {
195             printf("The Method works well\n");
196         }
197         else
198         {
199             printf("The Method don't work so well\n");
200             printf("method: %.17lf precision: %.17lf\n", fabs
201 (new_norm_vec - norm_vec) / new_norm_vec, precision);
202         }
203
204         FILE *time_record_file;
205         char timefilename[100] = "time_mpi_";
206         strcat(timefilename, input_filename);
207         printf("%s\n", timefilename);
208         time_record_file = fopen(timefilename, "a+");
209         fprintf(time_record_file, "%.10lf\n", (t2.tv_sec - t1
210 .tv_sec) + (t2.tv_usec - t1.tv_usec) / 1e6);
211         fclose(time_record_file);
212     }
213
214     MPI_Finalize();
215
216     return 0;

```

```

214 }
215
216
217 double normalize_vec(int n_vertexs, double *vector)
218 {
219
220     double sum_elements = 0;
221     for (int i = 0; i < n_vertexs; i++)
222     {
223         sum_elements += pow(vector[i], 2);
224     }
225
226     for (int i = 0; i < n_vertexs; i++)
227     {
228         vector[i] /= sqrt(sum_elements);
229     }
230
231     return sqrt(sum_elements);
232 }
233
234 void mat_mult_AdjList(struct Graph *graph, double *vector,
235                     double *new_vector, int *jobs, int size_jobs)
236 {
237
238     for (int i = 0; i < size_jobs; i++)
239     {
240         struct Node *ptr = graph->head[jobs[i]];
241
242         if (ptr == NULL)
243         {
244         }
245         else
246         {
247             while (ptr != NULL)
248             {
249                 new_vector[i] += mult_pointers(ptr->weight,
250 vector[ptr->dest]);
251                 ptr = ptr->next;
252             }
253         }
254     }
255
256 void printfvector(double *vector, int n_vertexs)
257 {
258     for (int i = 0; i < n_vertexs; i++)
259     {
260         printf("%lf ", vector[i]);
261     }

```

```

261     printf("\n");
262 }
263
264 void printfIntVector(int *vector, int n_vertexs)
265 {
266     for (int i = 0; i < n_vertexs; i++)
267     {
268         printf("%d ", vector[i]);
269     }
270     printf("\n");
271 }
272
273 void cleanVector(double *Clean_vector, int n_vertexs)
274 {
275     double zero = 0;
276     for (int i = 0; i < n_vertexs; i++)
277     {
278         Clean_vector[i] = zero;
279     }
280 }
281
282 double mult_pointers(double num1, double num2)
283 {
284     double aux1 = num1;
285     double aux2 = num2;
286     double mult_value = aux1 * aux2;
287     return mult_value;
288 }
289
290 void copy_vec(double *vector, double *new_vector, int
n_vertexs)
291 {
292     for (int i = 0; i < n_vertexs; i++)
293     {
294         double aux = new_vector[i];
295         vector[i] = aux;
296     }
297 }

```

Listing 4: Power iteratrion - Parallel MPI

4 Performance analysis

For the performance analysis, the test files available in the compiled file `powerit_test.tar.gz` were used. With this in mind, error bar plots were performed to verify the processing time of each code.

4.1 Small

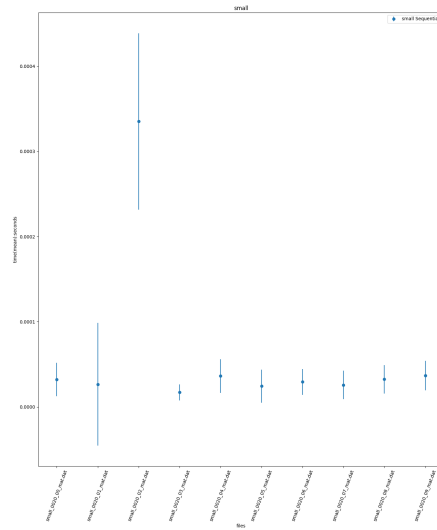


Figure 4: Sequential

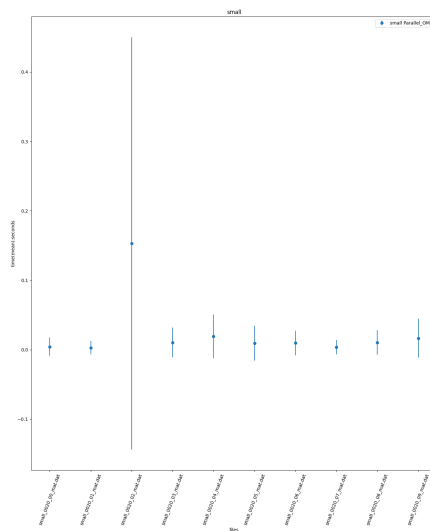


Figure 5: Parallel OpenMP

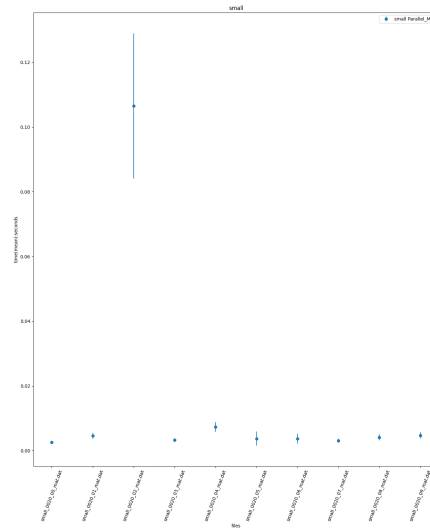


Figure 6: Parallel MPI

4.2 Medium

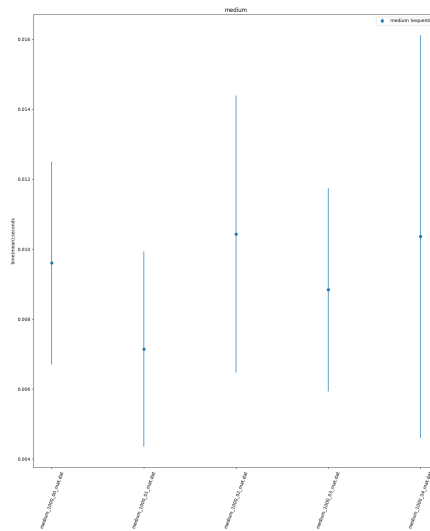


Figure 7: Sequential

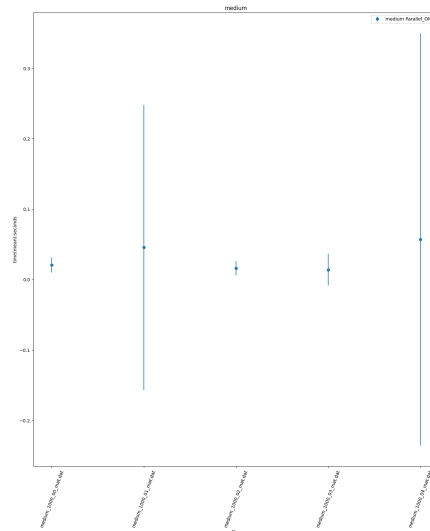


Figure 8: Parallel OpenMP

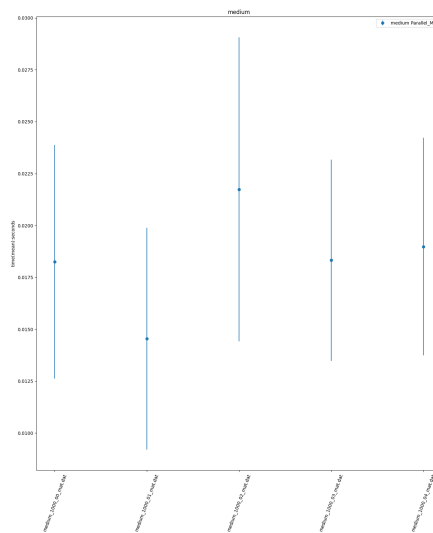


Figure 9: Parallel MPI

4.3 Large

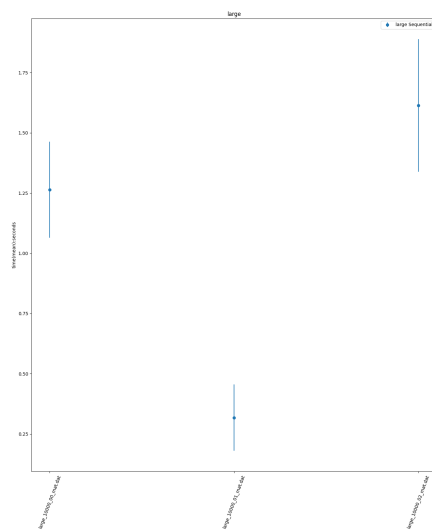


Figure 10: Sequential

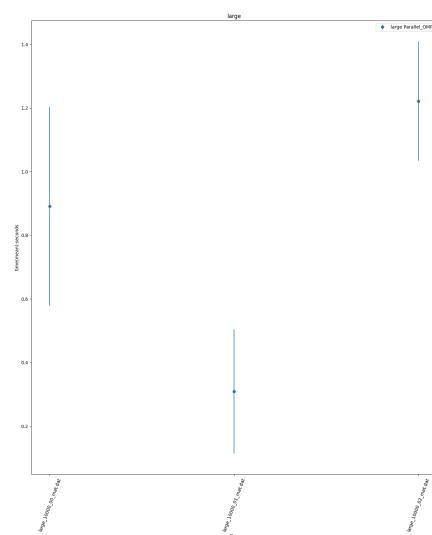


Figure 11: Parallel OpenMP

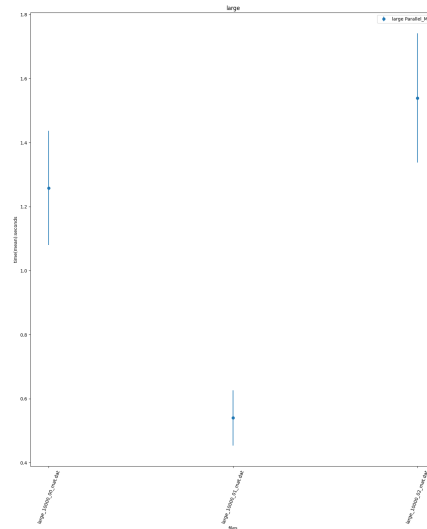


Figure 12: Parallel MPI

4.4 Huge

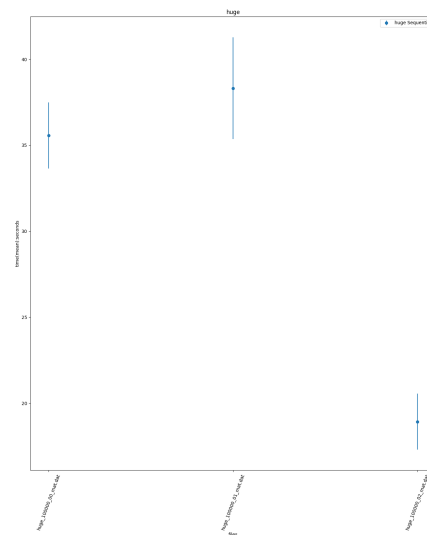


Figure 13: Sequential

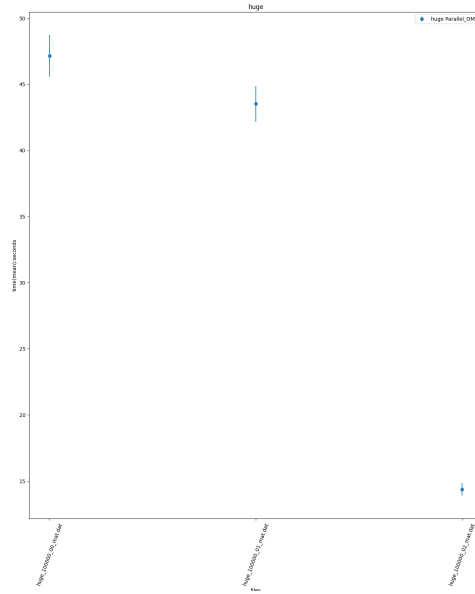


Figure 14: Parallel OpenMP

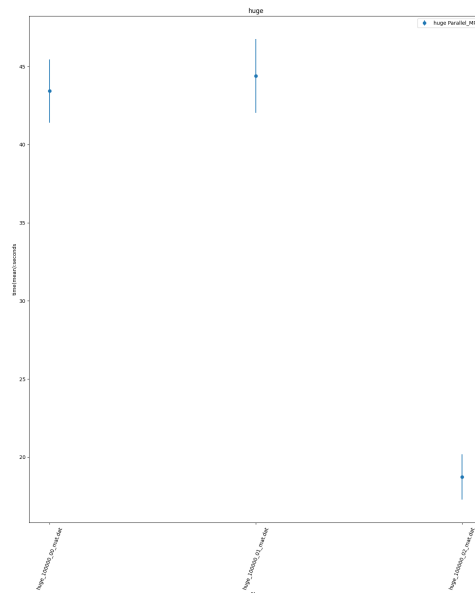


Figure 15: Parallel MPI

4.5 All size files

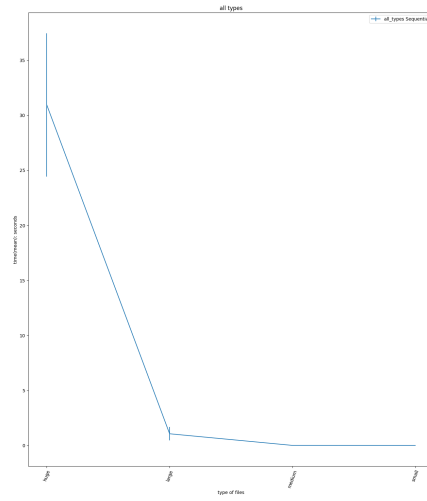


Figure 16: Mean of each type file - Sequential

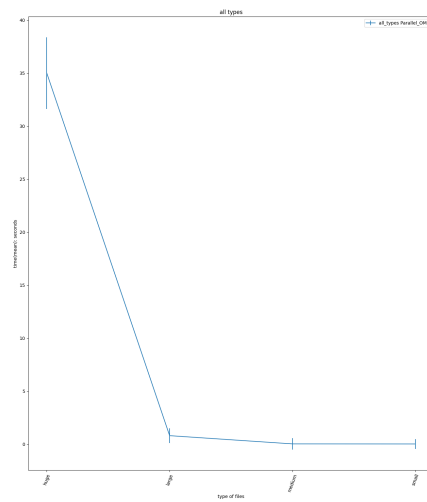


Figure 17: Mean of each type file - Parallel OpenMP

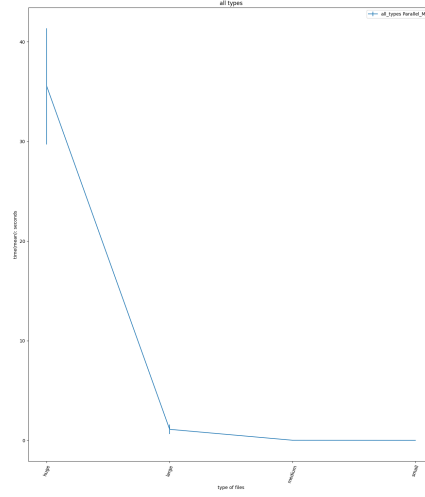


Figure 18: Mean of each type file - Parallel MPI

4.6 Comparisons

In view of the images presented above, an image was made with the implementations together for better analysis.

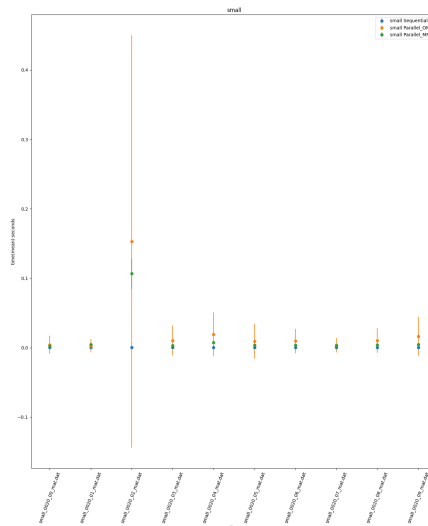


Figure 19: Small

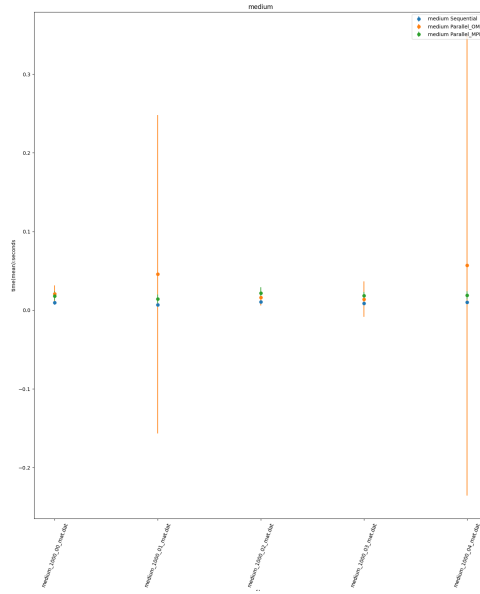


Figure 20: Medium

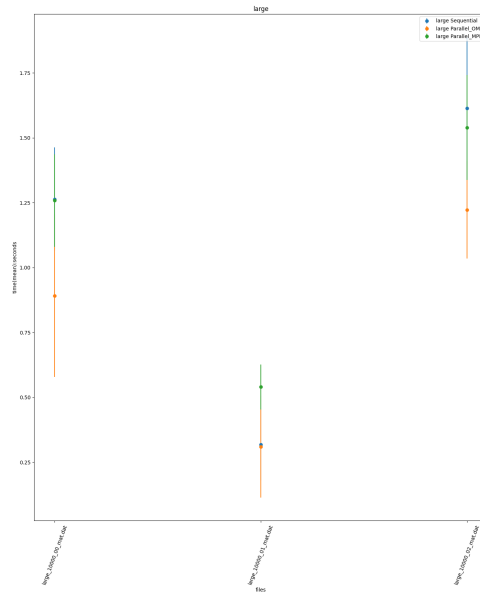


Figure 21: Large

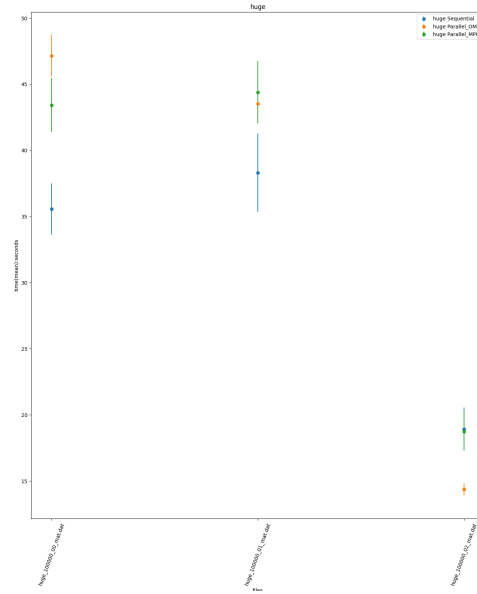


Figure 22: Huge

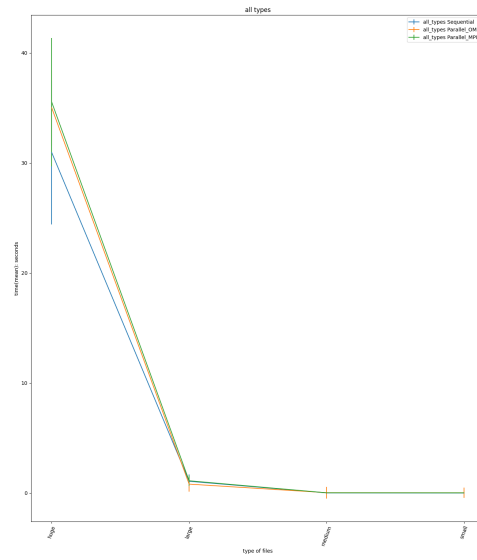


Figure 23: All Types

5 Conclusions

In short, among the three implementations made in this report, the routine that had the best performance was the sequential code. However, it would be expected that the parallel codes would have better performance due to the partition of work between the processors. This result was obtained because the machine that was used contained only two processors, so the passing of messages needed during the parallelization procedures took a large portion of time, leading to an increase in time that exceeded the work divided into the processors.

6 Reference

- [1] Links with the images of Graph and Adjancecy List
- [2] da Silva, Éverton Luís Mendes. Codes from this project
- [3] da Silva, Éverton Luís Mendes. Codes and images from this project in Google Drive