Санкт-Петербургский государственный университет
Кафедра системного программирования

Улучшение производительности алгоритма достижимости с регулярными ограничениями на графическом ускорителе

Козенко Дмитрий Сергеевич, группа 23.Б08-мм

Научный руководитель: к.ф.-м.н. С.В. Григорьев, доцент кафедры системного программирования

Санкт-Петербург
2025

Графовые базы данных и запросы с регулярными ограничениями

- Запросы с регулярными ограничениями активно используются в графовых базах данных
- Алгоритм RPQ[1] основан на операциях линейной алгебры, которые отлично подходят вычислений на GPU
- Текущая реализация RPQ[2] на GPU на наборе данных Wikidata проигрывает CPU почти в 4 раза

[1] RPQ с линейной алгеброй: https://se.math.spbu.ru/thesis/texts/Beljanin_Georgij_Olegovich_Spring_practice_2nd_year_2024_text.pdf (Дата доступа: 19.06.2025)

[2] GitHub репозитория с реализацией алгоритма на GPU: https://github.com/mitya-y/rpq/tree/cb2583e (Дата доступа: 19.06.2025)

Постановка задачи

Целью работы является анализ узких мест реализации алгоритма на GPU и улучшение ее производительности

Задачи:

1. Воспроизвести тестирование из прошлой работы, проанализировать результаты и найти узкие места алгоритма
2. Улучшить производительность алгоритма
3. Провести эксперименты с итоговой версией алгоритма
4. Перенести свою ветку update-cuda-12.6 в основной репозиторий cuBool, организовать репозиторий с алгоритмами, использующими cuBool, и добавить в нее свой алгоритм

Описание данных и стендов

В ходе работы над анализом алгоритма были проведены измерения на следующих наборах данных

- Wikidata[3]: набор данных из реального мира вместе с 660 запросами
- RPQ-Bench[4]: синтетический набор данных, 20 типов запросов. Было создано 2 датасета: с графом на 10^7 вершин, по 1000 запросов каждого типа и с графом на 10^8 вершин и 100 запросов каждого вида

Были использованы следующие тестовые стенды

- Intel i5-10400f и Nvidia RTX 3050 (8 Gb VRAM), 16 Gb RAM, Manjaro 6.14
- AMD Ryzen 9 7900X и Nvidia RTX 3090 (24 Gb VRAM), 128 Gb RAM, Ubuntu 24.04

[3] База знаний WikiData: https://www.wikidata.org/wiki/Wikidata:Database_download (Дата доступа 19.06.25)

[4] Набор данных RPQ-Bench: `https://github.com/Mamenglu/LD-RPQB` (Дата доступа 19.06.25)

Улучшение алгоритма

Для улучшения производительности алгоритма были применены следующие изменения:

1. Изменена реализация функции сложения матриц — используется схожая поэлементному умножению матриц реализация вместо библиотеки Nsparse
2. Исправлена ошибка в cmake с конфигурациями сборки
3. Добавлена возможно передавать транспонированные матрицы в алгоритм

Таблица: Сравнение времени работы алгоритма до и после изменений

Версия алгоритма	Время выполнения 520 запросов, с	Время загрузки, с
Без изменений	273.09	5704.66
С изменениями	84.61	666.15

Организация репозитория cuBoolGraph[5]

```
cuBoolGraph [add-rpq] tree -d -L 2
.
├── deps
│   ├── cuBool
│   ├── fast_matrix_market
│   └── googletest
├── include
├── src
└── tests
    └── test_data
```

Рис.: Структура репозитория cuBoolGraph

Особенности проекта

- Система сборки — `CMake`
- Зависимости скачиваются через `git submodule`
- CI/CD: GitHub Actions (сборка и тесты)

[5] GitHub репозиторий cuBoolGraph: `https://github.com/SparseLinearAlgebra/cuBoolGraph` (Дата доступа: 19.06.2025)

Эксперимент

Для дальнейшего анализа производительности алгоритма были поставлены следующие задачи

1. Замерить время работы реализаций на CPU и GPU и проанализировать типы запросов, работающих на GPU медленее, чем на CPU
2. Изучить масштабируемость реализации для GPU, сравнив время работы на двух разных по мощности тестовых стендах

Суммарное время работы выполнения всех запросов:

Данные	Время GPU,	Время CPU,	Ускорение GPU
Wikidata	115.73	136.30	1.18
RPQ-bench 10^7	185.28	55.19	0.30

Таблица: Результаты сравнения CPU и GPU

Анализ набора данных Wikidata

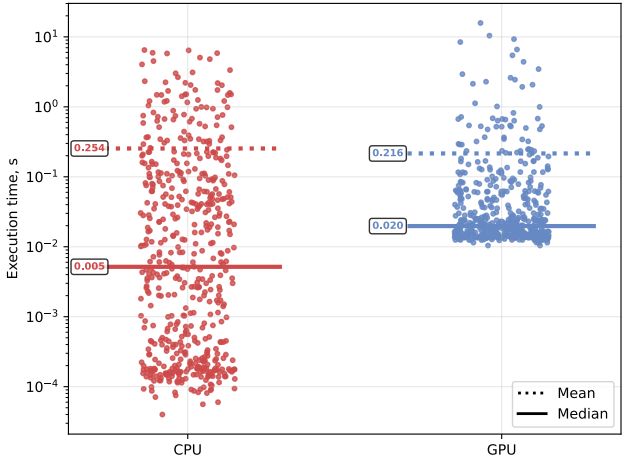

Рис.: Облако распределения времени исполнения запросов

Анализ набора данных RPQ-bench

Таблица: Время работы каждой операции на наборе данных RPQ-bench 10^7 в реализациях на GPU и CPU

Операция	GPU, s	CPU, s
`MxM`	148.96 (80.11%)	20.37 (37.13%)
`VxM`	21.19 (11.49%)	8.08 (14.73%)
`cuBool_EwiseAdd`	6.55 (3.52%)	—
`cuBool_EwiseMulInverted`	9.23 (4.96%)	—
`GrB_Matrix_assign`	—	26.22 (47.79%)

Анализ масштабируемости алгоритма

Суммарное время работы выполнения всех запросов:

Таблица: Результаты замеров на RTX 3050 и RTX 3090

Набор данных	RTX 3050, s	RTX 3090, s	Ускорение
wikidata	116.50	100.99	1.15
RPQ-bench 10^7	185.28	131.46	1.41
RPQ-bench 10^8	318.86	300.17	1.06

Таблица: Время работы каждой операции на RTX 3050 и RTX 3090 на датасете RPQ-bench на 10^8 вершин

Операция	RTX 3050,	RTX 3090,	Ускорение
`MxM`	274.71 (85.00%)	269.33 (90.27%)	1.02
`VxM`	22.65 (7.00%)	11.22 (3.76%)	2.02
`EwiseAdd`	15.28 (4.73%)	11.67 (3.91%)	1.31
`EwiseMulInverted`	10.57 (4.27%)	6.15 (2.06%)	1.72

Результаты

- Найдены узкие места в алгоритме и исправлены ошибки, вследствие чего производительность выросла
- Проведено сравнение финальной версии реализации на GPU с реализацией на CPU, показавшее, что на крупных запросах GPU может дать выигрыш, а на маленьких предпочтительнее использовать CPU
- Проведено исследование масштабируемости алгоритма, который выявил узкое место — операция умножения матриц
- Изменения cuBool были перенесены в основной репозиторий, а исходный код в cuBoolGraph[6]

[6] GitHub репозиторий cuBoolGraph: `https://github.com/SparseLinearAlgebra/cuBoolGraph` (Дата доступа: 19.06.2025)