Санкт-Петербургский государственный университет

Кафедра системного программирования

Группа 17Б.11-мм

Орачев Егор Станиславович

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

Отчёт по преддипломной практике

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

Санкт-Петербург
2020

Оглавление

Введение

Все чаще современные системы аналитики и рекомендаций строятся на основе анализа данных, структурированных с использованием *графовой модели*. В данной модели основные сущности представляются вершинами графа, а отношения между сущностями — ориентированными ребрами с различными метками. Подобная модель позволяет относительно легко и практически в явном виде моделировать сложные иерархические структуры, которые не так просто представить, например, в классической *реляционной модели*. В качестве основных областей применения графовой модели можно выделить следующие: графовые базы данных [4], анализ RDF данных [6], биоинформатика [23] и статический анализ кода [14].

Поскольку графовая модель используется для моделирования отношений между объектами, при решении прикладных задач возникает необходимость в выявлении более сложных взаимоотношений между объектами. Для этого чаще всего формируются запросы в специализированных программных средствах для управления графовыми базами данных. В качестве запроса можно использовать некоторый *шаблон* на путь в графе, который будет связывать объекты, т.е. выражать взаимосвязь между ними. В качестве такого шаблона можно использовать формальные грамматики, например, регулярные или контекстно-свободные (КС). Используя вычислительно более выразительные грамматики, можно формировать более сложные запросы и выявлять нестандартные и скрытые ранее взаимоотношения между объектами. Например, *same-generation queries* [1], сходные с сбалансированными скобочными последовательностями Дика, могут быть выражены КС грамматиками, в отличие от регулярных.

Результатом запроса может быть множество пар объектов, между которыми существует путь в графе, удовлетворяющий заданным ограничениям. Также может возвращаться один экземпляр такого пути для каждой пары объектов или итератор всех путей, что зависит от семантики запроса. Поскольку один и тот же запрос может иметь разную семантику, требуются различные программные и алгоритмические средства для его выполнения.

Запросы с регулярными ограничениями изучены достаточно хорошо, языковая и программная поддержка выполнения подобных запросов присутствует в некоторых в современных графовых базах данных. Однако, полноценная поддержка запросов с КС ограничениями до сих пор не представлена. Существуют алгоритмы [3,6,7,16,18] для вычисления запросов с КС ограничениями, но потребуется еще время, прежде чем появиться полноценная высокпроизводительная реализация одного из алгоритмов, способная обрабатывать реальные графовые данные.

Работы [8, 12] в качестве реализации алгоритма [3] для выполнения запросов с КС ограничениями с семантикой достижимости и семантикой одного пути показывают, что возможно использовать GPGPU для выполнения наиболее вычислительно сложных частей алгоритма, что дает *существенный* прирост в производительности.

Недавно представленный алгоритм [7] для вычисления запросов с КС ограничениями полагается на операции линейной алгебры: произведение Кронекера (частный случай тензорного произведения), умножение и сложение матриц в полукольце булевой алгебры. Важной задачей является реализация данного алгоритма, так как он в сравнении с [8] позволяет выполнять запросы для всех ранее упомянутых семантик, потенциально поддерживает бо́льшие по размеру КС запросы, с незначительными накладными расходами позволяет выполнять запросы с регулярными ограничениями, а с реализацией на GPGPU позволит получить потенциально приемлемое время выполнения запрсов.

1. Цель и задачи

Целью работы является реализация алгоритма поиска путей в графовых базах данных через тензорное произведение на GPGPU. Для ее выполнения были поставлены следующие задачи:

- Реализация библиотеки для работы с примитивами булевой алгебры на GPGPU
- Реализация интерфейса для работы с примитивами библиотеки в тестовой инфраструктуре
- Реализация алгоритма поиска путей с КС ограничениями
- Апробация алгоритма с использованием синтетических и реальных данных

2. Обзор предметной области

2.1. Терминология

В этой секции изложены основные определения и факты из теории графов и формальных языков, необходимые для понимания предметной области.

Ориентированный граф с метками $\mathcal{G} = \langle V, E, L \rangle$ это тройка объектов, где V конечное непустое множество вершин графа, $E \subseteq V \times L \times V$ конечное множество ребер графа, L конечное множество меток графа. Здесь и далее будем считать, что вершины графа индексируются целыми числами, т.е. $V = \{0 \ldots |V| - 1\}$.

Граф $\mathcal{G} = \langle V, E, L \rangle$ можно представить в виде матрицы смежности M размером $|V| \times |V|$, где $M[i, j] = \{l \mid (i, l, j) \in E\}$. Используя булеву матричную декомпозицию, можно представить матрицу смежности в виде набора матриц $\mathcal{M} = \{M^l \mid l \in L, M^l[i, j] = 1 \iff l \in M[i, l]\}$.

Путь π в графе $\mathcal{G} = \langle V, E, L \rangle$ это последовательность ребер e_0, e_1, e_{n-1}, где $e_i = (v_i, l_i, u_i) \in E$ и для любых $e_i, e_{i+1} : u_i = v_{i+1}$. Путь между вершинами v и u будем обозначать как $v\pi u$. Слово, которое формирует путь $\pi = (v_0, l_0, v_1), \ldots, (v_{n-1}, l_{n-1}, v_n)$ будем обозначать как $\omega(\pi) = l_0 \ldots l_{n-1}$, что является конкатенацией меток вдоль этого пути π.

Контекстно-свободная (КС) грамматика $G = \langle \Sigma, N, P, S \rangle$ это четверка объектов, где Σ конечное множестве терминалов или алфавит, N конечное множество нетерминалов, P конечное множество правил вывода вида $A \to \gamma, \gamma \in (N \cup \Sigma)^*$, $S \in N$ стартовый нетерминал.

Язык L над конечным алфавитом символов Σ — множество всевозможных слов, составленных из символов этого алфавита, т.е. $L = \{\omega \mid w \in \Sigma^*\}$.

2.2. Поиск путей с ограничениями

При вычислении запроса p на поиск путей в графе $\mathcal{G} = \langle V, E, L \rangle$ в качестве ограничения выступает некоторый язык L, которому должны удовлетворять результирующие пути.

Поиск путей в графе с семантикой **достижимости**, это поиск всех таких пар вершин (v, u), что между ними существует путь $v\pi u$ такой, что $\omega(\pi) \in L$. Результат запроса обозначается как $R = \{(v, u) \mid \exists v\pi u : \omega(\pi) \in L\}$.

Поиск путей в графе с семантикой **всех путей**, это поиск всех таких путей $v\pi u$, что $\omega(\pi) \in L$. Результат запроса обозначается как $\Pi = \{v\pi u \mid v\pi u : \omega(\pi) \in L\}$.

Необходимо отметить, что множество Π может быть бесконечным, поэтому в качестве результата запроса предполагается не всё множество в явном виде, а некоторый *итератор*, который позволит последовательно извлекать все пути.

Семантика **одного пути** является ослабленной формулировкой семантики всех путей, так как для получения результата достаточно найти всего один путь вида $v\pi u : \omega(\pi) \in L$ для каждой пары $(v, u) \in R$.

Поскольку язык L может быть бесконечным, при составлении запросов используют не множество L в явном виде, а некоторое правило формирования слов этого языка. В качестве таких правил и выступают регулярные выражения или КС грамматики. При именовании запросов отталкиваются от типа правил, поэтому запросы именуются как регулярные или КС соответственно.

2.3. Существующие решения

Впервые проблема выполнения запросов с контекстно-свободными ограничениями была сформулирована в 1990 году в работе Михалиса Яннакакиса [27]. С того времени были представлены многие работы, в которых так или иначе предлагалось решение данной проблемы. Однако в недавнем исследовании Йохем Куиджперс и др. [13] на основе сравнения нескольких алгоритмов [3, 16, 24] для выполнения запросов с контекстно-свободными ограничениями заключили, что существующие алгоритмы неприменимы для анализа реальных данных в силу того, что обработка таких данных занимает значительное время. Стоит отметить, что алгоритмы, используемые в статье, были реализованы на языке программирования *Java* и исполнялись в среде *JVM* в однопоточном режиме, что не является сколь-угодно производительным решением.

Это подтверждают результаты работы [8], в которой с использование программных и аппаратных средств NVIDIA CUDA был реализован алгоритм Рустама Азимова [3]. В данном алгоритме задача поиска путей с КС ограничениями для семантики одного пути сведена к операциям линейной алгебры, что позволяет использовать высокопроизводительные библиотеки для выполнения данных операций на GPGPU.

2.4. Вычисления на GPGPU

GPGPU (от англ. General-purpose computing on graphics processing units) — техника использования графического процессора видеокарты компьютера для осуществления неспециализированных вычислений, которые обычно проводит центральный процессор. Данная техника позволяет получить значительной прирост производительности, когда необходимо обрабатывать большие массивы данных с фиксированным набором команд по принципу *SIMD*.

Исторически видеокарты в первую очередь использовались как графические ускорители для создания высококачественной трехмерной графики в режиме реального времени. Однако, позже стало ясно, что мощность графического процессора можно использовать не только для графических вычислений. Так появились программируемые вычислительные блоки (англ. compute shaders), которые позволяют выполнять на видеокарте неграфические вычисления.

На данный момент существует несколько промышленных стандартов программирования вычислений на видеокарте, одними из которых являются Vulkan [26], OpenGL [25], Direct3D [10] как API для преимущественно графических задач, а также OpenCL [22], NVIDIA CUDA [19] как API для неспециализированных вычислений.

3. Поиск путей с КС ограничениями через произведение Кронекера

Недавно представленный алгоритм [7] для выполнения КС запросов использует подобную технику сведения вычислений к операциям булевой алгебры: произведению Кронекера, матричному умножению и сложению. Однако структура алгоритма такова, что он позволяет выполнять запросы сразу в семантике достижимости и семантике всех путей, способен работать с КС грамматиками существенно большего размера, также имеет относительно небольшие накладные расходы при вычислении запросов с регулярными ограничениями, что делает его потенциально применимым для решения этого класса проблем.

3.1. Рекурсивный автомат

Для представления входной грамматики КС запроса алгоритм [7] использует *рекурсивный автомат*. Данный формализм является своего рода обобщением *детерминированного конечного автомата* на случай КС языков. Для понимания того, как он устроен, обратимся к теории формальных языков.

Конечный автомат (КА) $F = \langle \Sigma, Q, Q_s, Q_f, \delta \rangle$ это пятерка объектов, где Σ конечное множество входных символов или алфавит, Q конечное множество состояний, $Q_s \subseteq Q$ множество стартовых состояний, $Q_f \subseteq Q$ множество конечных состояний, $\delta : \Sigma \times Q \to 2^Q$ функция переходов автомата. Язык, допускаемый автоматом F будем обозначать как $L(F)$. Любое регулярное выражение может быть преобразовано в соответствующий КА [17].

Рекурсивный автомат (РА) $R = \langle M, m, \{C_i\}_{i \in M} \rangle$ это тройка объектов, где M конечное множество меток компонентных КА, называемых далее *модули*, m метка стартового модуля, $\{C_i\}$ множество модулей, где модуль $C_i = \langle \Sigma \cup M, Q_i, S_i, F_i, \delta_i \rangle$: состоит из:

- $\Sigma \cup M$ множество символов модуля, $\Sigma \cap M = \emptyset$

- Q_i конечное множество состояний модуля, $Q_i \cap Q_j = \emptyset, \forall i \neq j$
- $S_i \subseteq Q_i$ множество стартовых состояний модуля
- $F_i \subseteq Q_i$ множество конечных состояний модуля
- $\delta_i : (\Sigma \cup M) \times Q_i \to 2^{Q_i}$ функция переходов

Рекурсивный автомат ведет себя как набор КА или модулей [2]. Эти модули очень сходны с КА при обработке входных последовательностей символов, однако они способны обрабатывать дополнительные *рекурсивные вызовы* за счет неявного *стека вызовов*, который присутствует во время работы РА. С точки зрения прикладного программиста это похоже на рекурсивные вызовы одних функций из других с той разницей, что вместо функций здесь выступают модули РА.

Рекурсивные автоматы по своей вычислительной мощности эквивалентны автоматам на основе стека [2]. А поскольку подобный стековый автомат способен распознавать КС грамматику [17], рекурсивные автоматы эквивалентны КС грамматикам. Это позволяет корректно использовать РА для представления входной КС грамматики запроса.

3.2. Пересечение рекурсивного автомата и графа

Классический алгоритм [17] *пересечения* двух КА F^1 и F^2 позволяет построить новый КА F с таким свойством, что он допускает пересечение исходных регулярных языков, т.е. $L(F) = L(F^1) \cap L(F^2)$. Формально, для $F^1 = \langle \Sigma, Q^1, Q^1_S, Q^1_F, \delta^1 \rangle$ и $F^2 = \langle \Sigma, Q^2, Q^2_S, Q^2_F, \delta^2 \rangle$ строится новый КА $F = \langle \Sigma, Q, Q_S, Q_F, \delta \rangle$, где $Q = Q^1 \times Q^2$, $Q_S = Q^1_S \times Q^2_S$, $Q_F = Q^1_F \times Q^2_F$, $\delta : \Sigma \times Q \to Q$ и $\delta(s, \langle q_1, q_2 \rangle) = \langle q'_1, q'_2 \rangle$, если $\delta^1(s, q_1) = q'_1$ и $\delta^2(s, q_2) = q'_2$.

Интерпретируя ориентированный граф с метками как некоторый конечный автомат, в котором все вершины графа являются одновременно начальными и конечными состояниями автомата, а ребра графа — переходами между состояниями автомата, возможно пересечь этот граф и некоторый КА, используя алгоритм пересечения, описанный выше.

Однако, если представить граф и функцию переходов КА, тоже интерпретируемую как граф, в виде матриц смежности, можно использовать *произведение Кронекера* для построения функции переходов автомата пересечения.

Произведение Кронекера для двух матриц A и B размера $m_1 \times n_1$ и $m_2 \times n_2$ с поэлементной операцией умножения $\cdot$ дает матрицу $M = A \otimes B$ размером $m_1 * m_2 \times n_1 * n_2$, где $M[u * m_2 + v, p * n_2 + q] = A[u, p] \cdot B[v, q]$.

Поскольку РА состоит из набора модулей, которые по своей структуре не сильно отличаются от классических КА, это дает идею для применения похожего алгоритма пересечения РА и графа, с той разницей, что процесс пересечения будет итеративным и будет включать в себя транзитивное замыкание, чтобы учесть *рекурсивные вызовы*, присутствующие в РА.

3.3. Описание алгоритма

В листинге 1 представлен псевдокод алгоритма [7]. Необходимо отметить, что алгоритм использует булеву матричную декомпозицию в строках **3** $-$ **4** для представления матрицы переходов РА и матрицы смежности графа, а также использует матричное умножение, сложение и произведение Кронекера в строках **14** $-$ **16**.

Данный алгоритм является относительно простым и компактным, так как всю сложность выполнения он перекладывает на операции линейной алгебры, которые должны быть реализованы в сторонних высокопроизводительных библиотеках.

Listing 1 Поиск путей через произведение Кронекера

function KroneckerProductBasedCFPQ(G, $\mathcal{G}$)
  $R \leftarrow$ Рекурсивный автомат для грамматики G
  $\mathcal{M}_1 \leftarrow$ Матрица переходов R в булевой форме
  $\mathcal{M}_2 \leftarrow$ Матрица смежности $\mathcal{G}$ в булевой форме
  $C_3 \leftarrow$ Пустая матрица
  for $s \in \{0, ..., dim(\mathcal{M}_1) - 1\}$ **do**
    for $S \in getNonterminals(R, s, s)$ **do**
      for $i \in \{0, ..., dim(\mathcal{M}_2) - 1\}$ **do**
        $M_2^S[i, i] \leftarrow \{1\}$
      end for
    end for
  end for
  while Матрица смежности $\mathcal{M}_2$ изменяется **do**
    $\mathcal{M}_3 \leftarrow \mathcal{M}_1 \otimes \mathcal{M}_2$ ▷ Вычисление произведения Кронекера
    $M_3' \leftarrow \bigvee_{M_3^a \in \mathcal{M}_3} M_3^a$ ▷ Слияние матриц в одну булеву матрицу достижимости
    $C_3 \leftarrow transitiveClosure(M_3')$ ▷ Транзитивное замыкание для учета рекурсивных вызовов
    $n \times n \leftarrow \dim(M_3)$
    for $(i, j) \in \{0, ..., n - 1\} \times \{0, ..., n - 1\}$ **do**
      if $C_3[i, j]$ **then**
        $s, f \leftarrow getStates(C_3, i, j)$
        $x, y \leftarrow getCoordinates(C_3, i, j)$
        for $S \in getNonterminals(R, s, f)$ **do**
          $M_2^S[x, y] \leftarrow \{1\}$
        end for
      end if
    end for
  end while
  return $\mathcal{M}_2, C_3$
end function

4. Разработка библиотеки матричных операций

4.1. Мотивация

Для эффективной реализации алгоритмов [3, 7] требуются высокопроизводительные библиотеки операций линейной алгебры. В качестве такой библиотеки для выполнения матричных операций на ЦПУ нередко используются *SuiteSparse* [11], как эталонная реализация стандарта *GraphBLAS* [15], который был разработан как некоторый инструмент для реализации алгоритмов обработки графов на языке линейной алгебры.

Экспериментальное исследование [12] по реализации алгоритма Рустама Азимова [3] на GPGPU с использованием операций над плотными булевыми матрицами показало, что вычисление на графическом процессоре дает значительный прирост производительности при обработке синтетических данных и данных среднего размера. Однако реальные графовые данные насчитывают порядка 10^7 – 10^9 вершин и являются сильно разреженными, т.е. количество ребер в графе сравнимо с количеством вершин, поэтому плотные матрицы не подходят для обработки такого типа данных.

Естественным решением было бы использование библиотеки линейной алгебры над разреженными матрицами, которая использовала бы для своих вычислений графический процессор системы. Библиотеки *NVIDIA cuSPARSE* [20] и *CUSP* [5] для платформы NVIDIA CUDA предоставляют подобную функциональность, однако они имеют специализацию только для стандартных типов данных, таких как *float*, *double*, *int* и *long*. Для реализации алгоритмов [3, 7] требуются операции над разреженными булевыми матрицами, поэтому требуется специализация вышеуказанных библиотек для работы с булевыми примитивами. С одной стороны, библиотека *cuSPARSE* имеет закрытый исходный код, что делает невозможным ее модификацию, с другой стороны, библиотека *CUSP* имеет открытый исходный код и свободную лицензию, однако используемый ею алгоритм умножения разреженных матриц *слишком* требователен к ресурсам памяти, что делает его неприменимым для обработки графовых данных большого размера.

В работе [8] была предпринята попытка реализовать с нуля алгоритм [21] умножения разреженных матриц и специализировать его для конкретно булевых значений. Данный алгоритм эксплуатирует возможности видеокарт от NVIDIA и за счет бóльшего времени на обработку позволяет снизить количество расходуемой видеопамяти. Эксперименты показали, что подобный подход позволяет не только снизить в разы количество расходуемой видеопамяти, но и снизить общее время работы алгоритма [3] по сравнению с его реализацией на *CUSP*.

4.2. Примитивы линейной алгебры

4.3. Описание реализации

5. Текущий прогресс

Прогресс в работе на данный момент:

- Выбран технологический стек для реализации библиотеки матричных примитивов: C/C++ для реализации интерфейса библиотеки и ее функциональности, CMake для сборки проекта, NVIDIA CUDA Toolkit 10 для написания кода, исполняемого на CUDA-совместимой видеокарте, NVIDIA Thrust для автоматизации работы с неуправляемыми ресурсами GPU.

- Создан репозиторий проекта [9], настроена автоматическая сборка с использованием инструментария *Github Actions*. Добавлено описание проекта и инструкция для сборки.

- Создан C совместимый интерфейс для работы с примитивами библиотеки, а также добавлена непосредственно реализация интерфейса: создание и удаление матриц, запись и чтение значений матрицы, операции умножения, сложения, произведение Кронекера.

- Добавлен набор *unit*-тестов для проверки корректности работы операций на основе сравнения с эталонной реализации тестируемых операций на ЦПУ.

Список литературы

[1] Abiteboul Serge, Hull Richard, Vianu Victor. Foundations of Databases. — 1995. — 01. — ISBN: 0-201-53771-0.

[2] Analysis of Recursive State Machines / Rajeev Alur, Michael Benedikt, Kousha Etessami et al. // ACM Trans. Program. Lang. Syst. — 2005. — Jul. — Vol. 27, no. 4. — P. 786–818. — Access mode: https://doi.org/10.1145/1075382.1075387.

[3] Azimov Rustam, Grigorev Semyon. Context-free path querying by matrix multiplication. — 2018. — 06. — P. 1–10.

[4] Barceló Baeza Pablo. Querying Graph Databases // Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems. — PODS '13. — New York, NY, USA : Association for Computing Machinery, 2013. — P. 175–188. — Access mode: https://doi.org/10.1145/2463664.2465216.

[5] CUSP: A C++ Templated Sparse Matrix Library // Github. — 2020. — Access mode: https://github.com/cusplibrary/cusplibrary (online; accessed: 09.12.2020).

[6] Context-Free Path Queries on RDF Graphs / Xiaowang Zhang, Zhiyong Feng, Xin Wang et al. // CoRR. — 2015. — Vol. abs/1506.00743. — 1506.00743.

[7] Context-Free Path Querying by Kronecker Product / Egor Orachev, Ilya Epelbaum, Rustam Azimov, Semyon Grigorev. — 2020. — 08. — P. 49–59. — ISBN: 978-3-030-54831-5.

[8] Context-Free Path Querying with Single-Path Semantics by Matrix Multiplication / Arseniy Terekhov, Artyom Khoroshev, Rustam Azimov, Semyon Grigorev. — 2020. — 06. — P. 1–12.

[9] CuBool: Linear Boolean algebra primitives written in NVIDIA

CUDA // Github. — 2020. — Access mode: https://github.com/JetBrains-Research/cuBool (online; accessed: 08.12.2020).

[10] Direct3D 12 Graphics // Microsoft Online Documents. — 2018. — Access mode: https://docs.microsoft.com/ru-ru/windows/win32/direct3d12/direct3d-12-graphics?redirectedfrom=MSDN (online; accessed: 08.12.2020).

[11] Dr. Timothy Alden Davis. SuiteSparse: a suite of sparse matrix software // SuiteSparse website. — 2020. — Access mode: https://people.engr.tamu.edu/davis/suitesparse.html (online; accessed: 08.12.2020).

[12] Evaluation of the Context-Free Path Querying Algorithm Based on Matrix Multiplication / Nikita Mishin, Iaroslav Sokolov, Egor Spirin et al. — 2019. — 06. — P. 1–5.

[13] An Experimental Study of Context-Free Path Query Evaluation Methods / Jochem Kuijpers, George Fletcher, Nikolay Yakovets, Tobias Lindaaker // Proceedings of the 31st International Conference on Scientific and Statistical Database Management. — SSDBM '19. — New York, NY, USA : ACM, 2019. — P. 121–132. — Access mode: http://doi.acm.org/10.1145/3335783.3335791.

[14] Fast Algorithms for Dyck-CFL-Reachability with Applications to Alias Analysis / Qirun Zhang, Michael R. Lyu, Hao Yuan, Zhendong Su // SIGPLAN Not. — 2013. — Jun. — Vol. 48, no. 6. — P. 435–446. — Access mode: https://doi.org/10.1145/2499370.2462159.

[15] GraphBLAS Graph Linear Algebra API // graphblas. — 2020. — Access mode: https://graphblas.github.io/ (online; accessed: 09.12.2020).

[16] Hellings Jelle. Path Results for Context-free Grammar Queries on Graphs. — 2015. — 02.

[17] Hopcroft John E., Motwani Rajeev, Ullman Jeffrey D. Introduction to Automata Theory, Languages, and Computation (3rd Edition). — USA : Addison-Wesley Longman Publishing Co., Inc., 2006. — ISBN: 0321455363.

[18] Medeiros Ciro, Musicante Martin, Costa Umberto. An Algorithm for Context-Free Path Queries over Graph Databases. — 2020. — 04.

[19] NVIDIA. CUDA Toolkit Documentation // NVIDIA Developer Zone. — 2020. — Access mode: https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html (online; accessed: 01.12.2020).

[20] NVIDIA. cuSPARSE reference guide // NVIDIA Developer Zone. — 2020. — Access mode: https://docs.nvidia.com/cuda/cusparse/index.html (online; accessed: 09.12.2020).

[21] Nagasaka Yusuke, Nukada Akira, Matsuoka Satoshi. High-Performance and Memory-Saving Sparse General Matrix-Matrix Multiplication for NVIDIA Pascal GPU. — 2017. — 08. — P. 101–110.

[22] OpenCL: Open Standard for Parallel Programming of Heterogeneous Systems // Khronos website. — 2020. — Access mode: https://www.khronos.org/opencl/ (online; accessed: 08.12.2020).

[23] Quantifying variances in comparative RNA secondary structure prediction / James Anderson, Adám Novák, Zsuzsanna Sükösd et al. // BMC bioinformatics. — 2013. — 05. — Vol. 14. — P. 149.

[24] Santos Fred, Costa Umberto, Musicante Martin. A Bottom-Up Algorithm for Answering Context-Free Path Queries in Graph Databases. — 2018. — 01. — P. 225–233. — ISBN: 978-3-319-91661-3.

[25] The Khronos Working Group. OpenGL 4.4 Specification // Khronos Registry. — 2014. — Access mode: https://www.khronos.org/registry/OpenGL/specs/gl/glspec44.core.pdf (online; accessed: 08.12.2020).

[26] The Khronos Working Group. Vulkan 1.1 API Specification // Khronos Registry. — 2019. — Access mode: https://www.khronos.org/registry/vulkan/specs/1.1/html/vkspec.html (online; accessed: 08.12.2020).

[27] Yannakakis Mihalis. Graph-Theoretic Methods in Database Theory // Proceedings of the Ninth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems. — PODS '90. — New York, NY, USA : Association for Computing Machinery, 1990. — P. 230–242. — Access mode: https://doi.org/10.1145/298514.298576.