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

5 июня 2021

Введение

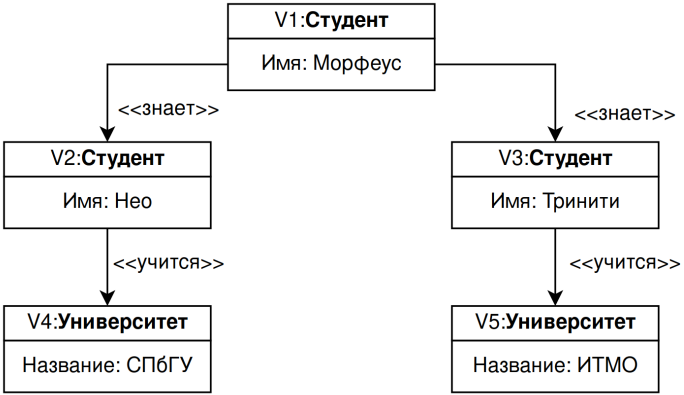

Рисунок: Пример графовой БД

- 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

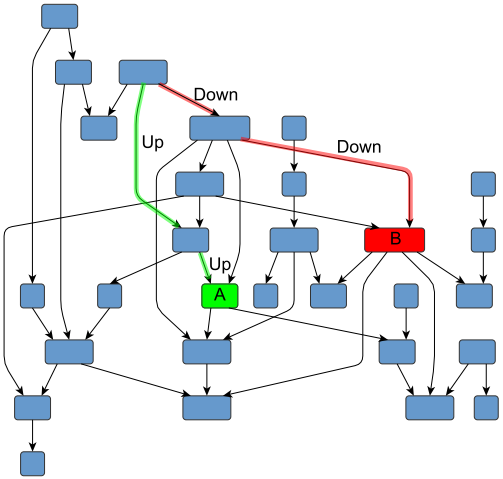

Рисунок: Пример графа

- Мотивация: навигация в графе
- Пример: находятся ли вершины A и B на одном уровне иерархии?
- Применение: анализ RDF данных, биоинформатика, статический анализ кода
- Проблема: отсутствие поддержки КС запросов как в современных СУБД, так и в отдельных инструментах

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. для булевых значений
- Структура матриц: разреженная
- Для эффективной реализации требуется библиотека примитивов разреженной булевой линейной алгебры на GPU

[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

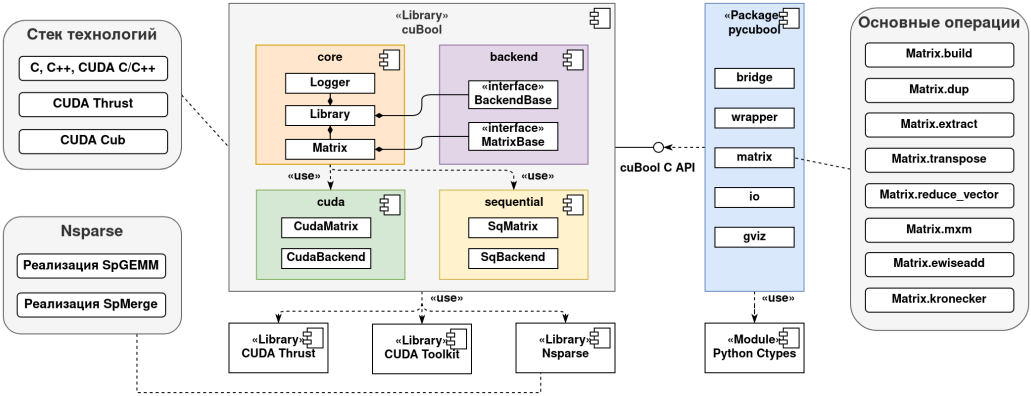

Рисунок: Архитектура разработанной библиотеки

Последовательность обработки операций

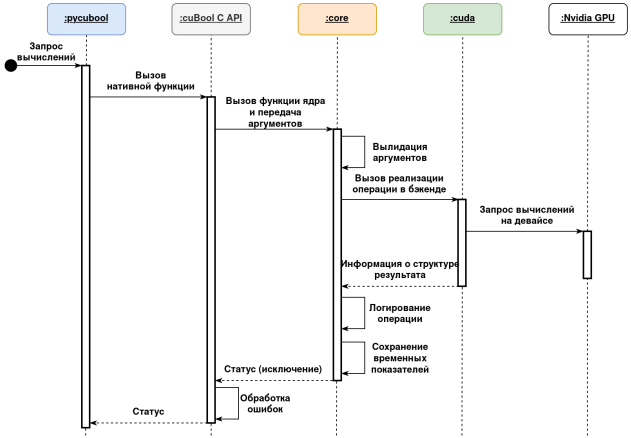

Рисунок: Последовательность выполнения вычислительной матричной операции на Nvidia GPU с использованием pycubool

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;
}
```

Рисунок: 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
```

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

Алгоритм поиска путей с КС ограничениями через тензорное произведение

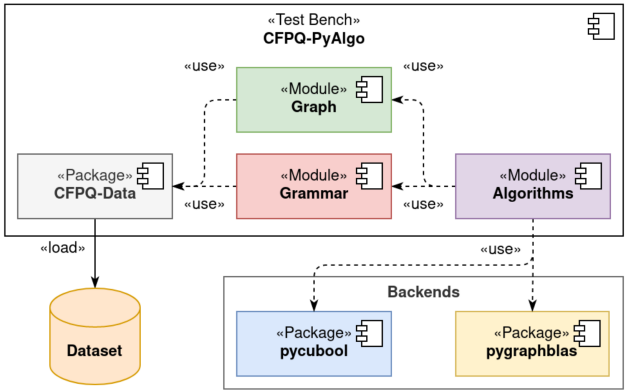

Рисунок: Архитектура CFPQ-PyAlgo стенда, в котором доступна реализация алгоритма

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?

B1: 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

Рисунок: Разреженные матричные данные

B1: Матричное умножение

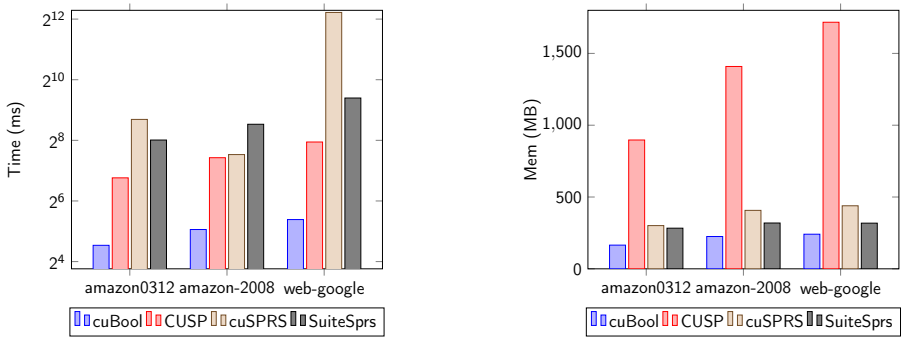

Рисунок: Поэлементное матричное умножение, время (t) в миллисекундах, память (m) в мегабайтах, отклонение в пределах 10%

B1: Поэлементное матричное сложение

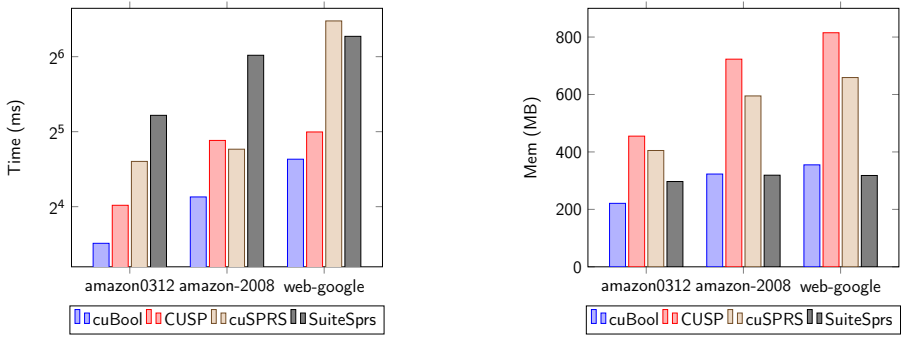

Рисунок: Поэлементное матричное сложение, время (t) в миллисекундах, память (m) в мегабайтах, отклонение в пределах 10%

B2: Набор данных

Граф $\mathcal{G}$	$\lvert V\rvert$	$\lvert E\rvert$	Кол-во sco	Кол-во $type$	Кол-во bt	Кол-во a	Кол-во d
go-hierarchy	45 007	980 218	490 109	0	—	—	—
enzyme	48 815	109 695	8 163	14 989	—	—	—
eclass_514en	239 111	523 727	90 512	72 517	—	—	—
go	272 770	534 311	90 512	58 483	—	—	—
geospecies	450 609	2 201 532	0	89 062	20 867	—	—
taxonomy	5 728 398	14 922 125	2 112 637	2 508 635	—	—	—
arch	3 448 422	5 940 484	—	—	—	671 295	2 298 947
crypto	3 464 970	5 976 774	—	—	—	678 408	2 309 979
drivers	4 273 803	7 415 538	—	—	—	858 568	2 849 201
fs	4 177 416	7 218 746	—	—	—	824 430	2 784 943

Рисунок: RDF графы и графы программ для экспериментов

B2: Запросы с КС ограничениями

$$S \to \overline{subClassOf}\ \ S\ \ subClassOf \mid \overline{type}\ \ S\ \ type \mid \overline{subClassOf}\ \ subClassOf \mid \overline{type}\ \ type \tag{1}$$

$$S \to \overline{subClassOf}\ \ S\ \ subClassOf \mid subClassOf \tag{2}$$

$$S \to broaderTransitive\ \ S\ \ \overline{broaderTransitive} \mid broaderTransitive\ \ \overline{broaderTransitive} \tag{3}$$

$$S \to \overline{d}\ V\ d$$
$$V \to ((S?)\overline{a})^*(S?)(a(S?))^* \tag{4}$$

Рисунок: КС грамматики запросов для анализа. Запросы, для анализа RDF данных: G_1 (1), G_2 (2), G_{geo} (3), для анализа указателей в графах программ — G_{ma} (4).

B2: Анализ RDF данных

Граф $\mathcal{G}$	G_1								G_2							
	Tns_{gpu}		Tns_{cpu}		Mtx_{gpu}		Mtx_{cpu}		Tns_{gpu}		Tns_{cpu}		Mtx_{gpu}		Mtx_{cpu}	
	t	m	t	m	t	m	t	m	t	m	t	m	t	m	t	m
go-hierarchy	0.15	315	0.17	265	0.11	208	0.08	254	0.27	484	0.24	252	0.06	6	0.09	255
enzyme	0.02	9	0.04	137	0.01	<1	0.01	181	0.01	4	0.02	132	0.01	<1	0.01	181
eclass_514en	0.10	57	0.24	205	0.04	14	0.07	180	0.09	36	0.27	193	0.02	2	0.06	181
go	1.67	176	1.58	282	0.43	68	1.00	244	0.82	119	1.27	243	0.18	12	0.94	246
taxonomy	2.21	1266	4.42	2018	0.71	364	1.13	968	0.90	969	3.56	1776	0.24	91	0.72	1175

Рисунок: Анализ RDF данных с использованием запросов G_1 и G_2, время (t) в секундах, память (m) в мегабайтах, отклонение в пределах 10%

B2: Анализ RDF данных

Граф $\mathcal{G}$	G_1								G_2							
	Tns_{gpu}		Tns_{cpu}		Mtx_{gpu}		Mtx_{cpu}		Tns_{gpu}		Tns_{cpu}		Mtx_{gpu}		Mtx_{cpu}	
	t	m	t	m	t	m	t	m	t	m	t	m	t	m	t	m
go-hierarchy	0.15	315	0.17	265	0.11	208	0.08	254	0.27	484	0.24	252	0.06	6	0.09	255
enzyme	0.02	9	0.04	137	0.01	<1	0.01	181	0.01	4	0.02	132	0.01	<1	0.01	181
eclass_514en	0.10	57	0.24	205	0.04	14	0.07	180	0.09	36	0.27	193	0.02	2	0.06	181
go	1.67	176	1.58	282	0.43	68	1.00	244	0.82	119	1.27	243	0.18	12	0.94	246
taxonomy	2.21	1266	4.42	2018	0.71	364	1.13	968	0.90	969	3.56	1776	0.24	91	0.72	1175

Рисунок: Анализ RDF данных с использованием запросов G_1 и G_2, время (t) в секундах, память (m) в мегабайтах, отклонение в пределах 10%

Граф $\mathcal{G}$	Tns_{gpu}		Tns_{cpu}		Mtx_{gpu}		Mtx_{cpu}	
	t	m	t	m	t	m	t	m
geospecies	*err*	*err*	26.32	19537	1.17	5272	7.48	7645

Рисунок: Анализ RDF данных с использованием запроса G_{geo}, время (t) в секундах, память (m) в мегабайтах, отклонение в пределах 10%

B2: Анализ указателей

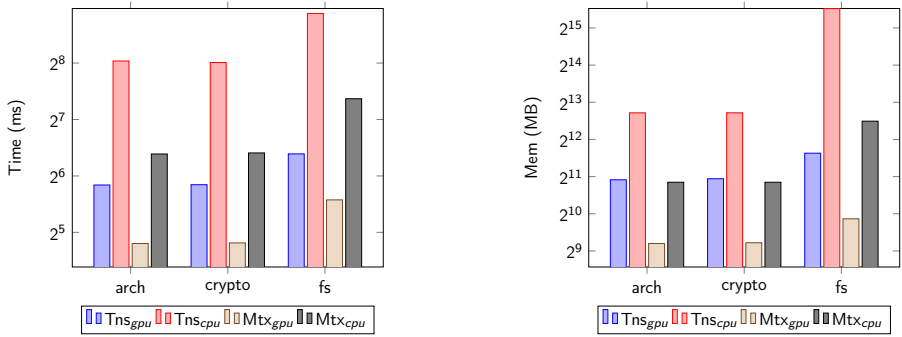

Рисунок: Анализ указателей с использованием запроса G_{ma}, время (t) в секундах, память (m) в мегабайтах, отклонение в пределах 10%

Заключение

- Спроектирована библиотека примитивов линейной булевой алгебры для работы с разреженными данными на GPGPU
- Реализована библиотека cuBool[2] в соответствии с разработанной архитектурой, опубликован соответствующий python-пакет[3]
- 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[4]

[3] GitHub cuBool: `https://github.com/JetBrains-Research/cuBool`
[3] PyPI pycubool: `https://pypi.org/project/pycubool/`
[4] Workshop on Graphs, Architectures, Programming, and Learning: `https://hpc.pnl.gov/grapl/`