Санкт-Петербургский Государственный университет

Реализация алгоритма поиска путей в графовых базах данных через тензорное произведение на GPGPU

Орачев Егор Станиславович, 17.Б11-мм
Научный руководитель: доцент кафедры информатики,
к.ф.-м.н. С.В. Григорьев
Рецензент: разработчик биоинформатического ПО,
ЗАО “БИОКАД” А.С. Хорошев

Программная инженерия

10 апреля 2021

Введение

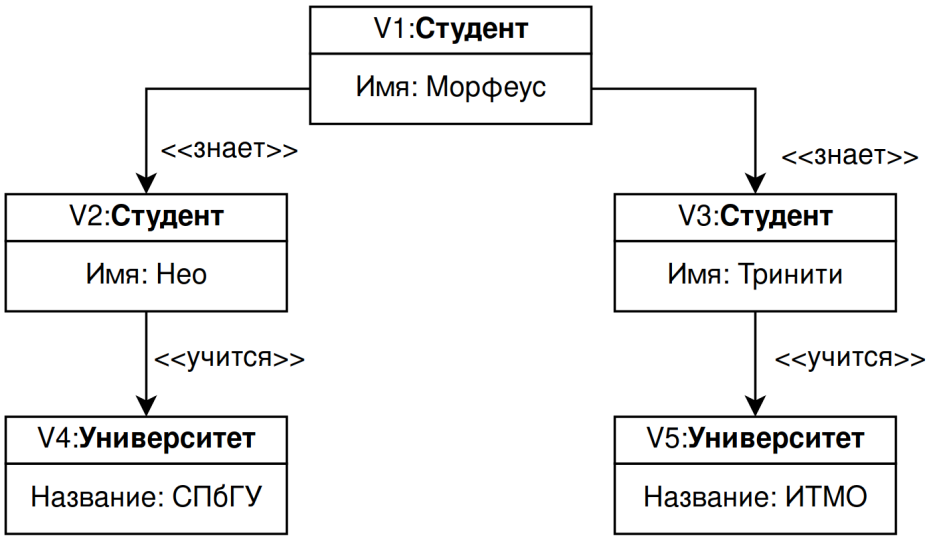

Figure: Пример графовой БД

- Графовая модель данных
- Графовые базы данных
- Граф программы, потока управления, данных и т.д
- Запрос к графовой БД: ограничение на пути

Запросы с КС ограничениями

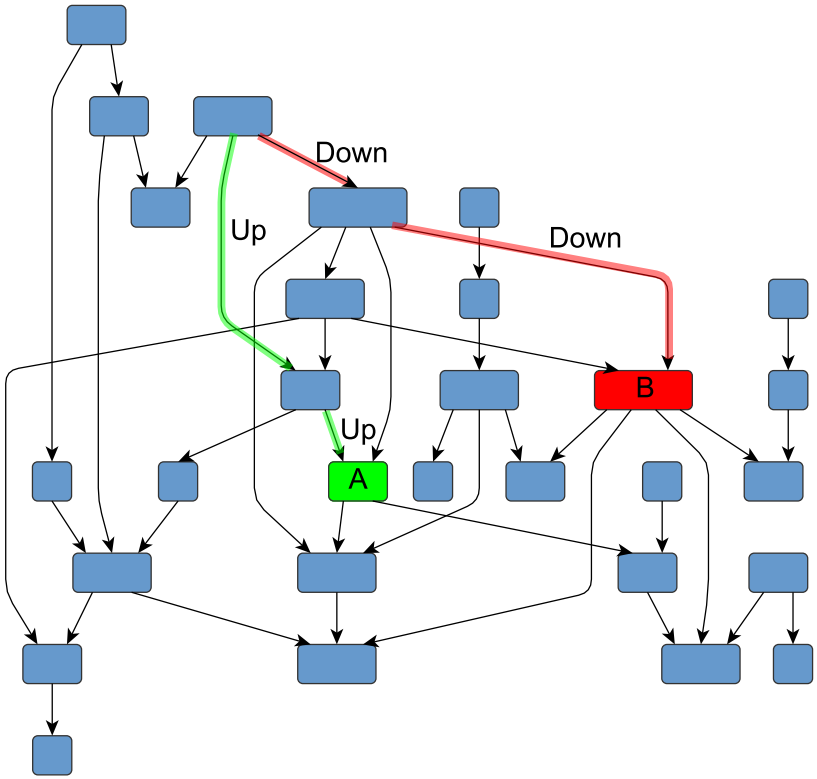

Figure: Пример графа

- Мотивация: навигация в графе
- Пример: находятся ли вершины A и B на одном уровне иерархии?
- Применение: анализ RDF данных, биоинформатика, статический анализ кода

Алгоритмы поиска путей с КС ограничениями

- На основе линейной алгебры:
  - Алгоритм Рустама Азимова
  - **Алгоритм на основе тензорного произведения**[1]
- Операции: матричное умножение, поэлементное сложение, произведение Кронекера и т.д. в булевом полукольце
- Структура матриц: разреженная
- Для эффективной реализации требуется библиотека примитивов разреженной булевой линейной алгебры на GPGPU

[1] Context-Free Path Querying by Kronecker Product, Egor Orachev, Ilya Epelbaum, Rustam Azimov, Semyon Grigorev. Дата обращения: 9.04.2021. Ссылка на статью: `https://link.springer.com/chapter/10.1007/978-3-030-54832-2_6`

Цель и задачи

Целью данной работы является реализация алгоритма поиска путей в графовых базах данных через тензорное произведение на GPGPU.
Задачи:

- Разработка архитектуры библиотеки примитивов разреженной линейной булевой алгебры для вычислений на GPGPU
- Реализация библиотеки в соответствии с разработанной архитектурой
- Реализация алгоритма поиска путей с КС ограничениями через тензорное произведение с использованием разработанной библиотеки
- Экспериментальное исследование полученных результатов

Требования к библиотеке

- Поддержка вычислений на Cuda-устройстве
- Поддержка вычислений на CPU
- C-совместимый API для работы с библиотекой
- Python-пакет для работы с примитивами и операциями библиотеки в управляемой высокоуровневой среде языка Python
- Поддержка логирования, функций для отладки и прототипирования конечных пользовательских алгоритмов

Архитектура библиотеки

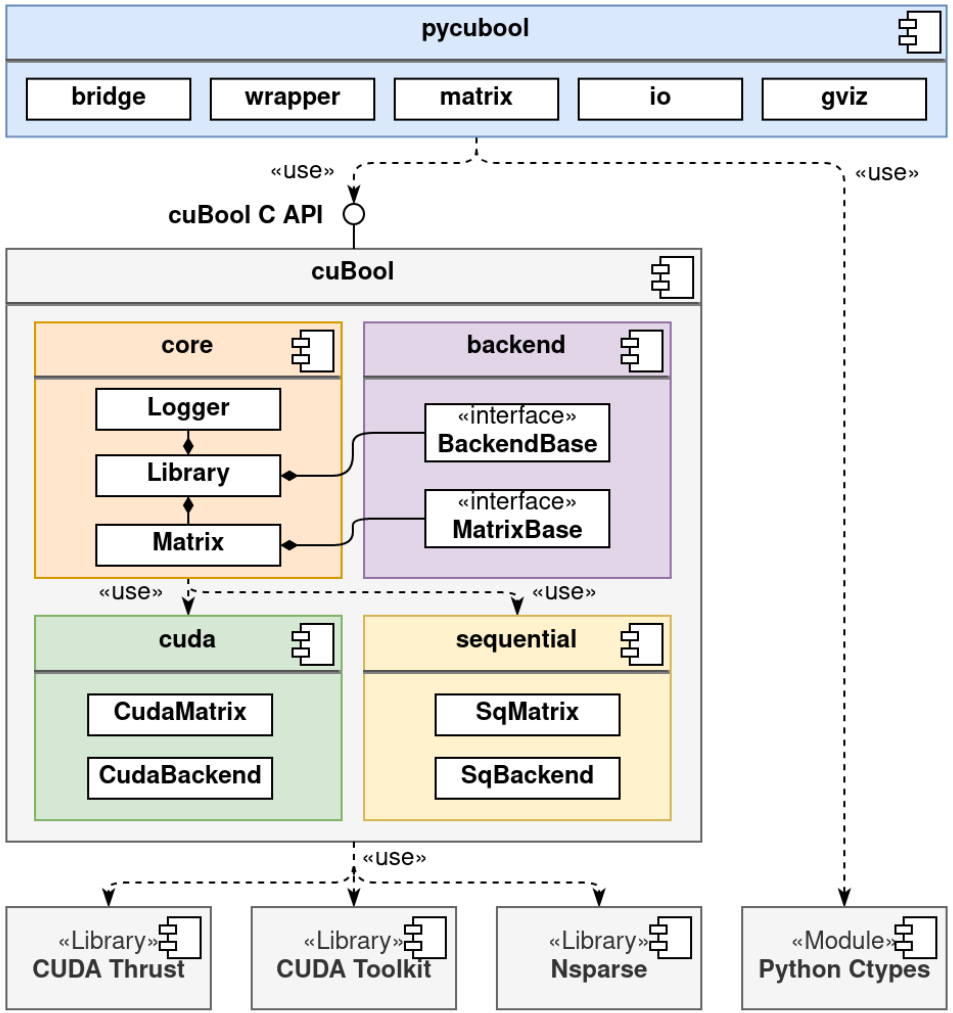

Figure: Архитектура разработанной библиотеки

Последовательность обработки операций

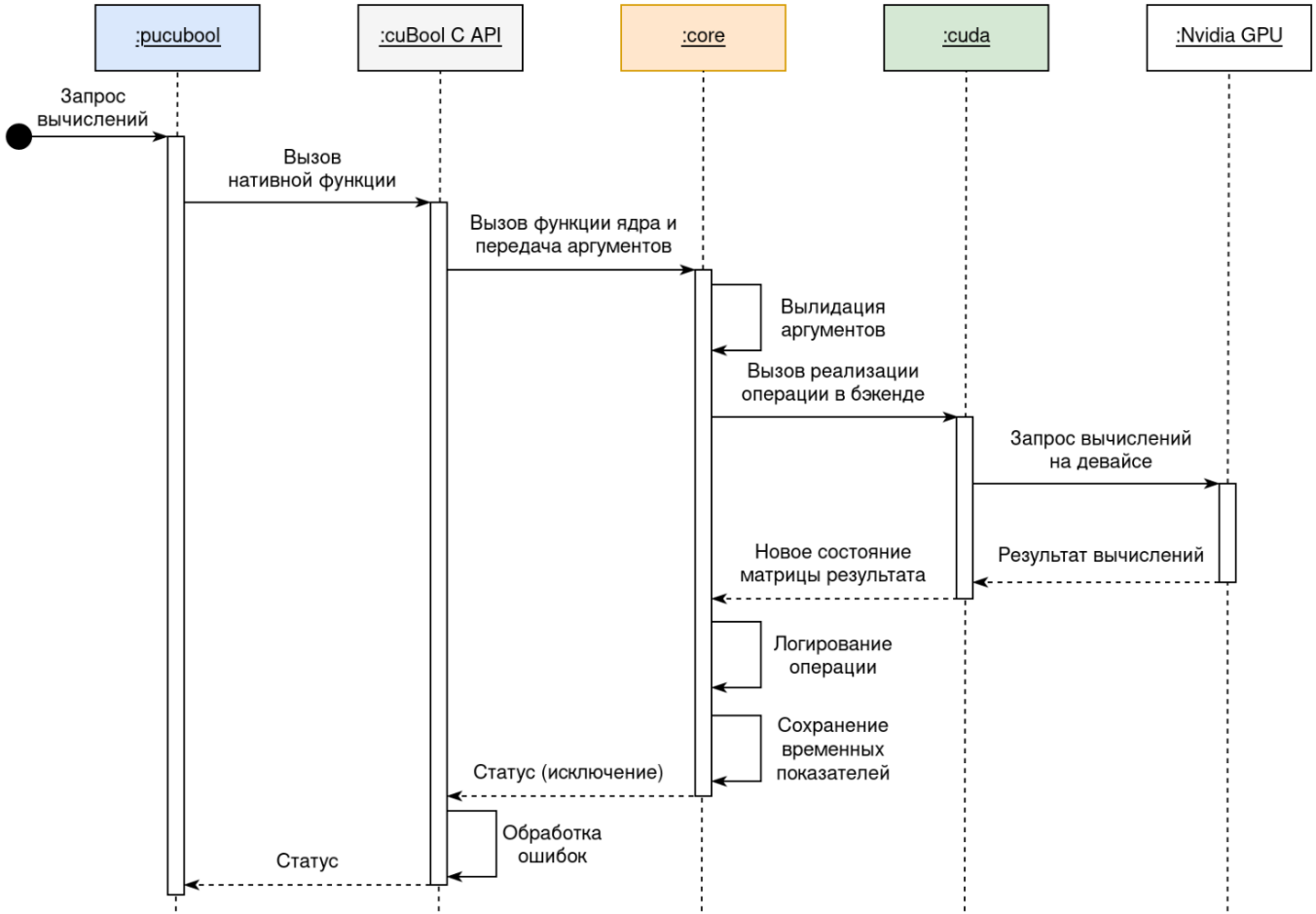

Figure: Последовательность выполнения вычислительной матричной операции на Nvidia GPU с использованием pycubool

Детали реализации

- C, C++, CUDA C++, Python
- Разреженная матрица булевых значений формате CSR
- Размер в памяти $sizeof(unsigned\ int) * (rows + 1 + nvals)$
- Операции:
  - Умножение
  - Поэлементное сложение
  - Произведение Кронекера
  - Транспонирование
  - Редуцирование к вектор-столбцу
  - Извлечение подматрицы

Пример использования C API

```c
#include <cubool/cubool.h>

cuBool_Status TransitiveClosure(cuBool_Matrix A, cuBool_Matrix* T) {
    cuBool_Matrix_Duplicate(A, T);                          /* Копируем матрицу смежности A */

    cuBool_Index total = 0;
    cuBool_Index current;
    cuBool_Matrix_Nvals(*T, &current);                      /* Количество ненулевых значений */

    while (current != total) {                              /* Пока результат меняется  */
        total = current;
        cuBool_MxM(*T, *T, *T, CUBOOL_HINT_ACCUMULATE);     /* T += T x T */
        cuBool_Matrix_Nvals(*T, &current);
    }

    return CUBOOL_STATUS_SUCCESS;
}
```

Figure: Вычисление транзитивного замыкания для ориентированного графа без меток с использованием **cuBool C API**

Пример использования Python API

```python
import pycubool

def transitive_closure(a: pycubool.Matrix):
    t = a.duplicate()                       # Копируем матрицу смежности A
    total = 0                               # Количество ненулевых значений результата

    while total != t.nvals:                 # Пока результат меняется
        total = t.nvals
        t.mxm(t, out=t, accumulate=True)  # t += t x t

    return t
```

Figure: Вычисление транзитивного замыкания для ориентированного графа без меток с использованием **pycubool**

Алгоритм поиска путей с КС ограничениями через тензорное произведение

- Алгоритм реализован с использованием разработанного пакета pycubool
- Реализация доступна в рамках проекта PyAlgo-CFPQ
- Вход: ориентированный граф с метками на ребрах и КС грамматика в виде рекурсивного автомата
- Выход: матрица достижимости, а также индекс для восстановления всех пути в графе соответствии с входной грамматикой

Экспериментальное исследование

- Рабочая станция: Intel Core i7-6790, 3.40GHz, RAM DDR4 64Gb, GeForce GTX 1070 с 8Gb VRAM, ОС Ubuntu 20.04
- Данные, необходимые для замеров, предварительно загружаются в RAM или VRAM в требуемом формате
- Исследовательские вопросы:
  - **B1:** Какова производительность отдельных операций реализованной библиотеки примитивов разреженной линейной булевой алгебры на GPGPU по сравнению с существующими аналогами?
  - **B2:** Какова производительность реализованного алгоритма поиска путей через тензорное произведение на GPGPU по сравнению с существующими аналогами, также полагающимися на примитивы линейной алгебры?

Набор данных

Матрица M	Кол-во Строк R	Nnz M	Nnz$/R$	Max Nnz$/R$	Nnz M^2	Nnz $M + M^2$
wing	62,032	243,088	3.9	4	714,200	917,178
luxembourg_osm	114,599	239,332	2.0	6	393,261	632,185
amazon0312	400,727	3,200,400	7.9	10	14,390,544	14,968,909
amazon-2008	735,323	5,158,388	7.0	10	25,366,745	26,402,678
web-Google	916,428	5,105,039	5.5	456	29,710,164	30,811,855
roadNet-PA	1,090,920	3,083,796	2.8	9	7,238,920	9,931,528
roadNet-TX	1,393,383	3,843,320	2.7	12	8,903,897	12,264,987
belgium_osm	1,441,295	3,099,940	2.1	10	5,323,073	8,408,599
roadNet-CA	1,971,281	5,533,214	2.8	12	12,908,450	17,743,342
netherlands_osm	2,216,688	4,882,476	2.2	7	8,755,758	13,626,132

Figure: Разреженные матричные данные

Матричное умножение

Матрица M	cuBool		CUSP		cuSPRS		clSPRS		SuiteSprs	
	t	m	t	m	t	m	t	m	t	m
wing	1.9	93	5.2	125	20.1	155	4.2	105	7.9	22
luxembourg_osm	2.4	91	3.7	111	1.7	151	6.9	97	3.1	169
amazon0312	23.2	165	108.5	897	412.8	301	52.2	459	257.6	283
amazon-2008	33.3	225	172.0	1409	184.8	407	77.4	701	369.5	319
web-Google	41.8	241	246.2	1717	4761.3	439	207.5	1085	673.3	318
roadNet-PA	18.1	157	42.1	481	37.5	247	56.6	283	66.6	294
roadNet-TX	22.6	167	53.1	581	46.7	271	70.4	329	80.7	328
belgium_osm	23.2	151	32.9	397	26.7	235	68.2	259	56.9	302
roadNet-CA	32.0	199	74.4	771	65.8	325	98.2	433	114.5	344
netherlands_osm	35.3	191	51.0	585	51.4	291	102.8	361	90.9	311

Figure: Матричное умножение, время (t) в миллисекундах, память (m) в мегабайтах, отклонение в пределах 10%

Поэлементное матричное сложение

Матрица M	cuBool		CUSP		cuSPRS		clSPRS		SuiteSprs	
	t	m	t	m	t	m	t	m	t	m
wing	1.1	95	1.4	105	2.4	163	-	-	2.3	176
luxembourg_osm	1.7	95	1.0	97	0.8	151	-	-	1.6	174
amazon0312	11.4	221	16.2	455	24.3	405	-	-	37.2	297
amazon-2008	17.5	323	29.5	723	27.2	595	-	-	64.8	319
web-Google	24.8	355	31.9	815	89.0	659	-	-	77.2	318
roadNet-PA	16.9	189	11.2	329	11.6	317	-	-	36.6	287
roadNet-TX	19.6	209	14.5	385	16.9	357	-	-	45.3	319
belgium_osm	19.5	179	10.2	303	10.5	297	-	-	28.5	302
roadNet-CA	30.5	259	19.4	513	20.2	447	-	-	65.2	331
netherlands_osm	30.1	233	14.8	423	18.3	385	-	-	50.2	311

Figure: Поэлементное матричное сложение, время (t) в миллисекундах, память (m) в мегабайтах, отклонение в пределах 10%

Заключение

- Спроектирована библиотека примитивов линейной булевой алгебры для работы с разреженными данными на GPGPU
- Реализована библиотека cuBool[2] в соответствии с разработанной архитектурой
- С использованием pycubool реализован алгоритм поиска путей с КС ограничениями через тензорное произведение
- Выполнено экспериментальное исследование полученных артефактов

- Результаты исследования были представлены на конференции GrAPL 2021[3]

---

[2] cuBool: `https://github.com/JetBrains-Research/cuBool`, pycubool: `https://test.pypi.org/project/pycubool/`

[3] GrAPL 2021: Workshop on Graphs, Architectures, Programming, and Learning. Дата обращения: 1.04.2021. Сайт конференции: `https://hpc.pnl.gov/grapl/`.