В листингах 3.8, 3.9 представлена параллельная реализация муравьиного алгоритма.

Листинг 3.8 – Параллельный муравьиный алгоритм (основной поток)

```
pair<double, vector<int>> ant_alg_thr1(vector<vector<double>>&
    matrix, double alpha, double beta, double ro, int days, int thrs)
{
    mutex m, m1;
    int places = matrix.size();
    int ants = places;
    vector<thread> threads;
    vector<parameters> arg(thrs);
    int delta = days / thrs;
    double min_dist = numeric_limits<double>::max();
    vector<int> best_way;
    auto phero = calc_pheromones(places);
    auto vis = calc_visibility(matrix, places);
    for (int th = 0; th < thrs; ++th)
    {
        arg[th].places = places;
        arg[th].alpha = alpha;
        arg[th].matrix = matrix;
        arg[th].q = calc_q(arg[th].matrix, places);
        arg[th].min_dist = numeric_limits<double>::max();
        arg[th].pheromones = phero;
        arg[th].visibility = vis;
        arg[th].ro = ro;
        arg[th].mute1 = &m1;
        arg[th].mute = &m;
        arg[th].min_dist = min_dist;
        arg[th].best_way = best_way;
        if (th < thrs - 1)
            arg[th].to_do = delta;
        else
            arg[th].to_do = delta + days % thrs;
        threads.push_back(thread(thread_work1, ref(arg[th])));
    }
    for (int th = 0; th < thrs; ++th)
        threads[th].join();
    return pair<double, vector<int>>(min_dist, best_way);
}
```

Листинг 3.9 – Параллельный муравьиный алгоритм (вспомогательный поток)

```
1  void thread_work1(parameters& arg)
2  {
3      srand(time(nullptr));
4      int ants = arg.places;
5      double beta = 1 - arg.alpha;
6      for (int i = 0; i < arg.to_do; ++i)
7      {
8          vector<int> route(arg.places);
9          iota(begin(route), end(route), 0);
10         arg.visited = calc_visited_places(route, ants);
11         for (int ant = 0; ant < ants; ++ant)
12         {
13             while (arg.visited[ant].size() != ants)
14             {
15                 vector<double> pk = find_ways(arg.pheromones,
                       arg.visibility, arg.visited, arg.places, ant,
                       arg.alpha, beta);
16                 int chosen_place =
                       choose_next_place_by_possibility(pk);
17                 arg.visited[ant].push_back(chosen_place - 1);
18             }
19             double cur_length = calc_length(arg.matrix,
                   arg.visited[ant]);
20             if (cur_length < arg.min_dist)
21             {
22                 arg.mute->lock();
23                 if (cur_length < arg.min_dist)
24                 {
25                     arg.min_dist = cur_length;
26                     arg.best_way = arg.visited[ant];
27                 }
28                 arg.mute->unlock();
29             }
30         }
31         auto p = update_pheromones(arg.matrix, arg.places,
               arg.visited, arg.pheromones, arg.q, arg.ro);
32         arg.mute1->lock();
33         arg.pheromones = p;
34         arg.mute1->unlock();
35     }
36 }
```