Московский Государственный Университет имени М.В. Ломоносова

Факультет вычислительной математики и кибернетики

Кафедра суперкомпьютеров и квантовой информатики

Отчёт по курсу

«Суперкомпьютерное моделирование и технологии»

# Анализ масштабируемости графового алгоритма SSSP в реализации Groute на высокопроизводительной вычислительной системе Polus

Работу выполнил:

Козлов Михаил Владимирович

623 группа

Москва, 2019

## Формулировка задания и описание используемой реализации

При выполнении задания необходимо исследовать масштабируемость алгоритма SSSP (Single Source Shortest Path) для ориентированных взвешенных графов в реализации библиотеки Groute[1]. Каждый из запусков должен производиться на одном из узлов системы Polus, оборудованным двумя Nvidia P100 GPU. Оценки масштабируемости исследуемой реализации следует провести с использованием синтетических входных графы трех типов: RMAT, случайных равномерных и SSCA2.

## Математическое описание задачи и алгоритма

Пусть задан граф $G = (V, E)$ с весами $f(e)$ и выделенной вершиной-источником $u$. Последовательность рёбер $P(u, v)$ $e_1 = (u, w_1), e_2 = (w_1, w_2), \dots, e_k = (w_{k-1}, v)$ называется путём, идущим от вершины $u$ к вершине $v$. Суммарный вес этого пути равен $f\big(P(u,v)\big) = \sum_{i=1}^{k} f(e_i)$. Требуется для каждый вершины $v$, достижимой из вершины-источника $u$, указать путь $P^*(u, v)$, имеющий наименьший возможный суммарный вес: $f^*(v) = f\big(P^*(u,v)\big) = \min f(P(u,v))$.

Решение задачи удовлетворяет принципу оптимальности: если путь $P^*(u, w)$ является частью кратчайшего пути $P^*(u, v)$, то $P^*(u, w)$ является кратчайшим путём от источника $u$ до вершины $w$.

Принцип оптимальности означает, что для каждой вершины $v$ вместо всего кратчайшего пути $P^*(u, v)$ достаточно хранить его последнее ребро $e_v^*$. Отсюда следует оценка объёма памяти $O(m)$.

## Ресурс параллелизма алгоритма

Кратчайший путь от $u$ к $v$ может быть восстановлен за время $O(|P^*(u, v)|)$, если, начиная с вершины $v$, проходить рёбра $e_w^*$ в обратном направлении до тех пор, пока не будет посещена вершина $u$.

Рёбра $e_v^*$ могут быть восстановлены за время $O(m)$ перебором рёбер для каждой вершины: $e_v^* \in \{(w, v) \in E \mid f^*(v) = f^*(w) + f^*((v, w))\}$.

Перебор может осуществляться параллельно.

Для поиска кратчайших расстояний $f^*(v)$ по набору рёбер $e_v^*$, необходимо построить из них дерево, на котором выполнить поиск в ширину за время $O(m)$ (с возможностью параллелизации).

## Платформа тестирования

Анализ производительности реализации был выполнен на одном узле системы Polus[2]. Так как Groute является библиотекой, ориентированной только лишь на вычисления на GPU будет приведено более подробное описание графических ускорителей, установленных на узлах системы Polus. Каждый узел системы оборудован двумя NVIDIA P100 GPU архитектуры Pascal, подключенных к центральному процессору с использованием NVLINK. Каждый графический ускоритель P100 оснащен 3584 легковесными CUDA-ядрами с тактовой частотой 1190 MHz, в совокупности предоставляющими производительность в 9.3 TFLOPs при вычислениях с одинарной точностью. CUDA-ядра сгруппированы в потоковые мультипроцессоры, по 64 ядра в каждом.

Память устройства графического ускорителя P100 состоит из четырех стеков памяти HBM2, имеющих суммарный объем 16 ГБ и пропускную способность до 700 ГБ/с. Также в иерархию память P100 GPU входят разделяемый между всеми ядрами кэш L2 размера 4 MB, и кэш L1/разделяемая память размера 64 KB, приватная для каждого из потоковых мультипроцессоров.

## Настройка библиотеки и опции компиляции

При выполнении задания использовалась библиотека Groute. Библиотека была скачана из официального репозитория[3] на суперкомпьютер Polus[2]. Для компиляции библиотеки необходимо установить *cmake,* что было сделано через *Anaconda*. Далее, следуя документации, был запущен файл *setup.sh,* который компилирует библиотеку.

## Генерация и свойства входных графов

Для анализа масштабируемости исследуемой библиотеки были сгенерированы ориентированные взвешенные графы с количеством вершин $2^{17}-2^{25}$ и средней степенью связности 32. Генерация была произведена предоставленным генератором *generator.cpp*. Для каждого из размеров было сгенерировано по 3 графа следующих типов RMAT, случайных равномерных и SSCA2.

Библиотека Groute принимает на вход графы в формате CSR, поэтому предложенный генератор был доработан таким образом, чтобы выходной файл содержал граф в нужном формате.

## Исходный код программы для анализа производительности

Библиотека Groute имеет несколько встроенных функций для работы с графами. Одной из них является функция SSSP, которая и использовалась для анализа производительности.

Для запусков на кластере Polus был написан bash-скрипт *polus.sh*, в котором определяются параметры запуска. Написанный скрипт принимает на вход тип графа и ставит в очередь на выполнение программы для всех размеров графа, которые исследуются.

## Анализ масштабируемости алгоритма и интерпретация результатов

В качестве основной метрики для оценки производительности целевого компьютера на реализациях графовых алгоритмов используется TEPS (Traversed Edges Per Second) – число ребер графа, которое алгоритм обходит (обрабатывает) за одну секунду. Таким образом, производительность компьютера на реализации графового алгоритма можно вычислить по следующей формуле: $TEPS = \frac{\text{число рёбер в графе}}{\text{время выполнения в секундах}}$.

На рисунках 1-3 изображены полученные результаты тестирования алгоритма SSSP в библиотеке Groute.

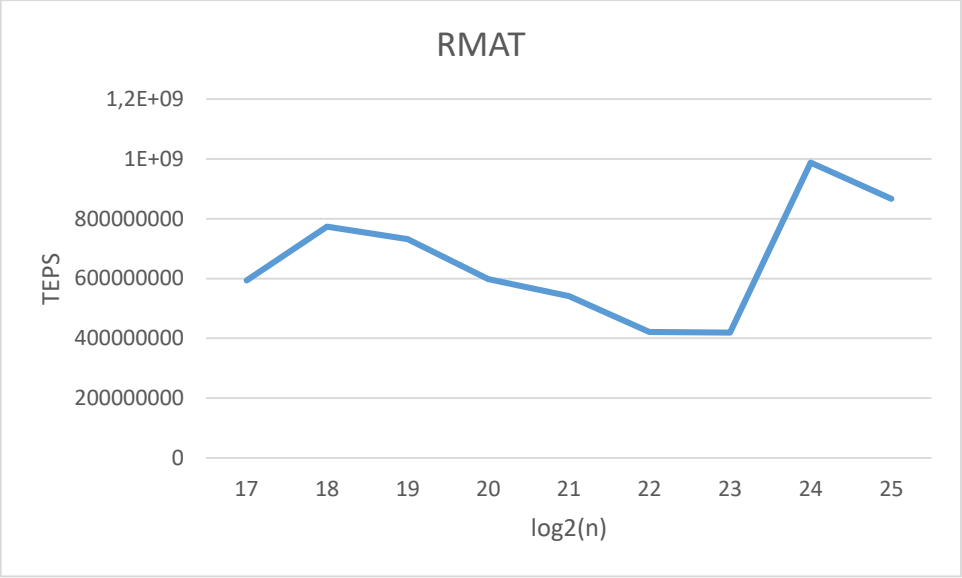


*Рисунок 1 Производительность реализации алгоритма SSSP из библиотеки Groute на графах типа RMAT.*

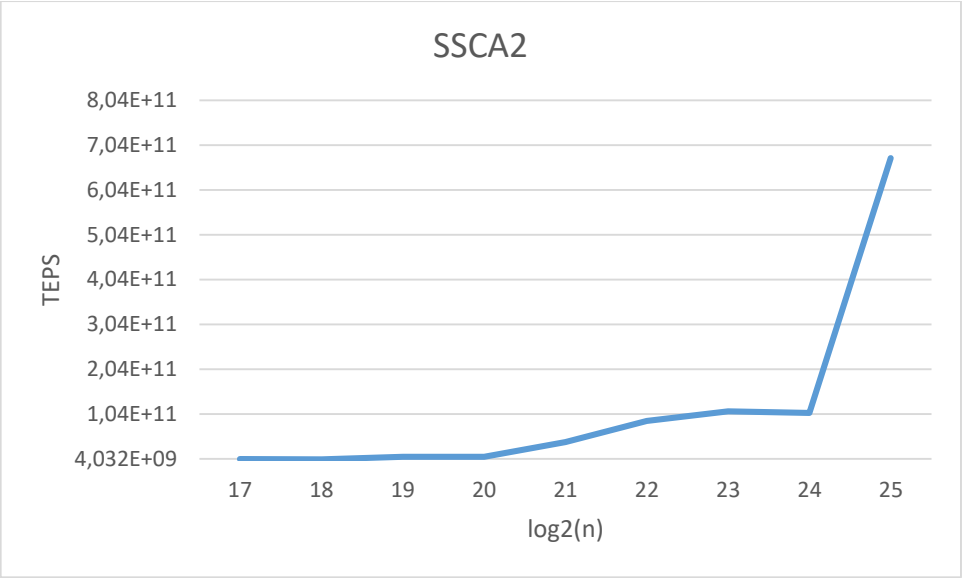


*Рисунок 2 Производительность реализации алгоритма SSSP из библиотеки Groute на графах типа SSCA2.*

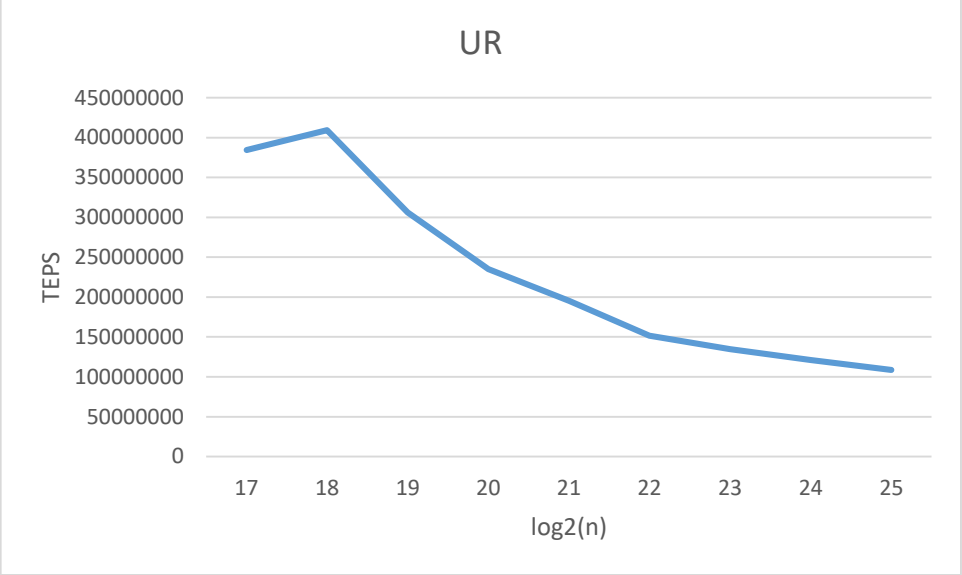


*Рисунок 3 Производительность реализации алгоритма SSSP из библиотеки Groute на графах типа равномерно-случайный граф(UR).*

Максимальные значения производительности для графов следующие:

- RMAT: 987 MTEPS (для графа $2^{24}$)
- SSCA2: 674 GTEPS (для графа $2^{25}$)
- UR: 40 MTEPS (для графа $2^{18}$)

Для графов SSCA2 производительность значительно выше, чем для двух других типов графов. Это объясняется тем, что при одинаковом количестве вершин, в SSCA2 графах гораздо меньше рёбер. По этой же причине не получается определить пик производительности для графов SSCA2, так как графы размером больше $2^{25}$ не исследуются. Дальнейшее падение производительности для графов RMAT и UR, объясняется тем, что графы невозможно эффективно кэшировать. Сдвиг пиков для данных графов можно объяснить разной структурой: в RMAT много одинаковых рёбер (с одинаков вершиной-источником и вершиной-целью) и много компонент связности, в отличие от UR.

## Проверка корректности

Корректность алгоритма проводилась с помощью встроенной проверки в библиотеке Groute. Проверка производилась для всех графов.

## Список литературы

[1] https://www.cse.huji.ac.il/~talbn/docs/groute-ppopp17.pdf

[2] http://hpc.cmc.msu.ru/polus

[3] https://github.com/groute/ppopp17-artifact