Санкт-Петербургский государственный университет

Кафедра системного программирования

Группа 20Б.07-мм

Гарбар Кирилл Анатольевич

Сложение разреженных матриц 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
2022

Оглавление

Введение

Одной из самых известных структур данных информатики является граф. Графам находится применение в многих естественных науках, таких как биология [10], физика [6] и химия [2]. Помимо естественных наук, графы пользуются в различных областях математики — теории групп [1], топологии [12], теории вероятностей [3]. Алгоритмы над графами получили широкое распространение в анализе социальных сетей. [8].

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 [7]. Алгоритмы обработки графов с матрицами смежности, написанные на языке линейной алгебры, были широко освещены в литературе [5].

Развитие идей о связи графов и разреженной линейной алгебры послужило толчком к созданию стандарта GraphBLAS[1][9], определяющего алгоритмы обработки графов на языке линейной алгебры с использованием разреженных матриц и векторов, заданных над полукольцами. Ярким примером таких алгоритмов является алгоритм вычисления достижимости пар вершин графа[2]. Для этого булева матрица смежности $A \in \mathbb{B}^{n \times n}$ возводится в степень. Операциями сложения и умножения в булевом полукольце являются логическое ИЛИ и И соответственно. Таким образом, ячейка (i, j) матрицы A^l отвечает за наличие пути из вершины i в вершину j с длиной не более, чем l.

Граф, содержащий небольшое количество дуг относительно количе-

[1]Описание GraphBLAS и ссылками на связанные материалы — https://graphblas.org/ (дата обращения: 20.05.2022)

[2]Описание алгоритмов над графами через марицы смежности — https://users.math-cs.spbu.ru/~okhotin/teaching/algorithms_2020/okhotin_algorithms_2020_l5.pdf (дата обращения: 20.05.2022)

ства вершин, разумно представлять в виде разреженной, а не плотной матрицы. Одним из форматов разреженных матриц, активно используемых в GraphBLAS, является CSR[3] формат. Данный формат представляет матрицу в виде трёх массивов. Первый массив, называемый массивом значений, хранит все ненулевые[4] элементы матрицы. Второй массив — массив столбцов — в ячейке с номером i хранит номер столбца элемента с индексом i в массиве значений. Наконец, третий массив, в ячейке с номером i количество ненулевых элементов, содержащихся в строках с номером до $i-1$ включительно. Помимо CSR формата, используется также координатный формат. Первые два массива координатного представления матрицы идентичны массивам значений и столбцов из CSR формата, а третий — массив строк — аналогичен массиву столбцов.

С момента создания стандарта GraphBLAS появилось множество его реализаций. На данный момент, полной и образцовой реализацией выступает SuiteSparse:GraphBLAS[5][4]. Помимо SuiteSparse существуют также GraphBLAST[6][11], CUSP[7] и pggraphblas[8]. Многие такие проекты работают с некоторыми ограничениями. Например, в SuiteSparse вычисления происходят на центральных процессорах, без поддержки параллельных вычислений на графических ускорителях, имеющих в данной области большой потенциал. Платформа OpenCL[9], в свою очередь, предоставляет поддержку как центральных, так и различных графических процессоров. Следует также заметить, что практически все реализации GraphBLAS написаны на языках C и C++, и реализация на более высокоуровневом языке программирования, таком как F#,

[3]Описание различных форматов разреженных матриц — https://en.wikipedia.org/wiki/Sparse_matrix (дата обращения: 20.05.2022)

[4]В разреженных матрицах, как правило, не хранят нейтральный по сложению элемент полукольца

[5]Описание SuiteSparse и ссылки на связанные материалы — https://people.engr.tamu.edu/davis/suitesparse.html (дата обращения: 20.05.2022)

[6]Репозиторий проекта GraphBLAST — https://github.com/gunrock/graphblast (дата обращения: 20.05.2022)

[7]Репозиторий проекта CUSP — http://cusplibrary.github.io/index.html (дата обращения: 20.05.2022)

[8]Репозиторий проекта pggraphblas — https://github.com/michelp/pggraphblas (дата обращения: 20.05.2022)

[9]Описание платформы OpenCL — https://www.khronos.org/opencl/ (дата обращения: 20.05.2022)

может иметь ряд преимуществ, таких как гибкая система типов, обилие статических проверок кода и функции высших порядков. В новых версиях C++ реализуются некоторые из приведённых приемуществ, но в F# всё это было практически с момента его создания и поэтому реализовано более качественно. Проектом, стремящимся реализовать GraphBLAS на языке F# с поддержкой платформы OpenCL является GraphBLAS-sharp[10].

Множество реализаций и растущая популярность не могли не породить множество дискуссий. Одной из активно обсуждаемых проблем, связанных с GraphBLAS, является проблема явных нулей[11]. Проблема связана с разреженными матрицами, а именно с тем, что такие матрицы, как правило, не хранят нулевой элемент полукольца, над которыми они заданы. Такие нули называются неявными. Однако, в результате операций полукольца, в матрице может оказаться нулевое значение, и, в зависимости от ситуации, может либо понадобиться сохранить его в матрице как явный ноль, либо удалить. Кроме того, иногда явный и неявный ноль могут иметь совершенно различную природу. Например, в задачах, связанных со взвешенными графами, неявный ноль может означать отсутствие ребра между двумя вершинами, а явный — ребро с нулевым весом. Существует несколько подходов к решению данной проблемы. К примеру, в SuiteSparse для удаления явных нулей из матрицы используется отдельный метод. Недостатками такого способа может выступать потеря в производительности и банальное неудобство. Этого можно избежать, если операции, работающие с элементами матрицы, будут работать над типами, которые могут принимать как нулевое значение, которое будет являться явным, так и никакое значение вовсе, которое будет неявным нулём. В языке F# такое можно естественным образом реализовать с помощью option типов.

[10]Репозиторий проекта GraphBLAS-sharp — https://github.com/YaccConstructor/GraphBLAS-sharp (дата обращения: 20.05.2022)

[11]Обсуждение проблемы явных нулей с одним из основоположников стандарта GraphBLAS — https://github.com/GraphBLAS/LAGraph/issues/28 (дата обращения: 5.06.2022)

1. Постановка задачи

Целью данной работы является реализация операций разреженной линейной алгебры для GraphBLAS-sharp с использованием библиотеки Brahma.FSharp

Были сформулированы следующие задачи.

- Реализовать поэлементное сложение разреженных матриц, представленных в CSR формате.
- Реализовать поддержку option типов в операции сложения элементов матриц, и предоставить возможность сложения матриц разных типов.
- С использованием различных графических процессоров произвести сравнение производительности полученного сложения матриц с SuiteSpase и CUSP, а также с Math.NET Numerics и с уже реализованным в GraphBLAS-sharp сложением в координатном формате.

2. Обзор

Обзор некоторых из существующих реализаций GraphBLAS, предыдущих результатов и библиотек, используемых в работе, представлен в данном разделе.

2.1. Существующие реализации GraphBLAS

Аналоги, с которыми происходит сравнение, были отобраны по следующим критериям:

- Присутствие ссылки на проект в репозитории[12], содержащем ссылки на связанные с GraphBLAS материалы.
- Наличие в поисковой выдаче Google по запросам “GraphBLAS” и “Sparse linear algebra”.

Критерии по которым аналоги сравниваются между собой, следующие:

- Поддержка платформы для параллельных вычислений.
- Язык программирования.

Результаты сравнения приведены в таблице 1.

Название	CPU	GPU Nvidia	GPU AMD	Язык
SuiteSparse	Да	Нет	Нет	C
GraphBLAST	Нет	Да	Нет	C++
CUSP	Нет	Да	Нет	C++
pggraphblas	Да	Нет	Нет	C

Таблица 1: Сравнение существующих решений

Как видно в таблице 1, не все реализации поддерживают вычисления на графических процессорах, а те, которые поддерживают, делают это с некоторыми ограничениями, такими как отсутствие поддержки центральных процессоров или поддержка графических процессоров

[12]Репозиторий с ссылками на материалы про GraphBLAS — https://github.com/GraphBLAS/GraphBLAS-Pointers (дата обращения: 20.05.2022)

только одной из фирм производителей. Также, все такие реализации написаны на языках C и C++, не обладающих гибким высокоуровневым интерфейсом.

2.2. Математические библиотеки с поддержкой операций линейной алгебры

В связи с тем, что в GraphBLAS-sharp реализованы некоторые операции линейной алгебры, принято решение произвести сравнение производительности с одной из математических библиотек платформы .NET — Math.NET Numerics[13]. Math.NET Numerics выбран как одна из самых популярных математических библиотек платформы .NET, используемая в MATLAB[14] и множестве других проектов[15].

2.3. Используемая библиотека Brahma.FSharp

Поддержка платформы OpenCL, используемой в GraphBLAS-sharp, реализована на множестве платформ разных видов и производителей. Возможность писать код на высокоуровневом языке, исполняемый на платформе OpenCL, предоставляется библиотекой Brahma.FSharp[16], 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. Примитивом синхронизации в данном случае выступает очередь, реализуемая с помощью MailboxProcessor[17].

[13]Описание Math.NET Numerics — https://numerics.mathdotnet.com/ (дата обращения: 20.05.2022)

[14]Описание MATLAB — https://www.mathworks.com/products/matlab.html (дата обращения: 20.05.2022)

[15]Проекты, использующие Math.NET Numerics — https://numerics.mathdotnet.com/Users.html (дата обращения: 20.05.2022)

[16]Репозиторий проекта Brahma.FSharp — https://github.com/YaccConstructor/Brahma.FSharp (дата обращения: 20.05.2022)

[17]Описание MailboxProcessor — https://fsharp.github.io/fsharp-core-docs/reference/fsharp-control-fsharpmailboxprocessor-1.html (дата обращения: 20.05.2022)

2.4. Предыдущие результаты

Для GraphBLAS-sharp уже была реализована операция поэлементного сложения разреженных матриц в координатном формате и префиксная сумма, результаты этой[18] работы будут использованы для рещения одной из поставленных задач — поэлементного сложения матриц в CSR формате.

[18]Работа в которой было реализовано сложение координатных матриц — https://github.com/YaccConstructor/articles/blob/master/2021/diploma/Artem%20Chernikov/report/Chernikov-report.pdf (дата обращения: 20.05.2022)

3. Поэлементное сложение матриц

Описание решения первой из поставленных задач, а именно поэлементного сложения матриц в CSR формате, представлено в данном разделе.

В GraphBLAS-sharp CSR матрица реализована как запись $CSRMatrix$, содержащая количесвто строк и столбцов — $RowCount$ и $ColumnCount$ соответственно, и трёх массивов — $Values$, $Columns$ и $RowPointers$ — аллоцируемых в памяти устройства, на котором будут производиться вычисления.

3.1. Реализация сложения

Для сложения был выбран стандартный алгоритм, который также используется в CUSP. Три этапа, составляющие данный алгоритм, следующие.

1. Конвертация матрицы из CSR формата в координатный.
2. Поэлементное сложение матриц в координатном формате.
3. Конвертация результирующей матрицы обратно в CSR формат.

Массивы значений и столбцов для координатного и CSR форматов идентичны, а по массиву, содержащему количество ненулевых элементов в строках матрицы, естественным образом получается массив строк, аналогичный массиву столбцов. Таким образом, для конвертации матрицы в координатный формат, массивы $Values$ и $Columns$ копируются, а количество ненулевых элементов в i-ой строке матрицы определяется как $RowPointers[i+1] - RowPointers[i]$. На каждый ненулевой элемент в строке i в результирующий массив кладётся i, притом таким образом, что массив отсортирован по неубыванию. Затем матрицы, представленные в координатном формате, складываются по уже реализованному алгоритму.

Для обратной конвертации массивы $Values$ и $Columns$ копируются, а $Rows$ необходимо преобразовать в $RowPointers$. Для этого считается битовая карта уникальных элементов массива строк — $bitmap$. На i-ой позиции в $bitmap$ стоит 0 тогда и только тогда, когда $bitmap[i] = bitmap[i + 1]$. Затем, по массиву $bitmap$ считается исключающая префиксная сумма. Получается массив и значение суммы — $positions$ и $totalSum$ соответственно. Массив $positions$ для каждого элемента $Rows$ хранит порядковый номер этого элемента как уникального. После этого создаются два массива длины $totalSum$ — $nonZeroRowsIndices$ и $nonZeroRowsPointers$. Первый содержит номера непустых строк матрицы, а второй — в ячейке i хранит количество ненулевых элементов в строках до i включительно. Для этого элемент с индексом i массива $Rows$ кладётся в $nonZeroRowsIndices[positions[i]]$, а $i + 1$ — в $nonZeroRowsPointers[positions[i]]$. Затем считается количество элементов в каждой непустой строке, полученные числа выставляются на соответствующие им места в результирующем массиве и с помощью включающей префиксной суммы конвертация завершается.

4. Option и сложение матриц разных типов

Ранее тип операции сложения был следующий: $typeA \rightarrow \ typeA \rightarrow typeA$, то есть по двум элементам одинакового типа получался третий элемент того же типа. Благодаря внедрению поддержки union типов в транслятор, частным случаем которых являются option типы, теперь можно задать операцию с сигнатурой $Option\ typeA \rightarrow Option\ typeA \rightarrow Option\ typeA$. Благодаря этому, пользователь, задающий бинарную операцию над элементами матриц, сможет сам решить, стоит ли сохранять получившиеся нули в явном виде, или нет. Следующим этапом повышения гибкости операции сложения стала возможность складывать матрицы разных типов. К примеру, написав соответствующую операцию сложения, теперь можно произвести сложение по маске или другие, более экзотические операции. Таким образом, сигнатура функции сложения стала выглядеть так: $Option\ typeA \rightarrow Option\ typeB \rightarrow Option\ typeC$. Однако, метод с такой сигнатурой не защищает пользователя от ошибки, ведь можно написать операцию, возвращающую по двум неявным нулям явное значение. В таком случае, ожидаемый результат будет далёк от реального, ведь разреженные матрицы не хранят неявные нули. Чтобы этого избежать, был введён новый тип — $AtLeastOne<typeA, typeB>$, объявление типа приведено в листинге 1. Данный тип гаранитурет, что среди пары элементов хотя бы один имеет явное значение. Итак, результатом преобразований стала сигнатура: $AtLeastOne<typeA, typeB> \rightarrow Option\ typeC$.

```
type AtLeastOne<'typeA, 'typeB when 'typeA: struct and 'typeB:
    struct> =
    | Both of 'typeA * 'typeB
    | Left of 'typeA
    | Right of 'typeB
```

Листинг 1: Объявление типа AtLeastOne

4.1. Реализация поддержки сложения матриц разных типов и option

Для сложения матриц разных типов достаточно внести некотрые изменения в алгоритм сложения матриц в координатном формате. Первым этапом данного сложения является слияние трёх массивов, представляющих первую матрицу, с соответствующими массивами второй матрицы. Благодаря этому, если в обеих матрицах на одной и той же позиции хранится ненулевой элемент, в результате слияния эти элементы окажутся в соседних ячейках получившихся массивов. Так как теперь суммируемые матрицы могут хранить элементы различных типов, вместо одного массива значений в результате слияния получается два массива, хранящих элементы одного типа. В случае, когда на одной позиции элемент есть только в одной из матриц, необходимо знать, в каком из двух массивов после слияния он окажется. Для этого создаётся битовая карта, определяющий взят ли этот элемент из первой матрицы.

Следующим этапом является сложение пар элементов, получившихся в результате слияния. В случае, когда на одной позиции и в первой, и во второй матрице есть значение, всё остаётся по-прежнему. В случае, если элемент есть только в первой матрице, он складывается с неявным нулём правой матрицы, иначе элемент складывается с неявным нулём левой матрицы. В случае, когда результатом операции стал неявный ноль, никакое значение не кладётся в результирующий массив значений.

5. Сравнительные эксперименты

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

5.1. Постановка эксперимента

Для постановки экспериментов в GraphBLAS-sharp был создан отдельный проект под названием GraphBLAS-sharp.Benchmarks, в основе которого лежит инструмент BenchmarkDotNet[19]. В проекте реализована загрузка матриц в формате Matrix Market[20] 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[21]. Были отобраны матрицы разной степени разреженности и размера. Для каждой матрицы также заранее был посчитан её квадрат, с которым и складывалась исходная матрица. Выбранные матрицы приведены в таблице 2.

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 сложения матриц происходили следующим образом. Сперва матрица и её квадрат загружались из файлов как координатные матрицы, которые затем преобразовывались в формат CSR или COO, в зависимости от эксперимента. После этого для каждой матрицы происходило десять разогревочных итераций эксперимента, за которыми следовало сто замеряемых итераций, в результате которых высчитывалось среднее значение и стандартное отклонение времени выполнения операции сложения.

[19]Обзор инструмента BenchmarkDotNet — https://benchmarkdotnet.org/articles/overview.html (дата обращения: 20.05.2022)

[20]Описание Matrix Market формата — https://math.nist.gov/MatrixMarket/formats.html (дата обращения: 20.05.2022)

[21]Источник матриц для экспериментов — https://sparse.tamu.edu/ (дата обращения: 20.05.2022)

Название	Размер	Количество ненулевых элементов	Количество ненулевых элементов у возведённой в квадрат
wing	62 032	243 088	714,200
luxembourg_osm	114 599	119 666	4 582
amazon0312	400 727	3 200 440	14 390 544
amazon-2008	735 323	5 158 388	25 366 745
web-Google	916 428	5 105 039	30 811 855
webbase-1M	1 000 005	3 105 536	51 111 996
cit-Patents	3 774 768	16 518 948	469

Таблица 2: Матрицы, на которых производилось сравнение

Название	GraphBLAS-sharp	SuiteSparse	CUSP
wing	$1,8 \pm 0,1$	$1,9 \pm 0,1$	$0,5 \pm 0,2$
luxembourg_osm	$2,9 \pm 0,3$	$1.9 \pm 0,5$	$0,5 \pm 0,1$
amazon0312	$17,0 \pm 0,8$	$28,9 \pm 0,2$	$2,8 \pm 0,1$
amazon-2008	$12,2 \pm 0,8$	$50,1 \pm 2,4$	$3,5 \pm 0,1$
web-Google	$18,4 \pm 0,6$	$58.8 \pm 0,7$	$3,6 \pm 0,1$
webbase-1M	$70,7 \pm 1,0$	$72,9 \pm 0,4$	$24,6 \pm 2,1$
cit-Patents	$54,6 \pm 1,2$	$157,4 \pm 1,2$	$8,5 \pm 1,2$

Таблица 3: Результаты сравнения библиотек на сложение в CSR формате, GTX 2070, среднее $\pm$ стандартное отклонение, мс

5.2. Результаты сравнения с SuiteSparse и CUSP

Наиболее интересным кандидатом для сравнения выступает SuiteSparse, ведь он является эталонной реализацией GraphBLAS. Также, для сравнения был выбран CUSP, так как алгоритмы сложения в GraphBLAS-sharp и CUSP аналогичны. Результаты замеров времени сложения приведены в таблице 3

Из результатов видно, что полученное решение выигрывает в производительности вплоть до четырёх раз у SuiteSparse практически на всех матрицах. По сравнению с CUSP наблюдается проигрыш от трёх до семи раз.

Название	GraphBLAS-sharp	Math.NET Numerics
wing	$1,8 \pm 0,1$	$5,5 \pm 0,2$
luxembourg_osm	$2,9 \pm 0,3$	$286,2 \pm 2,2$
amazon0312	$17,0 \pm 0,8$	–
amazon-2008	$12,2 \pm 0,8$	–
web-Google	$18,4 \pm 0,6$	–
webbase-1M	$70,7 \pm 1,0$	–
cit-Patents	$54,6 \pm 1,2$	–

Таблица 4: Результаты сравнения библиотек на сложение в CSR формате, GTX 2070, среднее $\pm$ стандартное отклонение, мс. Отсутствие данных означает, что среднее время превышает 100 секунд

5.3. Результаты сравнения с Math.NET Numerics

Одной из наиболее используемых математических библиотек с поддержкой операций разреженной линейной алгебры на платформе .NET является Math.NET Numerics. Math.NET Numerics также поддерживает использование провайдеров алгоритмов для ускорения работы требующих производительности операций, поэтому для сложения матриц был использован Intel Math Kernel Library[22]. Для хранения матриц в Math.NET Numerics используется CSR формат. Сравнение производительности CSR форматов в GraphBLAS-sharp и Math.NET Numerics приведены в таблице 4

По результатам можно сделать вывод, что Math.NET Numerics не поддерживает операции с разреженными матрицами достаточно большого размера.

5.4. Результаты сравнения CSR и COO сложения до и после внедрения option и сложения матриц разных типов

Вместе с внедрением поддержки option типов в транслятор, библиотека Brahma.FSharp претерпела множество изменений, в том числе

[22]Описание Intel MKL — https://en.wikipedia.org/wiki/Math_Kernel_Library (дата обращения: 20.05.2022)

Название	Версия без option		Версия с option	
	COO	CSR	COO	CSR
wing	$19,7 \pm 0,2$	$63,3 \pm 0,9$	$0,8 \pm 0,1$	$2,2 \pm 0,3$
luxembourg_osm	$20,5 \pm 0,3$	$62,3 \pm 1,3$	$0,9 \pm 0,2$	$2,8 \pm 0,3$
amazon0312	$27,8 \pm 0,4$	$87,9 \pm 1,6$	$5,4 \pm 0,6$	$17,1 \pm 0,7$
amazon-2008	$26,1 \pm 0,3$	$84,5 \pm 1,4$	$4,1 \pm 0,5$	$12,1 \pm 0,8$
web-Google	$27,6 \pm 0,4$	$87,7 \pm 1,2$	$5,2 \pm 0,2$	$18,3 \pm 0,8$
webbase-1M	$51,9 \pm 0,7$	$274,1 \pm 1,1$	$44,6 \pm 1,0$	$73,2 \pm 1,0$
cit-Patents	$30,4 \pm 0,5$	$145,6 \pm 1,4$	$13,9 \pm 0,7$	$54,5 \pm 0,9$

Таблица 5: Результаты сравнения сложения для координатного и CSR форматов на GTX 2070, с использованием option типов и без, среднее $\pm$ стандартное отклонение, мс

и оптимизационных. В связи с этим, а также потому, что внедрение option типов в операцию сложения предполагает некоторое изменение алгоритма сложения, было произведено сравнение старой и новой версии сложения. Результаты приведены в таблице 5

По результатам видно, что во всех случаях скорость сложения значительно возросла. Наибольшую роль в этом сыграла оптимизация библиотеки Brahma.FSharp.

5.5. Результаты сравнения координатного и CSR форматов

Как в GraphBLAS-sharp, так и в CUSP сложение в CSR формате использует сложение в координатном формате, поэтому может быть полезно сравнить время сложения в координатном и CSR форматах. Результаты приведены в таблице 6

Как видно в таблице, в CUSP разница в координатном и CSR сложении составила около двух раз, в то время как в GraphBLAS-sharp — от двух до четырёх раз.

Название	GraphBLAS-sharp		CUSP	
	COO	CSR	COO	CSR
wing	$0,8 \pm 0,1$	$1,8 \pm 0,1$	$0,2 \pm 0,1$	$0,5 \pm 0,2$
luxembourg_osm	$0,9 \pm 0,2$	$2,9 \pm 0,3$	$1,7 \pm 0,2$	$0,5 \pm 0,1$
amazon0312	$5,3 \pm 0,6$	$17,0 \pm 0,8$	$1,2 \pm 0,1$	$2,8 \pm 0,1$
amazon-2008	$4,2 \pm 0,5$	$12,2 \pm 0,8$	$1,7 \pm 0,2$	$3,5 \pm 0,1$
web-Google	$4,9 \pm 0,2$	$18,4 \pm 0,6$	$1,6 \pm 0,2$	$3,6 \pm 0,1$
webbase-1M	$44.8 \pm 1,1$	$70,7 \pm 1,0$	$14,2 \pm 1,1$	$24,6 \pm 2,1$
cit-Patents	$14,5 \pm 0,7$	$54,6 \pm 1,2$	$4,3 \pm 0,3$	$8,5 \pm 1,2$

Таблица 6: Сравнение времени сложения матриц в CSR и COO форматах для GrahBLAS-sharp и CUSP, GTX 2070, среднее ± стандартное отклонение, мс

Название	GTX 1070	GTX 2070	Vega 10 XTX
wing	$2,3 \pm 0,2$	$1,8 \pm 0,1$	$3,0 \pm 0,6$
luxembourg_osm	$3,3 \pm 0,3$	$2,9 \pm 0,3$	$3,4 \pm 0,4$
amazon0312	$19,7 \pm 0,7$	$17,0 \pm 0,8$	$22,3 \pm 1,3$
amazon-2008	$14,5 \pm 0,9$	$12,2 \pm 0,8$	$20,0 \pm 0,8$
web-Google	$21,2 \pm 0,6$	$18,4 \pm 0,6$	$26,0 \pm 1,3$
webbase-1M	$78,4 \pm 1,1$	$70,7 \pm 1,0$	$240,8 \pm 2,4$
cit-Patents	$59,8 \pm 1,1$	$54,6 \pm 1,2$	$94,7 \pm 0,1$

Таблица 7: Результаты сравнения сложения CSR матриц на разных устройствах, среднее ± стандартное отклонение, мс

5.6. Результаты сравнения на разных устройствах

Чтобы выяснить, как ведёт себя полученное решение на разных устройствах, были проведены замеры времени сложения матриц в формате CSR на GTX 2070, 8GB GDDR6, 1410 MHz и GTX 1070, 8GB GDDR5, 1506 MHz, а также на Vega 10 XTX, 8GB HBM2, 1500 MHz. Полученные результаты приведены в таблице 7

Из полученных результатов видно, что наибольшую производительность показала GTX 2070, в то время как GTX 1070 оказалась несколько медленнее. Наименьшую производительность показала Vega 10 XTX с особо существенным отставанием на больших матрицах.

Название	GraphBLAS-sharp	SuiteSparse
wing	$2,5 \pm 0,4$	$1,0 \pm 0,1$
luxembourg_osm	$2,6 \pm 0,3$	$1,4 \pm 0,3$
amazon0312	$13,0 \pm 1,0$	$23,0 \pm 0,9$
amazon-2008	$9,1 \pm 0,8$	$35,2 \pm 4,0$
web-Google	$14,7 \pm 0,8$	$43,9 \pm 0,2$
webbase-1M	$55,4 \pm 1,2$	$31,0 \pm 1,6$
cit-Patents	$47,9 \pm 0,9$	$107,9 \pm 0,4$

Таблица 8: Сравнение результатов поэлементного умножения матриц, GTX 2070, среднее $\pm$ стандартное отклонение, мс

5.7. Результаты сравнения поэлементного умножения в GraphBLAS-sharp и CUSP

Задав поэлементную операцию определённым образом, можно получить не только сложение матриц. Примером другой такой операции является поэлементное умножение, которое, в отличие от сложения, работает на пересечении множеств элементов двух матриц. В SuiteSparse для каждой из этих операций используется свой алгоритм, когда в GraphBLAS-sharp достаточно написать только новую бинарную операцию, как представлено на листинге 2. Результаты сравнения поэлементного умножения в GraphBLAS-sharp и SuiteSparse приведены в таблице 8

```
let float32Mul =
    <@ fun (values: AtLeastOne<float32, float32>) ->
        let mutable res = 0f

        match values with
        | Both (f, s) -> res <- f * s
        | _ -> res <- 0f

        if res = 0f then None else (Some res) @>
```

Листинг 2: Операция поэлементного умножения в GraphBLAS-sharp

По результатам эксперимента видно, что на большей части матриц наблюдается выигрыш в производительности, в то время как отстава-

ние на остальных матрицах не столь значительное.

5.8. Вывод

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

- Благодаря оптимизации транслятора была достигнута высокая производительности полученного решения.
- По сравнению с эталонной реализацией стандарта, использующей центральный процессор для вычислений, наблюдается выигрыш в производительности. Тем не менее, отставание от CUSP составило от трёх до семи раз.
- Сложение в CSR формате существенно отстаёт в производительности от сложения в координатном формате.
- Наилучшей производительности удалось добиться на устройствах компании Nvidia.
- Несмотря на то, что GraphBLAS-sharp написан на высокоуровневом языке платформы .NET, была достигнута приемлемая производительность даже по сравнению с реализациями на C/C++. Отсюда можно сделать вывод о конкурентоспособности полученного решения.

Заключение

В ходе данной работы были выполнены следующие задачи.

- Реализовано поэлементное сложение разреженных матриц, представленных в CSR формате.
- Реализована поддержка option типов в операции сложения элементов матриц, а также добавлена возможность сложения матриц разного типа.
- С использованием различных графических процессоров произведено сравнение производительности полученного сложения матриц с SuiteSpase и CUSP, а также с Math.NET Numerics и с уже реализованным в GraphBLAS-sharp сложением в координатном формате.

Помимо этого, была создана инфраструктура для дальнейшего проведения экспериментов по сравнению с аналогами. В дальнейшие планы входит как реализация большего количества строительных блоков для алгоритмов над графами, так и реализация самих алгоритмов.

Код доступен в репозитории на сервисе github[23]. Имя пользователя: kirillgarbar

[23]Репозиторий проекта GraphBLAS-sharp — https://github.com/kirillgarbar/GraphBLAS-sharp (дата обращения: 20.05.2022)

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] 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.

[6] Jr. G. David Forney. Codes on Graphs: Models for Elementary Algebraic Topology and Statistical Physics // IEEE Trans. Inf. Theory. — 2018. — Vol. 64, no. 12. — P. 7465–7487. — Access mode: https://doi.org/10.1109/TIT.2018.2866577.

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

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.