Санкт-Петербургский государственный университет

Кафедра системного программирования

Группа 21.М07-мм

Орачев Егор Станиславович

Разработка библиотеки обобщенной разреженной линейной алгебры для вычислений в multi-GPU инфраструктуре

Отчёт по учебной практике

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

Санкт-Петербург
2022

Saint Petersburg State University

Egor Orachev

Master's Thesis

Generalized sparse linear algebra framework for multi-GPU computations

Education level: master

Speciality *09.04.04 «Software Engineering»*

Programme *CB.5666.2021 «Software Engineering»*

Scientific supervisor:
C.Sc., docent S.V. Grigorev

Reviewer:

Saint Petersburg
2022

Оглавление

Введение

Данные, структурированные с использованием графовой модели, все чаще естественным образом появляются в реальных практических задачах, таких как вычисление запросов к базам данных [2], графовые базы данных [19], анализ социальных сетей [15] анализ RDF данных [3], биоинформатика [18] и статический анализ кода [10].

В графовой модели основные сущности представляются вершинами графа, а отношения между сущностями — ориентированными ребрами с различными метками. Подобная модель позволяет относительно легко моделировать предметную область, сохраняя в явном виде множество сложных отношений между объектами, что не так просто выполнить, например, в классической *реляционной модели*.

Достаточно больше количество реальных графовых данных имеет разреженную структуру, когда количество ребер сравнимо с количество вершин. Поэтому для решения прикладных задач наибольший интерес представляют инструменты, которые позволяю эффективно представлять в памяти такие данные. А поскольку граф может насчитывать десятки и сотни миллионов ребер [15], инструменты обработки графов должны предоставлять возможности для параллельной обработки с использованием нескольких вычислительных устройств или узлов.

За последние десятилетия исследовательское сообщество посвятило множество работ разработке различных инструментов для эффективного анализа реальных графовых данных. Можно отметить такие проекты как Gunrock [11], Ligra [21], GraphBLAST [26], SuiteSparse [7] и многие другие. Существующие инструменты имеют различные стратегии распараллеливания вычислений, используют для ускорения операций графический процессор, ускоряют вычисления с использованием нескольких потоков центрального процессора.

Как показывают различные исследования [26,27], использование графического процессора для ускорения вычислений является одним из перспективных направлений для получения высокопроизводительного анализа данных. Однако реализация прикладных алгоритмов и, в том

числе, библиотек для анализа графов является очень трудоемкой задачей. Далее выделы основные причины, объясняющие эту проблему.

- Программные интерфейсы для работы с GPU, такие как Nvidia CUDA и OpenCL, имеют низкоуровневую природу — они многословны, требуют множества вспомогательной работы для осуществления простых операций, что делает их недоступными для рядового программиста.

- Многие алгоритмы на графах имеют сходную структуру, однако в деталях сильно отличаются, что требует точечной и аккуратной реализации каждого алгоритма с применением *локальных* оптимизаций, что делает их неприменимыми для других алгоритмов.

- При использовании одного или нескольких GPU устройств, необходимо равномерно распределить работу между всеми вычислительными блоками, что проблематично в силу непредсказуемости нерегулярных шаблонов обращения к данным, используемых в графовых алгоритмах.

- Графовые алгоритмы в основном требуют множество обращений к неструктурированным данным в памяти, когда количество непосредственных вычислений сохраняется низким. Поэтому необходимо использовать специализированные структуры хранения данных.

Чтобы ответить на множество возникающих вопросов, исследовательское сообщество предложило перспективную концепцию использования аппарата разреженной линейной алгебры для решения прикладных задач в виде стандарта GraphBLAS [12], который предоставляет C API и позволяет выражать алгоритмы на графах в терминах операций над матрицами и векторами. Эффективная реализация только этих примитивов и операций позволяет получить готовую для вычислений реализацию алгоритма, не вдаваясь в делали низкоуровневого программирования.

Было представлено множество реализаций стандарта GraphBLAS, но полноценной его реализаций для вычислений на широком классе графических устройств в multi-GPU инфраструктуре на текущий момент не существует. Поэтому важной задачей является разработка библиотеки примитивов и операций разреженной линейной алгебры, предоставляющей возможность обобщения операций на произвольные пользовательские типы и операции, а также осуществляющей автоматическое распараллеливание вычислительных операций в multi-GPU инфраструктуре.

1. Цель и задачи

Целью данной работы является реализация библиотеки примитивов и операций обобщенной разреженной линейной алгебры для вычислений в multi-GPU инфраструктуре. Для ее выполнения были поставлены следующие задачи.

- Обзор существующих библиотек для анализа графов в терминах линейной алгебры.
- Обзор существующих инструментов и технологий для разработки программ для выполнения на GPU-устройствах.
- Разработка архитектуры библиотеки примитивов разреженной линейной алгебры.
- Реализация библиотеки в соответствии с разработанной архитектурой.
- Реализация набора основных алгоритмов с использованием библиотеки.
- Экспериментальное исследование библиотеки и реализаций алгоритмов.

2. Обзор предметной области

Для разработки библиотеки необходимо сперва рассмотреть базовую теорию, а также ознакомиться с существующими подходами к реализации. Для этого предлагается ознакомиться с концепциями, предлагаемыми GraphBLAS, а также рассмотреть существующие инструменты для работы с примитивами разреженной линейной алгебры на GPU, а также те проблемы, которые возникают при программировании подобных решений в GPU-среде. Это поможет обосновать необходимость разработки нового инструмента.

2.1. Концепции GraphBLAS

Стандарт GraphBLAS [12] представляет математическую нотацию, транслированную в некоторый C API. Данный стандарт оперирует концепциями линейной (разреженной) алгебры. Основные элементы данного стандарта изложены ниже.

- **Примитивы или контейнеры.** Основными контейнерами для хранения данных являются *матрица*, *вектор*, *скаляр* и *маска*. Контейнеры параметризуются типом элементов, которые они хранят. Имеется возможность создания пользовательских типов данных.

- **Алгебраические структуры.** В качестве основных алгебраических структур используются *полукольцо* и *моноид*. Данные структуры адаптированы для разреженных данных, поэтому они отличаются от тех, которые приняты в алгебре. Данные структуры определяют поэлементные функции, которые оперируют данными в контейнерах. Например, эти функции используются в качестве параметров *mult* и *add* при вычислении матричного произведения, где элементы строки и столбца сначала умножаются, а затем некоторым образом суммируются в конечный элемент.

- **Операции.** Основными операциями являются произведения матриц и векторов, поэлементные операции, транспонирование, операции свертки, применение масок для фильтрации элементов, а также операции для манипулирования данными.

2.2. Существующие инструменты

Существующие инструменты для анализа графов в терминах разреженной линейной алгебры можно разделить на две основные категории: специализированные библиотеки и библиотеки примитивов разреженной линейно алгебры (без фокуса на анализ графовых данных). Существующие инструменты используют разные типы вычислителей, включая один или несколько центральных процессоров, или GPU-ускоритель. Далее представленные наиболее популярные и влиятельные из них.

SuiteSparse

Библиотека SuiteSparse [7] является полной и эталонной реализацией стандарта GpraphBLAS для вычислений на центральном процессоре. Она доступна для использования через упоминаемый ранее GraphBLAS C API для программирования в языковой среде C или C++, а также имеет ряд открытых и поддерживаемых сообществом пакетов, для использования функций библиотеки в других языковых средах, таких как Python через pygraphblas [28].

В проекте ведется активная работа по поддержке технологии Nvidia Cuda для осуществления вычислений на Nvidia GPU-устройстве, однако информации о поддержке нескольких GPU-устройств в одном вычислителе нет. Кроме этого, также остается открытым вопрос о возможности создания собственных пользовательских типов и функций для вычислений на GPU, поскольку Nvidia Cuda API требует предкомпиляции кода, что на данный момент не предусмотрено в стандарте GraphBLAS.

GraphBLAST

Библиотека GraphBLAST [26] предоставляет примтивы и операции разреженной линейной алгебры для вычислений на GPU-устройстве с использованием технологии Nvidia Cuda. Данная библиотека использует схожие с GraphBLAS концепции, однако она имеет C++ интерфейс и использует обобщенное программирование с использованием Cuda C++ шаблонов для обеспечения возможности создания произвольных пользовательских типов и операций.

Использование подобного API оправдано, так как это упрощает написание прикладных алгоритмов и позволяет использовать компилятор для генерации требуемого вспомогательного кода. Однако существенным недостатком такого подхода является то, что весь код содержится в *заголовочных файлах*, что требует от пользователя постоянное наличие Nvidia NVCC компилятора и полную перекомпиляцию приложения при любых модификациях.

На данный момент проект находится на стадии разработки, и, несмотря на презентацию на тематической конференции, часть функциональности проекта еще недоступна.

Cusp

Библиотека cusp [6] предоставляет набор примитивов разреженной линейной алгебры и основных операций для вычислений на центральном процессоре (в однопоточном или параллельном режиме) и на Nvidia GPU-устройстве. Библиотека имеет C++ интерфейс на основе шаблонов, использует обобщенное программирование для параметризации операций. В своей основе она использует библиотеку примитивов для параллельного программирования Nvidia Thrust [13], которая предоставляет реализацию для таких операций, как *sort*, *reduce*, *gather*, *scan* и т.д.

Cusp имеет сходный с GraphBLAST интерфейс, однако данная библиотека была спроектирована для произвольных вычислений с использованием линейной алгебры без акцента на графовых данных. Поэтому

она не поддерживает ряд важных операций, таких как применение маски или редуцирование значений.

cuBool и SPbLA

Библиотеки cuBool и SPbLA [20] являются попыткой реализации примитивов и операций из стандарта GraphBLAS, но только для булевых значений. Существует множество алгоритмов, которые можно выразить с использованием операций булевой разреженной линейной алгебры, такие как достижимость в графе с регулярными или контекстно-свободными ограничениями [1, 4, 5, 9].

Библиотеки используют Nvidia CUDA и OpenCL API для вычислений на GPU. Кроме этого, библиотека cuBool имеет пакет pycubool [17] для работы в среде Python. Эти библиотеки были разработаны входе подобного исследования в 2021 году. Они могли бы быть взяты как основа в данной работе, однако архитектура решений не позволяет их расширить для использования произвольных типов и пользовательских функций.

Общие недостатки

Далее выделены основные и наиболее важные недостатки представленных решений.

- **Императивный интерфейс.** Стандарт GraphBLAS и многие сходные библиотеки имеют императивный интерфейс, где пользователь явно вызывает одну операцию за другой, используя соответствующие функции. Это существенно усложняет реализацию подобного интерфейса, а также вносит ряд ограничений на оптимизацию и распараллеливание, т.к. библиотека не обладает достаточной информацией о зависимостях между данными, а также о точках синхронизации.

- **Использование шаблонов.** Ряд библиотек, которые реализуют GraphBLAS в виде C++ API, используют шаблоны языка для

обобщенного программирования примитивов и операций. Это снижает количество вспомогательного кода, однако делает невозможным единовременную компиляцию такой библиотеки и ее распространение для конечных пользователей без требования локальной компиляции.

- **Использование Nvidia CUDA.** Библиотеки, использующие GPU-устройство для ускорения вычислений, полагаются чаще всего на Nvidia Cuda C++ API, так как оно имеет удобный механизм шаблонов. Однако Cuda технология поддерживается только на графических картах компании Nvidia, что существенно снижает количество потенциальных компьютеров для вычислений.

2.3. Вычисления на GPU-устройстве

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 значительный прирост производительности, когда необходимо обрабатывать большие массивы однородных данных фиксированным набором команд.

Термин *multi-GPU* (от англ. multiple GPUs) используется для обозначения вычислительной инфраструктуры, в которой доступно несколько GPU-устройств. Приложения, созданные для подобной среды, автоматически осуществляют распределение вычислений по всем доступным устройствам.

Существует несколько промышленных стандартов для создания программ, использующих графический процессор, одними из которых являются Vulkan [25], OpenGL [24], DirectX [8] как API для графических и неспециализированных вычислительных задач, а также OpenCL [16], Nvidia Cuda [14] как API для неспециализированных вычислений.

В существующих графовых инструментах основными технологиями для распараллеливания вычислений на графических устройствах явля-

ются OpenCL и Nvidia Cuda. Далее представлен краткий обзор каждой из технологий.

- **Nvida Cuda API** является проприетарной технологией компании Nvidia и доступен только на графических устройствах этой компании. Данный API имеет языковую поддержку как C так C++ возможностей, позволяет использовать шаблоны для обобщенного программирования, что упрощает разработку множества алгоритмов, таких как *sort*, *scan*, *reduce*, которые параметризуются типами элементов и операциями. Кроме того, компания Nvidia поставляет множество инструментов для отладки и профилирования Cuda-кода. Также Nvidia Cuda имеет средства для multi-GPU программирования.

- **OpenCL API** это открытый стандарт для создания программ, использующих различного типа ускорители для распараллеливания вычислений. Данный стандарт имеет поддержку на множестве платформ, включая Intel, Nvidia, AMD, Apple M1, что делает его применимым на широком классе устройств. Данный API спроектирован в виде C-интерфейса, он не имеет встроенной поддержки для осуществления обобщенного программирования, как в Cuda C++. Также OpenCL имеет средства для multi-GPU программирования. Так как данный стандарт является открытым, на разных платформах качество его поддержки, а также актуальная версия отличаются, что делает проблематичным разработку и отладку OpenCL-приложений.

В данной работе в качестве технологии для программирования GPU-вычислений используется OpenCL. Данный API выбран, поскольку его требуемая версия 1.2 для проекта имеет поддержку на всех актуальных устройствах. Также данный API позволяет динамически во время исполнения компилировать код для выполнения на GPU, что делает его удобным инструментов для создания *обобщенной* библиотеки, где пользователь сможет реализовывать свои примитивные типы и операции в виде набора текстовых строк.

3. Архитектура библиотеки

В данной секции предлагается рассмотреть архитектуру разрабатываемой библиотеки, ее основные компоненты и модули, а также последовательность обработки операций на GPU. Библиотека спроектирована с учетом основных аспектов, связанных с осуществлением вычислений на GPU, рассмотренных в предыдущей главе.

Структура библиотеки и ее конченная функциональность определяется следующими высокоуровневыми требованиями, которые продиктованы конечными вычислительными задачами и особенностями реализации существующих решений.

- **Использование OpenCL API.** Библиотека должна использовать OpenCL API для ускорения вычислений на одном или нескольких OpenCL-совместимых GPU-устройств в системе. Данный API поддерживается на множестве графических ускорителей, а также позволяет динамически во время исполнения компилировать GPU-код, что избавляет пользователя от установки и использования стороннего специализированного компилятора.

- **Пользовательские типы данных.** Библиотека предоставляет возможность для создания произвольных пользовательских типов. Тип представляется уникальным именем и размером элементов в байтах. Элементы типа являются POD-структурами, обрабатываются как обычные последовательности байтов.

- **Пользовательские функции.** Библиотека предоставляет возможность создания пользовательских функций, которые могут принимать на вход пользовательские (произвольные) типы, и которые могут быть использованы чтобы параметризовать математические операции. Функции определяются в виде набора сток с OpenCL-кодом, что позволяет создавать функции без использования сторонних инструментов для компиляции пользовательского кода.

- **Граф выражений.** Пользователь определяет задачи для вычислений в виде ориентированного ациклического графа выражений, используя интерфейс библиотеки для создания узлов графа и зависимостей между узлами в декларативном стиле. Затем пользователь передает весь граф библиотеке для выполнения, и может в блокирующем и неблокирующем режиме получить результат.

- **Автоматическое распределение вычислений.** Библиотека автоматически выполняет разбиение графа выражений на множество задач и подзадач, которые упорядочиваются в соответствии с зависимостями и распределяются между всеми доступными GPU-устройствами вычислителя. Данная работа скрыта от пользователя и не требует от него действий.

- **Автоматическое распределение данных**. Библиотека автоматически выполняет форматирование данных, хранит их в гибридном формате и автоматически распределяет между вычислительными узлами. Данная работа, формат хранения, детали распределения скрыты от пользователя и не требует от него действий.

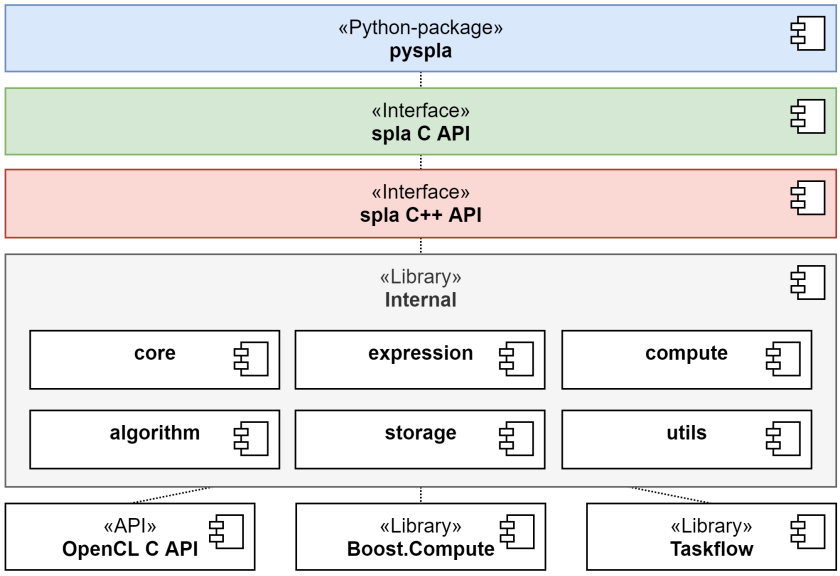

Рис. 1: Архитектура библиотеки

3.1. Компоненты библиотеки

Далее предлагается рассмотреть высокоуровневую структуру разрабатываемой библиотеки *spla*. Диаграмма основных компонентов представлена на рисунке 1. В данной диаграмме зависимости между компонентами направленны сверху вниз, модули расположены внутри одной компоненты имеют общий уровень в иерархии. Детальное описание отдельных частей диаграммы представлено ниже.

- Pyspla — это Python-пакет для работы с примитивами библиотеки в высокоуровной среде языка Python. Данный пакет предназначен для конечных пользователей. Он будет предоставлять средства для прототипирования и запуска алгоритмов, а также будет расширять функциональность исходной библиотеки за счет введения дополнительных операций и за счет синтаксического сахара, доступного в языке Python.

- Spla C API — это C99 совместимы интерфейс библиотеки для ее использования в C-приложения. Данный интерфейс требуется как для экспортирования методов и операций в Python-пакет за счет методов вызова функций из скомпилированной библиотеки с разделяемым кодом, так и для написания высокопроизводительных прикладных алгоритмах с минимальным количеством накладных расходов.

- Spla C++ API — основной интерфейс библиотеки. Предоставляет доступ ко всем примитивам и операциям библиотеки. Интерфейс не использует шаблоны. Для автоматизации управления временем жизни объектов используется встроенный механизма подсчета ссылок на объекты, что позволяет их передавать в любые вычислительные выражения.

- Intenral — детали реализации библиотеки, полностью скрытые для внешнего пользователя. Далее предлагается кратко рассмотреть основные модули реализации:

  - Core — ядро библиотеки. Поддерживает глобальное состояние, предоставляет доступ ко всем вспомогательным и основным подсистемам библиотеки.
  - Storage — предоставляет средства для автоматизированного гибридного хранения матриц и векторов. Матрицы и вектора хранятся в виде двумерной или одномерной сетки блоков. Каждый блок имеет свой собственный формат хранения.
  - Expression — механизм для трансляции входных графов выражений в ациклический граф задач, который можно выполнить на множестве потоков процессора.
  - Algorithm — модуль, который хранит множетсво алгоритмов для обработки блоков матриц и векторов в разных форматах и разных сочетаниях.
  - Compute — вспомогательные базовые алгоритмы, расширяющие функциональность библиотеки Boost.Compute.
  - Utils — утилиты, используемые в библиотеке.

- Сторонние зависимости. Для работы с GPU-устройством используется OpenCL API. Boost.Compute [22] является надстройкой над OpenCL. Он предоставляет реализацию базовых алгоритмов (*sort*, *scatter*, и т.д.) и контейнеров для данных (*vector*, *iterator*) для вычислений на GPU-устройстве. Taskflow [23] используется в качестве многопоточного менеджера задач, который позволяет выполнять пользовательские задачи в виде ориентированного графа с зависимостями.

4. Заключение

В рамках выполнения данной работы были получены следующие результаты.

- Проведен обзор существующих наиболее влиятельных и популярных библиотек для анализа графов в терминах разреженной линейной алгебры. Рассмотрены инструменты, особенности их использования и детали реализации. Также выявлен ряд недостатков, которыми данные решения обладают.

- Проведен обзор существующих инструментов и технологий для создания программ, использующих GPU-устройство для вычислений. В соответствии с достоинствами, недостатками и целями проекта выбрана технология OpenCL API для написания GPU-специфичного кода.

- Разработана архитектура библиотеки примитивов обобщенной разреженной линейной алгебры для вычислений в multi-GPU инфраструктуру. Данная архитектура основана на многочисленных требованиях к проекту, которые были получены на основе анализа существующих решений. Также начата разработка прототипа библиотеки, первой версии, пригодной для дальнейшего анализа, расширения и улучшения.

Разрабатываемая библиотека опубликована в сервисе GitHub. Страница проекта доступна по ссылке `https://github.com/JetBrains-Research/spla`.

Список литературы

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

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

[3] Zhang Xiaowang, Feng Zhiyong, Wang Xin, Rao Guozheng, and Wu Wenrui. Context-Free Path Queries on RDF Graphs // CoRR. — 2015. — Vol. abs/1506.00743. — 1506.00743.

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

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

[6] Dalton Steven, Bell Nathan, Olson Luke, and Garland Michael. Cusp: Generic Parallel Algorithms for Sparse Matrix and Graph Computations. — 2014. — Version 0.5.0. Access mode: http://cusplibrary.github.io/.

[7] Davis Timothy A. Algorithm 1000: SuiteSparse:GraphBLAS: Graph Algorithms in the Language of Sparse Linear Algebra // ACM Trans. Math. Softw. — 2019. — Dec. — Vol. 45, no. 4. — Access mode: https://doi.org/10.1145/3322125.

[8] 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).

[9] Mishin Nikita, Sokolov Iaroslav, Spirin Egor, Kutuev Vladimir, Nemchinov Egor, Gorbatyuk Sergey, and Grigorev Semyon. Evaluation of the Context-Free Path Querying Algorithm Based on Matrix Multiplication. — 2019. — 06. — P. 1–5.

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

[11] Wang Yangzihao, Davidson Andrew, Pan Yuechao, Wu Yuduo, Riffel Andy, and Owens John D. Gunrock // Proceedings of the 21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. — 2016. — Feb. — Access mode: http://dx.doi.org/10.1145/2851141.2851145.

[12] Kepner J., Aaltonen P., Bader D., Buluc A., Franchetti F., Gilbert J., Hutchison D., Kumar M., Lumsdaine A., Meyerhenke H., McMillan S., Yang C., Owens J. D., Zalewski M., Mattson T., and Moreira J. Mathematical foundations of the GraphBLAS // 2016 IEEE High Performance Extreme Computing Conference (HPEC). — 2016. — Sep. — P. 1–9.

[13] NVIDIA. CUDA Thrust // NVIDIA Developer Zone. — 2020. — Access mode: https://docs.nvidia.com/cuda/thrust/index.html (online; accessed: 16.12.2020).

[14] 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).

[15] Ching Avery, Edunov Sergey, Kabiljo Maja, Logothetis Dionysios, and Muthukrishnan Sambavi. One Trillion Edges: Graph Processing at Facebook-Scale // Proc. VLDB Endow. — 2015. — aug. —

Vol. 8, no. 12. — P. 1804–1815. — Access mode: https://doi.org/10.14778/2824032.2824077.

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

[17] Orachyov Egor, Alimov Pavel, and Grigorev Semyon. cuBool: sparse Boolean linear algebra for Nvidia Cuda. — 2020. — Access mode: https://pypi.org/project/pycubool/. Version 1.2.0.

[18] Anderson James, Novák Adám, Sükösd Zsuzsanna, Golden Michael, Arunapuram Preeti, Edvardsson Ingolfur, and Hein Jotun. Quantifying variances in comparative RNA secondary structure prediction // BMC bioinformatics. — 2013. — 05. — Vol. 14. — P. 149.

[19] Cailliau P., Davis T., Gadepally V., Kepner J., Lipman R., Lovitz J., and Ouaknine K. RedisGraph GraphBLAS Enabled Graph Database // 2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW). — 2019. — P. 285–286.

[20] Orachev Egor, Karpenko Maria, Khoroshev Artem, and Grigorev Semyon. SPbLA: The Library of GPGPU-Powered Sparse Boolean Linear Algebra Operations // 2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW). — 2021. — P. 272–275.

[21] Shun Julian and Blelloch Guy E. Ligra: A Lightweight Graph Processing Framework for Shared Memory // SIGPLAN Not. — 2013. — feb. — Vol. 48, no. 8. — P. 135–146. — Access mode: https://doi.org/10.1145/2517327.2442530.

[22] Szuppe Jakub. Boost.Compute: A Parallel Computing Library for C++ Based on OpenCL // Proceedings of the 4th International Workshop on OpenCL. — New York, NY, USA : Association for Computing Machinery. — 2016. — IWOCL '16. — Access mode: https://doi.org/10.1145/2909437.2909454.

[23] Huang Tsung-Wei, Lin Dian-Lun, Lin Chun-Xun, and Lin Yibo. Taskflow: A Lightweight Parallel and Heterogeneous Task Graph Computing System // IEEE Transactions on Parallel and Distributed Systems. — 2022. — Jun. — Vol. 33, no. 6. — P. 1303–1320. — Access mode: http://dx.doi.org/10.1109/TPDS.2021.3104255.

[24] 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).

[25] 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).

[26] Yang Carl, Buluç Aydın, and Owens John D. GraphBLAST: A High-Performance Linear Algebra-based Graph Framework on the GPU // arXiv preprint. — 2019.

[27] cuBool: sparse linera Boolean algebra for NVIDIA CUDA // Github. — 2020. — Access mode: https://github.com/JetBrains-Research/cuBool.

[28] pygraphblas: a Python wrapper around the GraphBLAS API. — Access mode: https://github.com/Graphegon/pygraphblas (online; accessed: 28.12.2022).