Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н. Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

ОТЧЕТ по практикуму
Задание №2

Тема практикума «Обработка и визуализация графов.»

Название «Обработка и визуализация графов в вычислительном комплексе Тераграф»

Дисциплина «Архитектура элекронно-вычислительных» машин

Студент: _____ Чыонг В. Х.
подпись, дата Фамилия, И.О.

Преподаватель: _____ Попов А. Ю.
подпись, дата Фамилия, И. О.

Москва — 2022 г.

Содержание

Цель работы

Практикум посвящен освоению принципов представления графов и их обработке с помощью вычислительного комплекса Терраграф. В ходе практикума необходимо ознакомиться с вариантами представления графов в виде объединения структур языка C/C++, изучить и применить на практике примеры решения некоторых задач на графах. По индивидуальному варианту необходимо разработать программу хост-подсистемы и программного ядра sw_kernel, выполняющего обработку и визуализацию графов.

1 Основные теоретические сведения

Визуализация графа — это графическое представление вершин и ребер графа. Визуализация строится на основе исходного графа, но направлена на получение дополнительных атрибутов вершин и ребер: размера, цвета, координат вершин, толщины и геометрии ребер. Помимо этого, в задачи визуализации входит определение масштаба представления визуализации. Для различных по своей природе графов, могут быть более применимы различные варианты визуализации. Таким образом задачи, входящие в последовательность подготовки графа к визуализации, формулируются исходя из эстетических и эвристических критериев.

2 Экспериментальная часть

2.1 Индивидуальное задание

Задание практикума выполнялось по варианту 11: Выполнить визуализацию неориентированного графа, представленного в формате tsv. Каждая строчка файла представляет собой описание ребра, сотоящее из трех чисел (Вершина,Вершина,Вес) или двух чисел (Вершина,Вершина). Во втором случае вес ребра принимается равным 1.

2.2 Результаты выполнения задания

2.2.1 Host

Листинг 2.1 – Измененный код хост-системы под индивидульное задание

```
#include "host_main.h"
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/ip.h>
#include <stdlib.h>
#include <assert.h>
#include <string.h>
#include <stdio.h>
#include <fstream>
#include <iostream>

// #define RAND_GRAPH
#define GRID_GRAPH
//#define GRID_LAYOUT
//#define GRID_SPIRAL_LAYOUT
//#define SPIRAL_LAYOUT
#define BOX_LAYOUT
//#define FORCED_LAYOUT
#define DEBUG
```

```
21 #define DATA_FILE
      "/iu_home/iu7127/worksp/btwc-dijkstra-xrt/host/src/data.tsv"
22
23
24 #define handle_error(msg) \
25           do { perror(msg); exit(EXIT_FAILURE); } while (0)
26
27 static void usage()
28 {
29     std::cout << "usage: <xclbin> <sw_kernel>\n\n";
30 }
31
32 static void print_table(std::string test, float value,
      std::string units)
33 {
34     std::cout << std::left << std::setfill(' ') << std::setw(50)
         << test << std::right << std::setw(20) << std::fixed <<
         std::setprecision(0) << value << std::setw(15) << units <<
         std::endl;
35     std::cout << std::setfill('-') << std::setw(85) << "-" <<
         std::endl;
36 }
37 const int port = 0x4747;
38 int server_socket_init() {
39     int sock_fd;
40     struct sockaddr_in srv_addr;
41     int client_fd;
42     sock_fd = socket(AF_INET, SOCK_STREAM, 0);
43     if (sock_fd == -1)
44         handle_error("socket");
45     memset(&srv_addr, 0, sizeof(srv_addr));
46     srv_addr.sin_family = AF_INET;
47     srv_addr.sin_port = htons(port);
48     srv_addr.sin_addr.s_addr = INADDR_ANY;
49     if (bind(sock_fd, (struct sockaddr *)&srv_addr,
         sizeof(srv_addr)) == -1)
50         handle_error("bind");
51     if (listen(sock_fd, 2) == -1)
52         handle_error("listen");
53     return sock_fd;
54 }
```

```
55
56 int main(int argc, char** argv)
57 {
58
59     unsigned int err = 0;
60     unsigned int cores_count = 0;
61     float LNH_CLOCKS_PER_SEC;
62     clock_t start, stop;
63
64     __foreach_core(group, core) cores_count++;
65
66     //Assign xclbin
67     if (argc < 3) {
68         usage();
69         throw std::runtime_error("FAILED_TEST\nNo␣xclbin␣
           specified");
70     }
71
72     //Open device #0
73     leonhardx64 lnh_inst = leonhardx64(0, argv[1]);
74     __foreach_core(group, core)
75     {
76         lnh_inst.load_sw_kernel(argv[2], group, core);
77     }
78
79     /*
80      *
81      * SW Kernel Version and Status
82      *
83      */
84     __foreach_core(group, core)
85     {
86         printf("Group␣#%d␣\tCore␣#%d\n", group, core);
87         lnh_inst.gpc[group][core]->start_sync(__event__(get_version));
88         printf("\tSoftware␣Kernel␣Version:\t0x%08x\n",
           lnh_inst.gpc[group][core]->mq_receive());
89         lnh_inst.gpc[group][core]->start_sync(__event__(get_lnh_status_hi
90         printf("\tLeonhard␣Status␣Register:\t0x%08x",
           lnh_inst.gpc[group][core]->mq_receive());
91         lnh_inst.gpc[group][core]->start_sync(__event__(get_lnh_status_lo
```

```
92          printf("_%08x\n",
                lnh_inst.gpc[group][core]->mq_receive());
93      }
94
95
96      //------------------------------------------------------------------------
97      // Измерение производительности Leonhard
98      //------------------------------------------------------------------------
99
100     float interval;
101     char buf[100];
102     err = 0;
103
104     time_t now = time(0);
105     strftime(buf, 100, "Start at local date: %d.%m.%Y.; local
            time: %H.%M.%S", localtime(&now));
106
107     printf("\nDISC system speed test v3.0\n%s\n\n", buf);
108     std::cout << std::left << std::setw(50) << "Test" <<
            std::right << std::setw(20) << "value" << std::setw(15) <<
            "units" << std::endl;
109     std::cout << std::setfill('-') << std::setw(85) << "-" <<
            std::endl;
110     print_table("Graph Processing Cores count (GPCC)",
            cores_count, "instances");
111
112
113
114
115     /*
116      *
117      * GPC frequency measurement for the first kernel
118      *
119      */
120     lnh_inst.gpc[0][LNH_CORES_LOW[0]]->start_async(__event__(frequency_me
121
122     // Measurement Body
123     lnh_inst.gpc[0][LNH_CORES_LOW[0]]->sync_with_gpc(); // Start
            measurement
124     sleep(1);
```

```
125     lnh_inst.gpc[0][LNH_CORES_LOW[0]]->sync_with_gpc(); // Start
            measurement
126     // End Body
127     lnh_inst.gpc[0][LNH_CORES_LOW[0]]->finish();
128     LNH_CLOCKS_PER_SEC =
            (float)lnh_inst.gpc[0][LNH_CORES_LOW[0]]->mq_receive();
129     print_table("Leonhard␣clock␣frequency␣(LNH_CF)",
            LNH_CLOCKS_PER_SEC / 1000000, "MHz");
130
131
132
133     /*
134      *
135      * Generate grid as a graph
136      *
137      */
138
139 #ifdef GRID_GRAPH
140
141     unsigned int u;
142
143     __foreach_core(group, core)
144     {
145         lnh_inst.gpc[group][core]->start_async(__event__(delete_graph));
146     }
147
148
149     unsigned int*
            host2gpc_ext_buffer[LNH_GROUPS_COUNT][LNH_MAX_CORES_IN_GROUP];
150     unsigned int messages_count = 0;
151     __foreach_core(group, core)
152     {
153             host2gpc_ext_buffer[group][core] = (unsigned
                    int*)lnh_inst.gpc[group][core]->external_memory_create_bu
154             offs = 0;
155             std::ifstream stream;
156             stream.open(DATA_FILE);
157             do{
158                     unsigned int u, v, w;
159                     stream >> u >> v >> w;
160                     EDGE(u, v, w);
```

```
161                     EDGE(v, u, w);
162                     messages_count += 2;
163             }while(!stream.eof());
164             lnh_inst.gpc[group][core]->external_memory_sync_to_device(0,
                    3*sizeof(unsigned int)*messages_count);
165     }
166 //      return 0;
167     __foreach_core(group, core)
168     {
169             lnh_inst.gpc[group][core]->start_async(__event__(insert_edges
170     }
171     __foreach_core(group, core) {
172             long long tmp =
                    lnh_inst.gpc[group][core]->external_memory_address();
173             lnh_inst.gpc[group][core]->mq_send((unsigned int)tmp);
174     }
175     __foreach_core(group, core) {
176             lnh_inst.gpc[group][core]->mq_send(3*sizeof(unsigned
                    int)*messages_count);
177     }
178
179
180     __foreach_core(group, core)
181     {
182             lnh_inst.gpc[group][core]->finish();
183     }
184     printf("Data␣graph␣created!\n");
185
186
187
188 #endif
189
190
191     /*
192      *
193      * Generate random graph
194      *
195      */
196
197 #ifdef RAND_GRAPH
198
```

```
199     __foreach_core(group, core)
200     {
201         lnh_inst.gpc[group][core]->start_async(__event__(delete_graph));
202     }
203
204
205     unsigned int*
            host2gpc_ext_buffer[LNH_GROUPS_COUNT][LNH_MAX_CORES_IN_GROUP];
206     unsigned int vertex_count = GRAPH_SIZE_X * GRAPH_SIZE_Y;
207     unsigned int edge_count = vertex_count;
208     unsigned int subgraph_count = 10;
209     unsigned int messages_count = 0;
210     unsigned int u, v, w;
211
212     __foreach_core(group, core)
213     {
214
215         host2gpc_ext_buffer[group][core] = (unsigned
                int*)lnh_inst.gpc[group][core]->external_memory_create_buffer(
                * 1048576 * sizeof(int));
                //2*3*sizeof(int)*edge_count);
216         offs = 0;
217
218         //Граф должен быть связным
219         u = rand() % vertex_count;
220         for (int edge = 0; edge < edge_count; edge++) {
221             do
222                 v = rand() % vertex_count;
223             while (v == u);
224             w = 1;
225             EDGE(u, v, w);
226             EDGE(v, u, w);
227             messages_count += 2;
228             u = v;
229         }
230
231
232         //Создание связанных подграфов для демонстрации алгоритма
                выделения сообществ
233         for (int subgraph = 0; subgraph < subgraph_count;
                subgraph++) {
```

```
234             //Связаны все вершины подграфа
235             unsigned int subgraph_vcount = rand() % 20;
236             unsigned int subgraph_vstart = rand() % (vertex_count
                    - subgraph_vcount);
237             for (int vi = subgraph_vstart; vi < subgraph_vstart +
                    subgraph_vcount; vi++) {
238                 for (int vj = vi + 1; vj < subgraph_vstart +
                        subgraph_vcount; vj++) {
239                     w = 1;
240                     EDGE(vi, vj, w);
241                     EDGE(vj, vi, w);
242                     messages_count += 2;
243                 }
244             }
245         }
246
247
248
249         lnh_inst.gpc[group][core]->external_memory_sync_to_device(0,
                3 * sizeof(unsigned int)*messages_count);
250     }
251     __foreach_core(group, core)
252     {
253         lnh_inst.gpc[group][core]->start_async(__event__(insert_edges));
254     }
255     __foreach_core(group, core) {
256         long long tmp =
                lnh_inst.gpc[group][core]->external_memory_address();
257         lnh_inst.gpc[group][core]->mq_send((unsigned int)tmp);
258     }
259     __foreach_core(group, core) {
260         lnh_inst.gpc[group][core]->mq_send(3 *
                sizeof(int)*messages_count);
261     }
262
263
264     __foreach_core(group, core)
265     {
266         lnh_inst.gpc[group][core]->finish();
267     }
268     printf("Data␣graph␣created!\n");
```

```
269
270
271
272 #endif
273
274
275     /*
276      *
277      * Run BTWC
278      *
279      */
280
281     start = clock();
282
283     __foreach_core(group, core)
284     {
285         lnh_inst.gpc[group][core]->start_async(__event__(btwc));
286     }
287
288
289     __foreach_core(group, core)
290     {
291         lnh_inst.gpc[group][core]->finish();
292     }
293
294     stop = clock();
295
296     printf("\nBTWC is done for %.2f seconds\n", (float(stop -
        start) / CLOCKS_PER_SEC));
297
298
299
300     /*
301      *
302      * Show btwc
303      *
304      */
305     int sock_fd = server_socket_init();
306     int client_fd;
307
308     printf("Create visualisation\n");
```

```
309     __foreach_core(group, core)
310     {
311 // #ifdef GRID_LAYOUT
312 //
    lnh_inst.gpc[group][core]->start_async(__event__(create_visualization)
313 //       lnh_inst.gpc[group][core]->mq_send(GRAPH_SIZE_X);
314 //       lnh_inst.gpc[group][core]->mq_send(GRAPH_SIZE_Y);
315 // #endif
316 // #ifdef GRID_SPIRAL_LAYOUT
317 //
    lnh_inst.gpc[group][core]->start_async(__event__(create_centrality_vis
318 // #endif
319 // #ifdef SPIRAL_LAYOUT
320 //
    lnh_inst.gpc[group][core]->start_async(__event__(create_centrality_sp
321 // #endif
322 #ifdef BOX_LAYOUT
323         lnh_inst.gpc[group][core]->start_async(__event__(create_communit
324 #endif
325 #ifdef FORCED_LAYOUT
326         lnh_inst.gpc[group][core]->start_async(__event__(create_communit
327 #endif
328
329 #ifdef DEBUG
330         //DEBUG
331         unsigned int handler_state;
332         unsigned int com_u, com_v, com_k, com_r, v_count,
              delta_mod, modularity;
333         short unsigned int x, y, color, size, btwc, first_vertex,
              last_vertex;
334
335         printf("1 этап: инициализация временных структур\n");
336         handler_state = lnh_inst.gpc[group][core]->mq_receive();
337         while (handler_state != 0) {
338             com_u = lnh_inst.gpc[group][core]->mq_receive();
339             com_v = lnh_inst.gpc[group][core]->mq_receive();
340             printf("Количество сообществ в очереди %u и в структур
                  е сообществ %u\n", com_u, com_v);
341             printf("Количество вершин в графе %u\n",
                  lnh_inst.gpc[group][core]->mq_receive());
```

```
342             handler_state =
                    lnh_inst.gpc[group][core]->mq_receive();
343         }
344
345         printf("II этап: выделение сообществ\n");
346         handler_state = lnh_inst.gpc[group][core]->mq_receive();
347         while (handler_state != 0) {
348             switch (handler_state) {
349             case -1:
350                 com_u = lnh_inst.gpc[group][core]->mq_receive();
351                 com_v = lnh_inst.gpc[group][core]->mq_receive();
352                 delta_mod =
                        lnh_inst.gpc[group][core]->mq_receive();
353                 modularity =
                        lnh_inst.gpc[group][core]->mq_receive();
354                 printf("Объединение в сообщество вершин %u и %u :
                        \tdM = %d\tM = %d\n", com_u, com_v, delta_mod,
                        modularity);
355                 break;
356             case -2:
357                 com_u = lnh_inst.gpc[group][core]->mq_receive();
358                 com_v = lnh_inst.gpc[group][core]->mq_receive();
359                 delta_mod =
                        lnh_inst.gpc[group][core]->mq_receive();
360                 printf("\tМодификация связности сообществ %u и %u
                        : \tdM = %d\n", com_u, com_v, delta_mod);
361                 break;
362             default: break;
363             }
364             handler_state =
                    lnh_inst.gpc[group][core]->mq_receive();
365         }
366
367         printf("Тест итераторов сообщества\n");
368         handler_state = lnh_inst.gpc[group][core]->mq_receive();
369         while (handler_state != 0) {
370             int community =
                    lnh_inst.gpc[group][core]->mq_receive();
371             int first_vertex =
                    lnh_inst.gpc[group][core]->mq_receive();
```

```
372             int last_vertex =
                    lnh_inst.gpc[group][core]->mq_receive();
373             printf("Сообщество %u. Начальная вершина %u - Конечная
                    вершина %u\n", community, first_vertex,
                    last_vertex);
374             handler_state =
                    lnh_inst.gpc[group][core]->mq_receive();
375             while (handler_state != 0) {
376                 int vertex =
                        lnh_inst.gpc[group][core]->mq_receive();
377                 printf("%u->", vertex);
378                 handler_state =
                        lnh_inst.gpc[group][core]->mq_receive();
379             }
380             printf("\n");
381             handler_state =
                    lnh_inst.gpc[group][core]->mq_receive();
382         }
383
384 #ifdef BOX_LAYOUT
385         printf("III этап: построение дерева сообществ\n");
386         handler_state = lnh_inst.gpc[group][core]->mq_receive();
387         while (handler_state != 0) {
388             switch (handler_state) {
389             case -3:
390                 com_u = lnh_inst.gpc[group][core]->mq_receive();
391                 com_v = lnh_inst.gpc[group][core]->mq_receive();
392                 printf("Количество сообществ в очереди %u и в стру
                        ктуре сообществ %u\n", com_u, com_v);
393                 break;
394             case -4:
395                 com_u = lnh_inst.gpc[group][core]->mq_receive();
396                 com_v = lnh_inst.gpc[group][core]->mq_receive();
397                 delta_mod =
                        lnh_inst.gpc[group][core]->mq_receive();
398                 modularity =
                        lnh_inst.gpc[group][core]->mq_receive();
399                 v_count = lnh_inst.gpc[group][core]->mq_receive();
400                 com_r = lnh_inst.gpc[group][core]->mq_receive();
401                 printf("Создание дерева сообществ из сообществ %u
                        и %u в  сообщество %u, количество вершин %u: 
```

```
                    \tdM␣=␣%d\tM␣=␣%d\n", com_u, com_v, com_r,
                    v_count, delta_mod, modularity);
402                 break;
403             default: break;
404             }
405             handler_state =
                    lnh_inst.gpc[group][core]->mq_receive();
406         }
407 #endif
408 #ifdef FORCED_LAYOUT
409         printf("III␣этап:␣Размещение␣сообществ␣силовым␣алгоритмом
                \n");
410         handler_state = lnh_inst.gpc[group][core]->mq_receive();
411         while (handler_state != 0) {
412             int u = lnh_inst.gpc[group][core]->mq_receive();
413             int x = lnh_inst.gpc[group][core]->mq_receive();
414             int y = lnh_inst.gpc[group][core]->mq_receive();
415             int displacement =
                    lnh_inst.gpc[group][core]->mq_receive();
416             printf("Размещение␣сообщества␣%u␣в␣области␣(%d,%d),␣
                    disp=%u\n", u, x, y, displacement);
417             handler_state =
                    lnh_inst.gpc[group][core]->mq_receive();
418         }
419 #endif
420 #ifdef BOX_LAYOUT
421         printf("IV␣этап:␣выделение␣прямоугольных␣областей\n");
422         handler_state = lnh_inst.gpc[group][core]->mq_receive();
423         while (handler_state != 0) {
424             com_u = lnh_inst.gpc[group][core]->mq_receive();
425             unsigned int v_count =
                    lnh_inst.gpc[group][core]->mq_receive();
426             short unsigned int x0 =
                    lnh_inst.gpc[group][core]->mq_receive();
427             short unsigned int y0 =
                    lnh_inst.gpc[group][core]->mq_receive();
428             short unsigned int x1 =
                    lnh_inst.gpc[group][core]->mq_receive();
429             short unsigned int y1 =
                    lnh_inst.gpc[group][core]->mq_receive();
```

```
430             short unsigned int is_leaf =
                    lnh_inst.gpc[group][core]->mq_receive();
431             printf("Выделение прямоугольной области для сообщества
                    %u, %u вершин, лист (%u), координаты:
                    (%d,%d)-(%u,%u)\n", com_u, v_count, is_leaf, x0,
                    y0, x1, y1);
432             handler_state =
                    lnh_inst.gpc[group][core]->mq_receive();
433         }
434 #endif
435 #ifdef FORCED_LAYOUT
436         printf("IV этап: масштабирование в границы оласти\n");
437         handler_state = lnh_inst.gpc[group][core]->mq_receive();
438         while (handler_state != 0) {
439             switch (handler_state) {
440             case -4: {
441                 unsigned int scale =
                        lnh_inst.gpc[group][core]->mq_receive();
442                 printf("Коэффициент масштабирования: %u /
                        1000\n", scale);
443                 break;}
444             case -5: {
445                 unsigned int u =
                        lnh_inst.gpc[group][core]->mq_receive();
446                 int x = lnh_inst.gpc[group][core]->mq_receive();
447                 int y = lnh_inst.gpc[group][core]->mq_receive();
448                 unsigned int distance =
                        lnh_inst.gpc[group][core]->mq_receive();
449                 printf("Размещение сообщества %u в область
                        (%d,%d), диаметр (%u)\n", u, x, y, distance);
450                 break;}
451             default: break;
452             }
453             handler_state =
                    lnh_inst.gpc[group][core]->mq_receive();
454         }
455 #endif
456 #ifdef BOX_LAYOUT
457         printf("V этап: определение координат вершин\n");
458         handler_state = lnh_inst.gpc[group][core]->mq_receive();
459         while (handler_state != 0) {
```

```
460             switch (handler_state) {
461             case -6:
462                 com_u = lnh_inst.gpc[group][core]->mq_receive();
463                 v_count = lnh_inst.gpc[group][core]->mq_receive();
464                 first_vertex =
                        lnh_inst.gpc[group][core]->mq_receive();
465                 last_vertex =
                        lnh_inst.gpc[group][core]->mq_receive();
466                 printf("Сообщество %u (вершины %u - %u), всего вер
                        шин (%u)\n", com_u, first_vertex, last_vertex,
                        v_count);
467                 break;
468             case -7:
469                 com_u = lnh_inst.gpc[group][core]->mq_receive();
470                 u = lnh_inst.gpc[group][core]->mq_receive();
471                 x = lnh_inst.gpc[group][core]->mq_receive();
472                 y = lnh_inst.gpc[group][core]->mq_receive();
473                 color = lnh_inst.gpc[group][core]->mq_receive();
474                 size = lnh_inst.gpc[group][core]->mq_receive();
475                 btwc = lnh_inst.gpc[group][core]->mq_receive();
476                 printf("Сообщество %u, вершина %u, координаты:
                        (%u,%u)\n", com_u, u, x, y);
477                 break;
478             default: break;
479             }
480             handler_state =
                    lnh_inst.gpc[group][core]->mq_receive();
481         }
482 #endif
483 #ifdef FORCED_LAYOUT
484         printf("V этап: раскладка сообществ в областях\n");
485         handler_state = lnh_inst.gpc[group][core]->mq_receive();
486         while (handler_state != 0) {
487             com_u = lnh_inst.gpc[group][core]->mq_receive();
488             int u = lnh_inst.gpc[group][core]->mq_receive();
489             int x = lnh_inst.gpc[group][core]->mq_receive();
490             int y = lnh_inst.gpc[group][core]->mq_receive();
491             //int displacement =
                    lnh_inst.gpc[group][core]->mq_receive();
492             //printf("Размещение сообщества %u: вершина %u помещае
                    тся в (%d,%d), disp=%d\n", com_u, u, x, y,
```

```
                displacement);
493             printf("Размещение сообщества %u: вершина %u помещается в (%d,%d)\n", com_u, u, x, y);
494             handler_state =
                    lnh_inst.gpc[group][core]->mq_receive();
495         }
496 #endif
497 #endif
498     }
499
500     printf("Wait for connections\n");
501     while ((client_fd = accept(sock_fd, NULL, NULL)) != -1) {
502         printf("New connection\n");
503         __foreach_core(group, core) {
504             lnh_inst.gpc[group][core]->start_async(__event__(get_first_ve
505             if (lnh_inst.gpc[group][core]->mq_receive() != 0) {
506                 do {
507                     u = lnh_inst.gpc[group][core]->mq_receive();
508                     lnh_inst.gpc[group][core]->start_async(__event__(get_
509                     lnh_inst.gpc[group][core]->mq_send(u);
510                     unsigned int adj_c =
                            lnh_inst.gpc[group][core]->mq_receive();
511                     unsigned int pu =
                            lnh_inst.gpc[group][core]->mq_receive();
512                     unsigned int du =
                            lnh_inst.gpc[group][core]->mq_receive();
513                     unsigned int btwc =
                            lnh_inst.gpc[group][core]->mq_receive();
514                     unsigned int x =
                            lnh_inst.gpc[group][core]->mq_receive();
515                     unsigned int y =
                            lnh_inst.gpc[group][core]->mq_receive();
516                     unsigned int size =
                            lnh_inst.gpc[group][core]->mq_receive();
517                     unsigned int color =
                            lnh_inst.gpc[group][core]->mq_receive();
518                     write(client_fd, &u, sizeof(u));
519                     write(client_fd, &btwc, sizeof(btwc));
520                     write(client_fd, &adj_c, sizeof(adj_c));
521                     write(client_fd, &x, sizeof(x));
522                     write(client_fd, &y, sizeof(y));
```

```
523                     printf("(x,y,size)=%u,%u,%u\n", x, y, size);
524                     printf("Вершина %u – центральность %u –
                           (x,y,size)=%u,%u,%u связность %u\n", u,
                           btwc, x, y, size, adj_c);
525                     write(client_fd, &size, sizeof(size));
526                     write(client_fd, &color, sizeof(color));
527                     for (int i = 0; i < adj_c; i++) {
528                         unsigned int v =
                               lnh_inst.gpc[group][core]->mq_receive();
529                         unsigned int w =
                               lnh_inst.gpc[group][core]->mq_receive();
530                         write(client_fd, &v, sizeof(v));
531                         write(client_fd, &w, sizeof(w));
532                         //printf("Ребро с вершиной %u, вес
                               %u\n",v,w);
533                     }
534                     lnh_inst.gpc[group][core]->start_async(__event__(get_
535                     lnh_inst.gpc[group][core]->mq_send(u);
536                 } while (lnh_inst.gpc[group][core]->mq_receive()
                       != 0);
537
538             }
539         }
540
541         close(client_fd);
542     }
543
544     now = time(0);
545     strftime(buf, 100, "Stop at local date: %d.%m.%Y.; local
           time: %H.%M.%S", localtime(&now));
546     printf("DISC system speed test v1.1\n%s\n\n", buf);
547
548     //------------------------------------------------------------------
549     // Shutdown and cleanup
550     //------------------------------------------------------------------
551
552     if (err)
553     {
554         printf("ERROR: Test failed\n");
555         return EXIT_FAILURE;
556     }
```

```
557     else
558     {
559         printf("INFO: Test completed successfully.\n");
560         return EXIT_SUCCESS;
561     }
562
563
564
565
566
567     return 0;
568 }
```

2.2.2 sw_kernel

Листинг 2.2 – Измененный код sw_kernel под индивидульное задание

```
1  /*
2   * gpc_test.c
3   *
4   * sw_kernel library
5   *
6   *  Created on: April 23, 2021
7   *      Author: A.Popov
8   */
9
10 #include <stdlib.h>
11 #include "lnh64.h"
12 #include "gpc_io_swk.h"
13 #include "gpc_handlers.h"
14 #include "dijkstra.h"
15
16 #define VERSION 26
17 #define DEFINE_LNH_DRIVER
18 #define DEFINE_MQ_R2L
19 #define DEFINE_MQ_L2R
20 #define ROM_LOW_ADDR 0x00000000
21 #define ITERATIONS_COUNT    1
22 #define MEASURE_KEY_COUNT   1000000
23 #define __fast_recall__
```

```
24
25 extern lnh lnh_core;
26 extern global_memory_io gmio;
27 volatile unsigned int event_source;
28
29 int main(void) {
30     ////////////////////////////////////////////////////
31     //                  Main Event Loop
32     ////////////////////////////////////////////////////
33     //Leonhard driver structure should be initialised
34     lnh_init();
35     //Initialise host2gpc and gpc2host queues
36     gmio_init(lnh_core.partition.data_partition);
37     for (;;) {
38         //Wait for event
39         while (!gpc_start());
40         //Enable RW operations
41         set_gpc_state(BUSY);
42         //Wait for event
43         event_source = gpc_config();
44         switch(event_source) {
45             ////////////////////////////////////////////
46             //  Measure GPN operation frequency
47             ////////////////////////////////////////////
48             case __event__(frequency_measurement) :
                   frequency_measurement(); break;
49             case __event__(get_lnh_status_low) :
                   get_lnh_status_low(); break;
50             case __event__(get_lnh_status_high) :
                   get_lnh_status_high(); break;
51             case __event__(get_version): get_version(); break;
52             case __event__(dijkstra): dijkstra(); break;
53             case __event__(insert_edges): insert_edges(); break;
54             case __event__(get_vertex_data): get_vertex_data();
                   break;
55             case __event__(get_first_vertex): get_first_vertex();
                   break;
56             case __event__(get_next_vertex): get_next_vertex();
                   break;
57             case __event__(delete_graph): delete_graph(); break;
```

```
58              case __event__(delete_visualization):
                    delete_visualization(); break;
59              case __event__(create_visualization):
                    create_visualization(); break;
60              case __event__(set_visualization_attributes):
                    set_visualization_attributes(); break;
61              case __event__(create_centrality_visualization):
                    create_centrality_visualization(); break;
62              case
                    __event__(create_centrality_spiral_visualization):
                    create_centrality_spiral_visualization(); break;
63              case
                    __event__(create_communities_forest_vizualization):
                    create_communities_forest_vizualization(); break;
64              case
                    __event__(create_communities_forced_vizualization):
                    create_communities_forced_vizualization(); break;
65              case __event__(btwc): btwc(); break;
66
67          }
68          //Disable RW operations
69          set_gpc_state(IDLE);
70          while (gpc_start());
71
72      }
73  }
74
75  //-------------------------------------------------------------------
76  //      Глобальные переменные (для сокращения объема кода)
77  //-------------------------------------------------------------------
78
79          unsigned int LNH_key;
80          unsigned int LNH_value;
81          unsigned int LNH_status;
82          uint64_t TSC_start;
83          uint64_t TSC_stop;
84          unsigned int interval;
85          int i,j;
86          unsigned int err=0;
87
88
```

```
89 //-------------------------------------------------------------------
90 //      Измерение тактовой частоты GPN
91 //-------------------------------------------------------------------
92
93 void frequency_measurement() {
94
95         sync_with_host();
96         lnh_sw_reset();
97         lnh_rd_reg32_byref(TSC_LOW,&TSC_start);
98         sync_with_host();
99         lnh_rd_reg32_byref(TSC_LOW,&TSC_stop);
100        interval = TSC_stop-TSC_start;
101        mq_send(interval);
102
103 }
104
105
106 //-------------------------------------------------------------------
107 //      Получить версию микрокода
108 //-------------------------------------------------------------------
109
110 void get_version() {
111
112        mq_send(VERSION);
113
114 }
115
116
117 //-------------------------------------------------------------------
118 //      Получить регистр статуса LOW Leonhard
119 //-------------------------------------------------------------------
120
121 void get_lnh_status_low() {
122
123        lnh_rd_reg32_byref(LNH_STATE_LOW,&lnh_core.result.status);
124        mq_send(lnh_core.result.status);
125
126 }
127
128 //-------------------------------------------------------------------
129 //      Получить регистр статуса HIGH Leonhard
```

```
130 //-------------------------------------------------------------------
131
132 void get_lnh_status_high() {
133
134         lnh_rd_reg32_byref(LNH_STATE_HIGH,&lnh_core.result.status);
135         mq_send(lnh_core.result.status);
136
137 }
```

2.2.3 Полученный граф

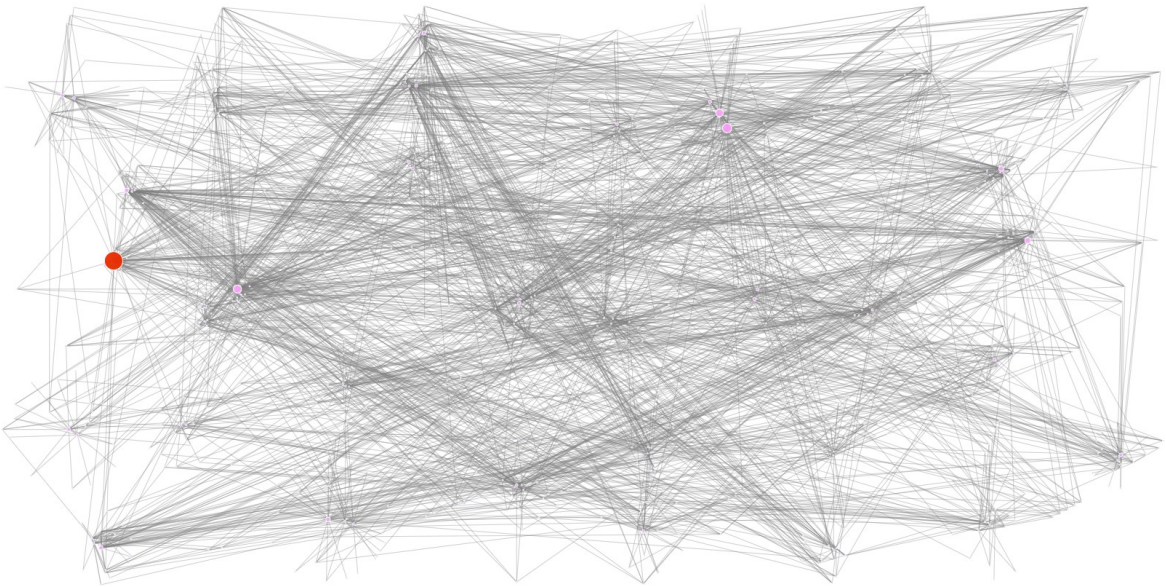

Рисунок 2.1 – Полученный граф по варианту 27