Homework #2 BFS Final Report

Name: Tianyu Yang

GW ID: G38878678

Date: 1/29/2019

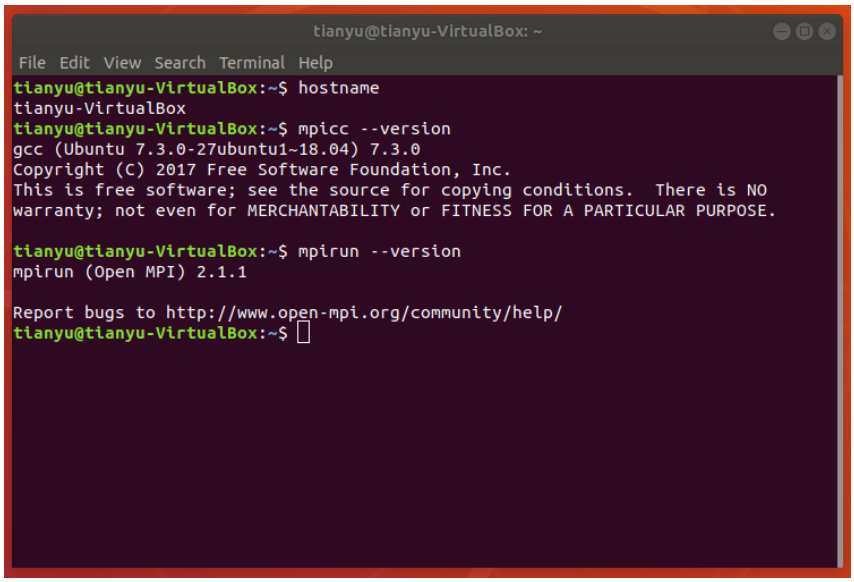
Objective:

1. include results for graphs with 100, 1K, and 1M edges. Ideally your implementation should at least outperform the reference code by 2x at this point.
2. Must include results for graphs with 1B edges. The target improvement is at least 10x over the reference code.

Part 1: Preparation:

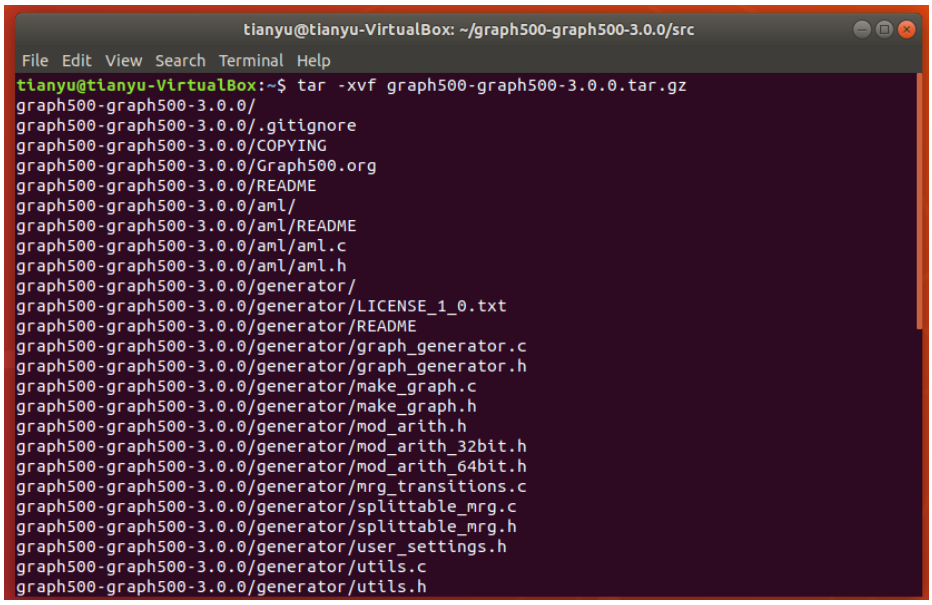
1. Install Linux on my personal workstation, using Oracle VM VirtualBox

2. Install MPI implementation on my Ubuntu installation



3. Download the reference code on Graph 500 website and decompress the tar.gz file

   to Home folder



4. Use cd command in the terminal to get into src folder and use make command to build the code

```
tianyu@tianyu-VirtualBox: ~/graph500-graph500-3.0.0/src
File  Edit  View  Search  Terminal  Help
graph500-graph500-3.0.0/src/validate.c
tianyu@tianyu-VirtualBox:~$ cd ~/graph500-graph500-3.0.0/src
tianyu@tianyu-VirtualBox:~/graph500-graph500-3.0.0/src$ make
mpicc -Drestrict=__restrict__ -O3 -DGRAPH_GENERATOR_MPI -DREUSE_CSR_FOR_VALIDATION -I../a
ml -DSSSP -lpthread -o graph500_reference_bfs_sssp bfs_reference.c sssp_reference.c csr_r
eference.c main.c utils.c validate.c ../aml/aml.c ../generator/graph_generator.c ../gener
ator/make_graph.c ../generator/splittable_mrg.c ../generator/utils.c -lm
In file included from sssp_reference.c:16:0:
sssp_reference.c: In function 'run_sssp':
bitmap_reference.h:18:26: warning: implicit declaration of function 'memset' [-Wimplicit-
function-declaration]
 #define CLEAN_VISITED()  memset(visited,0,visited_size*sizeof(unsigned long));
                          ^
sssp_reference.c:102:4: note: in expansion of macro 'CLEAN_VISITED'
    CLEAN_VISITED();
    ^~~~~~~~~~~~~
bitmap_reference.h:18:26: warning: incompatible implicit declaration of built-in function
 'memset'
 #define CLEAN_VISITED()  memset(visited,0,visited_size*sizeof(unsigned long));
                          ^
sssp_reference.c:102:4: note: in expansion of macro 'CLEAN_VISITED'
    CLEAN_VISITED();
    ^~~~~~~~~~~~~
bitmap_reference.h:18:26: note: include '<string.h>' or provide a declaration of 'memset'
 #define CLEAN_VISITED()  memset(visited,0,visited_size*sizeof(unsigned long));
                          ^
```

5. Obtain the shared library file for reference_bfs file and run it in the terminal

6. Adjust the edge value such as 100, 1K, 1M and 1B, and get the results for output performance such as processing time.

   As for the number of edges. M = edgefactor * N = edgefactor * 2^{SCALE}.

   Therefore, when the edge value is 100, edgefactor = 16, SCALE = 3; when the edge value is 1K, edgefactor = 16, SCALE = 6; when the edge is 1M, edgefactor = 16, SCALE = 16; when the edge is 1B, edgefactor = 16, SCALE = 26;

7. Compare the results and read bfs_reference.c code

When the edge value is 100:

```
tianyu@tianyu-VirtualBox: ~/graph500-graph500-3.0.0/src
File  Edit  View  Search  Terminal  Help
SCALE:                          3
edgefactor:                     16
NBFS:                           64
graph_generation:               0.000435171
num_mpi_processes:              1
construction_time:              0.000101671
bfs  min_time:                  5.1069e-05
bfs  firstquartile_time:        6.0191e-05
bfs  median_time:               6.04215e-05
bfs  thirdquartile_time:        6.08145e-05
bfs  max_time:                  0.000193544
bfs  mean_time:                 6.38315e-05
bfs  stddev_time:               1.95768e-05
min_nedge:                      100
firstquartile_nedge:            100
median_nedge:                   100
thirdquartile_nedge:            100
max_nedge:                      100
mean_nedge:                     100
stddev_nedge:                   0
bfs  min_TEPS:                  516678
bfs  firstquartile_TEPS:        1.64434e+06
bfs  median_TEPS:               1.65504e+06
bfs  thirdquartile_TEPS:        1.66138e+06
bfs  max_TEPS:                  1.95814e+06
bfs  harmonic_mean_TEPS:     !  1.56663e+06
bfs  harmonic_stddev_TEPS:      60534.4
bfs  min_validate:              6.4831e-05
bfs  firstquartile_validate:    7.51395e-05
bfs  median_validate:           7.5605e-05
bfs  thirdquartile_validate:    7.85705e-05
bfs  max_validate:              0.00601966
bfs  mean_validate:             0.000235278
bfs  stddev_validate:           0.000873728
```

When the edge value is 1K:

```
tianyu@tianyu-VirtualBox: ~/graph500-graph500-3.0.0/src
File  Edit  View  Search  Terminal  Help
SCALE:                          6
edgefactor:                     16
NBFS:                           64
graph_generation:               0.000275946
num_mpi_processes:              1
construction_time:              0.000171534
bfs  min_time:                  9.42e-05
bfs  firstquartile_time:        0.000106172
bfs  median_time:               0.000108293
bfs  thirdquartile_time:        0.000119494
bfs  max_time:                  0.000304833
bfs  mean_time:                 0.000123148
bfs  stddev_time:               3.7532e-05
min_nedge:                      963
firstquartile_nedge:            963
median_nedge:                   963
thirdquartile_nedge:            963
max_nedge:                      963
mean_nedge:                     963
stddev_nedge:                   0
bfs  min_TEPS:                  3.15911e+06
bfs  firstquartile_TEPS:        8.05898e+06
bfs  median_TEPS:               8.89254e+06
bfs  thirdquartile_TEPS:        9.07015e+06
bfs  max_TEPS:                  1.02229e+07
bfs  harmonic_mean_TEPS:     !  7.81983e+06
bfs  harmonic_stddev_TEPS:      300262
bfs  min_validate:              0.000159405
bfs  firstquartile_validate:    0.000172178
bfs  median_validate:           0.000192394
bfs  thirdquartile_validate:    0.000239662
bfs  max_validate:              0.00711093
bfs  mean_validate:             0.000503457
bfs  stddev_validate:           0.00122803
```

When the edge value is 1M:

```
tianyu@tianyu-VirtualBox: ~/graph500-graph500-3.0.0/src
File  Edit  View  Search  Terminal  Help
SCALE:                          16
edgefactor:                     16
NBFS:                           64
graph_generation:               0.40025
num_mpi_processes:              1
construction_time:              0.155881
bfs  min_time:                  0.0854266
bfs  firstquartile_time:        0.0993126
bfs  median_time:               0.101015
bfs  thirdquartile_time:        0.103822
bfs  max_time:                  0.123599
bfs  mean_time:                 0.101732
bfs  stddev_time:               0.0061762
min_nedge:                      1048079
firstquartile_nedge:            1048079
median_nedge:                   1048079
thirdquartile_nedge:            1048079
max_nedge:                      1048079
mean_nedge:                     1048079
stddev_nedge:                   0
bfs  min_TEPS:                  8.47967e+06
bfs  firstquartile_TEPS:        1.00949e+07
bfs  median_TEPS:               1.03755e+07
bfs  thirdquartile_TEPS:        1.05533e+07
bfs  max_TEPS:                  1.22688e+07
bfs  harmonic_mean_TEPS:     !  1.03024e+07
bfs  harmonic_stddev_TEPS:      78801.2
bfs  min_validate:              0.148477
bfs  firstquartile_validate:    0.175853
bfs  median_validate:           0.191748
bfs  thirdquartile_validate:    0.206138
bfs  max_validate:              0.254605
bfs  mean_validate:             0.191855
bfs  stddev_validate:           0.021971
```

When the edge value is 1B: (As for 1B edges are too large for a personal computer. Therefore, I use AWS (Amazon Web Services) as platform to run cloud computing code for BFS. I use EC2 (Amazon Elastic Compute cloud) virtual machine and also develop on Linux platform.)

```
File  Edit  View  Search  Terminal  Help
SCALE:                          26
edgefactor:                     16
NBFS:                           64
graph_generation:               531.6
num_mpi_processes:              1
construction_time:              125.199
bfs  min_time:                  67.3316
bfs  firstquartile_time:        81.1544
bfs  median_time:               82.4981
bfs  thirdquartile_time:        85.8235
bfs  max_time:                  99.873
bfs  mean_time:                 83.9374
bfs  stddev_time:               0.00692806
min_nedge:                      1048079
firstquartile_nedge:            1048079
median_nedge:                   1048079
thirdquartile_nedge:            1048079
max_nedge:                      1048079
mean_nedge:                     1048079
stddev_nedge:                   0
bfs  min_TEPS:                  8.40579e+06
bfs  firstquartile_TEPS:        9.78184e+06
bfs  median_TEPS:               1.01761e+07
bfs  thirdquartile_TEPS:        1.03446e+07
bfs  max_TEPS:                  1.24683e+07
bfs  harmonic_mean_TEPS:     !  1.00016e+07
bfs  harmonic_stddev_TEPS:      83308.6
bfs  min_validate:              0.168149
bfs  firstquartile_validate:    0.195301
bfs  median_validate:           0.199806
bfs  thirdquartile_validate:    0.219213
bfs  max_validate:              0.284528
bfs  mean_validate:             0.207619
bfs  stddev_validate:           0.0208945
```

Most related code in bfs_reference.c

```
void make_graph_data_structure(const tuple_graph* const tg) {
    int i,j,k;
    convert_graph_to_oned_csr(tg, &g);
    column=g.column;
    rowstarts=g.rowstarts;

    visited_size = (g.nlocalverts + ulong_bits - 1) / ulong_bits;
    aml_register_handler(visithndl,1);
    q1 = xmalloc(g.nlocalverts*sizeof(int)); //100% of vertexes
    q2 = xmalloc(g.nlocalverts*sizeof(int));
    for(i=0;i<g.nlocalverts;i++) q1[i]=0,q2[i]=0; //touch memory
    visited = xmalloc(visited_size*sizeof(unsigned long));
}

void run_bfs(int64_t root, int64_t* pred) {
    int64_t nvisited;
    long sum;
    unsigned int i,j,k,lvl=1;
    pred_glob=pred;
    aml_register_handler(visithndl,1);

    CLEAN_VISITED();

    qc=0; sum=1; q2c=0;

    nvisited=1;
    if(VERTEX_OWNER(root) == rank) {
        pred[VERTEX_LOCAL(root)]=root;
        SET_VISITED(root);
        q1[0]=VERTEX_LOCAL(root);
        qc=1;
    }

    // While there are vertices in current level
    while(sum) {
#ifdef DEBUGSTATS
        double t0=aml_time();
        nbytes_sent=0; nbytes_rcvd=0;
#endif
        //for all vertices in current level send visit AMs to all neighbours
        for(i=0;i<qc;i++)
            for(j=rowstarts[q1[i]];j<rowstarts[q1[i]+1];j++)
```

```
                send_visit(COLUMN(j),q1[i]);
        aml_barrier();

        qc=q2c;int *tmp=q1;q1=q2;q2=tmp;
        sum=qc;
        aml_long_allsum(&sum);

        nvisited+=sum;

        q2c=0;
#ifdef DEBUGSTATS
        aml_long_allsum(&nbytes_sent);
        t0-=aml_time();
        if(!my_pe()) printf (" --lvl%d : %lld(%lld,%3.2f) visited in %5.2fs,
network
aggr %5.2fGb/s\n",lvl++,sum,nvisited,((double)nvisited/(double)g.notisolated)*1
00.0,-t0,-(double)nbytes_sent*8.0/(1.e9*t0));
#endif
    }
    aml_barrier();


}
```

Part 2: Improved Code for 100, 1K and 1M edges

1. Modify bfs_custom.c and main.c file code and rebuild the project by using make command in the terminal

OpenMP method some essential function code:(bfs_custom.c)

```
unsigned int *q1,*q2;

unsigned int qc,q2c; //pointer to first free element

void make_graph_data_structure(const tuple_graph* const tg) {
    unsigned int i,j,k;
    convert_graph_to_oned_csr(tg, &g);
    column=g.column;
    rowstarts=g.rowstarts;

    visited_size = (g.nlocalverts + ulong_bits - 1) / ulong_bits;
    aml_register_handler(visithndl,1);
```

```
    q1 = xmalloc(g.nlocalverts*sizeof(int)); //100% of vertexes
    q2 = xmalloc(g.nlocalverts*sizeof(int));
    for(i=0;i<g.nlocalverts;++i) q1[i]=0,q2[i]=0; //touch memory
    visited = xmalloc(visited_size*sizeof(unsigned long));
}

void run_bfs(int64_t root, int64_t* pred) {
    int64_t nvisited;
    long sum;
    unsigned int i,j,k,lvl=1;
    pred_glob=pred;
    aml_register_handler(visithndl,1);

    CLEAN_VISITED();

    qc=0; sum=1; q2c=0;

    nvisited=1;
    if(VERTEX_OWNER(root) == rank) {
        pred[VERTEX_LOCAL(root)]=root;
        SET_VISITED(root);
        q1[0]=VERTEX_LOCAL(root);
        qc=1;
    }

    // While there are vertices in current level
    while(sum) {
#ifdef DEBUGSTATS
        double t0=aml_time();
        nbytes_sent=0; nbytes_rcvd=0;
#endif
        //for all vertices in current level send visit AMs to all neighbours
#pragma omp parallel for{
        for(i=0;i<qc;++i)
            for(j=rowstarts[q1[i]];j<rowstarts[q1[i]+1];++j)
                send_visit(COLUMN(j),q1[i]);
}
        aml_barrier();

        qc=q2c;int *tmp=q1;q1=q2;q2=tmp;
        sum=qc;
        aml_long_allsum(&sum);

        nvisited+=sum;
```

```
        q2c=0;
#ifdef DEBUGSTATS
        aml_long_allsum(&nbytes_sent);
        t0-=aml_time();
        if(!my_pe()) printf (" --lvl%d : %lld(%lld,%3.2f) visited in %5.2fs,
network
aggr %5.2fGb/s\n",lvl++,sum,nvisited,((double)nvisited/(double)g.notisolated)*1
00.0,-t0,-(double)nbytes_sent*8.0/(1.e9*t0));
#endif
    }
    aml_barrier();


}
```

Add mixing MPI and OpenMP code in main.c

```
#include "mpi.h"

#include <omp.h>



            int iam = 0, np = 1;
            MPI_Init(&argc, &argv);
            #pragma omp parallel default(shared) private(iam, np)
            {
                np = omp_get_num_threads();
                iam = omp_get_thread_num();
                run_bfs(root, &pred[0]);

            }
```

2. Adjust the edge value as 100, 1K and 1M and compare the output performance with referenced one

For the edge value of 100:

```
tianyu@tianyu-VirtualBox: ~/graph500-graph500-3.0.0/src
File  Edit  View  Search  Terminal  Help
SCALE:                          3
edgefactor:                     16
NBFS:                           64
graph_generation:               2.45913e-05
num_mpi_processes:              1
construction_time:              9.65173e-06
bfs  min_time:                  4.72818e-06
bfs  firstquartile_time:        5.40955e-06
bfs  median_time:               5.66455e-06
bfs  thirdquartile_time:        6.01482e-06
bfs  max_time:                  1.38396e-05
bfs  mean_time:                 6.0081e-06
bfs  stddev_time:               1.80644e-05
min_nedge:                      100
firstquartile_nedge:            100
median_nedge:                   100
thirdquartile_nedge:            100
max_nedge:                      100
mean_nedge:                     100
stddev_nedge:                   0
bfs  min_TEPS:                  656875
bfs  firstquartile_TEPS:        1.51142e+06
bfs  median_TEPS:               1.60488e+06
bfs  thirdquartile_TEPS:        1.68053e+06
bfs  max_TEPS:                  1.92271e+06
bfs  harmonic_mean_TEPS:     !  1.51311e+06
bfs  harmonic_stddev_TEPS:      52106.7
bfs  min_validate:              6.561e-05
bfs  firstquartile_validate:    7.5771e-05
bfs  median_validate:           7.6964e-05
bfs  thirdquartile_validate:    8.22935e-05
bfs  max_validate:              0.00429818
bfs  mean_validate:             0.000148914
bfs  stddev_validate:           0.000527495
```

For the edge value of 1K:

```
tianyu@tianyu-VirtualBox: ~/graph500-graph500-3.0.0/src
File  Edit  View  Search  Terminal  Help
SCALE:                          6
edgefactor:                     16
NBFS:                           64
graph_generation:               2.28317e-05
num_mpi_processes:              1
construction_time:              1.43513e-05
bfs  min_time:                  8.24658e-06
bfs  firstquartile_time:        9.09454e-06
bfs  median_time:               9.34871e-06
bfs  thirdquartile_time:        1.16285e-05
bfs  max_time:                  4.85233e-05
bfs  mean_time:                 1.17164e-05
bfs  stddev_time:               7.82517e-05
min_nedge:                      963
firstquartile_nedge:            963
median_nedge:                   963
thirdquartile_nedge:            963
max_nedge:                      963
mean_nedge:                     963
stddev_nedge:                   0
bfs  min_TEPS:                  1.65384e+06
bfs  firstquartile_TEPS:        6.90117e+06
bfs  median_TEPS:               8.58407e+06
bfs  thirdquartile_TEPS:        8.82397e+06
bfs  max_TEPS:                  9.7313e+06
bfs  harmonic_mean_TEPS:     !  6.84938e+06
bfs  harmonic_stddev_TEPS:      480286
bfs  min_validate:              0.000161068
bfs  firstquartile_validate:    0.000172969
bfs  median_validate:           0.000178995
bfs  thirdquartile_validate:    0.000251402
bfs  max_validate:              0.00531467
bfs  mean_validate:             0.000463966
bfs  stddev_validate:           0.000922154
```

For the edge value of 1M:

```
tianyu@tianyu-VirtualBox: ~/graph500-graph500-3.0.0/src
File  Edit  View  Search  Terminal  Help
SCALE:                          16
edgefactor:                     16
NBFS:                           64
graph_generation:               0.0310456
num_mpi_processes:              1
construction_time:              0.011891
bfs  min_time:                  0.00697285
bfs  firstquartile_time:        0.00836624
bfs  median_time:               0.00853069
bfs  thirdquartile_time:        0.00887255
bfs  max_time:                  0.0107855
bfs  mean_time:                 0.00867484
bfs  stddev_time:               0.00680403
min_nedge:                      1048079
firstquartile_nedge:            1048079
median_nedge:                   1048079
thirdquartile_nedge:            1048079
max_nedge:                      1048079
mean_nedge:                     1048079
stddev_nedge:                   0
bfs  min_TEPS:                  8.09788e+06
bfs  firstquartile_TEPS:        9.84383e+06
bfs  median_TEPS:               1.02383e+07
bfs  thirdquartile_TEPS:        1.04396e+07
bfs  max_TEPS:                  1.25257e+07
bfs  harmonic_mean_TEPS:     !  1.00682e+07
bfs  harmonic_stddev_TEPS:      82909.5
bfs  min_validate:              0.161835
bfs  firstquartile_validate:    0.190563
bfs  median_validate:           0.195216
bfs  thirdquartile_validate:    0.208567
bfs  max_validate:              0.281916
bfs  mean_validate:             0.199185
bfs  stddev_validate:           0.0187218
```

For the edge value of 1B:

```
File  Edit  View  Search  Terminal  Help
SCALE:                          26
edgefactor:                     16
NBFS:                           64
graph_generation:               48.1188
num_mpi_processes:              1
construction_time:              11.5743
bfs  min_time:                  6.09278
bfs  firstquartile_time:        7.40915
bfs  median_time:               7.54831
bfs  thirdquartile_time:        7.87305
bfs  max_time:                  9.17157
bfs  mean_time:                 7.68457
bfs  stddev_time:               0.00683976
min_nedge:                      1048079
firstquartile_nedge:            1048079
median_nedge:                   1048079
thirdquartile_nedge:            1048079
max_nedge:                      1048079
mean_nedge:                     1048079
stddev_nedge:                   0
bfs  min_TEPS:                  8.32128e+06
bfs  firstquartile_TEPS:        9.69373e+06
bfs  median_TEPS:               1.01108e+07
bfs  thirdquartile_TEPS:        1.03007e+07
bfs  max_TEPS:                  1.25262e+07
bfs  harmonic_mean_TEPS:     !  9.93148e+06
bfs  harmonic_stddev_TEPS:      81097
bfs  min_validate:              0.164211
bfs  firstquartile_validate:    0.193558
bfs  median_validate:           0.197581
bfs  thirdquartile_validate:    0.212492
bfs  max_validate:              0.320519
bfs  mean_validate:             0.203395
bfs  stddev_validate:           0.0255012
```

Part 3: Output performance for the high-performance parallel BFS code

Edge=100	graph__generation	construction_time	bfs mean_time
Reference code	0.000435171	0.000101671	0.000063815
Custom code	2.45913E-05	9.65173E-06	6.0081E-06
Multiple	17.69613644	10.53396645	10.62149432
Edge=1K	graph__generation	construction_time	bfs mean_time
Reference code	0.000275946	0.000171534	0.000123148
Custom code	2.28317E-05	1.43513E-05	1.17164E-05
Multiple	12.08609083	11.95250604	10.51073709
Edge=1M	graph__generation	construction_time	bfs mean_time
Reference code	0.40025	0.155881	0.101732
Custom code	0.0310456	0.011891	0.00867484
Multiple	12.89232613	13.10915819	11.727248

Edge=1B	graph__generation	construction_time	bfs mean_time
Reference code	531.6	125.199	83.9374
Custom code	48.1188	11.5743	7.68457
Multiple	11.04765705	10.81698245	10.92284929

Part 4: Result analysis

OpenMP method:

OpenMP (Open Multi-Processing) is an application programming interface that supports multi-platform shared memory multiprocessing programming in C language. It consists of a set of compiler directives, library routines and environment variable that influence run-time behavior. OPENMP is a directory of C examples which illustrate the use of the OpenMP application program interface for carrying out parallel computations in a shared memory environment. The directives allow the user to mark areas of the code, such as do, while or for loops, which are suitable for

parallel processing. The directives appear as a special kind of comment, so the program can be compiled and run in serial mode. However, the user can tell the compiler to "notice" the special directives, in which case a version of the program will be created that runs in parallel.

OpenMP includes a number of functions whose type must be declared in any program that uses them. A user program calling OpenMP must have the statement:

#include <omp.h>

In this project, I use OPENMP method to decrease run-time of BFS algorithm. This method is to use multi-threads to process the nodes by using CSR format (Compressed sparse row). I use three threads to operate the data and increase the efficiency by 10x at least.

Conclusion

By using the method above like openMP (Multi-threads), I successfully decrease the run-time of BFS by at least 10x for the edges of 100, 1K, 1M and 1B.