Санкт-Петербургский государственный университет

Кафедра системного программирования

Группа 20Б.09-мм

Гарбар Кирилл Анатольевич

Алгоритмы анализа графов в терминах линейной алгебры с использованием Brahma.FSharp

Отчёт по учебной практике

в форме «Производственное задание»

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

Санкт-Петербург
2023

Оглавление

Введение

Одной из самых известных структур данных информатики является граф. Графам находится применение в многих естественных науках, таких как биология [12], физика [8] и химия [2]. Помимо естественных наук, графы пользуются в различных областях математики — теории групп [1], топологии [15], теории вероятностей [3]. ААлгоритмы анализа графов получили широкое распространение в анализе социальных сетей. [10].

Классическое представление графа в виде множеств вершин и дуг не всегда удобно для программирования алгоритмов на графах, поэтому граф часто представляется иным образом. Один из способов представить граф в памяти машины — матрица смежности. Матрица смежности графа это квадратная матрица размера $N \times N$, где N — количество вершин графа. Ячейка (i, j) матрицы отвечает за наличие дуги между вершинами i и j. Такой способ представления графа используется на протяжении всей истории теории графов [9]. Алгоритмы обработки графов, представленных в виде матриц смежности, написанные на языке линейной алгебры, были широко освещены в литературе [7].

Граф, содержащий небольшое количество дуг относительно количества вершин, разумно представлять в виде разреженной, а не плотной матрицы. Одним из наиболее активно используемых форматов разреженных матриц является CSR[1] формат, сжимающий хранимые строковые индексы. Помимо CSR формата используется также координатный формат, хранящий индексы и значения элементов в виде троек.

Развитие идей о связи графов и разреженной линейной алгебры послужило толчком к созданию стандарта GraphBLAS[2][11], определяющего алгоритмы обработки графов на языке линейной алгебры с использованием разреженных матриц и векторов, заданных над полукольцами.

[1]Описание форматов разреженных матриц — https://en.wikipedia.org/wiki/Sparse_matrix (дата обращения: 30.05.2023)

[2]Описание GraphBLAS и ссылками на связанные материалы — https://graphblas.org/ (дата обращения: 30.05.2023)

Стандарт GraphBLAS обладает некоторыми проблемами и ограничениями, освещёнными в литературе[6]. Примером активно обсуждаемых в сообществе проблем могут служить явные и неявные нули[3].

Существует проект под названием GraphBLAS-sharp[4], который стремится реализовать операции линейной алгебры для алгоритмов анализа графов, направленный на решение некоторых из существующих в GraphBLAS проблем. Проект использует библиотеку Brahma.FSharp[5], осуществляющую трансляцию кода на языке F# в OpenCL C.

На данный момент в GraphBLAS-sharp уже имеется значительная часть операций линейной алгебры, поэтому существует возможность убедиться в работоспособности проекта, реализовав с его помощью некоторые алгоритмы анализа графов. С этой целью были выбраны базовые, но часто используемые алгоритмы анализа графов, требующие малого количества операций линейной алгебры. Такими алгоритмами являются обход графа в ширину, поиск кратчайшего пути из заданной вершины и ранжирование веб-страниц с помощью PageRank.

[3]Обсуждение проблемы явных нулей с одним из основоположников стандарта GraphBLAS — https://github.com/GraphBLAS/LAGraph/issues/28 (дата обращения: 30.05.2023)

[4]Репозиторий проекта GraphBLAS-sharp — https://github.com/YaccConstructor/GraphBLAS-sharp (дата обращения: 30.05.2023)

[5]Репозиторий проекта Brahma.FSharp — https://github.com/YaccConstructor/Brahma.FSharp (дата обращения: 30.05.2023)

1. Постановка задачи

Целью данной работы является проверка работоспособности проекта GraphBLAS-sharp путём реализации алгоритмов анализа графов в терминах линейной алгербы с помощью имеющихся операций, а также постановка сравнительных экспериментов и добавление в GraphBLAS-sharp новых операций линейной алгебры.

Были сформулированы следующие задачи.

- Реализовать умножение матрицы в CSR формате на плотный и разреженный вектор.
- С помощью имеющихся в GraphBLAS-sharp операций линейной алгербы реализовать алгоритм обхода графа в ширину, поиска кратчайшего пути и PageRank.
- Автоматизировать процесс постановки экспериментов по сравнению производительности с библиотеками-аналогами.
- Произвести сравнение производительности обхода в ширину с аналогичными реализациями.

2. Обзор

Обзор используемых в работе алгоритмов анализа графов, некоторых из существующих проектов для анализа графов, предыдущих результатов и библиотек, используемых в работе, представлен в данном разделе.

2.1. Предыдущие результаты

На данный момент в GraphBLAS-sharp имеется обширный набор обобщённых операций как для плотных, так и для разреженных алгебраических структур данных. Для предоставления пользователю гибкого и выразительного интерфейса проект полагается на функциональные особенности языка F#. Помимо этого, проект снабжён инфраструктурой для замеров производительности в целях проверки конкурентоспособности с аналогами. Всё это будет использовано в текущей работе.

2.2. алгоритмы анализа графов

В данном разделе представлены одни из наиболее известных и широко используемых алгоритмов, которые удачно выражаются в виде короткой последовательности матрично-векторных операций и поэтому подходят для проверки полученного решения.

2.2.1. Обход графа в ширину

Одним из самых известных и простых алгоритмов анализа графов является обход графа в ширину. Задача обхода состоит в том, чтобы посетить каждую вершину графа, которая достижима из заданной стартовой вершины. Порядок обхода вершин определяется расстоянием вершин от стартовой, выраженным в количестве рёбер. Таким образом, обход осуществляется по фронтам — множествам вершин, равноудалённым от стартовой.

В терминах линейной алгебры граф можно представить матрицей смежности $G \in \mathbb{B}^{n \times n}$, а фронт — вектором $front \in \mathbb{B}^n$. В качестве опе-

раций сложения и умножения возьмём стандартные. В таком случае новый фронт получается из старого путём умножения транспонированной матрицы G^T на вектор *front* с удалением ранее посещённых вершин применением маски. Результатом обхода может служить вектор, хранящий в компоненте с номером i расстояние от источника обхода до вершины с номером i, выраженное в количестве рёбер. Псевдокод данного алгоритма представлен в листинге 1.

Algorithm 1 Обход графа в ширину в терминах линейной алгебры

```
Input: Matrix G, source vertex number X
front, levels ← createVector(G.NumberOfRows)
front[X] ← true
level ← 0

while front not empty do
    level ← level + 1
    fillLevels(levels, level, front)
    newFront ← mxv(G, front)
    front ← maskComplemented(newFront, levels)
end while

return levels
```

Операция *createVector* создаёт вектор переданного ей размера и инициализирует его значениями *false*. С помощью *fillLevels* массив *levels* заполняется значением *level* только для тех вершин, которые находятся во фронте. Далее, после умножения матрицы G на вектор *front*, из фронта удаляются посещённые ранее вершины наложением дополненной маски. Операция маскирования вектора заключается в исключении из вектора значений, не покрытых вторым вектором-маской. Так как в данном случае требуется исключить все вершины, покрытые маской, используется операция дополненной маски, действующая противоположным образом. Так из фронта исключаются все вершины, результат для которых уже записан.

В процессе обхода графа, фронт обхода может становиться как плотным, так и разреженным, в зависимости от устройства графа. Чтобы не терять ресурсы, умножая матрицу на разреженный вектор в плот-

ном представлении, можно использовать умножение матрицы на разреженный вектор, когда заполненность фронта достигает определённого порога.

2.2.2. Поиск кратчайших путей от заданной вершины

Задача данного алгоритма содержится в его названии. В отличие от обхода в ширину, расстоянием является сумма весов дуг, составляющих путь.

Формулировка алгоритма в матрично-векторных терминах следующая. Имеется граф и два фронта — текущий и предыдущий, а также результирующий веткор $G \in \mathbb{R}^{n \times n}$, *front*, *prev*, *res* $\in \mathbb{R}^n$, но в качестве операций сложения и умножения используется минимум и стандартное сложение соответственно. Так же как и в обходе в ширину, новый фронт получается из старого умножением G^T на вектор *prev*. После этого в *res* и *prev* записывается минимум из старого и полученного во *front* значений. Процесс повторяется, пока обход всех достижимых вершин не завершится. Псевдокод алгоритма изображён на листинге 2.

Algorithm 2 Поиск кратчайшего пути из заданной вершины в терминах линейной алгебры

```
Input: Matrix G, source vertex number X
front, prev, res ← createVector(G.NumberOfRows)
prev[X] ← 0

while front not empty do
    front ← mxv(G, prev)

    fillWithMin(res, front, prev)
end while

return res
```

В отличие от обхода в ширину, создаваемые векторы инициализируются максимально возможным значением, это будет означать, что длина пути до вершины ещё не рассчитана. Операция *fillWithMin* заполняет массивы *res* и *prev* минимумом из старого и полученного во

front значений.

2.2.3. Ранжирование веб-страниц с помощью PageRank

Задачей алгоритма PageRank является ранжирование веб-страниц по качеству и количеству ведущих на них ссылок.

Идея алгоритма состоит в том, что о важности веб-страницы можно судить по ссылающимся на неё веб-страницам. Таким образом, чем больше ссылок имеется на данную веб-страницу, тем она релевантнее. Кроме того, разные ссылки имеют разный вес, зависящий от ссылающейся веб-страницы.

Формулировка алгоритма в матрично-векторных терминах следующая. Имеется матрица $G \in \mathbb{R}^{n \times n}$, представляющая граф, вершинами которого являются веб-страницы, а дугами — ссылки между страницами. Веса дуг, исходящих из каждой данной вершины одинаковы и в сумме дают единицу. Результатом работы алгоритма служит вектор $rank \in \mathbb{R}^n$, сопоставляющий каждой вершине её важность, выраженную числом. В начале алгоритма важность каждой вершины $1/n$.

На каждой итерации алгоритма все вершины получают вес от ссылающихся на них вершин. На языке линейной алгебры это выражается как умножение матрицы G на вектор *rank*. Так, на первой итерации алгоритма все вершины получают вес от своих прямых соседей, на второй — от соседей своих соседей и так далее. После некоторого количества итераций *rank* практически перестаёт изменяться и алгоритм останавливается. Чтобы подсчитать, насколько сильно меняется *rank*, можно использовать норму разности векторов. Псевдокод алгоритма изображён на листинге 3.

2.3. Умножение разреженной матрицы на плотный вектор

Для реализации выбранных алгоритмов в GraphBLAS-sharp не хватает такой алгебраической операции как умножение матрицы на плотный вектор. Выбранный алгоритм умножения был широко освещён в

Algorithm 3 Ранжирование веб-страниц с помощью PageRank

```
Input: Matrix G, accuracy
n ← G.Size
error ← accuracy
rank, prev ← createVector(G.NumberOfRows, 1/n)

while error ≥ accuracy do
    rank ← mxv(G, prev)
    error ← computeError(rank, prev)
    prev ← rank
end while

return rank
```

литературе[5]. Алгоритм включает в себя два этапа.

1. Умножение всех значений матрицы на соответствующие им значения из вектора.

2. Построчное сложение полученных на предыдущем этапе значений.

Первый этап алгоритма умножения реализуется самым наивным образом — поток с идентификатором i записывает результат умножения в массив по индексу i.

Для эффективной реализации второго этапа — построчного сложения полученных на первом этапе значений — используется разделяемая локальная память графического процессора, отличающаяся высокой скоростью доступа. В неё загружается массив, полученный на первом этапе. Затем осуществляется сложение элементов массива в локальной памяти по строкам матрицы.

2.4. Умножение разреженной матрицы на разреженный вектор

Вектор, используемый в описанных выше алгоритмах, часто оказывается разреженным, и для экономии ресурсов можно использовать умножение на разреженный вектор. Выбранный алгоритм умножения был

широко освещён в литературе[14]. Алгоритм включает в себя следующие этапы.

1. Отбор строк матрицы, которые будут участвовать в умножении.
2. Сортировка полученных элементов матрицы по столбцам.
3. Умножение элементов матрицы на соответствующий элемент вектора.
4. Суммирование полученных после умножения значений для каждого столбца.

На первом этапе отбираются только строки с такими номерами i, что вершина с номером i содержится в векторе.

После этого полученные значения сортируются по столбцам, например с помощью поразрядной или битонной сортировок.

Умножение элементов матрицы на элементы вектора тривиально — для каждого элемента матрицы из вектора берётся элемент, соответствующий номеру строки умножаемого элемента.

Суммирование значений по столбцам производится с помощью сегментного редуцирования, при котором между собой суммируются только элементы с одинаковым номером столбца.

2.5. Существующие библиотеки алгоритмов анализа графов

Чтобы убедиться в работоспособности полученного решения, необходимо отобрать уже существующие библиотеки-аналоги, предоставляющие набор операций линейной алгебры для алгоритмов анализа графов.

Самой известной и эталонной библиотекой, в полной мере реализующей стандарт GraphBLAS является SuiteSparse:GraphBLAS[6][4]. На её основе создана библиотека алгоритмов анализа графов LAGraph.

[6]Описание SuiteSparse и ссылки на связанные материалы — https://people.engr.tamu.edu/davis/suitesparse.html (дата обращения: 30.05.2023)

LAGraph поддерживает вычисления только на центральных процессорах, в то время как GraphBLAS-sharp использует преимущественно графические процессоры. По этой причине в качестве аналогичной библиотеки также будет рассмотрен GraphBLAST[7][13] — одна из наиболее высокопроизводительных библиотек операций линейной алгебры для графовых алгоритмов.

Рассмотренные ранее библиотеки достаточно строго реализуют стандарт GraphBLAS. Существуют также библиотеки, не стремящиеся в полной мере реализовать стандарт, но также реализующие операции линейной алгебры для анализа графов. Одной из таких библиотек является Spla[8]. Spla поддерживает высокопроизводительные математические вычисления с помощью OpenCL и позволяет реализовать алгоритмы анализа графов с помощью обобщённых алгебраических операций.

Помимо библиотек, оперирующих абстракциями линейной алгебры для работы с графами, существуют библиотеки, работающие с графами в терминах дуг и вершин. Примером такой библиотеки на платформе .NET служит QuikGraph[9], предоставляющий механизмы работы с обобщёнными графовыми структурами.

2.6. Используемая библиотека Brahma.FSharp

Поддержка OpenCL, используемого в GraphBLAS-sharp, реализована на множестве платформ разных видов и производителей. Возможность писать код на высокоуровневом языке платформы .NET, исполняемый с помощью OpenCL, предоставляется библиотекой Brahma.FSharp, осуществляющей трансляцию кода из F# в OpenCL. Разработчик может писать код на языке F#, называемый ядром, которой затем будет транслирован в язык OpenCL C и исполнен на графическом или центральном процессоре асинхронно.

[7]Репозиторий проекта GraphBLAST — https://github.com/gunrock/graphblast (14.04.2023)
[8]Репозиторий проекта Spla — https://github.com/SparseLinearAlgebra/spla (14.04.2023)
[9]Репозиторий проекта QuikGraph — https://github.com/KeRNeLith/QuikGraph (14.04.2023)

3. Умножение разреженной матрицы на вектор

Описание решения первой из поставленных задач, а именно умножения матрицы в CSR формате на плотный и разреженный вектор, представлено в данном разделе.

Единственный нетривиальным этапом умножения на плотный вектор является суммирование значений полученных в результате умножения соответствующих элементов матрицы и вектора. Этот этап полностью полагается на использование быстрой локальной памяти графического процессора. Размер локальной памяти ограничен и часто недостаточно велик, чтобы поместить туда весь массив суммируемых значений. По этой причине в начале алгоритма рассчитывается объём свободной локальной памяти и массив загружается в локальную память максимально возможными частями. Процесс повторяется пока массив не будет обработан полностью.

Для умножения матрицы на разреженный вектор необходима сортировка элементов матрицы по столбцам. Для этого была выбрана уже имеющаяся в GraphBLAS-sharp битонная сортировка.

На этапе умножения элементов требуется по элементу матрицы получить соответствующий элемент вектора. Так как вектор разреженный, сделать это за константное время невозможно. В свою очередь, создание плотного представления вектора слишком дорогостоящая операция, поэтому для поиска элемента вектора по номеру используется бинарный поиск.

Для последующего сложения элементов по столбцам для каждого столбца используется отдельный поток, суммирующий элементы последовательно.

4. AАлгоритмы анализа графов в терминах линейной алгебры

Детали реализации алгоритмов анализа графов при помощи операций из GraphBLAS-sharp описаны в этом разделе.

Основной всех реализованных алгоритмов служит умножение матрицы на вектор. Чтобы сэкономить ресурсы на инициализацию нового вектора, в GraphBLAS-sharp добавлен метод, записывающий результат в старый вектор.

Помимо этого также требуются операции, которые будут фильтровать векторы от лишних вершин или записывать в них какие-то значения. Примером таких операций может послужить удаление посещённых ранее вершин из фронта в обходе в ширину и обновление расстояний в поиске кратчайших расстояний. Все эти операции выражаются с помощью имеющейся в GraphBLAS-sharp операции *map2*, которая принимает бинарную операцию и два массива. Затем *map2* применяет переданную операцию к соответствующим элементам переданных массивов и сохраняет результат в другой массив. Так, например, передав в *map2* операцию, возвращающую минимум двух чисел, можно записать в массив кратчайших расстояний минимум из посчитанных ранее и полученных на новой итерации алгоритма расстояний. Также как и с умножением матрицы на вектор, результат записывается в уже существующие массивы там, где это возможно.

В процессе обхода графа фронт часто может содержать малое количество вершин относительно своего размера. В таком случае разумной идеей является представление фронта в разреженном виде. С этой целью помимо умножения матрицы на плотный вектор также было реализовано умножение на разреженный вектор. Таким образом, умножение на плотный и разреженный вектор чередуются в зависимости от заполненности фронта, вычисляемой в конце каждой итерации. Такой вариант реализации обхода в ширину также добавлен в GraphBLAS-sharp.

5. Экспериментальное исследование

В данном разделе описана постановка экспериментов по сравнению производительности и приведены полученные результаты, а также сделаны некоторые выводы.

5.1. Автоматизация экспериментов

Постановка экспериментов включает в себя множество рутинных задач, которые могут быть автоматизированы. В данном разделе описан процесс автоматизации замеров производительности библиотек с алгоритмами анализа графов.

Основным инструментом послужил сервис Github Actions[10], позволяющий автоматически запускать рабочие процессы на удалённых машинах. Работа велась в репозитории проекта graph-bench[11], где уже были написаны скрипты для сборки и запуска замеров производительности библиотек-аналогов.

Сравнительные эксперименты требовалось запускать вручную, находясь рядом с машиной физически или используя удалённый доступ. Чтобы у пользователя была возможность запустить эксперименты и получить оформленный результат прямо в браузере, был написан YAML[12] скрипт, выполняющий всю работу автоматически.

Теперь провдение замеров производительности выглядит следующим образом.

1. Пользователь отправляет коммит с конфигурационными файлами в репозиторий проекта на github.

2. Инструменты GraphBLAS-sharp, LAGraph, SuiteSparse, GraphBLAST и Spla собираются из исходных файлов.

[10]Описание Github Actions — https://github.com/features/actions (дата обращения: 30.05.2023)

[11]Репозиторий проекта graph-bench — https://github.com/kirillgarbar/graph-bench (дата обращения: 30.05.2023)

[12]Язык, воспринимаемый сервисом Github Actions

3. Матрицы, указанные в конфигурационном файле, загружаются на машину.

4. Происходит замер производительности выбранных инструментов и алгоритмов.

5. Результаты оформляются в таблицу и отправляются пользователю на github.

5.2. Постановка эксперимента

Для постановки экспериментов в GraphBLAS-sharp существует отдельный проект под названием GraphBLAS-sharp.Benchmarks, в основе которого лежит инструмент BenchmarkDotNet[13]. В проекте реализована загрузка матриц в формате Matrix Market[14] и их дальнейшие преобразования, выделение необходимой для эксперимента памяти и её чистка, а также замер времени исполнения нужного алгоритма над матрицами.

Матрицы, на которых ставился эксперимент, были взяты из The SuiteSparse Matrix Collection[15]. Были отобраны матрицы разной степени разреженности и размера. Выбранные матрицы приведены в таблице 1.

Автоматизированный замер производительности аналогов осуществлялся с помощью проекта graph-bench.

Название	Размер	Количество элементов	Средняя степень
wing	62 032	243 088	3,0
coAuthorsCiteseer	227 320	1 628 238	7,2
coPapersDBLP	540 486	15 245 729	56,4
hollywood-2009	1 139 905	113 891 327	98,9
roadNet-CA	1 971 281	5 533 214	2,8

Таблица 1: Матрицы, на которых производилось сравнение

[13]Обзор инструмента BenchmarkDotNet — https://benchmarkdotnet.org/articles/overview.html (дата обращения: 30.05.2023)

[14]Описание Matrix Market формата — https://math.nist.gov/MatrixMarket/formats.html (дата обращения: 30.05.2023)

[15]Источник матриц для экспериментов — https://sparse.tamu.edu/ (дата обращения: 30.05.2023)

Название	GBS	SS	GBT	Spla	QG
wing	72 ± 2	23 ± 1	18 ± 1	12 ± 0	48 ± 1
coAuthCite	27 ± 4	8 ± 4	4 ± 0	4 ± 1	243 ± 3
coPapersDBLP	89 ± 1	20 ± 4	16 ± 0	7 ± 1	2281 ± 9
hollywood-2009	244 ± 13	27 ± 1	52 ± 1	15 ± 1	13326 ± 17
roadNet-CA	307 ± 2	164 ± 5	149 ± 5	101 ± 1	1271 ± 11

Таблица 2: Результаты сравнения библиотек на время обхода графа в ширину, GTX 2070, среднее $\pm$ стандартное отклонение, мс. Библиотеки: GraphBLAS-sharp (GBS), SuiteSpare:GraphBLAS/LAGraph (SS), GraphBLAST (GBT), Spla, QG(QuikGraph).

Характеристики машины, на которой был поставлен эксперимент, следующие: Ubuntu 20.4, Intel Core i7-4790 CPU, 3.60GHz, DDR4 32GB RAM и GeForce GTX 2070, 8GB GDDR6, 1410 MHz.

Замеры производительности происходили следующим образом. Сперва матрица загружалась из файлов в координатном формате, а затем преобразовывались в формат CSR. После этого для каждой матрицы происходило пять разогревочных итераций эксперимента, за которыми следовало десять замеряемых итераций, в результате которых высчитывалось среднее значение и стандартное отклонение времени выполнения алгоритма над графом.

5.3. Результаты сравнения с аналогами

Результаты замеров времени на обход графа в ширину, поиск кратчайшего пути и PageRank приведены в таблицах 2, 3, 4, 5.

По результатам замеров, медленнее всего оказался QuikGraph. Это связано с тем, что QuikGraph не нацелен на высокопроизводительную обработку разреженных графов большого размера. Кроме того, библиотека поддерживает вычисления только на центральном процессоре.

GraphBLAS-sharp оказался до десяти раз быстрее QuikGraph, но от полутора до десяти раз медленне остальных аналогов. При этом наибольшая разница проявилась в алгоритме поиска кратчайшего пути, в то время как в обходе графа в ширину и PageRank разница не столь значительна.

Название	GBS Pull Only	GBS Push-Pull
wing	72 ± 2	101 ± 2
coAuthCite	27 ± 4	60 ± 3
coPapersDBLP	89 ± 1	101 ± 7
hollywood-2009	244 ± 13	194 ± 7
roadNet-CA	307 ± 2	2960 ± 23

Таблица 3: Результаты сравнения времени обхода графа в ширину с использованием плотных и разреженных векторов, GTX 2070, среднее $\pm$ стандартное отклонение, мс. Pull Only — обход с плотным фронтом, Push-Pull — обход в ширину с чередованием плотного и разреженного фронта.

Название	GBS	SS	GBT	Spla	QG
wing	63 ± 2	28 ± 1	32 ± 1	16 ± 1	142 ± 1
coAuthCite	620 ± 4	40 ± 4	12 ± 0	7 ± 0	944 ± 9
coPapersDBLP	3278 ± 14	91 ± 2	90 ± 1	17 ± 1	5030 ± 52
hollywood-2009	5236 ± 37	239 ± 1	323 ± 1	35 ± 1	17908 ± 73
roadNet-CA	1723 ± 12	1077 ± 7	367 ± 5	145 ± 5	5043 ± 22

Таблица 4: Результаты сравнения библиотек на время поиска кратчайшего пути, GTX 2070, среднее $\pm$ стандартное отклонение, мс. Библиотеки: GraphBLAS-sharp (GBS), SuiteSpare:GraphBLAS/LAGraph (SS), GraphBLAST (GBT), Spla, QG(QuikGraph)

Название	GBS	SS	GBT	Spla
wing	39 ± 2	3 ± 0	1 ± 0	1 ± 0
coAuthCite	80 ± 3	33 ± 1	1 ± 0	1 ± 0
coPapersDBLP	348 ± 7	370 ± 1	14 ± 0	4 ± 0
hollywood-2009	1300 ± 12	1327 ± 10	50 ± 1	19 ± 1
roadNet-CA	340 ± 3	282 ± 7	5 ± 0	14 ± 0

Таблица 5: Результаты сравнения библиотек на время ранжирования веб-страниц с помощью PageRank, GTX 2070, среднее $\pm$ стандартное отклонение, мс. Библиотеки: GraphBLAS-sharp (GBS), SuiteSpare:GraphBLAS/LAGraph (SS), GraphBLAST (GBT), Spla

Операции, используемые в алгоритмах GraphBLAS-sharp и аналогов схожи, поэтому нельзя считать, что виной этому послужил только более богатый набор операций у аналогов. Отсюда можно сделать вывод, что отставание в производительности в большей мере вызвано низкой производительностью отдельных матрично-векторных операций.

Можно также заметить, что попытка оптимизации обхода графа в ширину путём учёта разреженности фронта не принесла положительных результатов. Главной причиной этому послужил тот факт, что рассчёт заполненности фронта на каждой итерации требует существенных вычислительных ресурсов. Сложность алгоритма умножения матрицы на разреженный вектор также сыграла в этом не последнюю роль.

5.4. Вывод

Выводы, сформулированные в результате анализа данных, полученных в ходе замеров, следующие.

- GraphBLAS-sharp работает с приемлемой для платформы .NET производительностью, но значительно медленнее аналогичных библиотек на языке C/C++.
- Основная причина проигрыша в производительности — базовые операции GraphBLAS-sharp медленнее своих аналогов из других библиотек.
- Базовые матрично-векторные операции GraphBLAS-sharp требуют тщательной оптимизации.
- Имеющийся в GraphBLAS-sharp набор операций позволяет выражать основные алгоритмы анализа графов с потенциально приемлемой производительностью.

Заключение

В ходе данной работы были выполнены следующие задачи.

- Реализовано умножение матрицы в CSR формате на плотный и разреженный вектор.
- С помощью имеющихся в GraphBLAS-sharp операций линейной алгербы реализован алгоритм обхода графа в ширину, поиска кратчайшего пути и PageRank.
- Автоматизирован процесс постановки экспериментов по сравнению производительности с библиотеками-аналогами.
- Произведено сравнение производительности обхода в ширину с реализациями из SuiteSpare/LAGraph, GraphBLAST, Spla и QuikGraph.

Так как значительная часть необходимых матрично-векторных операций уже реализована GraphBLAS-sharp, а также имеются использующие их алгоритмы с графами, в дальнейшие планы входит оптимизация данных операций с целью достижения конкурентоспособной производительности.

Код проектов GraphBLAS-sharp и graph-bench доступен в репозиториях на сервисе github[16]. Имя пользователя: kirillgarbar.

[16]Репозиторий проекта GraphBLAS-sharp — https://github.com/kirillgarbar/GraphBLAS-sharp/tree/dev (дата обращения: 30.05.2023)
Репозиторий проекта graph-bench — https://github.com/kirillgarbar/graph-bench (дата обращения: 30.05.2023)

Список литературы

[1] Baginski Czeslaw and Grzeszczuk Piotr. On the generic family of Cayley graphs of a finite group // J. Comb. Theory, Ser. A. — 2021. — Vol. 184. — P. 105495. — Access mode: https://doi.org/10.1016/j.jcta.2021.105495.

[2] Nøjgaard Nikolai, Fontana Walter, Hellmuth Marc, and Merkle Daniel. Cayley Graphs of Semigroups Applied to Atom Tracking in Chemistry // J. Comput. Biol. — 2021. — Vol. 28, no. 7. — P. 701–715. — Access mode: https://doi.org/10.1089/cmb.2020.0548.

[3] Choi Hayoung, Lee Hosoo, Shen Yifei, and Shi Yuanming. Comparing large-scale graphs based on quantum probability theory // Appl. Math. Comput. — 2019. — Vol. 358. — P. 1–15. — Access mode: https://doi.org/10.1016/j.amc.2019.03.061.

[4] 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.

[5] Gao Jiaquan, Qi Panpan, and He Guixia. Efficient CSR-Based Sparse Matrix-Vector Multiplication on GPU // Mathematical Problems in Engineering. — 2016. — 01. — Vol. 2016. — P. 1–14.

[6] Gilbert John R. What did the GraphBLAS get wrong? // HPEC GraphBLAS BoF. — 2022. — Access mode: https://sites.cs.ucsb.edu/~gilbert/talks/talks.htm.

[7] Graph Algorithms in the Language of Linear Algebra / ed. by Kepner Jeremy and Gilbert John R. — SIAM, 2011. — Vol. 22 of Software, environments, tools. — ISBN: 978-0-89871-990-1. — Access mode: https://doi.org/10.1137/1.9780898719918.

[8] Jr. G. David Forney. Codes on Graphs: Models for Elementary Algebraic Topology and Statistical Physics // IEEE Trans. Inf. The-

ory. — 2018. — Vol. 64, no. 12. — P. 7465–7487. — Access mode: https://doi.org/10.1109/TIT.2018.2866577.

[9] Konig D. Graphen und Matrizen (Graphs and Matrices). — 1931.

[10] Rizi Fatemeh Salehi. Graph Representation Learning for Social Networks : Ph. D. thesis ; University of Passau, Germany. — 2021. — Access mode: https://opus4.kobv.de/opus4-uni-passau/frontdoor/index/index/docId/921.

[11] Mattson Tim, Bader David, Berry Jon, Buluc Aydin, Dongarra Jack, Faloutsos Christos, Feo John, Gilbert John, Gonzalez Joseph, Hendrickson Bruce, Kepner Jeremy, Leiserson Charles, Lumsdaine Andrew, Padua David, Poole Stephen, Reinhardt Steve, Stonebraker Mike, Wallach Steve, and Yoo Andrew. Standards for graph algorithm primitives // 2013 IEEE High Performance Extreme Computing Conference (HPEC). — 2013. — P. 1–2.

[12] Washietl Stefan and Gesell Tanja. Graph Representations and Algorithms in Computational Biology of RNA Secondary Structure // Structural Analysis of Complex Networks / ed. by Dehmer Matthias. — Birkhäuser / Springer, 2011. — P. 421–437. — Access mode: https://doi.org/10.1007/978-0-8176-4789-6_17.

[13] Yang Carl, Buluç Aydın, and Owens John D. GraphBLAST: A High-Performance Linear Algebra-Based Graph Framework on the GPU // ACM Trans. Math. Softw. — 2022. — feb. — Vol. 48, no. 1. — Access mode: https://doi.org/10.1145/3466795.

[14] Yang Carl, Wang Yangzihao, and Owens John. Fast Sparse Matrix and Sparse Vector Multiplication Algorithm on the GPU. — 2015. — 05. — P. 841–847.

[15] Xing Jie, Xu Ping, Yao Shuguang, Zhao Hui, Zhao Ziliang, and Wang Zhangjun. A novel weighted graph representation-based method for structural topology optimization // Adv. Eng. Softw. — 2021. —

Vol. 153. — P. 102977. — Access mode: https://doi.org/10.1016/j.advengsoft.2021.102977.