Санкт-Петербургский государственный университет

Кафедра системного программирования

Группа 18.Б10-мм

Панфилёнок Дмитрий Викторович

Высокоуровневая реализация спецификации GraphBLAS с поддержкой исполнения на GPU

Отчёт по учебной практике

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

Санкт-Петербург
2021

Оглавление

1. Введение

Графы являются одной из основных структур в информатике, а алгоритмы над ними чрезвычайно важны в анализе компьютерных и социальных сетей, статическом анализе кода, биоинформатике [10]. Графы, возникающие в данных областях, могут содержать миллионы узлов и ребер, поэтому распараллеливание алгоритмов для работы с ними оказывается необходимым условием достижения высокой производительности. Однако такое улучшение их эффективности происходит за счет более сложной модели программирования [15]. Результатом этого становится, в том числе, несоответствие между языками высокого уровня, на которых пользователи и разработчики графовых алгоритмов предпочли бы программировать (например, Python), и языками программирования для параллельного оборудования [15]. Таким образом, существующие решения либо просты в использовании (networkx[1]), либо высокопроизводительны (gunrock[2]). Проблемой является и то, что традиционные параллельные алгоритмы для анализа графов тяжело реализовывать и оптимизировать, а прирост производительности от роста числа параллельных процессов снижается [10].

С появлением эффективных структур и алгоритмов для разреженных матриц становится возможным использование подхода к вычислению над графами, основанного на линейной алгебре. Матрица смежности может представлять широкий спектр графов, включая ориентированные, взвешенные, двудольные. Ключевым свойством такого подхода является способность оперировать богатым набором графов различных типов с помощью небольшого набора матричных операций над полукольцами. Например, транспонирование матрицы смежности изменяет направления ребер в ориентированном графе, а умножение матрицы на вектор, как показано на рисунке 1, является шагом в алгоритме поиска в ширину.

[1]Репозиторий библиотеки networkx: https://github.com/networkx/networkx. Дата посещения: 13.12.2020

[2]Репозиторий библиотеки gunrock: https://github.com/gunrock/gunrock Дата посещения: 13.12.2020

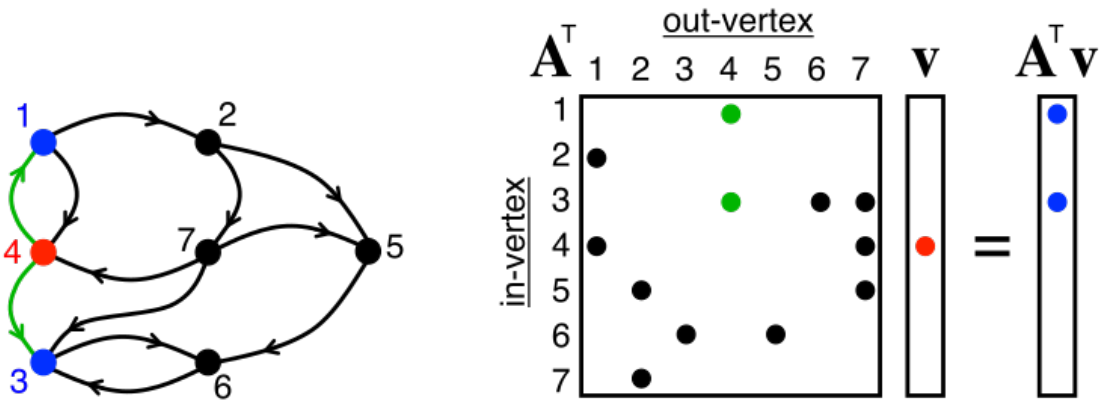

Рис. 1: Вычисление одного шага в алгоритме поиска в ширину[3]

Спецификация GraphBLAS [10] определяет базовые примитивы для построения графовых алгоритмов в терминах линейной алгебры. Разработчики считают, что эта область достаточно зрела, чтобы иметь потребность в стандартизации. Стандартизация позволяет сконцентрировать усилия исследователей на разработке инновационных алгоритмов для анализа и обработки графов, а не придумывать все новые, во многом пересекающиеся, низкоуровневые решения. Кроме того, благодаря такому подходу, по словам авторов [7], возможно решение некоторых проблем, описанных ниже.

1. **Переносимость.** Алгоритмы не требуют модификаций для достижения высокой производительности на конкретном устройстве.

2. **Лаконичность.** Алгоритмы выражаются гораздо меньшим числом строчек кода.

3. **Производительность.** Алгоритмы остаются высокопроизводительными.

4. **Масштабируемость.** Алгоритмы эффективны как на небольших, так и на огромных данных.

GraphBLAS описывает небольшое множество математических операций, которые необходимы для реализации широкого спектра операций над графами. В стандарте описаны следующие объекты:

[3]GraphBLAS [Электронный ресурс] // Википедия. Свободная энциклопедия. – URL: https://en.wikipedia.org/wiki/GraphBLAS (дата обращения: 13.12.2020).

- абстрактные структуры для хранения дынных (матрицы, векторы)
- алгебраические структуры (моноиды, полукольца, бинарные и унарные операторы)
- операции линейной алгебры над произвольными алгебраическими структурами (произведение матриц, поэлементное сложение и умножение, взятие подматрицы и т.д.)
- объекты управления (маски и дескрипторы)

На данный момент стандарт GraphBLAS уже имеет несколько полноценных реализаций[4], однако все они в основном ориентированы на исполнение на CPU. В то же время разработка инструмента с поддержкой исполнения на графических процессорах общего назначения является перспективным направлением исследований, так как их использование может существенно повысить производительность такого рода решений [6] [15]. На текущий момент нет стандартного подхода к реализации спецификации GraphBLAS на GPU — разработчики сталкиваются не только с проблемами, связанными с реализацией обобщенных операций на графических процессорах с помощью стандартных инструментов языка C++, но и с переносимостью решений, основанных на программно-аппаратной платформе CUDA. Одним из возможных подходов к реализации GraphBLAS на GPU является использование языка высокого уровня, а также библиотек, динамически транслирующих конструкции и объекты данного языка в низкоуровневый код, способный исполнятся на графическом процессоре видеокарты.

[4]Форум, посвященный стандарту GraphBLAS: https://graphblas.github.io/. Дата посещения: 04.06.2021

2. Постановка задачи

Предметом данной работы является реализация высокоуровневой, производительной и переносимой библиотеки на основе стандарта GraphBLAS с целью предоставить удобный инструмент анализа и обработки графов широкому кругу разработчиков и исследователей.

Задачи:

1. Реализовать пользовательский интерфейс, соответствующих спецификации GraphBLAS.
2. Реализовать базовые примитивы линейной алгебры для выполнения на графических процессорах общего назначения.
3. Провести экспериментальное исследование предложенной реализации и сравнить её с аналогичными решениями в предметной области.

3. Обзор

В таблице 1 представлено сравнение текущих реализаций спецификации GraphBLAS, а также некоторых других популярных библиотек для анализа графов на GPU, основанных на иных подходах к построению алгоритмов.

Одним из критериев оценки существующих решений стало наличие возможности использования графических процессоров видеокарты в качестве платформы для исполнения вычислений. Такая возможность может существенно повысить производительность решений, основанных на линейной алгебре, так как операции с разреженными матрицами поддаются массовому параллелизму [15]. Однако реализация GraphBLAS на GPU является открытой проблемой — как показано в таблице 1, на данный момент нет решений, которые бы полностью поддерживали вычисления на графическом процессоре. Это связано, в том числе с тем, что библиотеки, использующее для этого языки C и C++, сталкиваются с рядом проблем при попытке реализовать обобщенные операции на GPU. Кроме того, все существующие решения, которые разрабатываются для исполнения на GPU, используют программно-аппаратную архитектуру CUDA. Это означает, что они могут исполняться только на графических устройствах от NVIDIA, что негативно сказывается на переносимости таких библиотек. Решение, описываемое в данной работе, предлагает использовать открытую платформу OpenCL, которая позволяет осуществлять вычисления на любом устройстве с поддержкой данного стандарта, будь то GPU, CPU или FPGA. Такой подход не только позволит использовать всю вычислительную силу современных графических систем, но и облегчит разработку и тестирование алгоритмов широкому кругу исследовательских групп и независимых разработчиков.

Другим критерием оценки существующих библиотек стал язык программирования, на котором она написаны. Как показано в таблице 1, на текущий момент большинство реализаций написаны для языков C и C++. В то же время использование языков более высокого уровня име-

Реализация	Язык	Поддержка GPU
SuiteSparse [4]	C	Нет (поддержка CUDA в разработке)
IBM GraphBLAS[a]	C	Нет
GBTL [6]	C++	Нет (версия с CUDA устарела)
GraphBLAST [15]	C++	CUDA (в разработке)
CombBLAS [3]	C++	Частичная, CUDA
Graphulo[b]	Java	Нет
GraphMat[c]	C++	Нет
Gunrock [8]	C++	CUDA
CuSha[d]	C++	CUDA

[a]Репозиторий библиотеки IBM GraphBLAS: https://github.com/IBM/ibmgraphblas. Дата посещения: 13.12.2020

[b]Репозиторий библиотеки Graphulo: https://github.com/Accla/graphulo. Дата посещения: 13.12.2020

[c]Репозиторий библиотеки GraphMat: https://github.com/narayanan2004/GraphMat. Дата посещения: 13.12.2020

[d]Репозиторий библиотеки CuSha: https://github.com/farkhor/CuSha. Дата посещения: 13.12.2020

Таблица 1: Сравнение текущих реализаций GraphBLAS (верхний сегмент) и некоторых популярных библиотек для анализа графов на GPU, основанных на иных подходах к построению алгоритмов (нижний сегмент).

ет свои преимущества. Среди них можно выделить возможность дополнительных проверок во время компиляции, богатую экосистему некоторых платформ, абстрактную систему типов, благодаря которой можно более полно описывать абстракции линейной алгебры. Для функциональных языков программирования могут быть доступны некоторые техники суперкомпиляции и оптимизации, например kernel fusion [12] и deforestation [13].

В таблице 2 приведены инструменты для работы с OpenCL на платформах .NET и Java. Эти платформы достаточно популярны, чтобы предлагаемое решение было более востребовано. Библиотеки OpenCL.NET, Cloo, JavaCL и JOCL предоставляют высокоуровневый интерфейс для работы с ядрами, написанными на C. Эти библиотеки не позволяют естественным образом работать со сложными типами высокоуровневого языка, поэтому не подходят для данной работы. Среди оставшихся инструментов была выбрана библиотека Brahma.FSharp

.NET	Java
OpenCL.NET[a]	JavaCL[b]
Cloo[c]	JOCL[d]
FSCL[e]	Aparapi[f]
Brahma.FSharp[g]	ScalaCL[h]

[a]Репозиторий библиотеки OpenCL.NET: https://github.com/dgsantana/OpenCL.NET. Дата посещения: 01.06.2021

[b]Репозиторий библиотеки JavaCL: https://github.com/nativelibs4java/JavaCL. Дата посещения: 01.06.2021

[c]Репозиторий библиотеки Cloo: https://github.com/clSharp/Cloo. Дата посещения: 01.06.2021

[d]Репозиторий библиотеки JOCL: https://github.com/gpu/JOCL. Дата посещения: 01.06.2021

[e]Репозиторий библиотеки FSCL: https://github.com/FSCL/FSCL.Compiler. Дата посещения: 01.06.2021

[f]Репозиторий библиотеки Aparapi: https://github.com/aparapi/aparapi. Дата посещения: 01.06.2021

[g]Репозиторий библиотеки Brahma.FSharp: https://github.com/YaccConstructor/Brahma.FSharp. Дата посещения: 01.06.2021

[h]Репозиторий библиотеки ScalaCL: https://github.com/nativelibs4java/ScalaCL. Дата посещения: 01.06.2021

Таблица 2: Инструменты для работы с OpenCL на платформах .NET и Java

для языка программирования F#. На данный момент она активную разрабатывается на кафедре системного программирования СПбГУ, поэтому в случае возникновения ошибок можно ожидать их своевременного исправления.

4. Реализация

4.1. Архитектура решения

Архитектура предлагаемого решение приведена на рисунке 2.

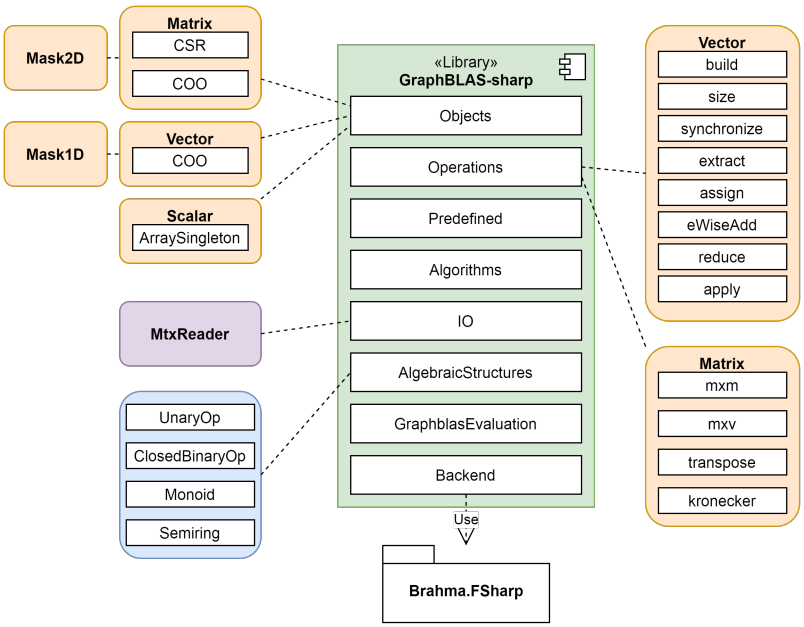

Рис. 2: Архитектура предлагаемой библиотеки GraphBLAS-sharp

В рамках данной работы были реализованы компоненты библиотеки GraphBLAS-sharp, описанные ниже.

- **Модули коллекций Matrix и Vector**. Модули содержат абстрактные классы для представления матрицы и вектора, реализованные в виде размеченного объединения, а также конкретные форматы их хранения.

- **Модули операций Matrix и Vector**. Модули содержат определение операций стандарта GraphBLAS для работы с абстрактными коллекциями.

- **Модуль AlgebraicStructures**. Модуль содержит реализацию некоторых алгебраических структур, таких как моноид и полукольцо.

- **Модуль Predefined**. Модуль содержит реализацию представителей классов алгебраических структур для встроенных типов данных. Так, например, в модуле реализованно стандартное булево полукольцо, стандартное арифметическое полукольцо над встроенными типами данных, а также тропическое полукольцо.

- **Модуль GraphblasEvaluation**. Модуль содержит вычислительное выражение GraphblasEvaluation. Оно выполняет 2 функции — во-первых, являясь по своей сути монадой Reader, позволяет выставлять глобальные и локальные параметры вычисления, а во-вторых, оно скрывает использование интерфейса для работы с OpenCL, который предоставляет библиотека Brahma.Fsharp.

- **Модуль MtxReader**. Модуль предназначен для импорта матриц из файлов в формате *mtx*.

- **Модуль Algorithms**. Модуль содержит небольшой набор классических алгоритмов на графах. В нем реализованы следующие алгоритмы:

  - алгоритм поиска в ширину из единственной вершины
  - алгоритм поиска кратчайшего пути из единственной вершины
  - алгоритм подсчета числа треугольников в графе
  - алгоритм вычисления меры центральности вершины

- **Модуль Backend**. Модуль содержит реализацию соответствующих операций для конкретных форматов хранения матрицы и вектора. В рамках данной работы были реализованны следующие операции:

  - умножение матрицы в CSR формате на разреженный вектор
  - транспонирование матрицы в CSR формате
  - создание разреженного вектора из списка элементов

4.2. Реализация коллекций

Особенностью стандарта GraphBLAS является то, что описываемые в нем объекты абстрактны — за реализацию внутреннего хранения объектов ответственен разработчик решения.

В качестве форматов хранения матрицы используются 2 формата — CSR и COO. Матрицы, которые выражают графы, обычно сильно разреженны, поэтому использовать плотную матрицу в качестве внутренней реализации нецелесообразно. CSR формат, по сравнению, например, с диагональным (DIA) или ELLPACK (ELL) форматом, лучше подходит для хранения произвольных разреженных матриц, так как не требует от матрицы соответствия определенному паттерну для эффективного хранения. По сравнению с COO он занимает $2 \cdot nnz + n + 1$ вместо $3 \cdot nnz$ памяти (здесь и далее nnz — число ненулевых элементов матрицы, n — число строк матрицы, m — число столбцов) и предоставляет доступ к произвольной строке за $O(1)$, что является существенным преимущество при реализации параллельных алгоритмов. В то же время для хранения сверхразреженных матриц, у которых $nnz > (m \cdot (n-1) - 1)/2$ лучше подходит формат COO.

4.3. Реализация операций

4.3.1. Умножение матрицы в CSR формате на разреженный вектор

Для реализации операции умножения матрицы на вектор были рассмотрены несколько алгоритмов. В статье за авторством W. Liu and B. Vinter [9] приводится алгоритм умножения разреженных матриц в CSR формате. Ключевой операцией в алгоритмах такого рода является операция слияния строк промежуточной матрицы. Так как все строки имеют разное число ненулевых элементов, то для данной операции особо важно выбрать эффективный метод балансирования нагрузки. В данной статье предлагается разделить все строки промежуточной матрицы на 38 групп в зависимости числа ненулевых элементов в них и

для каждой группы использовать один из четырех алгоритмов слияния. Однако, как показывает последние работы в данной области [11], существуют более производительные и простые в решения. Так, статья за авторством Y. Nagasaka и др. [11], которая также описывает алгоритм умножения разреженных матриц в CSR формате, предлагает для слияния строк использовать хеш-таблицу. По словам автора, этот алгоритм выигрывает у предыдущего и по времени работы, и по количеству потребляемой памяти. Однако для обновления данных в хеш-таблице алгоритм использует атомарные операции, а атомарные операции над произвольными типами на данный момент не поддерживаются библиотекой Brahma.FSharp, поэтому реализовать данный алгоритм оказалось невозможно. Алгоритм, описывающий умножение матрицы в CSR формате на разреженный вектор, за авторством Y. Tao и др. [1] также предполагает использование атомарных операций, что делает невозможным его реализацию. Другой метод авторов Yang, C., Wang, Y., и Owens, J. D [14] описывает умножения матрицы в CSR формате на разреженный вектор в контексте применения его в алгоритмах, подобных алгоритму поиска в ширину, а при оценке эффективности работы использует допущение о том, что вектор имеет небольшое число ненулевых элементов. Несмотря на то, что именно этот алгоритм реализован в аналогичной библиотеке GraphBLAST, было решено отказаться от него в пользу альтернативного варианта. В предлагаемой работе был реализован следующий алгоритм.

1. Каждая строка матрицы умножается на разреженный вектор.
    - Каждая строка обрабатывается 1 рабочей группой.
    - Каждый поток в рабочей группе обрабатывает 1 элемент левого вектора с шагом, равным размеру рабочей группы.
    - Значение с нужным индексом в правом векторе ищется бинарным поиском.
2. Полученный вектор промежуточных значений фильтруется от нулевых элементов с использованием префиксной суммы.

4.3.2. Транспонирование матрицы в CSR формате

Для транспонирования матрицы в CSR формате был реализован наивный алгоритм, предполагающий конвертацию матрицы в формат COO.

1. Матрица в CSR формате конвертируется в формат COO.
2. Индексы элементов сортируются битонической сортировкой так, чтобы соответствовать транспонированной матрице.
3. Матрица в COO формате конвертируется обратно в формат CSR.

4.4. Тестирование и непрерывная интеграция

Использование высокоуровневого языка для реализации стандарта GraphBLAS, в том числе, облегчает тестирование. В GraphBLAS внутреннее поведение операции зависит от объектов управления и от того, как именно реализованы абстрактные объекты коллекций, причем с ростом числа используемых реализаций число вариантов операции, которые нужно тестировать, растет экспоненциально. В реализациях на C++, в которых для тестирования применялся Boost Test Framework, отдельно тестируется каждый вариант и, как правило, только на одном типе данных.

В данной работе для тестирования используется библиотека Expecto[5]. За счет интеграции с библиотекой FsCheck[6], она поддерживает property-based тестирование, что позволяет автоматически генерировать наборы данных для тестов. Кроме того, она позволяет писать комбинаторные параметрические тесты, благодаря чему можно легко проверить все возможные варианты операции.

Возможность исполнения на CPU программ, которые используют OpenCL, позволила легко настроить тестирование библиотеки в сервисах непрерывной интеграции, таких как AppVeyor и GitHub Actions.

[5]Репозиторий библиотеки Expecto: https://github.com/haf/expecto. Дата посещения: 01.06.2021

[6]Репозиторий библиотеки FsCheck: https://github.com/fscheck/FsCheck. Дата посещения: 01.06.2021

5. Результаты

5.1. Эксперименты

Для апробации предлагаемого решения был реализован алгоритм поиска в ширину из единственной вершины. Предлагаемая реализация сравнивалась с другой библиотекой для анализа графов на платформе .NET — QuickGraph[7]. Набор данных состоит из 4 квадратных матриц, взятых из коллекции матриц SuiteSparse Matrix Collection [5]. Характеристики каждой матрицы представлены в таблице 3. Сравнение поводилось с помощью библиотеки BenchmarkDotNet[8] на ПК с операционной системой Windows 10 в конфигурации Intel Core i5-4690K CPU 3.50GHz, DDR3 8GB RAM, GeForce GTX 970, 4GB VRAM. Для каждой матрицы алгоритм запускался от 20 до 100 раз из случайной вершины. Перед запуском матрица была загружена в видеопамять, поэтому затраты на копирование матрицы не учитывались. Подобный принцип проведения экспериментов используется в эталонном решении для измерения производительности алгоритмов на графах GAP Benchmark Suite [2]. В таблицах 4 и 5 представлены результаты сравнения.

Название матрицы	Число строк/столбцов, $\|V\|$	Число ненулевых элементов, $\|E\|$
arc130	130	1282
linux-call-graph	324085	1208908
webbase-1M	1000005	3105536
cit-Patents	3774768	16518948

Таблица 3: Параметры используемых графов

Из результатов видно, что предлагаемое решение на несколько порядков проигрывает QuickGraph на небольших данных, но с ростом числа ненулевых элементов разница сокращается, хотя и остается очень большой.

[7]Репозиторий библиотеки QuickGraph: https://github.com/YaccConstructor/QuickGraph. Дата посещения: 01.06.2021

[8]Репозиторий библиотеки BenchmarkDotNet https://github.com/dotnet/BenchmarkDotNet. Дата посещения: 01.06.2021

Матрицы	Среднее, мс	Ошибка, мс	Медиана, мс
arc130	346	9.9	347.9
linux-call-graph	17453.4	5841.5	8765.3
webbase-1M	17367.1	5978.9	9867.5
cit-Patents	6345.4	2187.9	2888.4

Таблица 4: Оценка времени работы алгоритма в библиотеке GraphBLAS-sharp

Матрица	Среднее, мс	Ошибка, мс	Медиана, мс
arc130	0.063	0.01	0.063
linux-call-graph	8.5	0.7	9.6
webbase-1M	23.6	2.5	18.7
cit-Patents	137.6	2.6	136.6

Таблица 5: Оценка времени работы алгоритма в библиотеке QuickGraph

5.2. Выводы

Низкая производительная предлагаемого решения может иметь несколько причин.

- Во-первых, наличие больших накладных расходов на копирование данных между управляемой, неуправляемой и видеопамятью.
- Во-вторых, компиляция кода ядер во время исполнения может занимать много времени. Возможно, имеет смысл кэшировать уже скомпилированные ядра.
- В-третьих, отсутствие атомарных операций для произвольных типов данных не позволяет реализовывать некоторые алгоритмы, которые могут оказаться лучше аналогов.

Кроме того, использование OpenCL из высокоуровневого языка с управляемой памятью затрудняет профилирование кода, что усложняет поиск факторов, ограничивающих производительность библиотеки.

6. Заключение

В рамках данной работы были получены перечисленные ниже результаты.

1. Реализована[9] структура объектов и методов стандарта GraphBLAS на языке F#.

2. Реализовано подмножество операций линейной алгебры для выполнения на устройствах с поддержкой OpenCL

   - Умножение матрицы в CSR формате на разреженный вектор
   - Транспонирование матрицы в CSR формате

3. Проведено экспериментальное сравнение предложенной реализации с аналогом на платформе .NET и оценены результаты.

[9]Репозиторий проекта: https://github.com/YaccConstructor/GraphBLAS-sharp, имя аккаунта: dpanfilyonok

Список литературы

[1] Atomic reduction based sparse matrix-transpose vector multiplication on GPUs / Yuan Tao, Yangdong Deng, Shuai Mu et al. // 2014 20th IEEE International Conference on Parallel and Distributed Systems (ICPADS). — 2014. — P. 987–992.

[2] Beamer Scott, Asanovic Krste, Patterson David A. The GAP Benchmark Suite // CoRR. — 2015. — Vol. abs/1508.03619. — 1508.03619.

[3] Buluç Aydin, Gilbert John R. The Combinatorial BLAS: design, implementation, and applications // Int. J. High Perform. Comput. Appl. — 2011. — Vol. 25, no. 4. — P. 496–509. — Access mode: https://doi.org/10.1177/1094342011403516.

[4] Davis Timothy A. Algorithm 1000: SuiteSparse: GraphBLAS: Graph Algorithms in the Language of Sparse Linear Algebra // ACM Trans. Math. Softw. — 2019. — Vol. 45, no. 4. — P. 44:1–44:25. — Access mode: https://doi.org/10.1145/3322125.

[5] Davis Timothy A., Hu Yifan. The University of Florida Sparse Matrix Collection // ACM Trans. Math. Softw. — 2011. — Dec. — Vol. 38, no. 1. — Access mode: https://doi.org/10.1145/2049662.2049663.

[6] GBTL-CUDA: Graph Algorithms and Primitives for GPUs / Peter Zhang, Marcin Zalewski, Andrew Lumsdaine et al. // 2016 IEEE International Parallel and Distributed Processing Symposium Workshops, IPDPS Workshops 2016, Chicago, IL, USA, May 23-27, 2016. — IEEE Computer Society, 2016. — P. 912–920. — Access mode: https://doi.org/10.1109/IPDPSW.2016.185.

[7] Graphs, Matrices, and the GraphBLAS: Seven Good Reasons / Jeremy Kepner, David Bader, Aydın Buluç et al. // Procedia Computer Science. — 2015. — Vol. 51. — P. 2453–2462. — International Conference On Computational Science, ICCS 2015. Ac-

cess mode: https://www.sciencedirect.com/science/article/pii/S1877050915011618.

[8] Gunrock: GPU Graph Analytics / Yangzihao Wang, Yuechao Pan, Andrew A. Davidson et al. // CoRR. — 2017. — Vol. abs/1701.01170. — 1701.01170.

[9] Liu Weifeng, Vinter Brian. An Efficient GPU General Sparse Matrix-Matrix Multiplication for Irregular Data // 2014 IEEE 28th International Parallel and Distributed Processing Symposium. — 2014. — P. 370–381.

[10] Mathematical foundations of the GraphBLAS / J. Kepner, P. Aaltonen, D. Bader et al. // 2016 IEEE High Performance Extreme Computing Conference (HPEC). — 2016. — P. 1–9.

[11] Nagasaka Yusuke, Nukada Akira, Matsuoka Satoshi. High-Performance and Memory-Saving Sparse General Matrix-Matrix Multiplication for NVIDIA Pascal GPU // 2017 46th International Conference on Parallel Processing (ICPP). — 2017. — P. 101–110.

[12] Optimizing CUDA Code By Kernel Fusion—Application on BLAS / Jiri Filipovic, Matus Madzin, Jan Fousek, Ludek Matyska // CoRR. — 2013. — Vol. abs/1305.1183. — 1305.1183.

[13] Wadler Philip. Deforestation: transforming programs to eliminate trees // Theoretical Computer Science. — 1990. — Vol. 73, no. 2. — P. 231–248. — Access mode: https://www.sciencedirect.com/science/article/pii/030439759090147A.

[14] Wang Yangzihao, Yang Carl, Owens John D. Fast Sparse Matrix and Sparse Vector Multiplication Algorithm on the GPU. — 2015. — Access mode: https://escholarship.org/uc/item/1rq9t3j3.

[15] Yang Carl, Buluç Aydin, Owens John D. GraphBLAST: A High-Performance Linear Algebra-based Graph Framework on the GPU // CoRR. — 2019. — Vol. abs/1908.01407. — 1908.01407.