МИНОБРНАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

«САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО»

Институт компьютерных наук и кибербезопасности
Высшая школа технологий искусственного интеллекта
Направление 02.03.01 Математика и компьютерные науки

Отчет по лабораторной работе

по дисциплине «Архитектура суперкомпьютеров»

Разработка приложения, моделирующего работу механизма передачи сообщения в коммуникационной сети суперкомпьютера.

Обучающийся: _____ Гладков И.А.

Руководитель: _____ Чуватов М.В.

«_____»_____20___г.

Санкт-Петербург, 2024

Содержание

Введение

Отчет представляет собой описание выполненной лабораторной работы по дисциплине «Архитектура суперкомпьютеров». Лабораторная работа включает в себя реализацию приложения, моделирующего работу механизма передачи сообщения в коммуникационной сети суперкомпьютера. В приложении реализована функция отправки сообщения, которая вызывается в формате (АДРЕС_ИСТОЧНИКА ДЛИНА_СООБЩЕНИЯ АДРЕС_ПОЛУЧАТЕЛЯ) и которая включает в себя функцию вычисления маршрута передачи сообщения с целью минимизации времени, необходимого для передачи. ДЛИНА_СООБЩЕНИЯ – это число шагов, требуемых для его передачи по свободному каналу. Также реализована возможность продвигаться по шагам, и в начале каждого шага пользователь может указать от 0 до 10 заданий на пересылку сообщений с произвольными адресами источника и получателя. Важной особенностью является штраф за каждую пересылку через промежуточный узел или через коммутатор в 1 дополнительный шаг времени. В качестве использованной топологии была выбрана топология Dragonfly со следующими характеристиками:

- Число групп: 11
- Коммутаторов в группе: 5
- Узлов на коммутатор: 3
- Пропускная способность внутри группы: 4
- Пропускная способность между группами: 2

Работа была выполнена в среде Visual Studio 2022 на языке программирования C++.

1 Математическое описание

1.1 Коммуникационные сети суперкомпьютеров

Коммуникационные сети суперкомпьютеров — это критически важная часть архитектуры, которая обеспечивает связь между отдельными узлами (процессорами, серверами, вычислительными блоками) для выполнения параллельных вычислений. Эти сети должны быть чрезвычайно быстрыми и эффективными, чтобы минимизировать задержки и обеспечить максимальную пропускную способность для обмена данными.

Основные компоненты коммуникационной сети:

1. **Узлы** — это вычислительные элементы (процессоры, ядра или серверы), которые обмениваются данными. Каждый узел имеет сетевые адаптеры, через которые они подключены к сети.

2. **Коммутаторы** — это устройства, которые управляют потоком данных между узлами, перенаправляя данные от одного узла к другому по оптимальному пути.

3. **Маршрутизация** — процесс определения путей передачи данных между узлами через коммутаторы. Эффективные алгоритмы маршрутизации играют ключевую роль в минимизации задержек.

4. **Сетевые интерфейсы** — это интерфейсы, через которые узлы подключаются к сети (например, InfiniBand, Ethernet, или специализированные сети).

Ключевые характеристики коммуникационных сетей:

- **Пропускная способность** (Bandwidth) — максимальное количество данных, которое может передаваться по сети за единицу времени.

- **Задержка** (Latency) — время, которое требуется для передачи данных от одного узла к другому.

- **Прямое соединение** (Direct Connection) — возможность узлов обмениваться данными напрямую без участия коммутаторов.

- **Трафик** (Traffic Pattern) — типы передач данных, например, all-to-all (все ко всем), one-to-all (один ко всем), one-to-one (один к одному).

- **Отказоустойчивость** (Fault Tolerance) — сети суперкомпьютеров проектируются таким образом, чтобы справляться с отказами отдельных узлов или соединений без серьезного влияния на общую производительность.

1.2 Сетевые топологии

Сетевые топологии — это схема организации связи между узлами в вычислительной системе, такие как суперкомпьютеры, серверы или рабочие станции. Топология сети определяет, как данные передаются между узлами, насколько эффективно сеть масштабируется, и насколько быстро она обрабатывает запросы. Выбор топологии влияет на производительность, устойчивость к отказам и стоимость создания и поддержания сети.

Основные типы топологий сетей, используемых в сети суперкомпьютеров:

1. **Полносвязная топология** – каждый узел сети напрямую соединён со всеми другими узлами. Это минимизирует задержки при передаче данных, так как каждый узел может обмениваться данными без участия промежуточных узлов.

2. **Fat-tree** (Толстое дерево) – древовидная топология с высокой пропускной способностью на каждом уровне, благодаря расширенным (“жирным”) каналам между уровнями. У каждого уровня дерева есть более широкие каналы для соединения с более высоким уровнем. В этой топологии узлы на нижнем уровне связаны с узлами верхних уровней через несколько уровней коммутаторов.

3. **Тор** представляет собой двумерную или трёхмерную сетку узлов, в которой каждый узел соединён с ближайшими соседями, а крайние узлы соединяются друг с другом, создавая “кольцевую” структуру. В трёхмерной версии каждый узел связан с соседями по трём направлениям (например, x, y, z).

4. **Dragonfly** — это иерархическая сеть, где узлы объединены в группы, каждая из которых представляет собой почти полносвязную структуру. Между группами создаются ограниченные глобальные соединения, что снижает количество коммутаторов и кабелей, требуемых для межгрупповой связи.

5. **Гиперкуб** — это топология, в которой каждый узел представляет вершину многомерного куба. В двумерной версии (2D) каждый узел соединён с двумя соседями, а в трёхмерной версии (3D) узлы соединены с тремя соседями. С увеличением измерений количество узлов и соединений растёт экспоненциально.

6. **Butterfly** – данные передаются через несколько уровней узлов, где каждый уровень имеет узлы, соединённые с узлами следующего уровня по заранее определённой схеме, напоминающей "крылья бабочки". Это позволяет эффективно распределять данные между узлами через минимальное количество уровней.

1.3 Топология Dragonfly

Топология Dragonfly — это современная топология, разработанная для высокопроизводительных вычислительных систем, таких как суперкомпьютеры, с целью обеспечения

высокой производительности при минимальных затратах на межузловую коммуникацию. Dragonfly сочетает идеи из «Fat-tree», «Torus» и других топологий для создания эффективной, высокомасштабируемой сети.

Основные черты Dragonfly:

1. Иерархическая структура:
   - Группы: Сеть разбита на группы, внутри которых узлы соединены полносвязно или почти полносвязно. Группы могут состоять из подгрупп узлов, связанных между собой с высокой пропускной способностью.
   - Межгрупповые соединения: Узлы из разных групп соединены меньшим числом связей через специальные маршрутизаторы или коммутаторы, обеспечивая глобальные связи между группами.
2. Минимизация числа соединений: Каждая группа связана только с небольшим количеством других групп, что снижает общие требования к числу межгрупповых соединений и позволяет строить сети с тысячами или даже миллионами узлов.
3. Эффективное распределение нагрузки: Топология Dragonfly использует адаптивные алгоритмы маршрутизации, чтобы избежать перегрузок в межгрупповых каналах и сбалансировать нагрузку между узлами и группами.

Преимущества топологии Dragonfly:

- Отличная масштабируемость: Топология поддерживает большие кластеры с тысячами узлов, при этом минимизируя количество межгрупповых соединений.
- Высокая пропускная способность: Полносвязные группы и прямые соединения между группами обеспечивают высокую скорость передачи данных.
- Снижение задержек: Адаптивная маршрутизация позволяет минимизировать задержки и сбалансировать нагрузку.

Структура топологии Dragonfly представлена на рисунке 1.

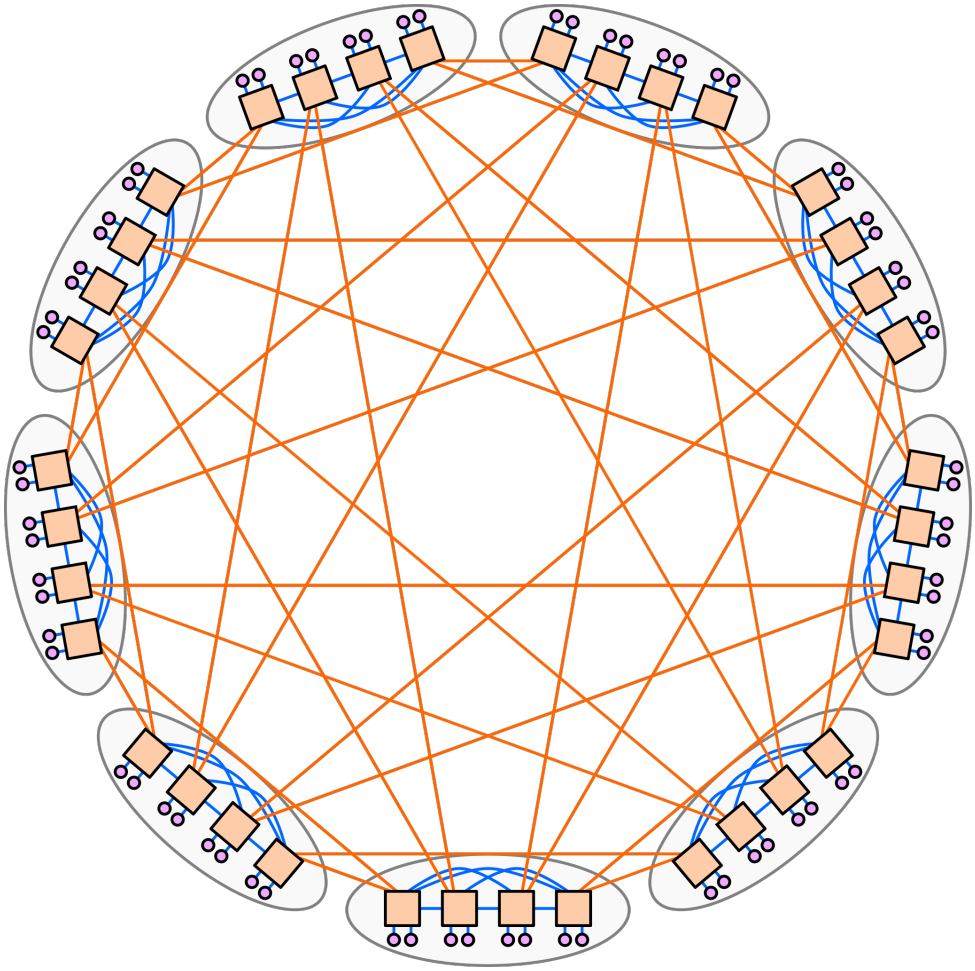

Рис. 1. Структура топологии Dragonfly

2 Особенности реализации

2.1 Класс Network

Класс `Network` предназначен для моделирования сетевой структуры заданной в варианте топологии, состоящей из нескольких групп, коммутаторов и узлов. Основная цель класса — управление сетевыми ресурсами, такими как пропускная способность внутри группы и между группами, а также обработка сообщений в сети.

Поля класса Network:

- `GROUPS` — количество групп в сети.
- `COMMUTATORS` — количество коммутаторов в каждой группе.
- `NODES` — количество узлов, подключенных к каждому коммутатору.
- `BANDWIDTH_IN_GROUP` — пропускная способность внутри группы.
- `BANDWIDTH_BETWEEN_GROUP` — пропускная способность между группами.
- `messages` — вектор, содержащий сообщения, обрабатываемые сетью.
- `status` — вектор строк, описывающих статус каждого сообщения.
- `path` — вектор векторов, содержащий пути передачи сообщений.
- `reserv` — вектор, описывающий количество зарезервированных ресурсов на каждом шаге передачи сообщения.
- `reminder` — вектор, содержащий информацию о том, сколько данных осталось передать между вершинами.
- `matrix_bandwidth` — матрица пропускных способностей сети.
- `matrix_load` — матрица загруженности сети.

2.2 Конструктор класса Network

Класс Network имеет параметризованный конструктор, который позволяет задать количество групп, коммутаторов, узлов, а также пропускную способность внутри группы и между группами. Помимо этого, он строит матрицу пропускных способностей и матрицу загруженности для заданной сети. В классе `Network`, для построения матрицы пропускных способностей и матрицы загруженности, нумерация элементов сети организована следующим образом:

- Узлы нумеруются первыми, начиная с 0 до $GN - 1$, где G — количество групп, а N — количество узлов в каждой группе.

- Коммутаторы идут следом и нумеруются с GN до $GN + GS - 1$, где S — количество коммутаторов в каждой группе.

Пример нумерации узлов и коммутаторов в таблице пропускных способностей представлен на рисунке 2.

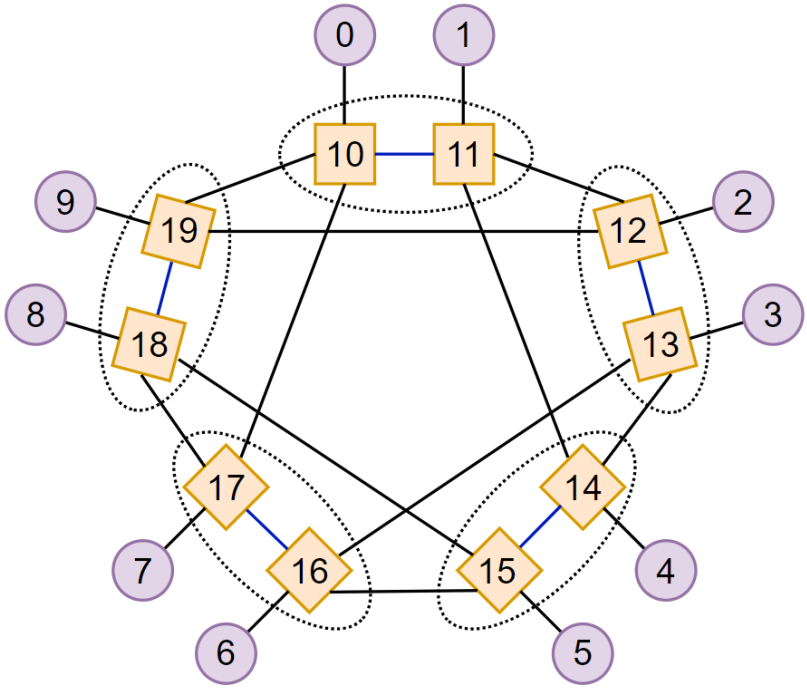

Рис. 2. Пример нумерации узлов и коммутаторов

Реализация конструктора представлена в листинге 1.

Листинг 1. Конструктор класса Network

```
1  Network::Network(int GROUPS, int COMMUTATORS, int NODES, int
       BANDWIDTH_IN_GROUP, int BANDWIDTH_BETWEEN_GROUP)  {
2      this->GROUPS = GROUPS;
3      this->COMMUTATORS = COMMUTATORS;
4      this->NODES = NODES;
5      this->BANDWIDTH_IN_GROUP = BANDWIDTH_IN_GROUP;
6      this->BANDWIDTH_BETWEEN_GROUP = BANDWIDTH_BETWEEN_GROUP;
7
8      count_commutators = GROUPS * COMMUTATORS;
9      count_nodes = count_commutators * NODES;
10     count_vertex = count_commutators + count_nodes;
11
12     vector<vector<int>> new_bandwidth(count_vertex, vector<int>(
       count_vertex, 0));
13     matrix_bandwidth = new_bandwidth;
```

```
14
15      for (int i = 0, ii=count_nodes, k = NODES; i < count_nodes;i++) {
16          k--;
17          matrix_bandwidth[i][ii] = BANDWIDTH_IN_GROUP; //пропускная способно
    сть в группе
18          matrix_bandwidth[ii][i] = BANDWIDTH_IN_GROUP;
19          if (k == 0) {
20              ii++;
21              k = NODES;
22          }
23      }
24
25      for (int i = 0, back=COMMUTATORS-1, group=GROUPS-1; i < COMMUTATORS; i
    ++, back--, group-=GROUPS/COMMUTATORS) {
26
27          for (int ii = count_nodes + i, group_right=group ; ii <
    count_vertex; ii+=COMMUTATORS, group_right++) {
28              if (group_right >= GROUPS) {
29                  group_right -= GROUPS;
30              }
31              for (int iii = 0, current_group = group_right; iii < GROUPS/
    COMMUTATORS; iii++) {
32                  current_group = (group_right - iii) < 0 ? GROUPS + (
    group_right - iii) : group_right - iii;
33                  int buf = count_nodes + current_group * COMMUTATORS + back;
34                  matrix_bandwidth[ii][count_nodes + current_group *
    COMMUTATORS + back] = BANDWIDTH_BETWEEN_GROUP;//пропускная способность м
    ежду группами
35                  matrix_bandwidth[count_nodes + current_group * COMMUTATORS
    + back][ii] = BANDWIDTH_BETWEEN_GROUP;
36              }
37          }
38      }
39
40      for (int i = 0; i < GROUPS; i++) {
41          for (int ii = count_nodes + i * COMMUTATORS; ii < count_nodes + (i
    + 1) * COMMUTATORS; ii++) {
42              for (int iii = ii; iii < count_nodes + (i + 1) * COMMUTATORS;
    iii++) {
43                  if (ii != iii) {
44                      matrix_bandwidth[ii][iii] = BANDWIDTH_IN_GROUP;//пропус
    кная способность в группе
45                      matrix_bandwidth[iii][ii] = BANDWIDTH_IN_GROUP;
46                  }
```

```
47                }
48            }
49        }
50        matrix_load = matrix_bandwidth;
51    }
```

2.3 Модифицированный алгоритм Дейкстры

Метод `Dijkstra_algorythm` реализует модифицированный алгоритм Дейкстры для поиска кратчайшего пути в сети с учетом загруженности соединений и штрафов за пересечение промежуточных узлов.

Метод принимает параметр `message` — вектор из трех элементов: `message[0]` — исходный узел, `message[1]` — длина сообщения, `message[2]` — конечный узел.

Реализация алгоритма представлена в листинге 2.

Листинг 2. Модифицированный алгоритм Дейкстры

```
1  vector<int> Network::Dijkstra_algorythm(vector<int> message) {
2      int index_from = message[0];
3      int index_to = message[2];
4      int cost = message[1];
5
6      int count = 0;
7      vector<int> vertex(count_vertex);
8      for (int i = 0; i < vertex.size(); i++) {
9          vertex[i] = INT_MAX;
10     }
11     vector<bool> vertex_check(count_vertex, false);
12     vector<vector<int>> route(count_vertex);
13     vertex[index_from] = 0;
14
15     int current;
16
17     while (CheckBoolVector(vertex_check) == false) {
18         current = Min_Index(vertex, vertex_check, cost) ;
19         if (current == -1)
20         break;
21         vertex_check[current] = true;
22
23         for (int i = 0; i < matrix_load[current].size(); i++) {
24             if (matrix_load[current][i] != 0) {
25                 int buf = cost / matrix_load[current][i];
26                 buf = cost % matrix_load[current][i]==0? buf :buf + 1;
27                 if (!(current == index_from || current == index_to)) {
```

```
28                      buf++; //прибавляем штраф
29                  }
30                  bool condition = vertex[current] + buf <= vertex[i];
31                  if (condition) {
32
33                      vertex_check[i] = false;
34                      route[i].clear();
35                      route[i] = route[current];
36                      route[i].push_back(current);
37                      vertex[i] = vertex[current] + buf;
38                  }
39              }
40          }
41      }
42      for (int i = 0; i < vertex.size(); i++) {
43          if (vertex[i] == INT_MAX || vertex[i] == INT_MAX + 1)
44          vertex[i] = 0;
45      }
46      for (int i = 0; i < route.size(); i++) {
47          if (route[i].size() != 0)
48          route[i].push_back(i);
49      }
50
51
52      vector<int> current_route;
53      for (int i = 0; i < route[index_to].size() - 1; i++) {
54          int index_1;
55          int index_2;
56          int cost_inner;
57          int broad;
58          if (i != route[index_to].size() - 2) {
59              index_1 = route[index_to][i];
60              index_2 = route[index_to][i + 1];
61              cost_inner = matrix_load[index_1][index_2];
62
63              broad = cost / cost_inner;
64              broad = cost % cost_inner == 0 ? broad : broad+1;
65              for (int ii = 0; ii < broad; ii++) {
66                  current_route.push_back(index_1);
67                  current_route.push_back(index_2);
68              }
69
70              //добавим штраф
71              current_route.push_back(index_2);
```

```
72                current_route.push_back(index_2);
73            }
74            else {
75                cost_inner = this->BANDWIDTH_IN_GROUP;
76                index_1 = route[index_to][i];
77                index_2 = route[index_to][i+1];
78
79                broad = cost / cost_inner;
80                broad = cost % cost_inner == 0 ? broad : broad+1;
81                for (int ii = 0; ii < broad; ii++) {
82                    current_route.push_back(index_1);
83                    current_route.push_back(index_2);
84                }
85            }
86        }
87
88        return current_route;
89    }
```

2.4 Добавление нового сообщения

Метод `adding_msg` отвечает за добавление сообщения в систему. Он запрашивает у пользователя ввод данных о сообщении в формате:

`АДРЕС_ИСТОЧНИКА ДЛИНА_СООБЩЕНИЯ АДРЕС_ПОЛУЧАТЕЛЯ`

где адреса находятся в пределах от 0 до `count_nodes - 1` включительно. Метод проверяет на корректность вводимых данных и вызывает метод AddMessage, который добавляет сообщение в вектор `messages`, вызывает алгоритм Дейкстры, чтобы получить путь для передачи сообщения, устанавливает статус для нового сообщения “Сообщение ожидает отправки”. Также метод определяет количество отправляемых данных, проверяя, возможно ли передать все данные по текущему пути, и обновляет матрицу нагрузки `matrix_load`.

Реализация методов `adding_msg` и представлена листинге 3.

Листинг 3. Добавление нового сообщения

```
1  void Network::adding_msg() {
2      string input;
3      int dep, length, arr;
4      char space1, space2;
5
6      cout << "\nВведите сообщение в формате АДРЕС_ИСТОЧНИКА ДЛИНА_СООБЩЕНИЯ
       АДРЕС_ПОЛУЧАТЕЛЯ, " << endl;
7      cout << "где адреса находятся в пределах от 0 до " << count_nodes - 1
       << " включительно" << endl;
```

```
8
9       do {
10          int flag_out1 = false;
11          int flag_out2 = false;
12          int flag_out3 = false;
13          getline(cin, input);
14          istringstream iss(input);
15
16          if (iss >> dep >> length >> arr) {
17
18              if (dep == arr) {
19                  cout << "Введите различные узел отправления и прибытия" <<
    endl;
20              }
21              else if (length <= 0) {
22                  cout << "Длина сообщения должна быть положительной!" <<
    endl;
23              }
24              else if (dep >= 0 && dep < count_nodes && arr >= 0 && arr <
    count_nodes && iss.eof()) {
25                  flag_out1 = true;
26              }
27              else {
28                  cout << "Некорретный ввод! Попробуйте снова" << endl;
29              }
30          }
31          else cout << "Некорретный ввод! Попробуйте снова" << endl;
32
33          if (messages.size() == 0) {
34              flag_out2 = true;
35          }
36          if(messages.size()>0){
37              for (int i = 0; i < messages.size(); i++) {
38                  if (dep == messages[i][0] &&
39                  length == messages[i][1] &&
40                  arr == messages[i][2]) {
41                      flag_out3 = true;
42                      break;
43                  }
44              }
45          }
46          if (flag_out3) {
47              cout << "Такое собщение уже существует! Попробуйте снова" <<
    endl;
```

```
48          }
49          if (flag_out1 && flag_out2)
50          break;
51          if (flag_out1 && (flag_out3 == false))
52          break;
53
54      } while (true);
55
56      this->AddMessage({ dep, length, arr });
57  }
58  void Network::AddMessage(vector<int> message) {
59      int index_from = message[0];
60      vector<int> messages_index; // индексы подходящих сообщений
61      for (int i = 0; i < messages.size(); i++) {
62          if (messages[i][0] == index_from && path[i].size() > 0 && path[i
        ][0] == index_from) {
63              messages_index.push_back(i);
64          }
65      }
66
67
68      for (int i = 0; i < messages_index.size(); i++) {
69          int index = messages_index[i];
70          int index_from = path[index][0];
71          int index_to = path[index][1];
72          if (messages_index.size() >= BANDWIDTH_IN_GROUP) {
73              if (i < BANDWIDTH_IN_GROUP) {
74                  matrix_load[index_from][index_to] += reserv[index];
75                  reserv[index] = 1;
76                  matrix_load[index_from][index_to] -= reserv[index];
77              }
78          }
79          else {
80              matrix_load[index_from][index_to] += reserv[index];
81              reserv[index] = BANDWIDTH_IN_GROUP / (messages_index.size() +
        1);
82              matrix_load[index_from][index_to] -= reserv[index];
83          }
84      }
85
86      bool can_make_path = false;
87      for (int i = 0; i < count_vertex; i++) {
88          if (matrix_load[message[0]][i]) {
89              can_make_path = true;
```

```
90              break;
91          }
92      }
93
94      vector <int> path;
95      this->messages.push_back(message);
96      if (can_make_path)
97      //if (Can_Make_Route(message))
98      if (Can_Dijkstra(message))
99      path = Dijkstra_algorythm(message);
100     else {
101         for (int i = 0; i < messages_index.size(); i++) {
102             int index = messages_index[i];
103             int index_from = this->path[index][0];
104             int index_to = this->path[index][1];
105             if (messages_index.size() >= BANDWIDTH_IN_GROUP) {
106                 if (i < BANDWIDTH_IN_GROUP) {
107                     matrix_load[index_from][index_to] += reserv[index];
108                     reserv[index] = 1;
109                     matrix_load[index_from][index_to] -= reserv[index];
110                 }
111             }
112             else {
113                 if (i == messages_index.size() - 1) {
114                     matrix_load[index_from][index_to] += reserv[index];
115                     reserv[index] = matrix_bandwidth[index_from][index_to]
    - matrix_bandwidth[index_from][index_to] / messages_index.size() * (
    messages_index.size() - 1);
116                     matrix_load[index_from][index_to] -= reserv[index];
117                 }
118                 else {
119                     matrix_load[index_from][index_to] += reserv[index];
120                     reserv[index] = BANDWIDTH_IN_GROUP / (messages_index.
    size());
121                     matrix_load[index_from][index_to] -= reserv[index];
122                 }
123             }
124         }
125     }
126     this->path.push_back(path);
127     this->status.push_back({ "Сообщение ожидает отправки" });
128     this->reminder.push_back(message[1]);
129
130     //if (Can_Make_Route(message)) {
```

```
131         if (Can_Dijkstra(message)) {
132             int can_send = message[1] < matrix_load[path[0]][path[1]] ?
    message[1] : matrix_load[path[0]][path[1]];
133             this->reserv.push_back(can_send);
134             matrix_load[path[0]][path[1]] -= can_send;
135         }
136         else
137         this->reserv.push_back(0);
138     }
```

2.5 Метод NextStep

Метод `NextStep` выполняет шаг передачи сообщений в сети. Он проходит по всем сообщениям и обрабатывает их в зависимости от текущего состояния. Основные действия метода:

- Итерирует по всем сообщениям в векторе `messages`.
- Если сообщение уже передается, проверяет, в каком состоянии находится передача:
  - Если сообщение находится в одном узле из-за штрафа, проверяет наличие свободного пути для передачи. Если путь доступен, вызывает алгоритм Дейкстры для нахождения нового маршрута и обновляет соответствующие параметры (остаток сообщения, матрицу нагрузки).
  - Если сообщение передается из одного узла в другой, проверяет доступную пропускную способность и обновляет матрицы нагрузки, резерв и остаток сообщения.
- После обработки каждого сообщения проверяет, нужно ли удалить текущий шаг маршрута. Если сообщение успешно доставлено, оно удаляется из всех векторов, а в противном случае — просто обновляется статус.
- В конце метода восстанавливает матрицу нагрузки до первоначального состояния с помощью `matrix_bandwidth`.

Реализация метода представлена листинге 4.

Листинг 4. Метод NextStep

```
void Network::NextStep() {
    for (int i = 0; i < messages.size(); i++) {
        int index_from;
        int index_to;
        int index_from_sec;
        int index_to_sec;
```

```
7           vector<int> messages_index; // индексы подходящих сообщений
8           if (path[i].size() > 0) {
9               index_from = path[i][0];
10              index_to = path[i][1];
11              if (path[i].size() > 2) {//для штрафных
12                  index_from_sec = path[i][2];
13                  index_to_sec = path[i][3];
14              }
15              for (int ii = 0; ii < messages.size(); ii++) {
16                  if (index_from != index_to && //не для штрафных
17                  path[ii].size() > 0 && path[ii][0] == index_from && path[ii
    ][1] == index_to) {
18                      messages_index.push_back(ii);
19                  }
20                  //для штрафных:
21                  if (path[ii].size() > 0 && path[i].size() > 2 &&
22                  index_from == index_to &&
23                  path[ii][0] == index_from_sec && path[ii][1] ==
    index_to_sec) {
24                      messages_index.push_back(ii);
25                  }
26                  if (path[ii].size() > 2 && path[i].size() > 2 &&
27                  index_from == index_to &&
28                  path[ii][0] == index_from && path[ii][1] == index_to &&
29                  path[ii][2] == index_from_sec && path[ii][3] ==
    index_to_sec) {
30                      messages_index.push_back(ii);
31                  }
32              }
33          }
34
35          if (path[i].size() > 0) {
36              if (index_from == index_to) {//для штрафных
37                  index_from = index_from_sec;
38                  index_to = index_to_sec;
39                  for (int iii = 0; iii < messages_index.size(); iii++) {
40                      reserv[messages_index[iii]] = 0;
41                  }
42                  if (matrix_load[index_from][index_to] != matrix_bandwidth[
    index_from][index_to])
43                  matrix_load[index_from][index_to] = matrix_bandwidth[
    index_from][index_to];
44              }
45
```

```
46              int summ=matrix_load[index_from][index_to];
47              for (int iii = 0; iii < messages_index.size(); iii++) {
48                  int index = messages_index[iii];
49                  summ += reserv[index];
50              }
51              if (summ != matrix_bandwidth[index_from][index_to]) {
52                  for (int iii = 0; iii < messages_index.size(); iii++) {
53                      int index = messages_index[iii];
54                      reserv[index] = 0;
55                  }
56              }
57
58              for (int ii = 0; ii < messages_index.size(); ii++) {
59                  int index = messages_index[ii];
60
61                  if (messages_index.size() >= matrix_bandwidth[index_from][
    index_to]) {
62                      if (ii < matrix_bandwidth[index_from][index_to]) {
63                          if (matrix_load[index_from][index_to] !=
    matrix_bandwidth[index_from][index_to])
64                          matrix_load[index_from][index_to] += reserv[index];
65                          reserv[index] = 1;
66                          matrix_load[index_from][index_to] -= reserv[index];
67                      }
68                  }
69                  else {
70                      if (ii == messages_index.size() - 1) {
71                          if (matrix_load[index_from][index_to] !=
    matrix_bandwidth[index_from][index_to])
72                          matrix_load[index_from][index_to] += reserv[index];
73                          int total = matrix_bandwidth[index_from][index_to]
    - matrix_bandwidth[index_from][index_to] / messages_index.size() * (
    messages_index.size() - 1);
74                          int can_send = reminder[index] < total ? reminder[
    index] : total;
75                          reserv[index] = can_send;
76                          matrix_load[index_from][index_to] -= reserv[index];
77                      }
78                      else {
79                          if (matrix_load[index_from][index_to] !=
    matrix_bandwidth[index_from][index_to])
80                          matrix_load[index_from][index_to] += reserv[index];
81                          int total = matrix_bandwidth[index_from][index_to]
    / (messages_index.size());
```

```
82                          int can_send = reminder[index] < total ? reminder[
    index] : total;
83                          reserv[index] = can_send;
84                          matrix_load[index_from][index_to] -= reserv[index];
85                      }
86                  }
87              }
88              for (int ii = 0; ii < messages_index.size(); ii++) {
89                  if (matrix_load[index_from][index_to] > 0) {
90                      for (int iii = 0; iii < messages_index.size(); iii++) {
91                          int index = messages_index[iii];
92                          if (reminder[index] > reserv[index]) {
93                              reserv[index]++;
94                              matrix_load[index_from][index_to]--;
95                          }
96                      }
97                  }
98              }
99
100             if (path[i][0] != path[i][1]) {
101                 int need_steps = reminder[i] / reserv[i];
102                 need_steps = reminder[i] % reserv[i] == 0 ? need_steps :
    need_steps + 1;
103
104                 int old_steps = 0;//считаем сколько было шагов
105                 for (int ii = 0; ii < path[i].size(); ii += 2) {
106                     if (index_from == path[i][ii] && index_to == path[i][ii
     + 1]) {
107                         old_steps++;
108                     }
109                     else {
110                         break;
111                     }
112                 }
113                 if (need_steps != old_steps) {
114                     path[i].erase(path[i].begin(), path[i].begin() +
    old_steps * 2);
115
116
117                     for (int iii = 0; iii < need_steps; iii++) {
118                         path[i].insert(path[i].begin(), { index_from,
    index_to });
119                     }
120
```

```
121                 }
122             }
123         }
124     }
125 
126     for (int i = 0; i < messages.size(); i++) {
127         if (status[i].size() == 0) {//рассматриваются ситуации когда сообще
    ние уже передается
128 
129             int local_from = path[i][0]; //точка в которой сейчас находится
     сообщение
130             int local_to = path[i][1]; //точка в которую передается шаг на
    данном шаге
131             int index_from = messages[i][0]; //отправитель
132             int index_to = messages[i][2]; //получатель
133             int message_cost = messages[i][1]; //длина сообщения
134 
135             if (local_from == path[i][1]) {// ситуация когда сообщение нахо
    дится в одном узле из за штрафа (то есть никуда не передается)
136                 bool check = false;
137                 if (reserv[i] == 0) {
138                     for (int ii = 0; ii < count_vertex; ii++) {
139                         if (matrix_load[ii][index_to] != 0) {
140                             check = true;
141                             break;
142                         }
143                     }
144                     if (check) {
145                         vector<int> new_path = Dijkstra_algorythm({
    local_from, message_cost, index_to });
146                         new_path.insert(new_path.begin(), { path[i][0],path
    [i][1] });
147                         path[i] = new_path;
148                         reminder[i] = message_cost;
149                     }
150                     else {
151                         string status = "Ожидает освобождения пути. Находит
    ся в " + to_string(local_from) + " коммутаторе";
152                         this->status[i].push_back({ status });
153                     }
154                 }
155             }
156             else { //ситуация когда передается сообщение из одного узда в д
    ругой
```

```
157                 reminder[i] -= reserv[i];
158             }
159 
160             //в конце шага нам нужно удалить данный шаг
161             if (path[i].size() > 2) {
162 
163                 if (path[i][2] == path[i][3]) {
164                     matrix_load[path[i][0]][path[i][1]] += reserv[i];
165                     reminder[i] = messages[i][1];
166                 }
167                 path[i].erase(path[i].begin());
168                 path[i].erase(path[i].begin());
169             }
170             else {
171                 messages.erase(messages.begin() + i);
172                 matrix_load[path[i][0]][path[i][1]] += reserv[i];
173                 matrix_load[path[i][1]][path[i][0]] += reserv[i];
174                 path.erase(path.begin() + i);
175                 status.erase(status.begin() + i);
176                 reminder.erase(reminder.begin() + i);
177                 reserv.erase(reserv.begin() + i);
178                 i--;
179             }
180         }
181         else if (path[i].size() > 0 && path[i][0] != messages[i][0]) {// ко
    гда в штрафе, а затем ожидание происходит
182 
183             int local_from = path[i][0]; //точка в которой сейчас находится
     сообщение
184             int local_to = path[i][1]; //точка в которую передается шаг на
    данном шаге
185             int index_from = messages[i][0]; //отправитель
186             int index_to = messages[i][2]; //получатель
187             int message_cost = messages[i][1]; //длина сообщения
188 
189             bool check = false;
190 
191             for (int ii = 0; ii < count_vertex; ii++) {
192                 if (matrix_load[ii][index_to] != 0) {
193                     check = true;
194                     break;
195                 }
196             }
197             if (check) {
```

```
198                 vector<int> new_path = Dijkstra_algorythm({ local_from,
    message_cost, index_to });
199                 new_path.insert(new_path.begin(), { path[i][0],path[i][1]
    });
200                 path[i] = new_path;
201                 reminder[i] = message_cost;
202
203                 int can_send = message_cost < matrix_load[path[i][2]][path[
    i][3]] ? message_cost : matrix_load[path[i][2]][path[i][3]];
204
205                 reserv[i] = can_send;
206                 matrix_load[path[i][2]][path[i][3]] -= can_send;
207                 matrix_load[path[i][3]][path[i][2]] -= can_send;
208             }
209         }
210     }
211
212     for (int i = 0; i < messages.size(); i++) {
213         if (status[i].size() != 0) {
214             if (path[i].size() > 0 && path[i][0] != messages[i][0]) { //ког
    да ожидает освобождения пути
215                 bool check = false;
216                 for (int ii = 0; ii < count_vertex; ii++) {
217                     if (matrix_load[ii][messages[i][2]] != 0) {
218                         check = true;
219                         break;
220                     }
221                 }
222                 if (check) {
223                     status[i].erase(status[i].begin());
224                 }
225             }
226             else {
227                 if (Can_Make_Route(messages[i])) {//это для начальных этапо
    в
228                     status[i].erase(status[i].begin());
229                     if (reserv[i] == 0) {
230                         path[i] = Dijkstra_algorythm(messages[i]);
231
232                         int index_from = messages[i][0];
233
234                         vector<int> messages_index; // индексы подходящих с
    ообщений
235                         for (int ii = 0; ii < messages.size(); ii++) {
```

```
236                             if (messages[ii][0] == index_from && path[ii].
    size() > 0 && path[ii][0] == index_from) {
237                                 messages_index.push_back(ii);
238                             }
239                         }
240 
241                         for (int ii = 0; ii < messages_index.size(); ii++)
    {
242                             int index = messages_index[ii];
243                             int index_from = path[index][0];
244                             int index_to = path[index][1];
245                             if (messages_index.size() >= matrix_bandwidth[
    index_from][index_to]) {
246                                 if (ii < matrix_bandwidth[index_from][
    index_to]) {
247                                     reserv[index] = 1;
248                                 }
249                             }
250                             else {
251 
252                                 if (ii == messages_index.size() - 1) {
253                                     reserv[index] = matrix_bandwidth[
    index_from][index_to] - matrix_bandwidth[index_from][index_to] /
    messages_index.size() * (messages_index.size() - 1);
254                                 }
255                                 else {
256                                     reserv[index] = matrix_bandwidth[
    index_from][index_to] / (messages_index.size());
257                                 }
258                             }
259                         }
260                     }
261                 }
262             }
263         }
264         else {
265             int index_from;
266             int index_to;
267             vector<int> messages_index; // индексы подходящих сообщений
268             if (path[i].size() > 0 && path[i][0] != path[i][1]) {
269                 index_from = path[i][0];
270                 index_to = path[i][1];
271                 for (int ii = 0; ii < messages.size(); ii++) {
272                     if (index_from != index_to && //не для штрафных
```

```
273                     path[ii].size() > 0 && path[ii][0] == index_from &&
    path[ii][1] == index_to) {
274                         messages_index.push_back(ii);
275                     }
276                 }
277 
278                 for (int ii = 0; ii < messages_index.size(); ii++) {
279                     int index = messages_index[ii];
280                     if (messages_index.size() >= matrix_bandwidth[
    index_from][index_to]) {
281                         if (ii < matrix_bandwidth[index_from][index_to]) {
282                             if (matrix_load[index_from][index_to] !=
    matrix_bandwidth[index_from][index_to])
283                             matrix_load[index_from][index_to] += reserv[
    index];
284                             reserv[index] = 1;
285                             matrix_load[index_from][index_to] -= reserv[
    index];
286                         }
287                     }
288                     else {
289                         if (ii == messages_index.size() - 1) {
290                             if (matrix_load[index_from][index_to] !=
    matrix_bandwidth[index_from][index_to])
291                             matrix_load[index_from][index_to] += reserv[
    index];
292                             int total = matrix_bandwidth[index_from][
    index_to] - matrix_bandwidth[index_from][index_to] / messages_index.size
    () * (messages_index.size() - 1);
293                             int can_send = reminder[index] < total ?
    reminder[index] : total;
294                             reserv[index] = can_send;
295                             matrix_load[index_from][index_to] -= reserv[
    index];
296                         }
297                         else {
298                             if (matrix_load[index_from][index_to] !=
    matrix_bandwidth[index_from][index_to])
299                             matrix_load[index_from][index_to] += reserv[
    index];
300                             int total = matrix_bandwidth[index_from][
    index_to] / (messages_index.size());
301                             int can_send = reminder[index] < total ?
    reminder[index] : total;
```

```
302                             reserv[index] = can_send;
303                             matrix_load[index_from][index_to] -= reserv[
    index];
304                         }
305                     }
306                 }
307                 for (int ii = 0; ii < messages_index.size(); ii++) {
308 
309                     if (matrix_load[index_from][index_to] > 0) {
310                         for (int iii = 0; iii < messages_index.size(); iii
    ++) {
311                             int index = messages_index[iii];
312                             if (reminder[index] > reserv[index]) {
313                                 reserv[index]++;
314                                 matrix_load[index_from][index_to]--;
315                             }
316                         }
317                     }
318                 }
319             }
320         }
321     }
322 }
```

2.6 Метод Can_Make_Route

Метод `Can_Make_Route` проверяет, возможно ли создать маршрут для заданного сообщения. Он выполняет следующие действия:

- Итерирует по всем сообщениям в векторе `messages` и проверяет, совпадает ли текущее сообщение с любым из предыдущих сообщений. Если найдено совпадение и резерв больше нуля, устанавливается флаг `flag1`.
- Проверяет наличие свободных маршрутов в матрице нагрузки для узла отправителя. Если хотя бы один узел имеет доступную пропускную способность, устанавливается флаг `flag2`.
- Возвращает `true`, если выполняется хотя бы одно из условий (флаги `flag1` или `flag2` равны `true`).

Реализация метода представлена листинге 5.

Листинг 5. Метод Can_Make_RouteStep

```
1 bool Network::Can_Make_Route(vector<int> message) {
```

```
    bool flag1 = false;
    bool flag2 = false;
    bool flag3 = false;
    for (int i = 0; i < messages.size(); i++) {
        if (message[0] == messages[i][0] &&
        message[1] == messages[i][1] &&
        message[2] == messages[i][2]){

            if (i < reserv.size() && reserv[i] > 0) {
                flag1 = true;
            }
        }
    }
    for (int i = 0; i < count_vertex; i++) {
        if (matrix_load[message[0]][i]) {
            flag2 = true;
            break;
        }
    }
    return flag1 || flag2;
}
```

3 Пример работы программы

```
Программа, моделирующая работу механизма передачи сообщения в коммуникационной сети суперкомпьютера

Выберите студента:
[1] Алексей Шихалев {13, 4, 3, 4, 3}
[2] Игорь Гладков {11, 5, 3, 4, 2}
[3] Никита Ромашко {10, 3, 4, 3, 3}
[4] Георгий Золоев {7, 6, 3, 2, 3}
[0] Выход из программы
2

Количество групп: 11
Количество коммутаторов в группе: 5
Количество узлов на коммутатор: 3
Пропускная способность внутри группы: 4
Пропускная способность между группами: 2


---


Шаг 0.

Список сообщений:

Нет сообщений для пересылки!

Выберите действие:
[1] Отправить сообщения (до 10 штук)
[0] Сменить студента
1
Введите количество сообщений, которое хотите отправить (от 0 до 10): 2

Введите сообщение в формате АДРЕС_ИСТОЧНИКА ДЛИНА_СООБЩЕНИЯ АДРЕС_ПОЛУЧАТЕЛЯ,
где адреса находятся в пределах от 0 до 164 включительно
0 3 3

Введите сообщение в формате АДРЕС_ИСТОЧНИКА ДЛИНА_СООБЩЕНИЯ АДРЕС_ПОЛУЧАТЕЛЯ,
где адреса находятся в пределах от 0 до 164 включительно
11 2 5

```

```
39 ----------------------------------------------------------------------------------------------------

40 Шаг 0.
41
42 Список сообщений:
43
44 {0,3,3} : Сообщение ожидает отправки
45 {11,2,5} : Сообщение ожидает отправки
46
47 Выберите действие:
48 [1] Отправить сообщения (до 10 штук)
49 [2] Следующий шаг
50 [0] Сменить студента
51 2
52
53 ----------------------------------------------------------------------------------------------------

54 Следующий шаг.
55 Шаг 1.
56
57 Список сообщений:
58
59 {0,3,3} : Из узла 0 в коммутатор 165 на данном шагу будет передано 3/3
60 {11,2,5} : Из узла 11 в коммутатор 168 на данном шагу будет передано 2/2
61
62 Выберите действие:
63 [1] Отправить сообщения (до 10 штук)
64 [2] Следующий шаг
65 [0] Сменить студента
66 2
67
68 ----------------------------------------------------------------------------------------------------

69 Следующий шаг.
70 Шаг 2.
71
72 Список сообщений:
73
74 {0,3,3} : Штраф. На данном шагу будет находится в 165 коммутаторе
75 {11,2,5} : Штраф. На данном шагу будет находится в 168 коммутаторе
76
77 Выберите действие:
78 [1] Отправить сообщения (до 10 штук)
79 [2] Следующий шаг
```

```
[0] Сменить студента
2

---

Следующий шаг.
Шаг 3.

Список сообщений:

{0,3,3} : Из коммутатора 165 в коммутатор 166 на данном шагу будет передано 3/3
{11,2,5} : Из коммутатора 168 в коммутатор 166 на данном шагу будет передано 2/2

Выберите действие:
[1] Отправить сообщения (до 10 штук)
[2] Следующий шаг
[0] Сменить студента
2

---

Следующий шаг.
Шаг 4.

Список сообщений:

{0,3,3} : Штраф. На данном шагу будет находится в 166 коммутаторе
{11,2,5} : Штраф. На данном шагу будет находится в 166 коммутаторе

Выберите действие:
[1] Отправить сообщения (до 10 штук)
[2] Следующий шаг
[0] Сменить студента
2

---

Следующий шаг.
Шаг 5.

Список сообщений:

```

```
{0,3,3} : Из коммутатора 166 в узел 3 на данном шагу будет передано 3/3
{11,2,5} : Из коммутатора 166 в узел 5 на данном шагу будет передано 2/2

Выберите действие:
[1] Отправить сообщения (до 10 штук)
[2] Следующий шаг
[0] Сменить студента
2

--------------------------------------------------------------------------------

Следующий шаг.
Шаг 6.

Список сообщений:

Нет сообщений для пересылки!

Выберите действие:
[1] Отправить сообщения (до 10 штук)
[0] Сменить студента
```

Заключение

В процессе выполнения работы было реализовано консольное приложение, моделирующее работу механизма передачи сообщения в коммуникационной сети суперкомпьютера. Все поставленные задачи были выполнены:

1. Заданная конфигурация соединений предложенной топологии была описана с помощью матриц смежности и пропускных способностей.

2. Реализана функция отправки сообщения, которая вызывается пользователем и включает в себя функцию вычисления маршрута передачи сообщения с целью минимизации времени, необходимого для передачи. Формат сообщения: “АДРЕС_ИСТОЧНИКА ДЛИНА_СООБЩЕНИЯ АДРЕС_ПОЛУЧАТЕЛЯ”.

3. Реализована возможность продвигаться по шагам, и в начале каждого шага пользователь может указать от 0 до 10 заданий на пересылку сообщений с произвольными адресами источника и получателя. ДЛИНА_СООБЩЕНИЯ — это есть число шагов (количество единиц времени), требуемых для его передачи по свободному каналу. Важной особенностью является штраф за каждую пересылку через промежуточный узел или через коммутатор в 1 дополнительный шаг времени.

Работа была выполнена в среде Visual Studio 2022 на языке программирования C++.

Список литературы

[1] В. Олифер, Н. Олифер. Компьютерные сети. Принципы, технологии, протоколы. — Питер, 2013. — С. 55. — 944 с. — 3000 экз.

[2] Russell J. Super-Connecting the Supercomputers: Innovations Through Network Topologies [Электронный ресурс]. HPCwire. 2019. URL: https://www.hpcwire.com/2019/07/15/super-connecting-the-supercomputers-innovations-through-network-topologies/ (дата обращения: 25.09.2024).

[3] Huawei Network. Dragonfly Topology | Test It, Believe It Series for Data Center Networks [Видео]. YouTube, 2019. URL: https://www.youtube.com/watch?v=atKMrPmTOXY (дата обращения: 25.09.2024).