Санкт-Петербургский государственный университет

Кафедра системного программирования

Высокоуровневая реализация спецификации GraphBLAS с поддержкой исполнения на GPU

Панфилёнок Дмитрий Викторович, 18.Б10-мм группа

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

Санкт-Петербург
2021

Введение

- Графы могут быть представлены разреженными матрицами
- Алгоритмы на графах можно выразить в терминах линейной алгебры над различными полукольцами
- Спецификация GraphBLAS описывает структуру объектов и методов для реализации таких алгоритмов в терминах линейной алгебры

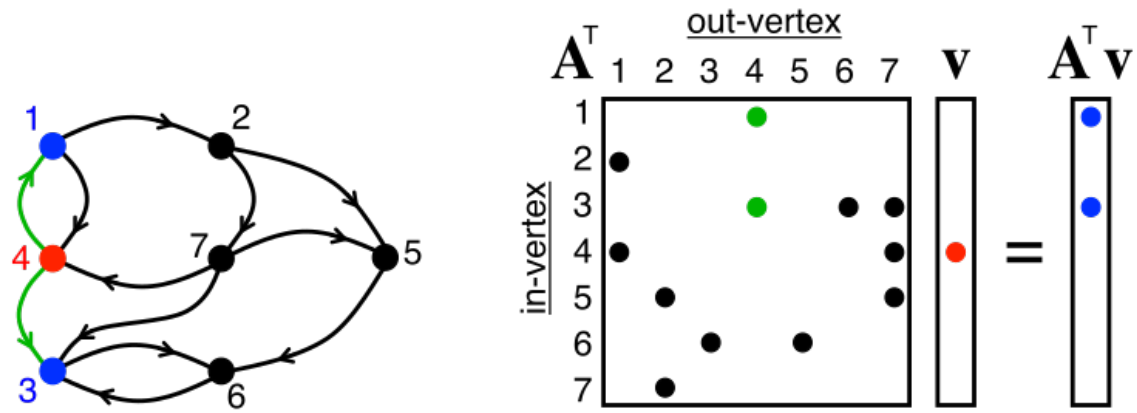

Рис.: Вычисление одного шага в алгоритме поиска в ширину[1]

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

Существующие решения

Реализация	Язык	Поддержка GPU
SuiteSparse	C	Нет
GBTL	C++	Нет (версия с CUDA устарела)
GraphBLAST	C++	CUDA
CombBLAS	C++	Частичная, CUDA
Graphulo	Java	Нет

Недостатки существующих решений:

- GraphBLAS на GPU — открытая проблема
- Низкая переносимость решений, основанных на CUDA
- Нет самостоятельных библиотек для высокоуровневых языков

Постановка задачи

Целью данной работы является реализация высокоуровневой, производительной и переносимой библиотеки на основе стандарта GraphBLAS.

Задачи:

- Реализовать структуру объектов и методов, соответствующих спецификации GraphBLAS
- Реализовать базовые примитивы линейной алгебры для выполнения на устройствах с поддержкой GPGPU
- Провести экспериментальное исследование предложенной реализации и сравнить её с аналогичными решениями в предметной области

Обзор инструментов

Программные интерфейсы для GPGPU:

- CUDA
- OpenCL
- Metal

Поддержка OpenCL в .NET и Java:

.NET	Java
OpenCL.NET	JavaCL
Cloo	JOCL
FSCL	Aparapi
Brahma.FSharp	ScalaCL

Предлагаемое решение

OpenCL

Стандарт GraphBLAS

- Матрицы, векторы
- Моноиды, полукольца, бинарные и унарные операторы
- Mxm, vxm, mxv, eWiseAdd eWiseMult, reduce и другие
- Маски, дескрипторы

Архитектура решения

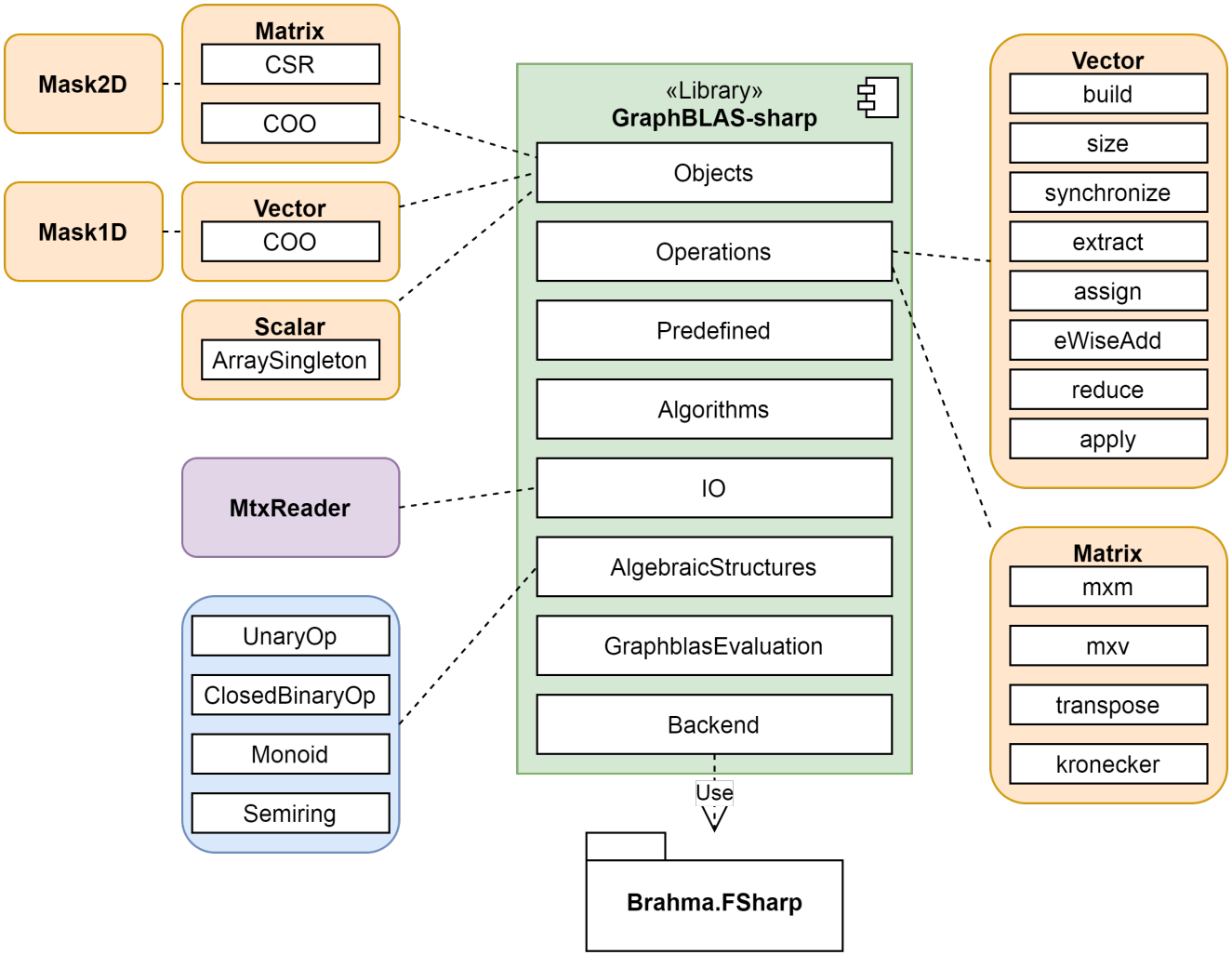

Рис.: Библиотека GraphBLAS-sharp

Выбор алгоритмов для реализации операций

- W. Liu and B. Vinter, "An Efficient GPU General Sparse Matrix-Matrix Multiplication for Irregular Data" , 2014[2]
- Y. Nagasaka, A. Nukada and S. Matsuoka, "High-Performance and Memory-Saving Sparse General Matrix-Matrix Multiplication for NVIDIA Pascal GPU" , 2017[3]
- Y. Tao et al., "Atomic reduction based sparse matrix-transpose vector multiplication on GPUs" , 2014[4]
- Yang, Carl, Yangzihao Wang, and John D. Owens. "Fast sparse matrix and sparse vector multiplication algorithm on the gpu" , 2015[5]

[2] https://ieeexplore.ieee.org/document/6877271
[3] https://ieeexplore.ieee.org/document/8025284
[4] https://ieeexplore.ieee.org/document/7097920
[5] https://escholarship.org/content/qt1rq9t3j3/qt1rq9t3j3.pdf

Реализованные операции

- Умножение матрицы в CSR формате на разреженный вектор
  1. Каждая строка матрицы умножается на разреженный вектор
  2. Получаем массив значений и массив, в котором для каждой строчки хранится количество произведений
  3. Из полученного плотного вектора удаляются элементы, для которых число произведений в соответствующем массиве нулевое
- Транспонирование матрицы в CSR формате
  1. Матрица в CSR формате конвертируется в формат COO
  2. Сортируются соответствующие массивы
  3. Матрица в COO формате конвертируется в формат CSR

Эксперименты

Название	|V|	|E|	GraphBLAS-sharp, мс	QuickGraph, мс
arc130	130	1282	347.9	0.063
linux-call-graph	324085	1208908	8765.3	9.6
webbase-1M	1000005	3105536	9867.5	18.7
cit-Patents	3774768	16518948	2888.4	136.6

Таблица: Медиана времени работы алгоритма поиска в ширину

- Windows 10, Intel Core i5-4690K CPU 3.50GHz, DDR3 8GB RAM, GeForce GTX 970, 4GB VRAM

Выводы и ограничения подхода

- Managed $\longrightarrow$ Native $\longrightarrow$ GPU
- Компиляция во время исполнения может занимать много времени
- Нет атомарных операций
- Сложно профилировать

Результаты

- Реализована[6] структура объектов и методов GraphBLAS на языке F#
- 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
- Поставлены эксперименты и проведено сравнение предложенной реализации с аналогами

[6] Репозиторий решения:
`https://github.com/YaccConstructor/GraphBLAS-sharp`

Приложение 1

```
1  let levelSingleSource (matrix: Matrix<int>) (source: int) = graphblas {
2      let vertexCount = Matrix.rowCount matrix
3      let! levels = Vector.zeroCreate vertexCount
4      let! frontier = Vector.ofList vertexCount [source, 1]
5      let mutable currentLevel = 0
6      let mutable break' = false
7      while not break' do
8          currentLevel <- currentLevel + 1
9          let! currentLevelScalar = Scalar.create currentLevel
10         let! frontierMask = Vector.mask frontier
11         do! Vector.fillSubVector levels frontierMask currentLevelScalar
12         let! levelsComplemented = Vector.complemented levels
13         do! Matrix.mxvWithMask AddMult.int levelsComplemented transposed
               frontier
14        >>= Vector.assignVector frontier
15         let! succ =
16             Vector.reduce AddMult.int frontier
17            >>= Scalar.exportValue
18         break' <- succ = 0
19     return levels
20 }
```

Листинг 1: Реализация поиска в ширину в GraphBLAS-sharp

Приложение 2

```
1  void bfs(Vector<float> *v, const Matrix<float> *A, Index s, Descriptor *desc)
       {
2      Index A_nrows;
3      CHECK(A->nrows(&A_nrows));
4      CHECK(v->fill(0.f));
5      Vector<float> q1(A_nrows);
6      Vector<float> q2(A_nrows);
7      std::vector<Index> indices(1, s);
8      std::vector<float> values(1, 1.f);
9      CHECK(q1.build(&indices, &values, 1, GrB_NULL));
10     float iter = 1;
11     float succ = 0.f;
12     do {
13         assign<float, float>(v, &q1, GrB_NULL, iter, GrB_ALL, A_nrows, desc);
14         CHECK(desc->toggle(GrB_MASK));
15         vxm<float, float, float, float>(
16             &q2, v, GrB_NULL, LogicalOrAndSemiring<float>(), &q1, A, desc);
17         CHECK(desc->toggle(GrB_MASK));
18         CHECK(q2.swap(&q1));
19         reduce<float, float>(&succ, GrB_NULL, PlusMonoid<float>(), &q1, desc)
               ;
20         iter++;
21     } while (succ > 0);
22 }
```

Листинг 2: Реализация поиска в ширину в GraphBLAST

Приложение 3

```
1  def bfs(matrix, start):
2      v = Vector.sparse(UINT8, matrix.nrows)
3      q = Vector.sparse(BOOL, matrix.nrows)
4      q[start] = True
5      not_done = True
6      level = 1
7      while not_done and level <= matrix.nrows:
8          v.assign_scalar(level, mask=q)
9          q = v.vxm(matrix, mask=v,
10                   desc=descriptor.ooco)
11         not_done = q.reduce_bool()
12         level += 1
13     return v
```

Листинг 3: Реализация поиска в ширину в pygraphblas

Приложение 4

```
// t4 <- sigma.[i - 1, *] * t3
do! Matrix.extractRow sigma (i - 1)
>>= Vector.apply (UnaryOp <@ float32 @>)
>>= fun x -> Vector.eWiseMult AddMult.float32 x t3
>>= Vector.assignVector t4
```

Листинг 4: GraphBLAS-sharp

```
// t4 = sigma[i - 1, :]
GrB_extract(t4, GrB_NULL, GrB_NULL, sigma, GrB_ALL, n, i - 1, GrB_DESC_T0);
// t4 = t4 * t3
GrB_eWiseAdd(*delta, GrB_NULL, GrB_NULL, GrB_PLUS_FP32, *delta, t4, GrB_NULL);
```

Листинг 5: SuiteSparse GraphBLAS

Приложение 5

Название	|V|	|E|	GraphBLAS-sharp, мс
arc130	130	1282	129.1
linux-call-graph	324085	1208908	732.9
webbase-1M	1000005	3105536	929.9
cit-Patents	3774768	16518948	2091.9

Таблица: Среднее время работы алгоритма транспонирования

- Windows 10, Intel Core i5-4690K CPU 3.50GHz, DDR3 8GB RAM, GeForce GTX 970, 4GB VRAM