Санкт-Петербургский государственный университет

Кафедра системного программирования

Группа 21.Б10-мм

Векторные операции в GraphBLAS–sharp

Ерин Игорь Антонович

Отчёт по учебной практике
в форме «Производственное задание»

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

Санкт-Петербург
2022

Оглавление

Введение

Граф — одна из важнейших абстракций ввиду их общности и интуитивно понятной структуры. Они применяются во многих сферах, вследствие чего алгоритмы, оперирующие оными, имеют критическую важность [3]. Также эти алгоритмы представляют множество проблем в реализации: сложность программного обеспечения, математическая сложность, теоретический анализ [2]. Алгебраический подход к анализу графов решает многие из упомянутых проблем, так как позволяет выразить соответствующие алгоритмы с помощью небольшого набора математических операций, что облегчает их анализ и упрощает программное обеспечение, их реализующее [1].

GraphBLAS[1] — стандарт, определяющий ключевые операции линейной алгебры, на основе которых может быть реализован широкий класс графовых алгоритмов [3]. Многие реализации данного стандарта, такие как SuiteSparse[2] и GraphBLAST[3] [4], предоставляют необходимые операции, но не предоставляют удобные для пользователя абстракции: функции высших порядков и алгебраические типы данных — вследствие использования языка C. Относительно вычислительных платформ, SuiteSparse не позволяет производить вычисления на графических процессорах, GraphBLAST поддерживает вычисления только на процессорах архитектуры CUDA, что также является существенным ограничением.

GraphBLAS-sharp[4]— проект, разработанный на кафедре системного программирования СПбГУ, вдохновленный стандартом GraphBLAS. GraphBLAS-sharp не только определяет операции, но и предоставляет их высокопроизводительную реализацию на языке высокого уровня F#, с поддержкой вычислений на графических процессорах, посредством трансляции кода F# в OpenCL. Тем самым позволяет пользоваться высокоуровневыми абстракциями языка F# и производить вы-

[1]Описание GraphBLAS: https://graphblas.org/. (Дата обращения: 29.11.2022)

[2]Описание SuiteSparse: https://people.engr.tamu.edu/davis/suitesparse.html (Дата обращения: 29.11.2022)

[3]Репозиторий проекта GraphBLAST: https://github.com/gunrock/graphblast (Дата обращения: 29.11.2022)

числения, как на центральных, так на графических процессорах, благодаря использованию OpenCL, что решает многие проблемы SuiteSparse и GraphBLAST.

К сожалению, в GraphBLAS-sharp долгое время не был реализован алгоритм поиска в ширину вследствие отсутствия необходимых для этого векторных операций, реализации которых посвящена эта работа.

[4]Репозиторий библиотеки GraphBLAS-sharp:
https://github.com/YaccConstructor/GraphBLAS-sharp. (Дата обращения: 29.11.2022)

Дата сборки: 13 декабря 2022 г.

1. Постановка задачи

Целью данной работы является реализация алгоритма поиска в ширину с помощью алгебраических операций на языке F# с поддержкой OpenCL. Для чего были поставлены следующие задачи.

- Реализовать необходимые векторные операции.
  - Операцию создания маски по вектору.
  - Поэлементные векторные операции.
  - Операцию заполнения вектора значением по маске.
- Сравнить производительность алгоритма поиска в ширину, реализованного с использованием операций, упомянутых выше, с аналогами в GraphBLAST и SuiteSparse.

2. Обзор

В данном разделе приведены обзор выражения алгоритма поиска в ширину с помощью операций линейной алгебры и обзор существующих решений.

2.1. Алгоритм поиска в ширину

Алгоритм поиска в ширину — важнейший алгоритм поиска и фундаментальный примитив для многих графовых алгоритмов. С помощью линейной алгебры алгоритм поиска в ширину выражается путем рекуррентного матрично-векторного произведения матрицы смежности, характеризующей граф, и предыдущего результата. Одно из таких произведений приведено в рисунке 1.

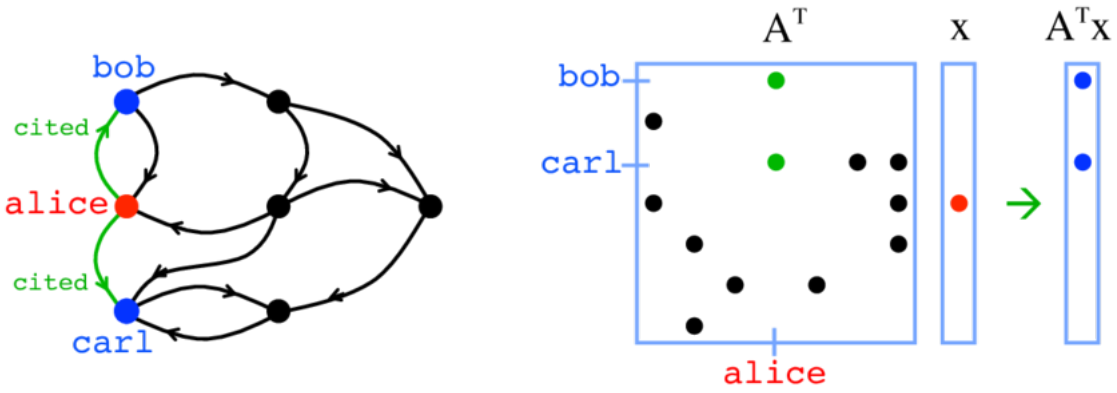

Рис. 1: Выражение алгоритма BFS языком линейной алгебры[5]

В листинге 1 приведён псевдокод алгоритма. Стоит отметить, что все вычисления происходят в булевом полукольце.

Перед циклом **while** происходит инициализация векторов **f** и **v**, представляющих активные вершины обхода и вершины, посещенные ранее, соответственно.

На каждой итерации, происходят следующие действия.

- Ныне активные вершины, представленные вектором **f**, маркируются с помощью счетчика **d**.

[5] https://graphblas.org/AdjacencyMatrixBFS.png (Дата обращения: 29.11.2022).

- Активные вершины умножаются на матрицу смежности, дабы найти следующие активные вершины, иллюстрация чего приведена в листинге 1. Вектор, полученный в результате умножения, **маскируются** дополненным вектором прежде посещенных вершин.

- Полученный таким образов вектор **f** редуцируется, результат чего записывается в переменную **c**.

- Счетчик **d** увеличивается.

Цикл продолжается пока не останется достижимых вершин.

Algorithm 1 Алгоритм поиска в ширину

1: **function** BFS($Graph\ A$, $Vector\ V$, $Source\ s$)
2: Initialize $d \leftarrow 1$
3: Initialize $f(i) \leftarrow \begin{cases} 1 & \text{if } i = s; \\ 0 & \text{if } i \neq s. \end{cases}$
4: Initialize $v \leftarrow \begin{bmatrix} 0, & 0, & \dots, & 0 \end{bmatrix}$
5: Initialize $c \leftarrow 1$
6: **while** $c > 0$ **do**
7: Update $v \leftarrow d \times f + v$;
8: Update $f \leftarrow A^T f. * \neg v$
9: Compute $c \leftarrow \sum_{i=0}^{n} f(i)$
10: Update $d \leftarrow d + 1$
11: **end while**
12: **end function**

2.2. Предыдущие результаты

Далее будет проведен обзор существующих в проекте GraphBLAS-sharp решений, которые были использованы, изменены или заменены данной работой.

Псевдокод алгоритма поиска в ширину: https://dl.acm.org/doi/pdf/10.1145/3466795 (Дата обращения: 13.12.2022)

- **Представление вектора**. Ранее в проекте были реализованы типы для представления векторов, как в разреженном формате, так и в плотном. Разреженный вектор представляется двумя массивами: массивом индексов и массивом значений. Плотный вектор представляется массивом, элементы которого имеют тип `Option`, это позволяет решить проблему неявных нулей [6], явного различия нуля-значения и нуля, сообщающего об отсутствии какого либо значения. К примеру, ноль в матрице смежности может говорить, как об отсутствии ребра между соответствующими вершинами, так и о том, что вес данного ребра равен нулю, в случае взвешенного графа.

- **Маски**. Ранее в проект были реализованы маски [5], которые представлялись отдельной сущностью. При этом маска, если необходимо, явно дополнялась перед использованием.

- **Поэлементные операции**. Ранее в проекте были реализованы поэлементные операции для матриц [5], в данной реализации поэлементных операций предполагалось, что результатом применения любой бинарной операции к двум нулям будет ноль.

- **Замеры производительности.** Ранее в GraphBLAS-sharp для постановки экспериментов был создан отдельный проект [6], в основе которого лежит инструмент BenchmarkDotNet.

2.3. Вывод

В проекте реализовано все необходимое для написания алгоритма поиска в ширину, за исключением векторных операций, упомянутых выше.

Репозиторий библиотеки BenchmarkDotNet: https://github.com/dotnet/BenchmarkDotNet (Дата обращения: 7.12.2022)

3. Реализация

В данном разделе описано решение поставленных задач, упомянуты возникшие при этом проблемы.

3.1. Маски

На стадии проектирования было предложено отказаться от уже реализованных одномерных масок, и использовать для представления масок сам вектор. Данное решение позволяет не создавать отдельную сущность “маска” по вектору каждый раз при необходимости, а использовать сам вектор в качестве маски, соответствующим образом трактуя его элементы. В случае разреженного представления вектора, существование значения с соответствующим индексом свидетельствует о маскировании позиции с этим индексом. В случае плотного вектора, элемент в маскируемом векторе подлежит маскированию, если на соответствующей позиции маски представлен элемент вида `Some`.

3.2. Поэлементные операции

Ранее в проекте уже были реализованы поэлементные операции для матриц. В случае векторов оные отличаются лишь размерностью, что приводит к дубликации кода. Которая усугубляется тем, что даже в случае матриц уже существовало две почти идентичных реализации поэлементных операций для двух типов бинарных операций, представленных в листинге 1 и 2.

Листинг 1: Первый тип бинарных поэлементных операций

```
'a option → 'b option → 'c option
```

Листинг 2: Второй тип бинарных поэлементных операций

```
AtLeastOne<'a, 'b> → 'c option
```

Первая позволяет пользователю взаимодействовать с элементами, представленными типом `Option`, что, как было упомянуто выше, ре-

шает проблему неявных нулей. Вторая оперирует лишь теми парами элементов, среди которых есть по крайней мере один ненулевой, с помощью типа `AtLeastOne`, определение которого приведено в листинге 3. Последний защищает пользователя от написания операции, возвращающей явное значение после применения к двум неявным нулям [6].

Листинг 3: Определение типа AtLeastOne

```
AtLeastOne<'a, 'b when 'a: struct and 'b: struct> =
    | Both of 'a * 'b
    | Left of 'a
    | Right of 'b
```

А значит, в случае реализации соответствующих векторных поэлементных операций, общее количество реализаций могло достичь четырех.

Данная проблема была решена, приведением операций второго типа к первому перед применением, с помощью функции представленной в листинге 4.

Листинг 4: Второй тип бинарных поэлементных операций

```
let atLeastOneToOption (op: AtLeastOne<'a, 'b> → 'c option) =
    fun (leftItem: 'a option) (rightItem: 'b option) →
        match leftItem, rightItem with
        | Some left, Some right → op (Both(left, right))
        | None, Some right → op (Right right)
        | Some left, None → op (Left left)
        | None, None → None
```

3.3. Масочные операции

На стадии проектирования предполагалось реализовать операции заполнения по маске, как и другие всевозможные масочные операции, с помощью поэлементных операций и соответствующей бинарной операции, представленной в листинге 5.

Листинг 5: Бинарная масочная операция

```
let maskOp (left: 'a option) (right: 'b option) =
    match right with
    | Some _ → left
    | _ → None
```

Реализовать операции с дополненной маской предлагалось тем же образом, не дополняя вектор явно, а используя дополненную бинарную операцию, представленную в листинге 6.

Листинг 6: Бинарная дополненная масочная операция

```
let complementedMaskOp (left: 'a option) (right: 'b option) =
    match right with
    | None → left
    | _ → None
```

Что оказалось проблемой в случае разреженных векторов, так как в текущей реализации поэлементных операций не рассматривается случай, когда на соответствующих позициях маски и вектора стоят нули, что необходимо в случае применения дополненной поэлементной операции, использующей нули маски.

Чтобы решить эту проблему, на данный момент разреженный вектор приводится к плотному формату перед применением операции дополненной маски, что нельзя назвать эффективным решением. Ввиду чего в репозитории проекта заведена соответствующая issue.

Проблема операции дополненной маски. https://github.com/YaccConstructor/GraphBLAS-sharp/issues/51 (Дата обращения: 10.12.2022)

4. Эксперимент

В данном разделе приведены условия сравнительных экспериментов и их результаты.

4.1. Цель эксперимента

Целью эксперимента является сравнение производительности алгоритма поиска в ширину в проекте GraphBLAS-sharp и его аналогов в SuiteSparse и GraphBLAST.

4.2. Условия эксперимента

Симметричные матрицы, различных размеров, представляющие одну компоненту связности, были отобраны из SuiteSparse Matrix Collection и представлены в таблице 1.

Таблица 1: Выбранные матрицы

Матрица	Вершины	Ребра
coAuthorsCiteseer	227.3K	1.6M
hollywood-2009	1.1M	113.8M
roadNet-CA	7.5M	194.1M

Эксперименты проведены с помощью библиотеки BenchmarkDotNet, на машине, имеющей следующие характеристики: Ubuntu 20.4, Intel Core i7-4790 CPU, 3.60GHz, DDR4 32GB RAM и GeForce GTX 2070, 8GB GDDR6, 1410 MHz.

Замеры времени выполнения алгоритма проведены из нулевой вершины для каждой матрицы. Ввиду влияния выбора исходной вершины на количество действий, совершаемых алгоритмом.

Источник матриц для экспериментов —https://sparse.tamu.edu/. (Дата обращения 7.12.2022)

4.3. Результаты эксперимента

Результаты эксперимента приведены в таблице 2.

Таблица 2: Результаты измерений.
Среднее время ± стандартное отклонение, мс.

Dataset	GraphBLAS-sharp	GraphBLAST	SuiteSparse
coAuthorsCiteseer	789.6 ± 13.3	4.2 ± 0.1	7.0 ± 0.1
hollywood-2009	4040.4 ± 27.8	52.1 ± 0.1	26.5 ± 0.4
roadNet-CA	16791.3 ± 97.3	149.2 ± 0.1	162.1 ± 5.4

4.4. Анализ

Из приведенных в таблице 2 результатов видно, что реализация алгоритма поиска в ширину в GraphBLAS-sharp заметно медленнее аналогов. Наблюдается проигрыш в производительности в 113 и 188 раз на матрице меньшего размера, с ростом матрицы проигрыш сокращается до 103 и 112 раз по сравнению с SuiteSparse и GraphBLAST, соответственно.

4.5. Выводы

Текущая реализация алгоритма поиска в ширину в GraphBLAS-sharp требует оптимизации.

Заключение

В ходе данной работы были выполнены следующие задачи.

- **Выполненные задачи.**
  - Реализованы все необходимые векторные операции.
  - Проведено сравнение производительности алгоритма поиска в ширину с аналогами.
- **Дальнейшая работа.** В ближайшее время будет проведено профилирование и последующая оптимизация алгоритма поиска в ширину. После чего предполагается продолжить работу над проектом для реализации основных графовых алгоритмов.

Код работы доступен в репозитории на сервисе GitHub. Имя пользователя: IgorErin.

Репозиторий проекта GraphBLAS-sharp — https://github.com/YaccConstructor/GraphBLAS-sharp (дата обращения: 7.12.2022)

Список литературы

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

[2] 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. URL: https://www.sciencedirect.com/science/article/pii/S1877050915011618.

[3] 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.

[4] Yang Carl, Buluç Aydın, 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. — 51 p. — URL: https://doi.org/10.1145/3466795.

[5] Артем Черников. Реализация операций линейной алгебры на графическом процессоре с использованием F#. — 2021. — URL: https://github.com/YaccConstructor/articles/blob/master/2021/diploma/Artem%20Chernikov/report/Chernikov-report.pdf.

[6] Кирилл Гарбар. Сложение разреженных матриц с использованием Brahma.FSharp. — 2022. — URL: https://github.com/YaccConstructor/articles/blob/master/2022/diploma/Kirill%20Garbar/text/Garbar-report.pdf.