



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»

Отчёт по практикуму №2 по курсу «Архитектура ЭВМ»

Тема Обработка и визуализация графов в вычислительном комплексе Тераграф

Студент Карпова Е.О.

Группа ИУ7-52Б

Преподаватель Дубровин Е.Н.

Москва — 2022 г.

Оглавление

Введение	3
1 Аналитическая часть	4
1.1 Конвейер визуализации графов	4
1.2 Использование библиотеки шаблонов leonhard-x64-xrt-v2 для обработки графов	5
2 Ход выполнения работы	6
Заключение	10

Введение

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 Аналитическая часть

1.1 Конвейер визуализации графов

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. Графы можно визуализировать, используя:

- 2D графическую сцену - наиболее часто применяемый случай, обладающий приемлемой вычислительной сложностью;
- 3D графическую сцену - такой вариант позволяет выполнять перемещение камеры наблюдения, что увеличивает возможное количество визуализируемых вершин;
- Иерархическое представление - граф представляется в виде иерархически вложенных подграфов (уровней), что позволяет более наглядно представить тесно связанные компоненты первоначального графа.

Для визуализации графов было определено несколько показателей качества, позволяющие получить количественные оценки эстетики и удобства графического представления. Алгоритмы раскладки, в большинстве случаев, нацелены на оптимизацию следующих показателей:

- Меньшее количество пересечений ребер: выравнивание вершин и ребер для получения наименьшего количества пересечений ребер делает визуализацию более понятной и менее запутанной.
- Минимум наложений вершин и ребер.
- Распределение вершин и/или ребер равномерно.
- Более тесное расположение смежных вершин.
- Формирование сообществ вершин из сильно связанных групп вершин.

Для больших графов метод выделения сообществ является предпочтительным, так как позволяет выявить скрытые кластеры вершин, показать

центральную вершину сообщества, минимизировать количество внешних связей между сообществами. Таким образом визуализация графов представляет собой многостадийный алгоритмический процесс. Кратко стадии процесса визуализации представлены на следующем рисунке.

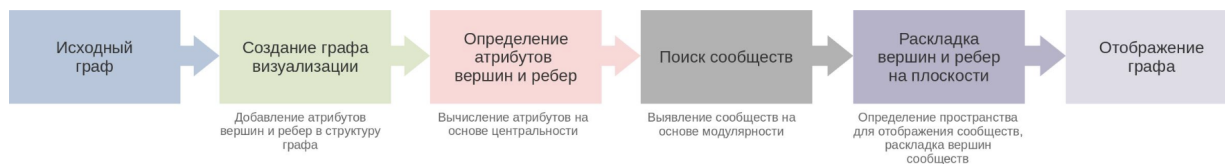


Рисунок 1.1 – Схема визуализации графов

1.2 Использование библиотеки шаблонов `leonhard-x64-xrt-v2` для обработки графов

Для реализации алгоритмов обработки графов необходимо представить операции над множествами (в том числе, множествами вершин и ребер графа) в виде набора команд дискретной математики DISC. Все команды обработки структур данных изменяют регистр статуса, по которому можно определить, было ли выполнение команды успешным (Регистр `LNH_STATE`, бит `SPU_ERROR_FLAG`). Результаты, влияющие на работу программы, должны быть учтены в общем алгоритме. После завершения основания команд, основанных на поиске (`SEARCH`, `DELETE`, `MAX`, `MIN`, `NEXT`, `PREV`, `NSM`, `NGR`) в очередь данных попадают ключ и значение найденных записей (`KEY`, `VALUE`), которые могут быть использованы в алгоритме программного ядра CPE riscv32. Для команд И-ИЛИ-НЕ (пересечение, объединение, дополнение) передаются операнды номеров структур (`R,A,B`). Операнд `R` указывает на номер структуры, в которой будет сохранен результат. Структуры `A` и `B` используются в И-ИЛИ-НЕ операциях и срезах в качестве исходных.

2 Ход выполнения работы

Все задания практикума выполнялись по варианту 11.

Выполнить визуализацию неориентированного графа, представленного в формате tsv. Каждая строчка файла представляет собой описание ребра, состоящее из трех чисел (Вершина,Вершина,Вес).

В листинге 2.1 представлен код программы по индивидуальному варианту из файла *host_main.cpp*.

Листинг 2.1 – Код программы по индивидуальному варианту

host_main.cpp

```
1 #ifdef MY_GRAPH
2
3     __foreach_core(group, core)
4     {
5         lnh_inst.gpc[group][core]->start_async(__event__(delete_graph));
6     }
7
8
9     unsigned int*
10         host2gpc_ext_buffer[LNH_GROUPS_COUNT][LNH_MAX_CORES_IN_GROUP];
11     unsigned int messages_count = 0;
12     unsigned int u, v, w;
13
14     __foreach_core(group, core)
15     {
16         host2gpc_ext_buffer[group][core] = (unsigned
17             int*)lnh_inst.gpc[group][core]->external_memory_create_buffer(16
18             * 1048576 * sizeof(int));
19         offs = 0;
20
21         std::ifstream file(argv[3], std::ios::in);
22
23         if (!file.is_open())
24         {
25             std::cout << "Error opening file." << std::endl;
26
27             return EXIT_FAILURE;
28         }
29
30         for (std::string line; std::getline(file, line); )
31         {
32             std::vector<std::string> tokens = split(line, '\t');
33
34             if (tokens.size() != 2)
```

```

32         {
33             std::cout << "Incorrect tokens count: expected 2, got "
34                 << tokens.size() << "." << std::endl;
35
36             return EXIT_FAILURE;
37         }
38
39         u = std::stoul(tokens[0]);
40         v = std::stoul(tokens[1]);
41         w = 1;
42
43         EDGE(u, v, w);
44         EDGE(v, u, w);
45         messages_count += 2;
46     }
47
48     lnh_inst.gpc[group][core]->external_memory_sync_to_device(0, 3 *
49         sizeof(unsigned int)*messages_count);
50 }
51 __foreach_core(group, core)
52 {
53     lnh_inst.gpc[group][core]->start_async(__event__(insert_edges));
54 }
55 __foreach_core(group, core) {
56     long long tmp =
57         lnh_inst.gpc[group][core]->external_memory_address();
58     lnh_inst.gpc[group][core]->mq_send((unsigned int)tmp);
59 }
60
61 __foreach_core(group, core) {
62     lnh_inst.gpc[group][core]->mq_send(3 *
63         sizeof(int)*messages_count);
64 }
65
66 __foreach_core(group, core)
67 {
68     lnh_inst.gpc[group][core]->finish();
69 }
70 printf("Data graph created!\n");
71 #endif

```

В рисунке 2.1 представлена команда для сборки проекта.

```

lu7031@dl580:~/worksp/task_02$ ./host/host_main leonhard_2cores_267mhz.xclbin ./sw-kernel/sw_kernel.rawbinary data/kronecker_var01.tsv

```

Рисунок 2.1 – команда для сборки проекта

В рисунке 2.2 представлен результат запуска.

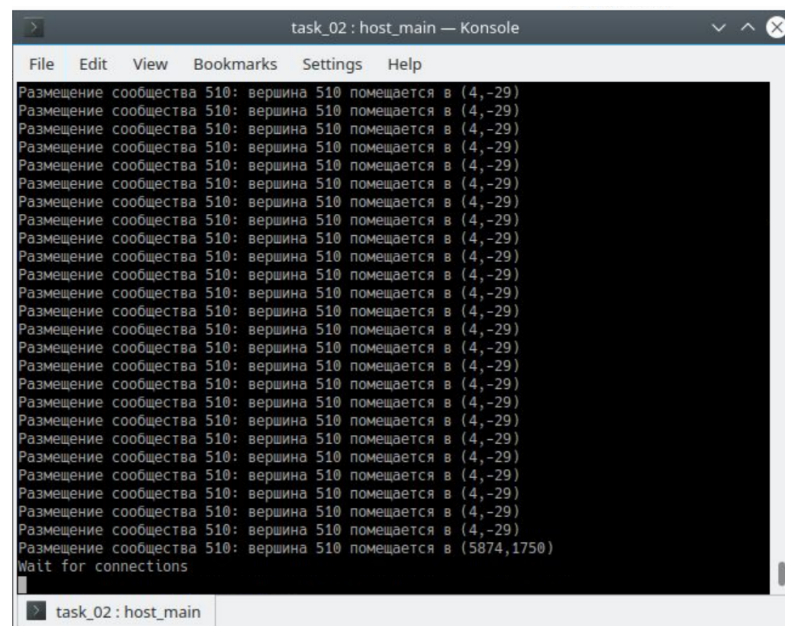


Рисунок 2.2 – результат запуска

В рисунке 2.3 представлена команда для запуска сервера bokeh.

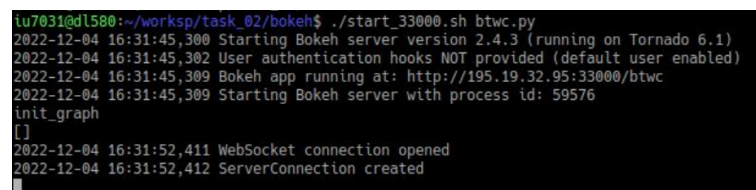


Рисунок 2.3 – команда для запуска сервера bokeh

В рисунке 2.4 результат визуализации.

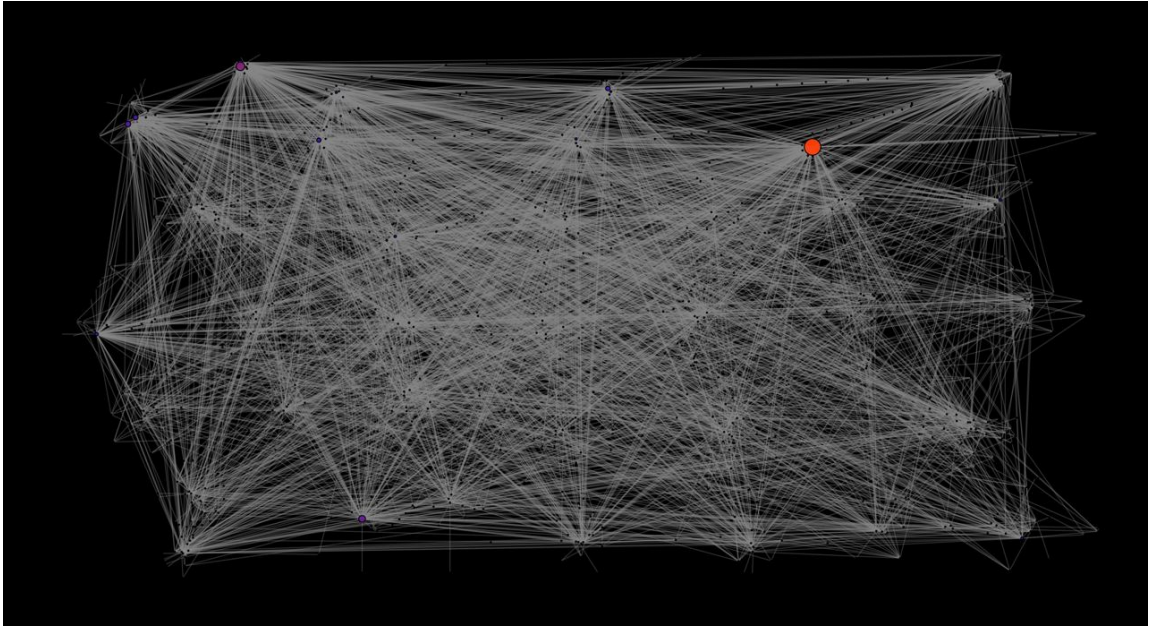


Рисунок 2.4 – результат визуализации

Заключение

В данном практикуме были изучены представления графов и их обработка с помощью вычислительного комплекса Тераграф. В ходе практикума было проведено знакомство с вариантами представления графов в виде объединения структур языка C/C++, изучено и применено на практике для решения некоторых задач на графах. По индивидуальному варианту была разработана программа хост-подсистемы и программного ядра `sw_kernel`, выполняющая обработку и визуализацию графов.