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: _____ Динь Вьет Ань
подпись, дата Фамилия, И.О.

Преподаватель: _____ Ибрагимов С. В.
подпись, дата Фамилия, И. О.

Москва — 2023 г.

Содержание

Цель работы

Практикум посвящен освоению принципов представления графов и их обработке с помощью вычислительного комплекса Тераграф. 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

```
1 #include "host_main.h"
2 #include <unistd.h>
3 #include <sys/types.h>
4 #include <sys/socket.h>
5 #include <netinet/ip.h>
6 #include <stdlib.h>
7 #include <assert.h>
8 #include <string.h>
9 #include <stdio.h>
10
11 #include <fstream>
12 #include <iostream>
13 #include <string>
14 #include <vector>
15 #include <sstream>
16
17 #define SRC_FILE "graf.tsv"
18
19 using namespace std;
20
21 #define RAND_GRAPH
```

```
22 //#define GRID_GRAPH
23 #define BOX_LAYOUT
24 //#define FORCED_LAYOUT
25 #define DEBUG
26
27 #define handle_error(msg) \
28 do { perror(msg); exit(EXIT_FAILURE); } while (0)
29
30 int get_edge_count(std::string filename)
31 {
32     std::ifstream fin(filename);
33     printf("%d\n", fin.is_open());
34     string line;
35     int count = 0;
36     while (getline(fin, line, '\n')) {
37         ++count;
38     }
39     fin.close();
40     return count;
41 }
42
43 static void usage()
44 {
45     std::cout << "usage: <xclbin> <sw_kernel>\n\n";
46 }
47
48 static void print_table(std::string test, float value,
      std::string units)
49 {
50     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;
51     std::cout << std::setfill('-') << std::setw(85) << "-" <<
          std::endl;
52 }
53 const int port = 0x4747;
54 int server_socket_init() {
55     int sock_fd;
56     struct sockaddr_in srv_addr;
57     int client_fd;
```

```
58      sock_fd = socket(AF_INET, SOCK_STREAM, 0);
59      if (sock_fd == -1)
60      handle_error("socket");
61      memset(&srv_addr, 0, sizeof(srv_addr));
62      srv_addr.sin_family = AF_INET;
63      srv_addr.sin_port = htons(port);
64      srv_addr.sin_addr.s_addr = INADDR_ANY;
65      if (bind(sock_fd, (struct sockaddr *)&srv_addr,
           sizeof(srv_addr)) == -1)
66      handle_error("bind");
67      if (listen(sock_fd, 2) == -1)
68      handle_error("listen");
69      return sock_fd;
70  }
71
72  int main(int argc, char** argv)
73  {
74
75      unsigned int err = 0;
76      unsigned int cores_count = 0;
77      float LNH_CLOCKS_PER_SEC;
78      clock_t start, stop;
79
80      __foreach_core(group, core) cores_count++;
81
82      //Assign xclbin
83      if (argc < 3) {
84          usage();
85          throw std::runtime_error("FAILED_TEST\nNo␣xclbin␣
               specified");
86      }
87
88      //Open device #0
89      leonhardx64 lnh_inst = leonhardx64(0, argv[1]);
90      __foreach_core(group, core)
91      {
92          lnh_inst.load_sw_kernel(argv[2], group, core);
93      }
94
95      /*
96      *
```

```
97      * SW Kernel Version and Status
98      *
99      */
100     __foreach_core(group, core)
101     {
102         printf("Group #%d \tCore #%d\n", group, core);
103         lnh_inst.gpc[group][core]->start_sync(__event__(get_version));
104         printf("\tSoftware Kernel Version:\t0x%08x\n",
                lnh_inst.gpc[group][core]->mq_receive());
105         lnh_inst.gpc[group][core]->start_sync(__event__(get_lnh_status_hi
106         printf("\tLeonhard Status Register:\t0x%08x",
                lnh_inst.gpc[group][core]->mq_receive());
107         lnh_inst.gpc[group][core]->start_sync(__event__(get_lnh_status_lo
108         printf("_%08x\n",
                lnh_inst.gpc[group][core]->mq_receive());
109     }
110
111
112     //-------------------------------------------------------------------
113     // Измерение производительности Leonhard
114     //-------------------------------------------------------------------
115
116     float interval;
117     char buf[100];
118     err = 0;
119
120     time_t now = time(0);
121     strftime(buf, 100, "Start at local date: %d.%m.%Y.; local
            time: %H.%M.%S", localtime(&now));
122
123     printf("\nDISC system speed test v3.0\n%s\n\n", buf);
124     std::cout << std::left << std::setw(50) << "Test" <<
            std::right << std::setw(20) << "value" << std::setw(15) <<
            "units" << std::endl;
125     std::cout << std::setfill('-') << std::setw(85) << "-" <<
            std::endl;
126     print_table("Graph Processing Cores count (GPCC)",
            cores_count, "instances");
127
128
129
```

```
130
131     /*
132     *
133     * GPC frequency measurement for the first kernel
134     *
135     */
136     lnh_inst.gpc[0][LNH_CORES_LOW[0]]->start_async(__event__(frequency_me
137
138     // Measurement Body
139     lnh_inst.gpc[0][LNH_CORES_LOW[0]]->sync_with_gpc(); // Start
            measurement
140     sleep(1);
141     lnh_inst.gpc[0][LNH_CORES_LOW[0]]->sync_with_gpc(); // Start
            measurement
142     // End Body
143     lnh_inst.gpc[0][LNH_CORES_LOW[0]]->finish();
144     LNH_CLOCKS_PER_SEC =
            (float)lnh_inst.gpc[0][LNH_CORES_LOW[0]]->mq_receive();
145     print_table("Leonhard␣clock␣frequency␣(LNH_CF)",
            LNH_CLOCKS_PER_SEC / 1000000, "MHz");
146
147
148
149     /*
150     *
151     * Generate grid as a graph
152     *
153     */
154
155     #ifdef GRID_GRAPH
156
157     unsigned int u;
158
159     __foreach_core(group, core)
160     {
161         lnh_inst.gpc[group][core]->start_async(__event__(delete_graph));
162     }
163
164
165     unsigned int*
            host2gpc_ext_buffer[LNH_GROUPS_COUNT][LNH_MAX_CORES_IN_GROUP];
```

```
166
167     __foreach_core(group, core)
168     {
169         host2gpc_ext_buffer[group][core] = (unsigned
                int*)lnh_inst.gpc[group][core]->external_memory_create_buffer(
170         offs = 0;
171         //Угловые вершины имеют 3 ребра
172         //Top Left
173         EDGE(0, 1, 2);                  //east
174         EDGE(0, GRAPH_SIZE_X, 2);     //south
175         EDGE(0, GRAPH_SIZE_X + 1, 3);    //south-east
176         //Top Right
177         EDGE(GRAPH_SIZE_X - 1, GRAPH_SIZE_X - 2, 2);          //west
178         EDGE(GRAPH_SIZE_X - 1, 2 * GRAPH_SIZE_X - 1, 2);
                //south
179         EDGE(GRAPH_SIZE_X - 1, 2 * GRAPH_SIZE_X - 2, 3);
                //south-west
180         //Bottom Left
181         EDGE(GRAPH_SIZE_X * (GRAPH_SIZE_Y - 1), GRAPH_SIZE_X *
                (GRAPH_SIZE_Y - 2), 2);  //north
182         EDGE(GRAPH_SIZE_X * (GRAPH_SIZE_Y - 1), GRAPH_SIZE_X *
                (GRAPH_SIZE_Y - 1) + 1, 2);  //east
183         EDGE(GRAPH_SIZE_X * (GRAPH_SIZE_Y - 1), GRAPH_SIZE_X *
                (GRAPH_SIZE_Y - 2) + 1, 3);  //north_east
184         //Bottom Right
185         EDGE(GRAPH_SIZE_X * GRAPH_SIZE_Y - 1, GRAPH_SIZE_X *
                (GRAPH_SIZE_Y - 1) - 1, 2);     //north
186         EDGE(GRAPH_SIZE_X * GRAPH_SIZE_Y - 1, GRAPH_SIZE_X *
               GRAPH_SIZE_Y - 2, 2);       //west
187         EDGE(GRAPH_SIZE_X * GRAPH_SIZE_Y - 1, GRAPH_SIZE_X *
                (GRAPH_SIZE_Y - 1) - 2, 3);     //north-west
188         //Left and Right sides
189         for (int y = 1; y < GRAPH_SIZE_Y - 1; y++) {
190             //Left
191             EDGE(GRAPH_SIZE_X * y, GRAPH_SIZE_X * (y - 1), 2);
                    //north
192             EDGE(GRAPH_SIZE_X * y, GRAPH_SIZE_X * (y + 1), 2);
                    //south
193             EDGE(GRAPH_SIZE_X * y, GRAPH_SIZE_X * y + 1, 2);
                    //east
```

```
194             EDGE(GRAPH_SIZE_X * y, GRAPH_SIZE_X * (y - 1) + 1,
                    3);  //north-east
195             EDGE(GRAPH_SIZE_X * y, GRAPH_SIZE_X * (y + 1) + 1,
                    3);  //south-east
196             //Right
197             EDGE(GRAPH_SIZE_X * (y + 1) - 1, GRAPH_SIZE_X * y -
                    1, 2);       //north
198             EDGE(GRAPH_SIZE_X * (y + 1) - 1, GRAPH_SIZE_X * (y +
                    2) - 1, 2);     //south
199             EDGE(GRAPH_SIZE_X * (y + 1) - 1, GRAPH_SIZE_X * (y +
                    1) - 2, 2);     //west
200             EDGE(GRAPH_SIZE_X * (y + 1) - 1, GRAPH_SIZE_X * y -
                    2, 3);       //north-west
201             EDGE(GRAPH_SIZE_X * (y + 1) - 1, GRAPH_SIZE_X * (y +
                    2) - 2, 3);     //south-west
202         }
203
204         for (int x = 1; x < GRAPH_SIZE_X - 1; x++) {
205             //Top
206             EDGE(x, x - 1, 2);  //east
207             EDGE(x, x + 1, 2);  //west
208             EDGE(x, GRAPH_SIZE_X + x, 2);        //south
209             EDGE(x, GRAPH_SIZE_X + x - 1, 3);    //south-east
210             EDGE(x, GRAPH_SIZE_X + x + 1, 3);    //south-west
211             //Bottom
212             EDGE(GRAPH_SIZE_X * (GRAPH_SIZE_Y - 1) + x,
                    GRAPH_SIZE_X * (GRAPH_SIZE_Y - 1) + x - 1, 2);
                    //east
213             EDGE(GRAPH_SIZE_X * (GRAPH_SIZE_Y - 1) + x,
                    GRAPH_SIZE_X * (GRAPH_SIZE_Y - 1) + x + 1, 2);
                    //west
214             EDGE(GRAPH_SIZE_X * (GRAPH_SIZE_Y - 1) + x,
                    GRAPH_SIZE_X * (GRAPH_SIZE_Y - 2) + x, 2);
                    //north
215             EDGE(GRAPH_SIZE_X * (GRAPH_SIZE_Y - 1) + x,
                    GRAPH_SIZE_X * (GRAPH_SIZE_Y - 2) + x - 1, 3);
                    //north-east
216             EDGE(GRAPH_SIZE_X * (GRAPH_SIZE_Y - 1) + x,
                    GRAPH_SIZE_X * (GRAPH_SIZE_Y - 2) + x + 1, 3);
                    //north-west
217         }
```

```
218
219         for (int y = 1; y < GRAPH_SIZE_Y - 1; y++)
220         for (int x = 1; x < GRAPH_SIZE_X - 1; x++) {
221             EDGE(x + GRAPH_SIZE_X * y, x + GRAPH_SIZE_X * (y -
                    1), 2);  //north
222             EDGE(x + GRAPH_SIZE_X * y, x + GRAPH_SIZE_X * (y +
                    1), 2);  //south
223             EDGE(x + GRAPH_SIZE_X * y, x + GRAPH_SIZE_X * y - 1,
                    2);         //east
224             EDGE(x + GRAPH_SIZE_X * y, x + GRAPH_SIZE_X * y + 1,
                    2);         //west
225             EDGE(x + GRAPH_SIZE_X * y, x + GRAPH_SIZE_X * (y - 1)
                    - 1, 3);  //north-east
226             EDGE(x + GRAPH_SIZE_X * y, x + GRAPH_SIZE_X * (y + 1)
                    - 1, 3);  //south-east
227             EDGE(x + GRAPH_SIZE_X * y, x + GRAPH_SIZE_X * (y - 1)
                    + 1, 3);  //north-west
228             EDGE(x + GRAPH_SIZE_X * y, x + GRAPH_SIZE_X * (y + 1)
                    + 1, 3);  //south-west
229         }
230         Inh_inst.gpc[group][core]->external_memory_sync_to_device(0,
                BIFFER_SIZE);
231     }
232     __foreach_core(group, core)
233     {
234         Inh_inst.gpc[group][core]->start_async(__event__(insert_edges));
235     }
236     __foreach_core(group, core) {
237         long long tmp =
                Inh_inst.gpc[group][core]->external_memory_address();
238         Inh_inst.gpc[group][core]->mq_send((unsigned int)tmp);
239     }
240     __foreach_core(group, core) {
241         Inh_inst.gpc[group][core]->mq_send(BIFFER_SIZE);
242     }
243
244
245     __foreach_core(group, core)
246     {
247         Inh_inst.gpc[group][core]->finish();
248     }
```

```
249     printf("Data graph created!\n");
250
251
252     #endif
253
254
255     /*
256     *
257     * Generate random graph
258     *
259     */
260
261     #ifdef RAND_GRAPH
262
263     __foreach_core(group, core)
264     {
265         lnh_inst.gpc[group][core]->start_async(__event__(delete_graph));
266     }
267
268
269     unsigned int*
           host2gpc_ext_buffer[LNH_GROUPS_COUNT][LNH_MAX_CORES_IN_GROUP];
270     // unsigned int vertex_count = GRAPH_SIZE_X * GRAPH_SIZE_Y;
271     // unsigned int edge_count = vertex_count;
272     // unsigned int subgraph_count = 10;
273     unsigned int edge_count = get_edge_count(SRC_FILE);
274     unsigned int messages_count = 0;
275     unsigned int u, v, w;
276
277     __foreach_core(group, core)
278     {
279
280         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);
281         offs = 0;
282
283         //Граф должен быть связным
284         // u = rand() % vertex_count;
285         // for (int edge = 0; edge < edge_count; edge++) {
```

```
286             //  do
287             //      v = rand() % vertex_count;
288             //  while (v == u);
289             //  w = 1;
290             //  EDGE(u, v, w);
291             //  EDGE(v, u, w);
292             //  messages_count += 2;
293             //  u = v;
294             // }
295
296         //Создание связанных подграфов для демонстрации алгоритма
                выделения сообществ
297         // for (int subgraph = 0; subgraph < subgraph_count;
                subgraph++) {
298             //  //Связаны все вершины подграфа
299             //  unsigned int subgraph_vcount = rand() % 20;
300             //  unsigned int subgraph_vstart = rand() %
                    (vertex_count - subgraph_vcount);
301             //  for (int vi = subgraph_vstart; vi <
                    subgraph_vstart + subgraph_vcount; vi++) {
302             //      for (int vj = vi + 1; vj <
                        subgraph_vstart + subgraph_vcount; vj++) {
303             //          w = 1;
304             //          EDGE(vi, vj, w);
305             //          EDGE(vj, vi, w);
306             //          messages_count += 2;
307             //      }
308             //  }
309             // }
310
311         ifstream fin(SRC_FILE);
312         cout << "Чтение данных из файла graf.tsv..." << endl;
313         for (int edge = 0; edge < edge_count; ++edge)  {
314             if (!(fin >> v >> u >> w))
315             w = 1;
316             cout << v << " " << u << " " << w << endl;
317             EDGE(u, v, w);
318             EDGE(v, u, w);
319             messages_count += 2;
320         }
321         cout << "Данные считаны!" << endl;
```

```
322         fin.close();
323
324         lnh_inst.gpc[group][core]->external_memory_sync_to_device(0,
                3 * sizeof(unsigned int)*messages_count);
325     }
326     __foreach_core(group, core)
327     {
328         lnh_inst.gpc[group][core]->start_async(__event__(insert_edges));
329     }
330     __foreach_core(group, core) {
331         long long tmp =
                lnh_inst.gpc[group][core]->external_memory_address();
332         lnh_inst.gpc[group][core]->mq_send((unsigned int)tmp);
333     }
334     __foreach_core(group, core) {
335         lnh_inst.gpc[group][core]->mq_send(3 *
                sizeof(int)*messages_count);
336     }
337
338
339     __foreach_core(group, core)
340     {
341         lnh_inst.gpc[group][core]->finish();
342     }
343     printf("Data graph created!\n");
344
345
346
347     #endif
348
349
350     /*
351     *
352     * Run BTWC
353     *
354     */
355
356     start = clock();
357
358     __foreach_core(group, core)
359     {
```

```
360             lnh_inst.gpc[group][core]->start_async(__event__(btwc));
361         }
362 
363 
364         __foreach_core(group, core)
365         {
366             lnh_inst.gpc[group][core]->finish();
367         }
368 
369         stop = clock();
370 
371         printf("\nBTWC is done for %.2f seconds\n", (float(stop -
                start) / CLOCKS_PER_SEC));
372 
373 
374 
375         /*
376         *
377         * Show btwc
378         *
379         */
380         int sock_fd = server_socket_init();
381         int client_fd;
382 
383         printf("Create visualisation\n");
384         __foreach_core(group, core)
385         {
386             //lnh_inst.gpc[group][core]->start_async(__event__(create_visual
387             //lnh_inst.gpc[group][core]->start_async(__event__(create_centra
388             //lnh_inst.gpc[group][core]->start_async(__event__(create_centra
389             #ifdef BOX_LAYOUT
390             lnh_inst.gpc[group][core]->start_async(__event__(create_communi
391             #endif
392             #ifdef FORCED_LAYOUT
393             lnh_inst.gpc[group][core]->start_async(__event__(create_communi
394             #endif
395 
396             #ifdef DEBUG
397             //DEBUG
398             unsigned int handler_state;
```

```
399         unsigned int com_u, com_v, com_k, com_r, v_count,
               delta_mod, modularity;
400         short unsigned int x, y, color, size, btwc, first_vertex,
               last_vertex;
401
402         printf("I␣этап:␣инициализация␣временных␣структур\n");
403         handler_state = lnh_inst.gpc[group][core]->mq_receive();
404         while (handler_state != 0) {
405             com_u = lnh_inst.gpc[group][core]->mq_receive();
406             com_v = lnh_inst.gpc[group][core]->mq_receive();
407             printf("Количество␣сообществ␣в␣очереди␣%u␣и␣в␣структур
                   е␣сообществ␣%u\n", com_u, com_v);
408             printf("Количество␣вершин␣в␣графе␣%u\n",
                   lnh_inst.gpc[group][core]->mq_receive());
409             handler_state =
                   lnh_inst.gpc[group][core]->mq_receive();
410         }
411
412         printf("II␣этап:␣выделение␣сообществ\n");
413         handler_state = lnh_inst.gpc[group][core]->mq_receive();
414         while (handler_state != 0) {
415             switch (handler_state) {
416                 case -1:
417                 com_u = lnh_inst.gpc[group][core]->mq_receive();
418                 com_v = lnh_inst.gpc[group][core]->mq_receive();
419                 delta_mod =
                       lnh_inst.gpc[group][core]->mq_receive();
420                 modularity =
                       lnh_inst.gpc[group][core]->mq_receive();
421                 printf("Объединение␣в␣сообщество␣вершин␣%u␣и␣%u␣:␣
                       \tdM␣=␣%d\tM␣=␣%d\n", com_u, com_v, delta_mod,
                       modularity);
422                 break;
423                 case -2:
424                 com_u = lnh_inst.gpc[group][core]->mq_receive();
425                 com_v = lnh_inst.gpc[group][core]->mq_receive();
426                 delta_mod =
                       lnh_inst.gpc[group][core]->mq_receive();
427                 printf("\tМодификация␣связности␣сообществ␣%u␣и␣%u␣
                       :␣\tdM␣=␣%d\n", com_u, com_v, delta_mod);
428                 break;
```

```
429                 default: break;
430             }
431             handler_state =
                    lnh_inst.gpc[group][core]->mq_receive();
432         }
433
434         printf("Тест итераторов сообщества\n");
435         handler_state = lnh_inst.gpc[group][core]->mq_receive();
436         while (handler_state != 0) {
437             int community =
                    lnh_inst.gpc[group][core]->mq_receive();
438             int first_vertex =
                    lnh_inst.gpc[group][core]->mq_receive();
439             int last_vertex =
                    lnh_inst.gpc[group][core]->mq_receive();
440             printf("Сообщество %u. Начальная вершина %u - Конечная
                    вершина %u\n", community, first_vertex,
                    last_vertex);
441             handler_state =
                    lnh_inst.gpc[group][core]->mq_receive();
442             while (handler_state != 0) {
443                 int vertex =
                        lnh_inst.gpc[group][core]->mq_receive();
444                 printf("%u->", vertex);
445                 handler_state =
                        lnh_inst.gpc[group][core]->mq_receive();
446             }
447             printf("\n");
448             handler_state =
                    lnh_inst.gpc[group][core]->mq_receive();
449         }
450
451         #ifdef BOX_LAYOUT
452         printf("III этап: построение дерева сообществ\n");
453         handler_state = lnh_inst.gpc[group][core]->mq_receive();
454         while (handler_state != 0) {
455             switch (handler_state) {
456                 case -3:
457                 com_u = lnh_inst.gpc[group][core]->mq_receive();
458                 com_v = lnh_inst.gpc[group][core]->mq_receive();
```

```
459                 printf("Количество сообществ в очереди %u и в стру
                        ктуре сообществ %u\n", com_u, com_v);
460                 break;
461                 case -4:
462                 com_u = lnh_inst.gpc[group][core]->mq_receive();
463                 com_v = lnh_inst.gpc[group][core]->mq_receive();
464                 delta_mod =
                        lnh_inst.gpc[group][core]->mq_receive();
465                 modularity =
                        lnh_inst.gpc[group][core]->mq_receive();
466                 v_count = lnh_inst.gpc[group][core]->mq_receive();
467                 com_r = lnh_inst.gpc[group][core]->mq_receive();
468                 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);
469                 break;
470                 default: break;
471             }
472             handler_state =
                    lnh_inst.gpc[group][core]->mq_receive();
473         }
474         #endif
475         #ifdef FORCED_LAYOUT
476         printf("III этап: Размещение сообществ силовым алгоритмом
                \n");
477         handler_state = lnh_inst.gpc[group][core]->mq_receive();
478         while (handler_state != 0) {
479             int u = lnh_inst.gpc[group][core]->mq_receive();
480             int x = lnh_inst.gpc[group][core]->mq_receive();
481             int y = lnh_inst.gpc[group][core]->mq_receive();
482             int displacement =
                    lnh_inst.gpc[group][core]->mq_receive();
483             printf("Размещение сообщества %u в области (%d,%d),
                    disp=%u\n", u, x, y, displacement);
484             handler_state =
                    lnh_inst.gpc[group][core]->mq_receive();
485         }
486         #endif
487         #ifdef BOX_LAYOUT
488         printf("IV этап: выделение прямоугольных областей\n");
```

```
489         handler_state = lnh_inst.gpc[group][core]->mq_receive();
490         while (handler_state != 0) {
491             com_u = lnh_inst.gpc[group][core]->mq_receive();
492             unsigned int v_count =
                    lnh_inst.gpc[group][core]->mq_receive();
493             short unsigned int x0 =
                    lnh_inst.gpc[group][core]->mq_receive();
494             short unsigned int y0 =
                    lnh_inst.gpc[group][core]->mq_receive();
495             short unsigned int x1 =
                    lnh_inst.gpc[group][core]->mq_receive();
496             short unsigned int y1 =
                    lnh_inst.gpc[group][core]->mq_receive();
497             short unsigned int is_leaf =
                    lnh_inst.gpc[group][core]->mq_receive();
498             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);
499             handler_state =
                    lnh_inst.gpc[group][core]->mq_receive();
500         }
501         #endif
502         #ifdef FORCED_LAYOUT
503         printf("IV этап: масштабирование в границы области\n");
504         handler_state = lnh_inst.gpc[group][core]->mq_receive();
505         while (handler_state != 0) {
506             switch (handler_state) {
507                 case -4: {
508                     unsigned int scale =
                            lnh_inst.gpc[group][core]->mq_receive();
509                     printf("Коэффициент масштабирования: %u /
                            1000\n", scale);
510                     break;}
511                 case -5: {
512                     unsigned int u =
                            lnh_inst.gpc[group][core]->mq_receive();
513                     int x =
                            lnh_inst.gpc[group][core]->mq_receive();
514                     int y =
                            lnh_inst.gpc[group][core]->mq_receive();
```

```
515                     unsigned int distance =
                            lnh_inst.gpc[group][core]->mq_receive();
516                     printf("Размещение сообщества %u в область
                            (%d,%d), диаметр (%u)\n", u, x, y,
                            distance);
517                     break;}
518                 default: break;
519             }
520             handler_state =
                    lnh_inst.gpc[group][core]->mq_receive();
521         }
522         #endif
523         #ifdef BOX_LAYOUT
524         printf("V этап: определение координат вершин\n");
525         handler_state = lnh_inst.gpc[group][core]->mq_receive();
526         while (handler_state != 0) {
527             switch (handler_state) {
528                 case -6:
529                 com_u = lnh_inst.gpc[group][core]->mq_receive();
530                 v_count = lnh_inst.gpc[group][core]->mq_receive();
531                 first_vertex =
                        lnh_inst.gpc[group][core]->mq_receive();
532                 last_vertex =
                        lnh_inst.gpc[group][core]->mq_receive();
533                 printf("Сообщество %u (вершины %u - %u), всего вер
                        шин (%u)\n", com_u, first_vertex, last_vertex,
                        v_count);
534                 break;
535                 case -7:
536                 com_u = lnh_inst.gpc[group][core]->mq_receive();
537                 u = lnh_inst.gpc[group][core]->mq_receive();
538                 x = lnh_inst.gpc[group][core]->mq_receive();
539                 y = lnh_inst.gpc[group][core]->mq_receive();
540                 color = lnh_inst.gpc[group][core]->mq_receive();
541                 size = lnh_inst.gpc[group][core]->mq_receive();
542                 btwc = lnh_inst.gpc[group][core]->mq_receive();
543                 printf("Сообщество %u, вершина %u, координаты:
                        (%u,%u)\n", com_u, u, x, y);
544                 break;
545                 default: break;
546             }
```

```
547             handler_state =
                    lnh_inst.gpc[group][core]->mq_receive();
548         }
549         #endif
550         #ifdef FORCED_LAYOUT
551         printf("V этап: раскладка сообществ в областях\n");
552         handler_state = lnh_inst.gpc[group][core]->mq_receive();
553         while (handler_state != 0) {
554             com_u = lnh_inst.gpc[group][core]->mq_receive();
555             int u = lnh_inst.gpc[group][core]->mq_receive();
556             int x = lnh_inst.gpc[group][core]->mq_receive();
557             int y = lnh_inst.gpc[group][core]->mq_receive();
558             //int displacement =
                    lnh_inst.gpc[group][core]->mq_receive();
559             //printf("Размещение сообщества %u: вершина %u помещае
                    тся в (%d,%d), disp=%d\n", com_u, u, x, y,
                    displacement);
560             printf("Размещение сообщества %u: вершина %u помещаетс
                    я в (%d,%d)\n", com_u, u, x, y);
561             handler_state =
                    lnh_inst.gpc[group][core]->mq_receive();
562         }
563         #endif
564         #endif
565     }
566
567     printf("Wait for connections\n");
568     while ((client_fd = accept(sock_fd, NULL, NULL)) != -1) {
569         printf("New connection\n");
570         __foreach_core(group, core) {
571             lnh_inst.gpc[group][core]->start_async(__event__(get_first_ve
572             if (lnh_inst.gpc[group][core]->mq_receive() != 0) {
573                 do {
574                     u = lnh_inst.gpc[group][core]->mq_receive();
575                     lnh_inst.gpc[group][core]->start_async(__event__(get_
576                     lnh_inst.gpc[group][core]->mq_send(u);
577                     unsigned int adj_c =
                            lnh_inst.gpc[group][core]->mq_receive();
578                     unsigned int pu =
                            lnh_inst.gpc[group][core]->mq_receive();
```

```
579                     unsigned int du =
                           lnh_inst.gpc[group][core]->mq_receive();
580                     unsigned int btwc =
                           lnh_inst.gpc[group][core]->mq_receive();
581                     unsigned int x =
                           lnh_inst.gpc[group][core]->mq_receive();
582                     unsigned int y =
                           lnh_inst.gpc[group][core]->mq_receive();
583                     unsigned int size =
                           lnh_inst.gpc[group][core]->mq_receive();
584                     unsigned int color =
                           lnh_inst.gpc[group][core]->mq_receive();
585                     write(client_fd, &u, sizeof(u));
586                     write(client_fd, &btwc, sizeof(btwc));
587                     write(client_fd, &adj_c, sizeof(adj_c));
588                     write(client_fd, &x, sizeof(x));
589                     write(client_fd, &y, sizeof(y));
590                     printf("(x,y,size)=%u,%u,%u\n", x, y, size);
591                     printf("Вершина␣%u␣-␣центральность␣%u␣-␣
                           (x,y,size)=%u,%u,%u␣связность␣%u\n", u,
                           btwc, x, y, size, adj_c);
592                     write(client_fd, &size, sizeof(size));
593                     write(client_fd, &color, sizeof(color));
594                     for (int i = 0; i < adj_c; i++) {
595                         unsigned int v =
                               lnh_inst.gpc[group][core]->mq_receive();
596                         unsigned int w =
                               lnh_inst.gpc[group][core]->mq_receive();
597                         write(client_fd, &v, sizeof(v));
598                         write(client_fd, &w, sizeof(w));
599                         //printf("Ребро с вершиной %u, вес
                               %u\n",v,w);
600                     }
601                     lnh_inst.gpc[group][core]->start_async(__event__(get_
602                     lnh_inst.gpc[group][core]->mq_send(u);
603                 } while (lnh_inst.gpc[group][core]->mq_receive()
                       != 0);
604
605             }
606         }
607
```

```
608             close(client_fd);
609         }
610
611         now = time(0);
612         strftime(buf, 100, "Stop at local date: %d.%m.%Y.; local
                time: %H.%M.%S", localtime(&now));
613         printf("DISC system speed test v1.1\n%s\n\n", buf);
614
615         //------------------------------------------------------------------
616         // Shutdown and cleanup
617         //------------------------------------------------------------------
618
619         if (err)
620         {
621             printf("ERROR: Test failed\n");
622             return EXIT_FAILURE;
623         }
624         else
625         {
626             printf("INFO: Test completed successfully.\n");
627             return EXIT_SUCCESS;
628         }
629
630         return 0;
631 }
```

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

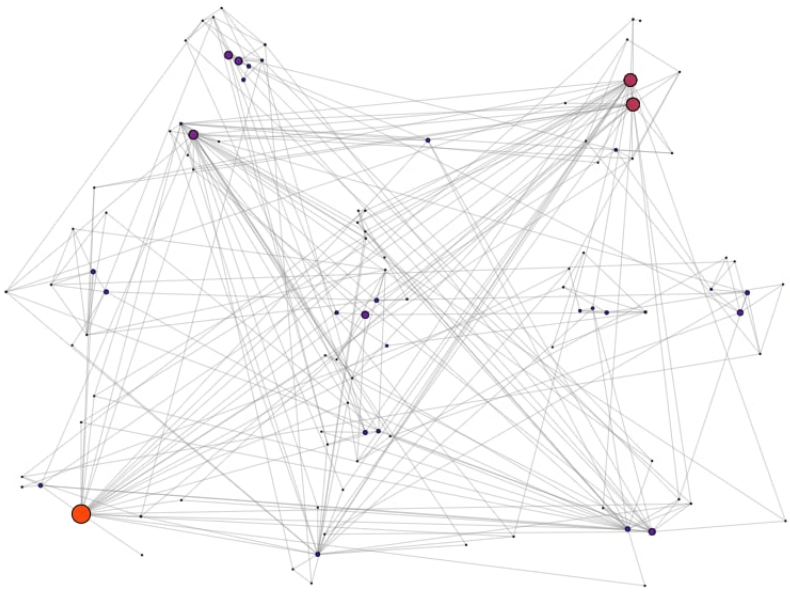

Рисунок 2.1 – Полученный граф по варианту 11