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»

НАПРАВЛЕНИЕ ПОДГОТОВКИ «09.03.04 Программная инженерия»

ОТЧЕТ

ПО ЧАСТИ ПРАКТИКУМА №1

Название: Изучение принципов работы микропроцессорного ядра RISC-V

Дисциплина: Архитектура ЭВМ

Студент	ИУ7-56Б		А.Д. Ковель
	Группа	Подпись, дата	И. О. Фамилия
Преподаватель			А. Ю. Попов
		Подпись, дата	И. О. Фамилия

Москва, 2022 г.

В данной работе будет выполнен 6 вариант.

Листинг 1 – Код чтения из файла

```
#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 <string>
# include <fstream>

# define SRC_FILE "data.tsv"


#define RAND_GRAPH
//#define GRID_GRAPH
#define BOX_LAYOUT
//#define FORCED_LAYOUT
#define DEBUG

#define handle_error(msg) \
do { perror(msg); exit(EXIT_FAILURE); } while (0)

int get_edge_count(std::string filename)
{
    std::ifstream fin(filename);
    printf("%d\n", fin.is_open());
    int a1, a2;
    int count = 0;
    while (fin >> a1 >> a2)
    ++count;
    fin.close();
    return count;
}

static void usage()
{
    std::cout << "usage: <xclbin> <sw_kernel>\n\n";
}

static void print_table(std::string test, float value, std::string units)
{
```

```
44      std::cout << std::left << std::setfill(' ') << std::setw(50) << test << std::r
45      std::cout << std::setfill('-') << std::setw(85) << "-" << std::endl;
46  }
47  const int port = 0x4747;
48  int server_socket_init() {
49      int sock_fd;
50      struct sockaddr_in srv_addr;
51      int client_fd;
52      sock_fd = socket(AF_INET, SOCK_STREAM, 0);
53      if (sock_fd == -1)
54      handle_error("socket");
55      memset(&srv_addr, 0, sizeof(srv_addr));
56      srv_addr.sin_family = AF_INET;
57      srv_addr.sin_port = htons(port);
58      srv_addr.sin_addr.s_addr = INADDR_ANY;
59      if (bind(sock_fd, (struct sockaddr *)&srv_addr, sizeof(srv_addr)) == -1)
60      handle_error("bind");
61      if (listen(sock_fd, 2) == -1)
62      handle_error("listen");
63      return sock_fd;
64  }
65
66  int main(int argc, char** argv)
67  {
68
69      unsigned int err = 0;
70      unsigned int cores_count = 0;
71      float LNH_CLOCKS_PER_SEC;
72      clock_t start, stop;
73
74      __foreach_core(group, core) cores_count++;
75
76      //Assign xclbin
77      if (argc < 3) {
78          usage();
79          throw std::runtime_error("FAILED_TEST\nNo xclbin specified");
80      }
81
82      //Open device #0
83      leonhardx64 lnh_inst = leonhardx64(0, argv[1]);
84      __foreach_core(group, core)
85      {
86          lnh_inst.load_sw_kernel(argv[2], group, core);
87      }
88
89      /*
90      *
91      * SW Kernel Version and Status
```

```
92      *
93      */
94      __foreach_core(group, core)
95      {
96          printf("Group #%d \tCore #%d\n", group, core);
97          lnh_inst.gpc[group][core]->start_sync(__event__(get_version));
98          printf("\tSoftware Kernel Version:\t0x%08x\n", lnh_inst.gpc[group][core]->
99          lnh_inst.gpc[group][core]->start_sync(__event__(get_lnh_status_high));
100         printf("\tLeonhard Status Register:\t0x%08x", lnh_inst.gpc[group][core]->m
101         lnh_inst.gpc[group][core]->start_sync(__event__(get_lnh_status_low));
102         printf("_%08x\n", lnh_inst.gpc[group][core]->mq_receive());
103     }
104
105
106     float interval;
107     char buf[100];
108     err = 0;
109
110     time_t now = time(0);
111     strftime(buf, 100, "Start at local date: %d.%m.%Y.; local time: %H.%M.%S", loc
112
113     printf("\nDISC system speed test v3.0\n%s\n\n", buf);
114     std::cout << std::left << std::setw(50) << "Test" << std::right << std::setw(2
115     std::cout << std::setfill('-') << std::setw(85) << "-" << std::endl;
116     print_table("Graph Processing Cores count (GPCC)", cores_count, "instances");
117
118
119
120
121     /*
122     *
123     * GPC frequency measurement for the first kernel
124     *
125     */
126     lnh_inst.gpc[0][LNH_CORES_LOW[0]]->start_async(__event__(frequency_measurement
127
128     // Measurement Body
129     lnh_inst.gpc[0][LNH_CORES_LOW[0]]->sync_with_gpc(); // Start measurement
130     sleep(1);
131     lnh_inst.gpc[0][LNH_CORES_LOW[0]]->sync_with_gpc(); // Start measurement
132     // End Body
133     lnh_inst.gpc[0][LNH_CORES_LOW[0]]->finish();
134     LNH_CLOCKS_PER_SEC = (float)lnh_inst.gpc[0][LNH_CORES_LOW[0]]->mq_receive();
135     print_table("Leonhard clock frequency (LNH_CF)", LNH_CLOCKS_PER_SEC / 1000000,
136
137
138
139     /*
```

```
140     *
141     * Generate grid as a graph
142     *
143     */
144
145     #ifdef GRID_GRAPH
146
147     unsigned int u;
148
149     __foreach_core(group, core)
150     {
151         lnh_inst.gpc[group][core]->start_async(__event__(delete_graph));
152     }
153
154
155     unsigned int* host2gpc_ext_buffer[LNH_GROUPS_COUNT][LNH_MAX_CORES_IN_GROUP];
156
157     __foreach_core(group, core)
158     {
159         host2gpc_ext_buffer[group][core] = (unsigned int*)lnh_inst.gpc[group][core
160         offs = 0;
161         //Top Left
162         EDGE(0, 1, 2);              //east
163         EDGE(0, GRAPH_SIZE_X, 2);   //south
164         EDGE(0, GRAPH_SIZE_X + 1, 3);   //south-east
165         //Top Right
166         EDGE(GRAPH_SIZE_X - 1, GRAPH_SIZE_X - 2, 2);        //west
167         EDGE(GRAPH_SIZE_X - 1, 2 * GRAPH_SIZE_X - 1, 2);    //south
168         EDGE(GRAPH_SIZE_X - 1, 2 * GRAPH_SIZE_X - 2, 3);    //south-west
169         //Bottom Left
170         EDGE(GRAPH_SIZE_X * (GRAPH_SIZE_Y - 1), GRAPH_SIZE_X * (GRAPH_SIZE_Y - 2)
171         EDGE(GRAPH_SIZE_X * (GRAPH_SIZE_Y - 1), GRAPH_SIZE_X * (GRAPH_SIZE_Y - 1)
172         EDGE(GRAPH_SIZE_X * (GRAPH_SIZE_Y - 1), GRAPH_SIZE_X * (GRAPH_SIZE_Y - 2)
173         //Bottom Right
174         EDGE(GRAPH_SIZE_X * GRAPH_SIZE_Y - 1, GRAPH_SIZE_X * (GRAPH_SIZE_Y - 1) -
175         EDGE(GRAPH_SIZE_X * GRAPH_SIZE_Y - 1, GRAPH_SIZE_X * GRAPH_SIZE_Y - 2, 2);
176         EDGE(GRAPH_SIZE_X * GRAPH_SIZE_Y - 1, GRAPH_SIZE_X * (GRAPH_SIZE_Y - 1) -
177         //Left and Right sides
178         for (int y = 1; y < GRAPH_SIZE_Y - 1; y++) {
179             //Left
180             EDGE(GRAPH_SIZE_X * y, GRAPH_SIZE_X * (y - 1), 2);      //north
181             EDGE(GRAPH_SIZE_X * y, GRAPH_SIZE_X * (y + 1), 2);      //south
182             EDGE(GRAPH_SIZE_X * y, GRAPH_SIZE_X * y + 1, 2);        //east
183             EDGE(GRAPH_SIZE_X * y, GRAPH_SIZE_X * (y - 1) + 1, 3);  //north-east
184             EDGE(GRAPH_SIZE_X * y, GRAPH_SIZE_X * (y + 1) + 1, 3);  //south-east
185             //Right
186             EDGE(GRAPH_SIZE_X * (y + 1) - 1, GRAPH_SIZE_X * y - 1, 2);      //nort
187             EDGE(GRAPH_SIZE_X * (y + 1) - 1, GRAPH_SIZE_X * (y + 2) - 1, 2);    /
```

```
188             EDGE(GRAPH_SIZE_X * (y + 1) - 1, GRAPH_SIZE_X * (y + 1) - 2, 2);     /
189             EDGE(GRAPH_SIZE_X * (y + 1) - 1, GRAPH_SIZE_X * y - 2, 3);       //nort
190             EDGE(GRAPH_SIZE_X * (y + 1) - 1, GRAPH_SIZE_X * (y + 2) - 2, 3);     /
191         }
192
193         for (int x = 1; x < GRAPH_SIZE_X - 1; x++) {
194             //Top
195             EDGE(x, x - 1, 2);  //east
196             EDGE(x, x + 1, 2);  //west
197             EDGE(x, GRAPH_SIZE_X + x, 2);       //south
198             EDGE(x, GRAPH_SIZE_X + x - 1, 3);   //south-east
199             EDGE(x, GRAPH_SIZE_X + x + 1, 3);   //south-west
200             //Bottom
201             EDGE(GRAPH_SIZE_X * (GRAPH_SIZE_Y - 1) + x, GRAPH_SIZE_X * (GRAPH_SIZE
202             EDGE(GRAPH_SIZE_X * (GRAPH_SIZE_Y - 1) + x, GRAPH_SIZE_X * (GRAPH_SIZE
203             EDGE(GRAPH_SIZE_X * (GRAPH_SIZE_Y - 1) + x, GRAPH_SIZE_X * (GRAPH_SIZE
204             EDGE(GRAPH_SIZE_X * (GRAPH_SIZE_Y - 1) + x, GRAPH_SIZE_X * (GRAPH_SIZE
205             EDGE(GRAPH_SIZE_X * (GRAPH_SIZE_Y - 1) + x, GRAPH_SIZE_X * (GRAPH_SIZE
206         }
207
208         for (int y = 1; y < GRAPH_SIZE_Y - 1; y++)
209         for (int x = 1; x < GRAPH_SIZE_X - 1; x++) {
210             EDGE(x + GRAPH_SIZE_X * y, x + GRAPH_SIZE_X * (y - 1), 2);  //north
211             EDGE(x + GRAPH_SIZE_X * y, x + GRAPH_SIZE_X * (y + 1), 2);  //south
212             EDGE(x + GRAPH_SIZE_X * y, x + GRAPH_SIZE_X * y - 1, 2);        //east
213             EDGE(x + GRAPH_SIZE_X * y, x + GRAPH_SIZE_X * y + 1, 2);        //west
214             EDGE(x + GRAPH_SIZE_X * y, x + GRAPH_SIZE_X * (y - 1) - 1, 3);  //nort
215             EDGE(x + GRAPH_SIZE_X * y, x + GRAPH_SIZE_X * (y + 1) - 1, 3);  //sout
216             EDGE(x + GRAPH_SIZE_X * y, x + GRAPH_SIZE_X * (y - 1) + 1, 3);  //nort
217             EDGE(x + GRAPH_SIZE_X * y, x + GRAPH_SIZE_X * (y + 1) + 1, 3);  //sout
218         }
219         lnh_inst.gpc[group][core]->external_memory_sync_to_device(0, BIFFER_SIZE);
220     }
221     __foreach_core(group, core)
222     {
223         lnh_inst.gpc[group][core]->start_async(__event__(insert_edges));
224     }
225     __foreach_core(group, core) {
226         long long tmp = lnh_inst.gpc[group][core]->external_memory_address();
227         lnh_inst.gpc[group][core]->mq_send((unsigned int)tmp);
228     }
229     __foreach_core(group, core) {
230         lnh_inst.gpc[group][core]->mq_send(BIFFER_SIZE);
231     }
232
233
234     __foreach_core(group, core)
235     {
```

```
236         lnh_inst.gpc[group][core]->finish();
237     }
238     printf("Data graph created!\n");
239 
240 
241     #endif
242 
243 
244     /*
245     *
246     * Generate random graph
247     *
248     */
249 
250     #ifdef RAND_GRAPH
251 
252     // __foreach_core(group, core)
253     // {
254         //  lnh_inst.gpc[group][core]->start_async(__event__(delete_graph));
255         // }
256 
257 
258     // unsigned int* host2gpc_ext_buffer[LNH_GROUPS_COUNT][LNH_MAX_CORES_IN_GROUP]
259     // unsigned int vertex_count = GRAPH_SIZE_X * GRAPH_SIZE_Y;
260     // unsigned int edge_count = vertex_count;
261     // unsigned int subgraph_count = 10;
262     // unsigned int messages_count = 0;
263     // unsigned int u, v, w;
264 
265     // __foreach_core(group, core)
266     // {
267 
268         //  host2gpc_ext_buffer[group][core] = (unsigned int*)lnh_inst.gpc[group]
269         //  offs = 0;
270 
271 
272         //  for (int edge = 0; edge < edge_count; edge++) {
273             //      do
274             //          v = rand() % vertex_count;
275             //      while (v == u);
276             //      w = 1;
277             //      EDGE(u, v, w);
278             //      EDGE(v, u, w);
279             //      messages_count += 2;
280             //      u = v;
281             //  }
282 unsigned int subgraph_vcount = rand() % 20;
283             //      unsigned int subgraph_vstart = rand() % (vertex_count - subgra
```

```
284             //      for (int vi = subgraph_vstart; vi < subgraph_vstart + subgraph
285                 //          for (int vj = vi + 1; vj < subgraph_vstart + subgraph_
286                     //              w = 1;
287                     //              EDGE(vi, vj, w);
288                     //              EDGE(vj, vi, w);
289                     //              messages_count += 2;
290                     //          }
291                 //      }
292             //  }
293
294
295
296         //  lnh_inst.gpc[group][core]->external_memory_sync_to_device(0, 3 * sizeo
297         // }
298     // __foreach_core(group, core)
299     // {
300         //  lnh_inst.gpc[group][core]->start_async(__event__(insert_edges));
301         // }
302     // __foreach_core(group, core) {
303         //  long long tmp = lnh_inst.gpc[group][core]->external_memory_address();
304         //  lnh_inst.gpc[group][core]->mq_send((unsigned int)tmp);
305         // }
306     // __foreach_core(group, core) {
307         //  lnh_inst.gpc[group][core]->mq_send(3 * sizeof(int)*messages_count);
308         // }
309
310
311     // __foreach_core(group, core)
312     // {
313         //  lnh_inst.gpc[group][core]->finish();
314         // }
315     // printf("Data graph created!\n");
316
317
318     __foreach_core(group, core)
319     {
320         lnh_inst.gpc[group][core]->start_async(__event__(delete_graph));
321     }
322
323
324     unsigned int* host2gpc_ext_buffer[LNH_GROUPS_COUNT][LNH_MAX_CORES_IN_GROUP];
325     unsigned int edge_count = get_edge_count(SRC_FILE);
326     unsigned int messages_count = 0;
327     unsigned int u, v, w;
328
329     __foreach_core(group, core)
330     {
331         host2gpc_ext_buffer[group][core] = (unsigned int*)lnh_inst.gpc[group][core
```

```
332
333         std::ifstream fin(SRC_FILE);
334         for (int edge = 0; edge < edge_count; ++edge)
335         {
336             fin >> u >> v;
337             w = 1;
338             EDGE(u, v, w);
339             EDGE(v, u, w);
340             messages_count += 2;
341         }
342         fin.close();
343
344
345         lnh_inst.gpc[group][core]->external_memory_sync_to_device(0, 3 * sizeof(u
346     }
347     __foreach_core(group, core)
348     {
349         lnh_inst.gpc[group][core]->start_async(__event__(insert_edges));
350     }
351     __foreach_core(group, core) {
352         long long tmp = lnh_inst.gpc[group][core]->external_memory_address();
353         lnh_inst.gpc[group][core]->mq_send((unsigned int)tmp);
354     }
355     __foreach_core(group, core) {
356         lnh_inst.gpc[group][core]->mq_send(3 * sizeof(int)*messages_count);
357     }
358
359
360     __foreach_core(group, core)
361     {
362         lnh_inst.gpc[group][core]->finish();
363     }
364     printf("Data graph created!\n");
365
366
367
368
369
370     #endif
371
372
373     /*
374     *
375     * Run BTWC
376     *
377     */
378
379     start = clock();
```

```
380
381     __foreach_core(group, core)
382     {
383         lnh_inst.gpc[group][core]->start_async(__event__(btwc));
384     }
385
386
387     __foreach_core(group, core)
388     {
389         lnh_inst.gpc[group][core]->finish();
390     }
391
392     stop = clock();
393
394     printf("\nBTWC is done for %.2f seconds\n", (float(stop - start) / CLOCKS_PER_
395
396
397
398     /*
399     *
400     * Show btwc
401     *
402     */
403     int sock_fd = server_socket_init();
404     int client_fd;
405
406     printf("Create visualisation\n");
407     __foreach_core(group, core)
408     {
409         //lnh_inst.gpc[group][core]->start_async(__event__(create_visualization));
410         //lnh_inst.gpc[group][core]->start_async(__event__(create_centrality_visua
411         //lnh_inst.gpc[group][core]->start_async(__event__(create_centrality_spira
412         #ifdef BOX_LAYOUT
413         lnh_inst.gpc[group][core]->start_async(__event__(create_communities_forest
414         #endif
415         #ifdef FORCED_LAYOUT
416         lnh_inst.gpc[group][core]->start_async(__event__(create_communities_forced
417         #endif
418
419         #ifdef DEBUG
420         //DEBUG
421         unsigned int handler_state;
422         unsigned int com_u, com_v, com_k, com_r, v_count, delta_mod, modularity;
423         short unsigned int x, y, color, size, btwc, first_vertex, last_vertex;
424
425         handler_state = lnh_inst.gpc[group][core]->mq_receive();
426         while (handler_state != 0) {
427             com_u = lnh_inst.gpc[group][core]->mq_receive();
```

```
428             com_v = lnh_inst.gpc[group][core]->mq_receive();
429             printf("                                                    %u
430             printf("                                                %u\n", lnh_ins
431             handler_state = lnh_inst.gpc[group][core]->mq_receive();
432         }
433
434         printf("II          :                                       \n");
435         handler_state = lnh_inst.gpc[group][core]->mq_receive();
436         while (handler_state != 0) {
437             switch (handler_state) {
438                 case -1:
439                 com_u = lnh_inst.gpc[group][core]->mq_receive();
440                 com_v = lnh_inst.gpc[group][core]->mq_receive();
441                 delta_mod = lnh_inst.gpc[group][core]->mq_receive();
442                 modularity = lnh_inst.gpc[group][core]->mq_receive();
443                 printf("
444                 break;
445                 case -2:
446                 com_u = lnh_inst.gpc[group][core]->mq_receive();
447                 com_v = lnh_inst.gpc[group][core]->mq_receive();
448                 delta_mod = lnh_inst.gpc[group][core]->mq_receive();
449                 printf("\ t
450                 break;
451                 default: break;
452             }
453             handler_state = lnh_inst.gpc[group][core]->mq_receive();
454         }
455
456         printf("                                                    \n");
457         handler_state = lnh_inst.gpc[group][core]->mq_receive();
458         while (handler_state != 0) {
459             int community = lnh_inst.gpc[group][core]->mq_receive();
460             int first_vertex = lnh_inst.gpc[group][core]->mq_receive();
461             int last_vertex = lnh_inst.gpc[group][core]->mq_receive();
462             printf("                    %u.                                 %u
463             handler_state = lnh_inst.gpc[group][core]->mq_receive();
464             while (handler_state != 0) {
465                 int vertex = lnh_inst.gpc[group][core]->mq_receive();
466                 printf("%u->", vertex);
467                 handler_state = lnh_inst.gpc[group][core]->mq_receive();
468             }
469             printf("\n");
470             handler_state = lnh_inst.gpc[group][core]->mq_receive();
471         }
472
473         #ifdef BOX_LAYOUT
474         printf("III         :
475         handler_state = lnh_inst.gpc[group][core]->mq_receive();
```

```
476         while (handler_state != 0) {
477             switch (handler_state) {
478                 case -3:
479                 com_u = lnh_inst.gpc[group][core]->mq_receive();
480                 com_v = lnh_inst.gpc[group][core]->mq_receive();
481                 printf("
482                 break;
483                 case -4:
484                 com_u = lnh_inst.gpc[group][core]->mq_receive();
485                 com_v = lnh_inst.gpc[group][core]->mq_receive();
486                 delta_mod = lnh_inst.gpc[group][core]->mq_receive();
487                 modularity = lnh_inst.gpc[group][core]->mq_receive();
488                 v_count = lnh_inst.gpc[group][core]->mq_receive();
489                 com_r = lnh_inst.gpc[group][core]->mq_receive();
490                 printf("
                        %u,                                    %u: \tdM = %d\tM = %d\
491                 break;
492                 default: break;
493             }
494             handler_state = lnh_inst.gpc[group][core]->mq_receive();
495         }
496         #endif
497         #ifdef FORCED_LAYOUT
498         printf("III         :
499         handler_state = lnh_inst.gpc[group][core]->mq_receive();
500         while (handler_state != 0) {
501             int u = lnh_inst.gpc[group][core]->mq_receive();
502             int x = lnh_inst.gpc[group][core]->mq_receive();
503             int y = lnh_inst.gpc[group][core]->mq_receive();
504             int displacement = lnh_inst.gpc[group][core]->mq_receive();
505             printf("                                                %u
506             handler_state = lnh_inst.gpc[group][core]->mq_receive();
507         }
508         #endif
509         #ifdef BOX_LAYOUT
510         printf("IV          :
511         handler_state = lnh_inst.gpc[group][core]->mq_receive();
512         while (handler_state != 0) {
513             com_u = lnh_inst.gpc[group][core]->mq_receive();
514             unsigned int v_count = lnh_inst.gpc[group][core]->mq_receive();
515             short unsigned int x0 = lnh_inst.gpc[group][core]->mq_receive();
516             short unsigned int y0 = lnh_inst.gpc[group][core]->mq_receive();
517             short unsigned int x1 = lnh_inst.gpc[group][core]->mq_receive();
518             short unsigned int y1 = lnh_inst.gpc[group][core]->mq_receive();
519             short unsigned int is_leaf = lnh_inst.gpc[group][core]->mq_receive();
520             printf("
521             handler_state = lnh_inst.gpc[group][core]->mq_receive();
522         }
```

```
523         #endif
524         #ifdef FORCED_LAYOUT
525         printf("IV          :
526         handler_state = lnh_inst.gpc[group][core]->mq_receive();
527         while (handler_state != 0) {
528             switch (handler_state) {
529                 case -4: {
530                     unsigned int scale = lnh_inst.gpc[group][core]->mq_receive();
531                     printf("
532                     break;}
533                 case -5: {
534                     unsigned int u = lnh_inst.gpc[group][core]->mq_receive();
535                     int x = lnh_inst.gpc[group][core]->mq_receive();
536                     int y = lnh_inst.gpc[group][core]->mq_receive();
537                     unsigned int distance = lnh_inst.gpc[group][core]->mq_receive
538                     printf("                                                %u
539                     break;}
540                 default: break;
541             }
542             handler_state = lnh_inst.gpc[group][core]->mq_receive();
543         }
544         #endif
545         #ifdef BOX_LAYOUT
546         printf("V          :
547         handler_state = lnh_inst.gpc[group][core]->mq_receive();
548         while (handler_state != 0) {
549             switch (handler_state) {
550                 case -6:
551                 com_u = lnh_inst.gpc[group][core]->mq_receive();
552                 v_count = lnh_inst.gpc[group][core]->mq_receive();
553                 first_vertex = lnh_inst.gpc[group][core]->mq_receive();
554                 last_vertex = lnh_inst.gpc[group][core]->mq_receive();
555                 printf("                    %u (                %u - %u),
556                 break;
557                 case -7:
558                 com_u = lnh_inst.gpc[group][core]->mq_receive();
559                 u = lnh_inst.gpc[group][core]->mq_receive();
560                 x = lnh_inst.gpc[group][core]->mq_receive();
561                 y = lnh_inst.gpc[group][core]->mq_receive();
562                 color = lnh_inst.gpc[group][core]->mq_receive();
563                 size = lnh_inst.gpc[group][core]->mq_receive();
564                 btwc = lnh_inst.gpc[group][core]->mq_receive();
565                 printf("                    %u,                 %u,
566                 break;
567                 default: break;
568             }
569             handler_state = lnh_inst.gpc[group][core]->mq_receive();
570         }
```

```
571         #endif
572         #ifdef FORCED_LAYOUT
573         printf("V         :
574         handler_state = lnh_inst.gpc[group][core]->mq_receive();
575         while (handler_state != 0) {
576             com_u = lnh_inst.gpc[group][core]->mq_receive();
577             int u = lnh_inst.gpc[group][core]->mq_receive();
578             int x = lnh_inst.gpc[group][core]->mq_receive();
579             int y = lnh_inst.gpc[group][core]->mq_receive();
580             //int displacement = lnh_inst.gpc[group][core]->mq_receive();
581             //printf("                                                %u:
582             printf("                                               %u:                 %
583             handler_state = lnh_inst.gpc[group][core]->mq_receive();
584         }
585         #endif
586         #endif
587     }
588
589     printf("Wait for connections\n");
590     while ((client_fd = accept(sock_fd, NULL, NULL)) != -1) {
591         printf("New connection\n");
592         __foreach_core(group, core) {
593             lnh_inst.gpc[group][core]->start_async(__event__(get_first_vertex));
594             if (lnh_inst.gpc[group][core]->mq_receive() != 0) {
595                 do {
596                     u = lnh_inst.gpc[group][core]->mq_receive();
597                     lnh_inst.gpc[group][core]->start_async(__event__(get_vertex_da
598                     lnh_inst.gpc[group][core]->mq_send(u);
599                     unsigned int adj_c = lnh_inst.gpc[group][core]->mq_receive();
600                     unsigned int pu = lnh_inst.gpc[group][core]->mq_receive();
601                     unsigned int du = lnh_inst.gpc[group][core]->mq_receive();
602                     unsigned int btwc = lnh_inst.gpc[group][core]->mq_receive();
603                     unsigned int x = lnh_inst.gpc[group][core]->mq_receive();
604                     unsigned int y = lnh_inst.gpc[group][core]->mq_receive();
605                     unsigned int size = lnh_inst.gpc[group][core]->mq_receive();
606                     unsigned int color = lnh_inst.gpc[group][core]->mq_receive();
607                     write(client_fd, &u, sizeof(u));
608                     write(client_fd, &btwc, sizeof(btwc));
609                     write(client_fd, &adj_c, sizeof(adj_c));
610                     write(client_fd, &x, sizeof(x));
611                     write(client_fd, &y, sizeof(y));
612                     printf("(x,y,size)=%u,%u,%u\n", x, y, size);
613                     printf("                %u -                                %u - (
614                     write(client_fd, &size, sizeof(size));
615                     write(client_fd, &color, sizeof(color));
616                     for (int i = 0; i < adj_c; i++) {
617                         unsigned int v = lnh_inst.gpc[group][core]->mq_receive();
618                         unsigned int w = lnh_inst.gpc[group][core]->mq_receive();
```

```
619                         write(client_fd, &v, sizeof(v));
620                         write(client_fd, &w, sizeof(w));
621                         //printf("                                    %u,         %u\n"
622                     }
623                     lnh_inst.gpc[group][core]->start_async(__event__(get_next_vert
624                     lnh_inst.gpc[group][core]->mq_send(u);
625                 } while (lnh_inst.gpc[group][core]->mq_receive() != 0);
626 
627             }
628         }
629 
630         close(client_fd);
631     }
632 
633     now = time(0);
634     strftime(buf, 100, "Stop at local date: %d.%m.%Y.; local time: %H.%M.%S", loca
635     printf("DISC system speed test v1.1\n%s\n\n", buf);
636 
637     //-------------------------------------------------------------------------
638     // Shutdown and cleanup
639     //-------------------------------------------------------------------------
640 
641     if (err)
642     {
643         printf("ERROR: Test failed\n");
644         return EXIT_FAILURE;
645     }
646     else
647     {
648         printf("INFO: Test completed successfully.\n");
649         return EXIT_SUCCESS;
650     }
651 
652 
653 
654 
655 
656     return 0;
657 }
```

Изображение работы программы: