Санкт-Петербургский государственный университет

Кафедра системного программирования

Группа 21.Б10-мм

Патов Артемий Сергеевич

Произведение Кронекера в GraphBLAS-sharp

Отчёт по учебной практике
в форме «Производственное задание»

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

Санкт-Петербург
2023

Оглавление

1. Введение

Использование линейной алгебры — один из способов высокопроизводительного анализа графов. Графы представляются в виде матриц смежности, которые, зачастую, имеют небольшую плотность, в таком случае, в памяти компьютера их удобно представлять в виде разреженных матриц, используя координатный[1] или CSR[2] форматы. Множество преобразований графов выражаются в виде операций над соответствующими им матрицами смежности. Стандарт GraphBLAS[3] определяет базовые структуры и функции для вычисления данных операций.

Одна из базовых операций спецификации — произведение Кронекера[4] — используется для генерации графов, например, в Графах Кронекера [3] — конструкции, которая генерирует множество графов, принимая на вход начальный граф и повторно применяя произведения Кронекера, сохраняя результат на каждой итерации. Помимо этого, произведение Кронекера используется в алгоритмах для выполнения контекстно-свободных запросов [1, 5] при построении пересечения двух автоматов.

Несмотря на явное наличие сферы применения, произведение Кронекера отсутствует во многих реализациях стандарта GraphBLAS. Причиной этому, вероятно, служит сложность размещения в памяти результирующей матрицы, ведь ее размер равен произведению размеров входных матриц. А из-за того, что операции кастомизируются пользователем, априори неизвестно, каков будет результат при оперировании с неявными нулями. Это сильно усложняет подсчет позиций элементов в результирующей матрице.

GraphBLAS-sharp[5] — одна из реализаций спецификации GraphBLAS

[1]COO формат: https://en.wikipedia.org/wiki/Sparse_matrix#Compressed_sparse_row_(CSR,_CRS_or_Yale_format). (Дата обращения: 15.05.2023)

[2]CSR формат: https://en.wikipedia.org/wiki/Sparse_matrix#Coordinate_list_(COO). (Дата обращения: 15.05.2023)

[3]Спецификация GraphBLAS: https://graphblas.org/. (Дата обращения: 15.05.2023)

[4]Произведение Кронекера: https://en.wikipedia.org/wiki/Kronecker_product. (Дата обращения: 15.05.2023)

[5]Репозиторий проекта GraphBLAS-sharp: https://github.com/yaccConstructor/graphBLAS-sharp. (Дата обращения: 15.05.2023)

с использованием параллельных вычислений на графическом процессоре на языке F#. Утилизация графического процессора достигация посредством трансляции исходного кода в OpenCL C при помощи библиотеки Brahma.FSharp[6]. Ранее в GraphBLAS-sharp уже было реализовано множество векторных и матричных операций, в рамках этой работы предполагается пополнить существующий набор, а также на основе него реализовать произведение Кронекера.

[6]Репозиторий библиотеки Brahma.FSharp: https://github.com/YaccConstructor/Brahma.FSharp. (Дата обращения: 15.05.2023)

2. Постановка задачи

Цель данной работы — реализовать произведение Кронекера для разреженных матриц в формате CSR для GraphBLAS-sharp с использованием библиотеки Brahma.FSharp.

Для достижения цели были поставлены следующие задачи:

1. Реализовать произведение Кронекера для CSR формата.
2. Сравнить производительность реализованного алгоритма с аналогом из SuiteSparse[7].

[7]Описание проекта SuiteSparse: https://people.engr.tamu.edu/davis/suitesparse.html. (Дата обращения: 15.05.2023)

3. Обзор

В данном разделе даётся формальное определение операции, вокруг которой строится текущая работа, проводится анализ реализаций в других библиотеках, обозреваются предыдущие результаты и приводится описание используемой библиотеки.

3.1. Произведение Кронекера

Для двух матриц $\mathbf{A} = [a_{i,j}]$ и $\mathbf{B} = [b_{i,j}]$ размерами $n \times m$ и $n' \times m'$ соответственно произведение Кронекера — это блочная матрица $\mathbf{C}$ размером $(n \cdot n') \times (m \cdot m')$, определяющаяся следующим образом:

$$\mathbf{C} = \mathbf{A} \otimes \mathbf{B} = \begin{pmatrix} a_{1,1}\mathbf{B} & a_{1,2}\mathbf{B} & \dots & a_{1,m}\mathbf{B} \\ a_{2,1}\mathbf{B} & a_{2,2}\mathbf{B} & \dots & a_{2,m}\mathbf{B} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1}\mathbf{B} & a_{n,2}\mathbf{B} & \dots & a_{n,m}\mathbf{B} \end{pmatrix}.$$

3.2. Предыдущие результаты

- **Представление матрицы.** Ранее в проекте были поддержаны различные форматы хранения разреженных матриц. Матрица, записанная в координатном формате (COO) представляется в виде трех массивов одинакового размера: массива строк, массива столбцов и массива значений. Каждому значению из последнего массива соответствует один номер строки и один номер столбца из предыдущих двух. Формат CSR (CSC) позволяет значительно сжать массив строк (столбцов), именно в таком формате алгоритм принимает матрицы на вход. Построчный формат (LIL[8]) позволяет хранить матрицу в виде списка строк, которые, в свою очередь, имеют тип `Sparse<'a> Option`, то есть представляются в виде разреженного вектора.

[8]Построчный формат хранения матриц: https://en.wikipedia.org/wiki/Sparse_matrix#List_of_lists_(LIL). (Дата обращения: 15.05.2023)

- **Поэлементные операции.** Тип бинарных поэлементных операций, принимаемых на вход алгоритмом:

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

  Тип элементов `Option` решает проблему неявных нулей [4]: позволяет явно различать ноль-значение и отсутствие значения. При поэлементных операциях неявный ноль представляется типом `None`.

- **Merge Path [2].** Ранее в проекте был реализован алгоритм Merge Path, позволяющий эффективно сливать две COO матрицы в одну. При этом данная реализация позволяет сливать пересекающиеся матрицы: на случай, если в обеих матрицах элементы имеют одинаковые индексы, выделяется два массива, оба элементы сохраняются.

- **Битонная сортировка**[9]**.** Ранее в проекте была реализована битонная сортировка — параллельный алгоритм сортировки битонных последовательностей — последовательностей, которые сначала монотонно не убывают, а затем монотонно не возрастают.

3.3. Реализации Произведения Кронекера в других библиотеках разреженной линейной алгебры

Были изучены различные библиотеки, реализующие GraphBLAS API, такие как: GraphBLAST, SuiteSparse и библиотека разреженной линейной алгебры CUSP. Из данных библиотек произведение Кронекера было реализовано лишь в SuiteSparse. Однако SuiteSparse позволяет производить вычисления только на CPU. Помимо этого, бинарные поэлементные операции, поддерживаемые SuiteSparse, не решают проблему неявных нулей.

[9]Битонная сортировка: https://en.wikipedia.org/wiki/Bitonic_sorter. (Дата обращения: 15.05.2023)

3.4. Используемая библиотека Brahma.FSharp

Brahma.FSharp — библиотека, написанная на языке F#, которая предоставляет возможность исполнения параллельных вычислений на графическом процессоре при помощи трансляции квазицитат[10] в OpenCL C.

[10] F# Quotations: https://learn.microsoft.com/en-us/dotnet/fsharp/language-reference/code-quotations. (Дата обращения: 15.05.2023)

4. Реализация

В данном разделе представлено описание решения поставленной задачи и возникшие при разработке проблемы. Код работы доступен на сервисе GitHub[11]. Имя пользователя: artemiipatov.

На вход подаются две матрицы: **A** и **B** в формате CSR, элементы которых имеют тип `'a` и `'b` соответственно, и бинарная поэлементная операция, имеющая вид

`'a option → 'b option → 'c option`.

Результирующая матрица представляется в формате COO, тип хранимых значений: `'c`.

В ходе работы было реализовано три алгоритма, их описание и сравнение представлены ниже.

4.1. CSR → CSR → COO с использованием алгоритма Merge Path

Одна из идей реализации произведения Кронекера предполагает итеративное построение результирующей матрицы **C**: на каждом из этапов правая матрица умножается на один из элементов левой, полученная матрица сливается с результирующей при помощи Merge Path. Такой алгоритм с одной стороны позволяет работать с бинарными операциями, которые при оперировании с неявными нулями, могут вернуть ненулевой элемент, ведь в данном алгоритме не используется знание о размере матрицы **C** из-за аллокации новой матрицы на каждом шаге при слиянии. С другой стороны, из-за частого копирования результирующей матрицы, этот алгоритм неэффективен по времени и по памяти.

[11]Репозиторий проекта: https://github.com/yaccConstructor/graphBLAS-sharp (дата обращения: 15.05.2023)

4.2. CSR → CSR → LIL с конвертацией исходных матриц в LIL

Построчный формат хранения матриц (LIL) довольно удобен для написания алгоритма, так как, во-первых предоставляет возможность хранения нулевых матриц: в таком случае все строки будут иметь тип `None`. Во-вторых, в случае с произведением Кронекера, он позволяет строить результирующую матрицу построчно, умножая поэлементно вектор на скаляр и сливая векторы. Сливать векторы гораздо эффективнее по памяти, ведь в данном случае приходится переаллоцировать лишь малую часть матрицы. Однако конвертация исходных матриц из CSR в LIL и слишком частое слияние двух векторов влечет за собой неэффективность по времени.

4.3. CSR → CSR → COO с преподсчетом размера

В данном алгоритме размер итоговой матрицы предвычисляется. Аллоцируется битовая карта, размер которой равен размеру правой матрицы $\mathbf{B}$; ненулевые элементы матрицы по очереди умножаются на матрицу $\mathbf{B}$, значение ячейки (i, j) битовой карты инкрементируется, если результат умножения элемента левой матрицы $\mathbf{A}$ на элемент $\mathbf{B}[i, j]$ имеет тип `Some`. Элементы итоговой битовой карты складываются, результат суммы есть размер результата произведения Кронекера. Стоит отметить, что результат бинарной операции с двумя неявными нулями вычисляется отдельно, и, если имеет тип `Some`, то произведение количеств нулевых элементов матриц $\mathbf{A}$ и $\mathbf{B}$ прибавляется к размеру.

Матрица нужного размера аллоцируется в координатном формате, затем правая матрица поочередно умножается на ненулевые элементы левой, результаты вставляются в итоговую матрицу. Умножение матрицы $\mathbf{B}$ на неявный ноль происходит всего один раз, если в результате получается ненулевая матрица, то ее элементы в нужные моменты вставляются в результирующую матрицу. После всех преобразований,

индексы элементов образуют битонные последовательности, которые можно отсортировать битонной сортировкой.

4.4. Вывод

Алгоритм	**Среднее, мс**	**Среднеквадратичное отклонение, мс**
С использованием Merge Path	809	22
С использованием формата LIL	5375	80
С преподсчетом размера	185	6

Таблица 1: Матрица[12] размера 47. Типы значений: `float32`. Бинарная операция: сложение.

В результате, для итоговой реализации был выбран алгоритм с преподсчетом размера матрицы, так как рассчет размера, хоть и требует дополнительных вычислений, позволяет провести аллокацию матрицы в самом начале и обойтись без переаллокаций в будущем. Засчет чего последний алгоритм эффективнее предыдущих двух как по памяти, так и по времени, что видно из замеров, представленных в таблице 1.

5. Эксперименты

В данном разделе представлены сравнительные эксперименты.

5.1. Цель эксперимента

Целью является сравнение производительности реализаций произведения Кронекера из библиотек GraphBLAS-sharp и SuiteSparse.

5.2. Условия эксперимента

Оперируемые матрицы представлены в таблице 3.

Матрица	**Размер**	**Количество ненулевых элементов**
can_634	634	3931
Si2	769	9285
lshp1561	1561	6121

Таблица 2: Набор матриц.

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

5.3. Результаты замера

Результаты замера приведены в таблице 3

Dataset	**GraphBLAS-sharp**	**SuiteSparse**
can_634	7242 ± 467	51 ± 2
Si2	19562 ± 801	279 ± 2
lshp1561	22674 ± 1300	123 ± 3

Таблица 3: Среднее время $\pm$ Среднеквадратичное отклонение, мс. Бинарная операция: умножение

5.4. Анализ результатов

Замеры показали, что реализация произведения Кронекера в GraphBLAS-sharp заметно медленнее аналога из SuiteSparse. На матрице с меньшим количеством ненулевых элементов наблюдается проигрыш в 139 раз. Однако с увеличением количества ненулевых элементов проигрыш уменьшается до 70 по сравнению с SuiteSparse.

5.5. Вывод

Реализация произведения Кронекера в GraphBLAS-sharp требует оптимизации.

6. Заключение

В ходе данной работы были выполнены следующие задачи.

- Реализовано произведение Кронекера в библиотеке GraphBLAS-sharp[13].
- Проведено сравнение реализованного алгоритма с аналогом из SuiteSparse.

Дальнейшая работа.

- В ближайшее время будет проведено профилирование, а затем оптимизация алгоритма произведения Кронекера.

[13]PR: https://github.com/YaccConstructor/GraphBLAS-sharp/pull/79. (Дата обращения: 16.05.2023)

Список литературы

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

[2] Green Oded, Mccoll Rob, Bader David. GPU merge path: a GPU merging algorithm. — 2012. — 06.

[3] Kronecker Graphs: An Approach to Modeling Networks / Jure Leskovec, Deepayan Chakrabarti, Jon Kleinberg et al. // Journal of Machine Learning Research. — 2008. — 12. — Vol. 11.

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

[5] Орачев Егор Станиславович. Реализация алгоритма поиска путей в графовых базах данных через тензорное произведение на GPGPU. — 2021. — URL: https://github.com/YaccConstructor/articles/blob/master/2021/diploma/Egor%20Orachev/Bachelor%20Thesis/kronecker_cfpq_gpu_diploma.pdf.