


Нижегородский государственный университет им. Н.И. Лобачевского

Институт информационных технологий, математики и механики

Нижегородский государственный университет им. Н.И. Лобачевского
Институт информационных технологий, математики и механики

ПАРАЛЛЕЛЬНАЯ ОБРАБОТКА ГРАФОВ

Лекция 7. GraphBLAS. Алгоритмы на графах и линейная алгебра

Пирова А.Ю.
Кафедра ВВиСП

Содержание

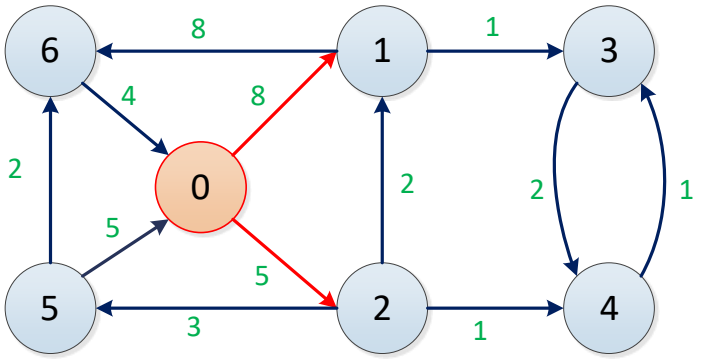
- Графы и линейная алгебра
- Мотивация
- Математические основы
- Основные операции
- Примеры
  - Поиск в ширину
  - Поиск кратчайших путей (алгоритм Беллмана-Форда)
  - Вычисление числа треугольников в графе
  - Вычисление локального коэффициента кластеризации
- Результаты вычислительных экспериментов
- Заключение

Особенности алгоритмов на графах

- Зависимость вычислений от данных
  - Вычислительная нагрузка неизвестна заранее
  - Алгоритмы часто сформулированы итеративно
- Неструктурированность задач
  - Статическое распределение данных может привести к дисбалансу вычислительной нагрузки
  - Отсутствует векторизация
- Низкая локальность
  - Нерегулярный доступ к памяти, кэш-промахи, ошибки предсказания ветвлений
- Низкая арифметическая интенсивность
  - Число операций обращения к памяти значительно превышает число арифметических операций над данными

Графы и линейная алгебра

- Алгоритмы на графах можно преобразовать к последовательности матрично-векторных операций
- Аналогия:
  - матрица смежности ~ разреженная матрица
  - выбранная вершина графа ~ единичный вектор



матрица смежности

	1	1				
			1	1		1
	1			1	1	
				1		
			1			
1						1
1						

Мотивация

- GraphBLAS – подход к реализации алгоритмов на графах, при котором весь алгоритм на графе представляется в виде операций над векторами и матрицами

Проблема	Решение
Библиотеки для обработки графов имеют большой объем кода, **оптимизацию и распараллеливание** каждого алгоритма надо выполнять **отдельно**	**Построение алгоритма из ограниченного числа блоков** упрощает разработку и оптимизацию кода.
Для графов различной структуры / в разных базах данных / **в разных библиотеках** / используются **различные** внутренние **форматы хранения**	**Унифицированное хранение графов** в виде разреженной матрицы подходит для графов разной структуры (однако можно применять разные форматы)
Параллельные алгоритмы на графах **нетривиально оптимизировать**	Достаточно применить **эффективные реализации матрично-векторных операций** и коммуникаций между процессами

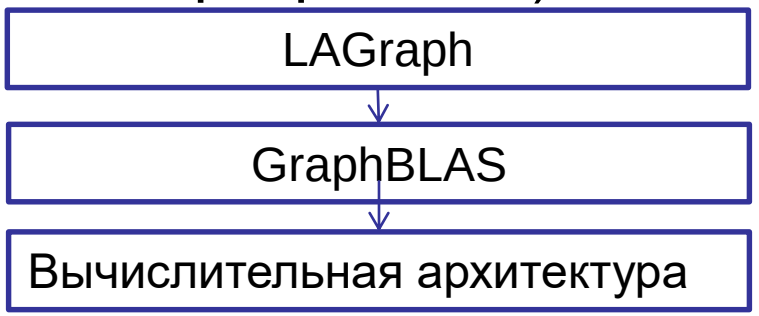
Мотивация

Проблема	Решение
Существующие реализации **сложно переносить на другие или новые вычислительные архитектуры**.	Переносимость обеспечивается за счет **переноса и оптимизации базовых** матрично-векторных **операций**
Производительность ограничена **пропускной способностью и латентностью** вычислительного устройства	Производительность ограничивается **пропускной способностью** вычислительного устройства

- О стандарте: http://graphblas.org
- **Коллекция ссылок: https://github.com/GraphBLAS/GraphBLAS-Pointers/tree/master**

История

- 2011 г. – сборник Кепнера, Гильберта по алгоритмам на графах в терминах линейной алгебры
- 2013 г. – стандартизация примитивов линейной алгебры, используемых для графовых задач – GraphBLAS
- 2017 г. – стандарт GraphBLAS для языка C
- 2019 г. – проект LAGraph – библиотека алгоритмов, использующих стандарт GraphBLAS
- 2020 г. – анонс разработки стандарта GraphBLAS для языка C++ и для распределенных систем (еще в разработке)



Реализации стандарта GraphBLAS

- Реализации стандарта:
  - SuiteSparse:GraphBLAS (C) https://github.com/DrTimothyAldenDavis/GraphBLAS
  - CombinatorialBLAS, CombBLAS (C++) https://github.com/PASSIONLab/CombBLAS
  - IBM GraphBLAS https://github.com/IBM/ibmgraphblas
  - GraphBLAS Template Library (C++) https://github.com/cmu-sei/gbtl/
  - Graphulo (Java): http://graphulo.mit.edu/
  - GraphBLAST (C++, CUDA) https://github.com/gunrock/graphblast
  - LAGraph (C++) https://github.com/GraphBLAS/LAGraph - библиотека алгоритмов на графах, использующая GraphBLAS
  - Python GraphBLAS https://github.com/python-graphblas/python-graphblas, https://github.com/python-graphblas/graphblas-algorithms

Параллелизм

Библиотека	Технология	Целевое устройство	Основной формат хранения матриц
SuiteSparse:GraphBLAS	OpenMP	Multicore CPU	CRS
CombBLAS	MPI + OpenMP	CPU, GPU	CSC, DCSC
GraphBLAST	CUDA	CPU, GPU	CRS, CSC

Математические основы

- Все операции в GraphBLAS выполняются над объектами полукольца $\langle D, \otimes, \oplus, 0 \rangle$, которое задается набором операторов и множеством значений аргументов D
- Множества допустимых значений элементов матрицы: $\boldsymbol{R}, \boldsymbol{C}, \boldsymbol{Z}, \boldsymbol{N}$
- Операторы:
  - Оператор сложения $\oplus$: $D \times D \to D$, $\langle D, \oplus, 0 \rangle$ - коммутативный моноид
  - Оператор умножения $\otimes$: $D \times D \to D$ – закрытый бинарный оператор

Математические основы

- Пусть a, b, c – матрицы смежности (инцидентности) размера $n \times m$
- Свойства операторов:
  - Оператор сложения $\oplus$: $c = a \oplus b$
    - ассоциативный $(a \oplus b) \oplus c = a \oplus (b \oplus c)$
    - коммутативный $a \oplus b = b \oplus a$
    - существует нейтральный элемент $a \oplus 0 = a$
  - Оператор умножения $\otimes$: $c = a \otimes b$
    - ассоциативный $(a \otimes b) \otimes c = a \otimes (b \otimes c)$
    - коммутативный $a \otimes b = b \otimes a$
    - дистрибутивный $a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c)$
- В терминах матриц $C = AB$

$$C_{ij} = A_{i1} \otimes B_{1j} \oplus A_{i2} \otimes B_{2j} \oplus \ \dots \oplus A_{i,n} \otimes B_{nj}$$

Математические основы

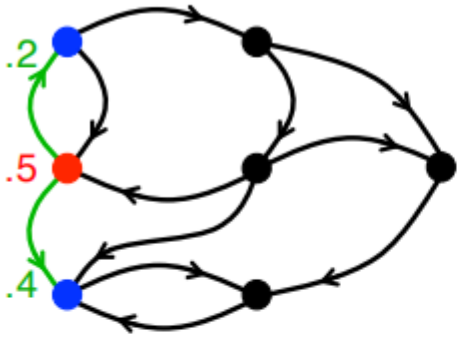
- Примеры полуколец:

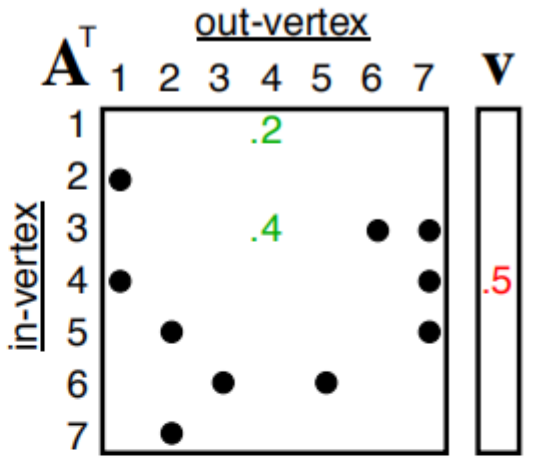
Название	$\oplus$	$\otimes$	Множество значений	Нулевой элемент	Смысл операции
стандартная арифметика	$+$	$\times$	$\boldsymbol{R}$	0	Произведение всех путей
целочисленная арифметика	$+$	$\times$	$\boldsymbol{N}$	0	Число всех путей
max-plus алгебра	Max	$+$	$\{-\infty, \boldsymbol{R}\}$	$-\infty$	Паросочетание
min-plus алгебра	Min	$+$	$\{\boldsymbol{R}, +\infty\}$	$+\infty$	Кратчайшие пути
Булева алгебра	xor	and	[0, 1]	0	достижимость
Алгебра множеств	$\cup$	$\cap$	$\boldsymbol{Z}$	$\emptyset$	

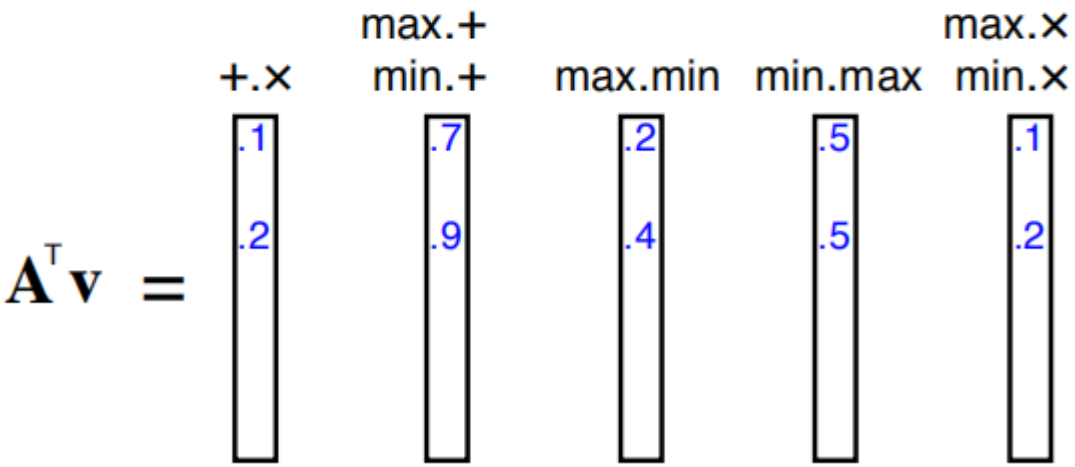
- Подробнее:
  Kepner J. GraphBLAS Mathematics-Provisional Release 1.0 //GraphBLAS. org, Tech. Rep. – 2017.

Математические основы

- Пример:







Kepner J. GraphBLAS Mathematics-Provisional Release 1.0 //GraphBLAS. org, Tech. Rep. – 2017.

Маски

- Стандарт учитывает наличие вектора-маски или матрицы-маски при выполнении операций умножения, сложения
  - Сокращение числа вычислительных операций
  - Способ задания подмножества вершин
  - Способ комбинирования матрично-векторных операций и поэлементных операций

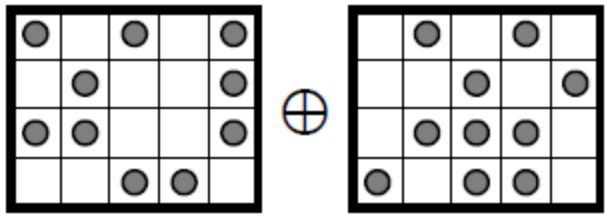
Основные операции GraphBLAS

- Обозначения:
  - A, B, C, M – матрицы
  - u, v, w, m – вектора
  - $\langle m \rangle, \langle M \rangle$ – маска
  - i, j – индексы
  - s, k – скаляры
  - $\oplus$ – сложение
  - $\otimes$ – умножение
  - $\odot$ – аккумулятор

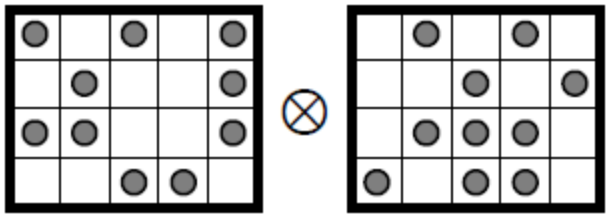
function name	description	GraphBLAS notation
`GrB_mxm`	matrix-matrix multiplication	$\mathbf{C}\langle\mathbf{M}\rangle = \mathbf{C} \odot \mathbf{AB}$
`GrB_vxm`	vector-matrix multiplication	$\mathbf{w}'\langle\mathbf{m}'\rangle = \mathbf{w}' \odot \mathbf{u}'\mathbf{A}$
`GrB_mxv`	matrix-vector multiplication	$\mathbf{w}\langle\mathbf{m}\rangle = \mathbf{w} \odot \mathbf{Au}$
`GrB_eWiseMult`	element-wise, set-union	$\mathbf{C}\langle\mathbf{M}\rangle = \mathbf{C} \odot (\mathbf{A} \otimes \mathbf{B})$ $\mathbf{w}\langle\mathbf{m}\rangle = \mathbf{w} \odot (\mathbf{u} \otimes \mathbf{v})$
`GrB_eWiseAdd`	element-wise, set-intersection	$\mathbf{C}\langle\mathbf{M}\rangle = \mathbf{C} \odot (\mathbf{A} \oplus \mathbf{B})$ $\mathbf{w}\langle\mathbf{m}\rangle = \mathbf{w} \odot (\mathbf{u} \oplus \mathbf{v})$
`GrB_extract`	extract submatrix	$\mathbf{C}\langle\mathbf{M}\rangle = \mathbf{C} \odot \mathbf{A}(\mathbf{i},\mathbf{j})$ $\mathbf{w}\langle\mathbf{m}\rangle = \mathbf{w} \odot \mathbf{u}(\mathbf{i})$
`GrB_assign`	assign submatrix	$\mathbf{C}\langle\mathbf{M}\rangle(\mathbf{i},\mathbf{j}) = \mathbf{C}(\mathbf{i},\mathbf{j}) \odot \mathbf{A}$ $\mathbf{w}\langle\mathbf{m}\rangle(\mathbf{i}) = \mathbf{w}(\mathbf{i}) \odot \mathbf{u}$
`GxB_subassign`	assign submatrix	$\mathbf{C}(\mathbf{i},\mathbf{j})\langle\mathbf{M}\rangle = \mathbf{C}(\mathbf{i},\mathbf{j}) \odot \mathbf{A}$ $\mathbf{w}(\mathbf{i})\langle\mathbf{m}\rangle = \mathbf{w}(\mathbf{i}) \odot \mathbf{u}$
`GrB_apply`	apply unary operation	$\mathbf{C}\langle\mathbf{M}\rangle = \mathbf{C} \odot f(\mathbf{A})$ $\mathbf{w}\langle\mathbf{m}\rangle = \mathbf{w} \odot f(\mathbf{u})$
`GxB_select`	apply select operation	$\mathbf{C}\langle\mathbf{M}\rangle = \mathbf{C} \odot f(\mathbf{A},\mathbf{k})$ $\mathbf{w}\langle\mathbf{m}\rangle = \mathbf{w} \odot f(\mathbf{u},\mathbf{k})$
`GrB_reduce`	reduce to vector reduce to scalar	$\mathbf{w}\langle\mathbf{m}\rangle = \mathbf{w} \odot [\oplus_j \mathbf{A}(:,j)]$ $s = s \odot [\oplus_{ij} \mathbf{A}(i,j)]$
`GrB_transpose`	transpose	$\mathbf{C}\langle\mathbf{M}\rangle = \mathbf{C} \odot \mathbf{A}'$
`GxB_kron`	Kronecker product	$\mathbf{C}\langle\mathbf{M}\rangle = \mathbf{C} \odot \mathrm{kron}(\mathbf{A},\mathbf{B})$

Основные операции GraphBLAS

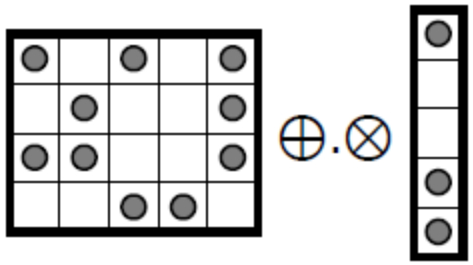
Element-wise addition:
union of non-zero elements



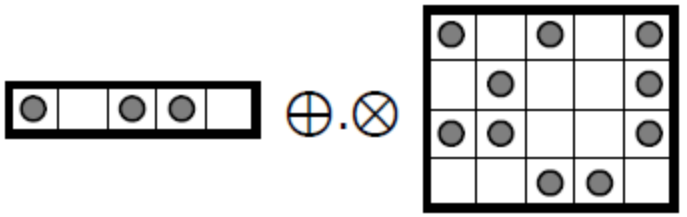
Element-wise multiplication:
intersection of non-zero elements



Sparse matrix times sparse vector:
process incoming edges



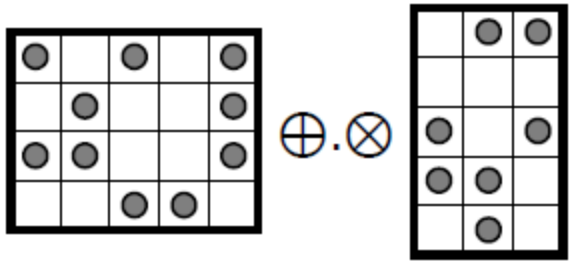
Sparse vector times sparse matrix:
process outgoing edges



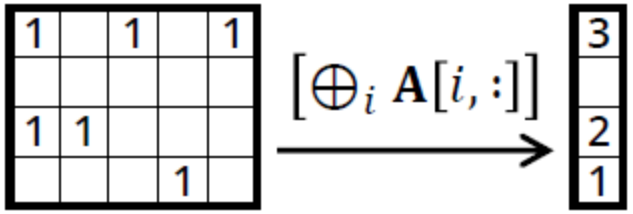
Slide from: Introduction to GraphBLAS: A linear algebraic approach for concise, portable, and high-performance graph algorithms by Gábor Szárnyas [pdf]

Основные операции GraphBLAS

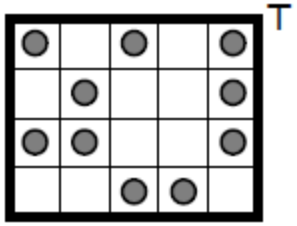
Sparse matrix times sparse matrix:
process connecting outgoing edges



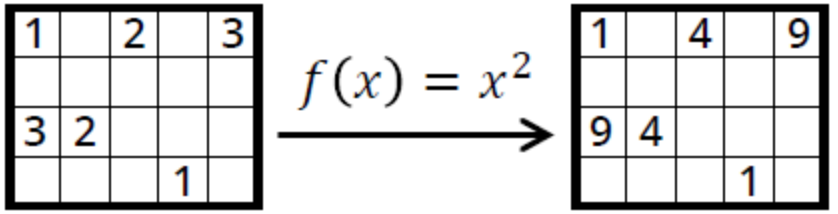
Reduction:
aggregate values in each row



Matrix transpose:
reverse edges

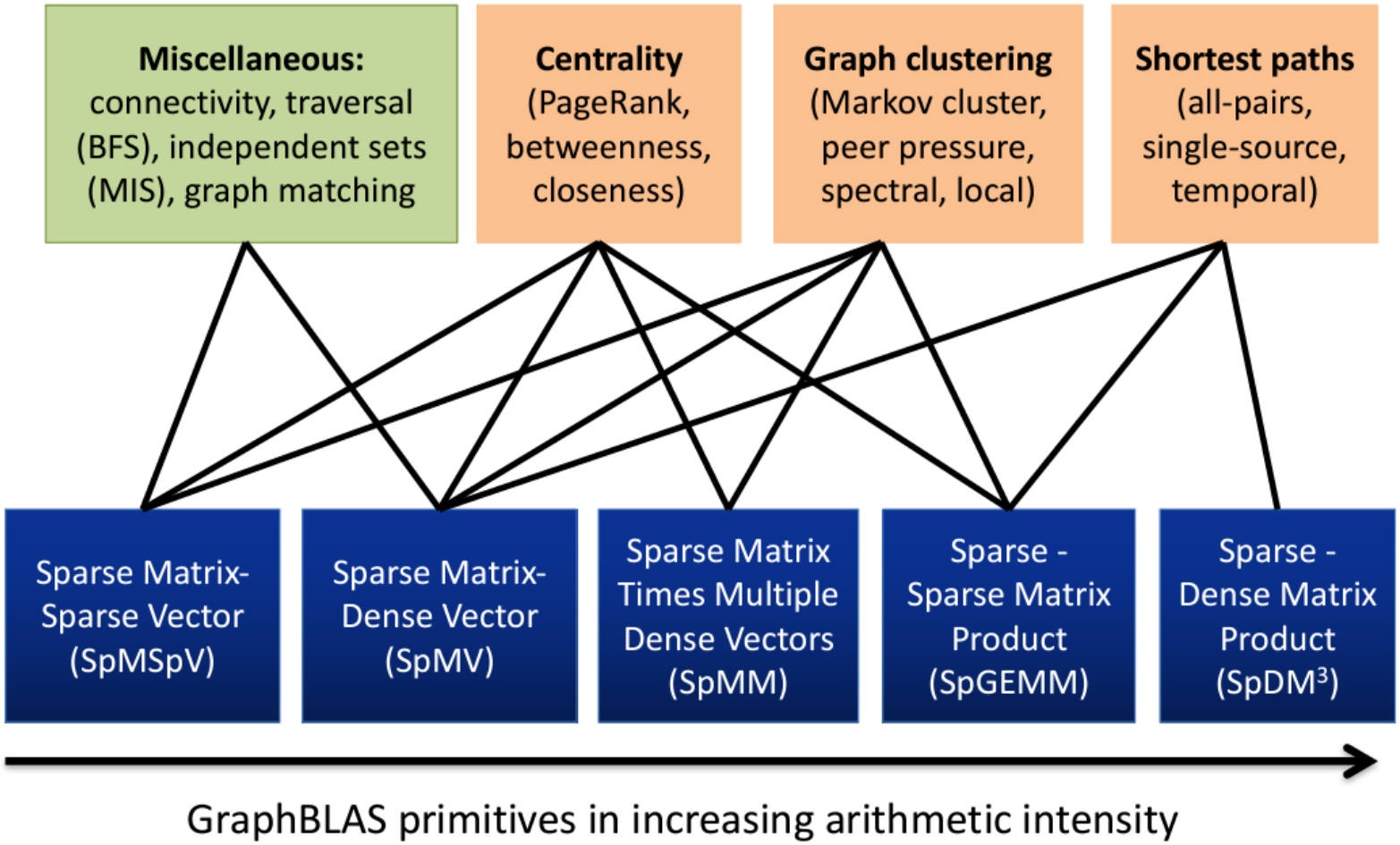


Apply:
apply unary operator on all values



Slide from: Introduction to GraphBLAS: A linear algebraic approach for concise, portable, and high-performance graph algorithms by Gábor Szárnyas [pdf]

Основные операции GraphBLAS



A. Buluç: Graph algorithms, computational motifs, and GraphBLAS,ECP Meeting 2018

Алгоритмы в терминах линейной алгебры

Алгоритм на графах	Алгоритм в терминах линейной алгебры
Инверсия ребер графа	Транспонирование матрицы
Обход графа	Умножение матрицы на вектор
Объединение графов	Сложение матриц
Пересечение графов	Умножение матриц
Выбор подграфа	Выбор подматрицы
Поиск кратчайших путей	Умножение матрицы на вектор
Нахождение паросочетания	Умножение матрицы на вектор
Подсчет числа треугольников	Умножение матриц

Алгоритмы в терминах линейной алгебры

problem	algorithm	canonical complexity Θ	LA-based complexity Θ
breadth-first search		m	m
single-source shortest paths	Dijkstra	$m + n \log n$	n^2
	Bellman-Ford	mn	mn
all-pairs shortest paths	Floyd-Warshall	n^3	n^3
minimum spanning tree	Prim	$m + n \log n$	n^2
	Borůvka	$m \log n$	$m \log n$
maximum flow	Edmonds-Karp	$m^2 n$	$m^2 n$
maximal independent set	greedy	$m + n \log n$	$mn + n^2$
	Luby	$m + n \log n$	$m \log n$

Slide from: Introduction to GraphBLAS: A linear algebraic approach for concise, portable, and high-performance graph algorithms by Gábor Szárnyas [pdf]

АЛГОРИТМЫ

Поиск в ширину

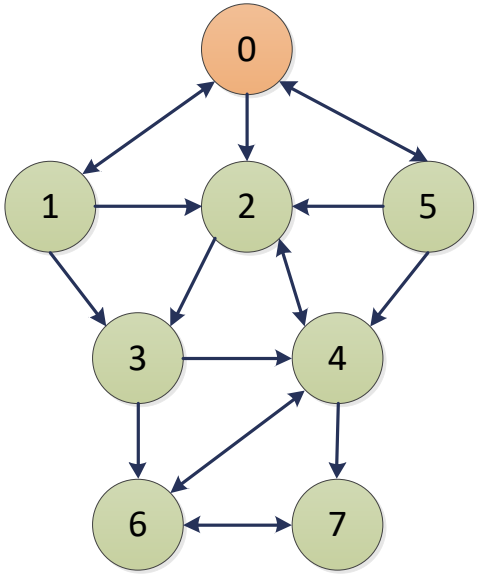
- Задача: в ненаправленном графе найти все вершины, достижимые из заданной вершины-источника, в порядке увеличения расстояния от источника
- **Алгоритм**:

Переменные: q – очередь из вершин текущего уровня (вектор-маска), v – вектор с номерами уровней

1. Инициализация. $q[s] = 1, level = 1$
2. Пока q не пусто:
   1. $v\langle q\rangle = level$ (`GrB_assign(v, q, level)`)
   2. $q\langle !\, v\rangle = A^T q$ (`GrB_vxm(q, A)`)
   3. Переход на новый уровень? (`GrB_Vector_reduce(`q`)`)

- Основная операция – умножение матрицы на вектор $A^T q$
- Полукольцо: $\oplus - OR$, $\otimes - AND$, значения {0, 1}, «нуль» - 0.

Поиск в ширину. Пример



Шаг 1

	1				1		
1							
1	1			1	1		
	1	1					
		1	1		1	1	
1							
			1	1			1
				1		1	

A^T

$\otimes$

1

q

$=$

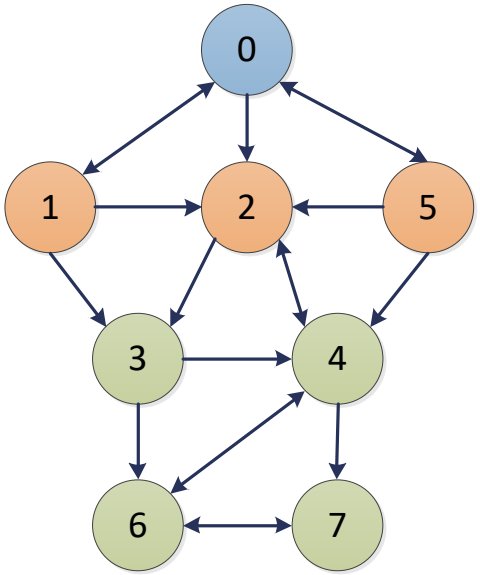
1
1
1

q_{new}

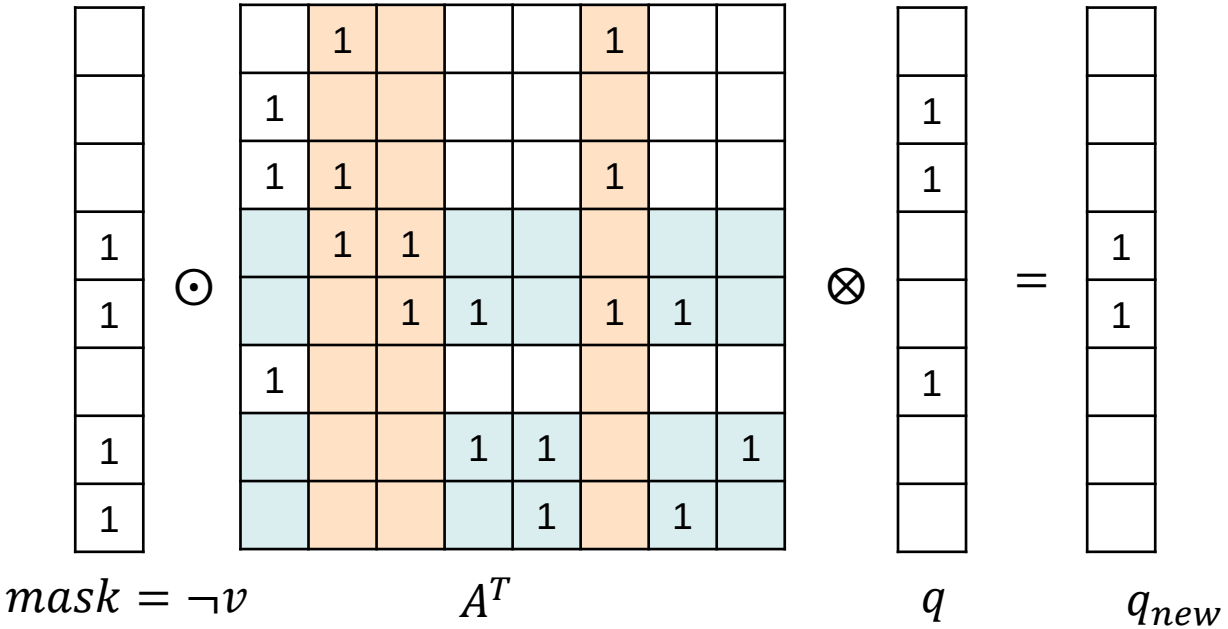
0
1
1
1

v

Поиск в ширину. Пример



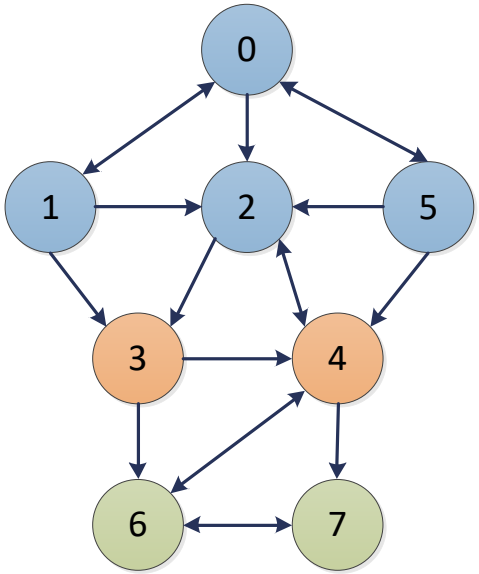
Шаг 2



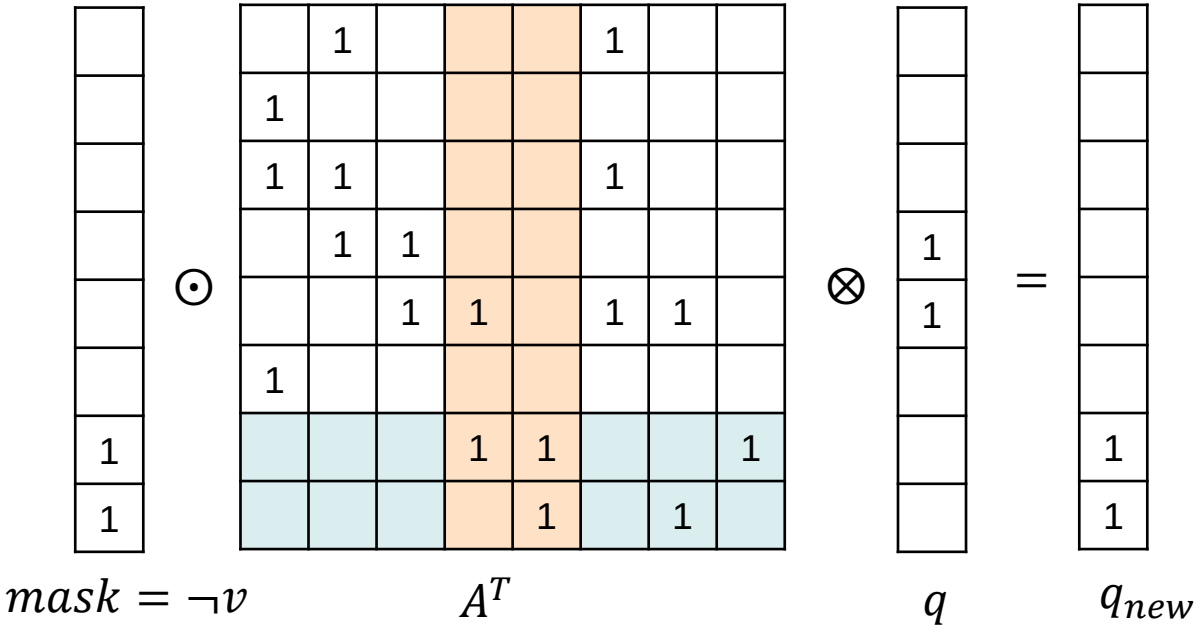
v

0	1	1	2	2	1		

Поиск в ширину. Пример



Шаг 3



v

0	1	1	2	2	1	3	3

Поиск в ширину. Код (из спецификации v. 2.0)

```
1  #include <stdlib.h>
2  #include <stdio.h>
3  #include <stdint.h>
4  #include <stdbool.h>
5  #include "GraphBLAS.h"
6
7  /*
8   * Given a boolean n x n adjacency matrix A and a source vertex s, performs a BFS traversal
9   * of the graph and sets v[i] to the level in which vertex i is visited (v[s] == 1).
10  * If i is not reacheable from s, then v[i] = 0. (Vector v should be empty on input.)
11  */
12 GrB_Info BFS(GrB_Vector *v, GrB_Matrix A, GrB_Index s)
13 {
14   GrB_Index n;
15   GrB_Matrix_nrows(&n,A);                          // n = # of rows of A
16
17   GrB_Vector_new(v,GrB_INT32,n);                   // Vector<int32_t> v(n)
18
19   GrB_Vector q;                                    // vertices visited in each level
20   GrB_Vector_new(&q,GrB_BOOL,n);                   // Vector<bool> q(n)
21   GrB_Vector_setElement(q,(bool)true,s);           // q[s] = true, false everywhere else
22
23   /*
24    * BFS traversal and label the vertices.
25    */
26   int32_t d = 0;                                   // d = level in BFS traversal
27   bool succ = false;                               // succ == true when some successor found
28   do {
29     ++d;                                           // next level (start with 1)
30     GrB_assign(*v,q,GrB_NULL,d,GrB_ALL,n,GrB_NULL);    // v[q] = d
31     GrB_vxm(q,*v,GrB_NULL,GrB_LOR_LAND_SEMIRING_BOOL,
32             q,A,GrB_DESC_RC);                      // q[!v] = q ||.&& A ; finds all the
33                                                    // unvisited successors from current q
34     GrB_reduce(&succ,GrB_NULL,GrB_LOR_MONOID_BOOL,
35                q,GrB_NULL);                        // succ = ||(q)
36   } while (succ);                                  // if there is no successor in q, we are done.
37
38   GrB_free(&q);                                    // q vector no longer needed
39
40   return GrB_SUCCESS;
41 }
```

Поиск в ширину с хранением родителя

- Алгоритм

  Переменные: p – вектор родителей, idx - вектор индексов, v – вектор с номерами уровней

  1. Инициализация. $q[s] = 1$, $level = 1$, p[s] = s
  2. Пока q не пусто:
     1. $q\langle !\, p\rangle = qA$ (`GrB_vxm(q, A)`)
     2. $p\langle q\rangle = q$
     3. $v\langle v\rangle = idx$
     4. Переход на новый уровень? (`GrB_Vector_reduce`(q))

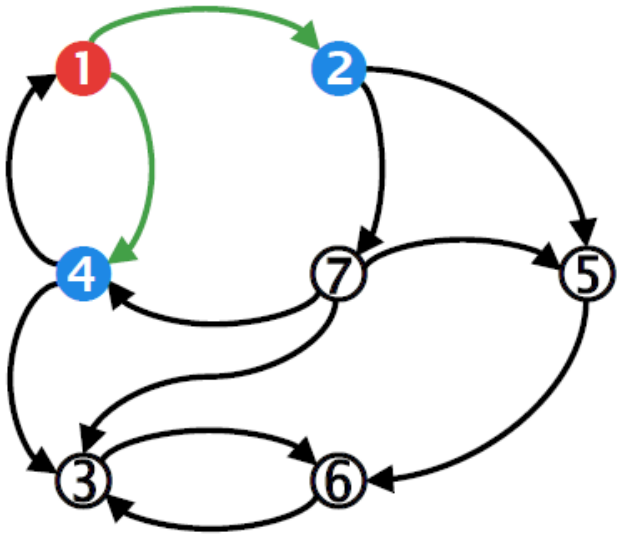
- Полукольцо: $\oplus - min$, $\otimes - first$, значения $\boldsymbol{N}$, «нуль» - 0.

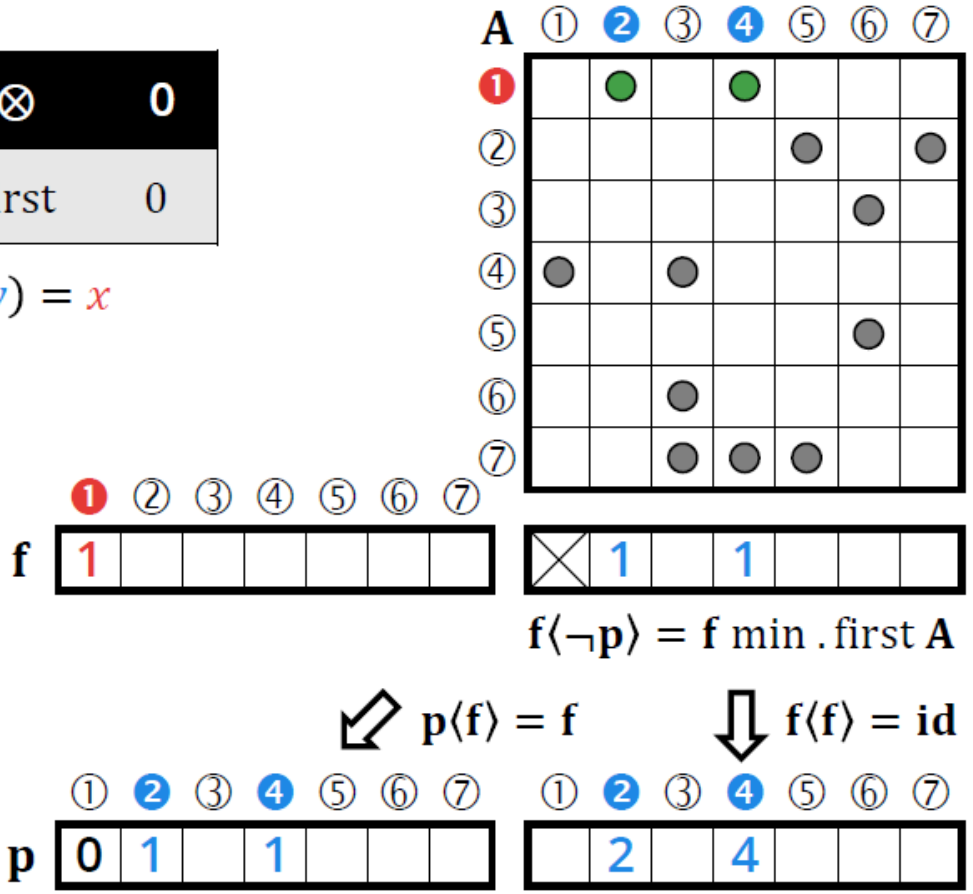
Пример

BFS – PARENTS

semiring	domain	⊕	⊗	0
min-first	$\mathbb{N}$	min	first	0

$$\text{first}(x, y) = x$$



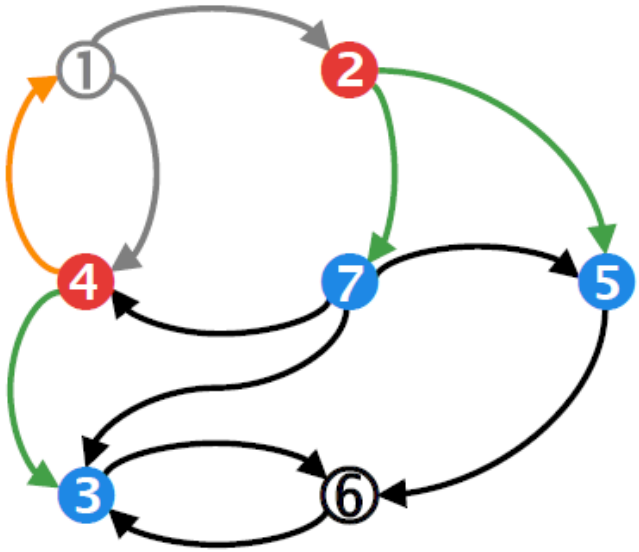


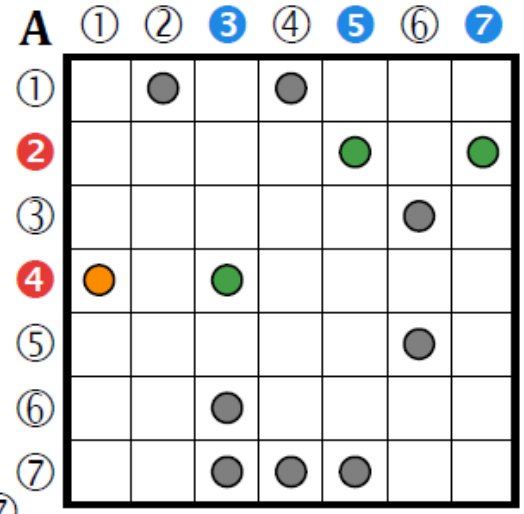
Пример

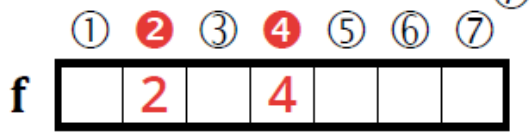
BFS – PARENTS

semiring	domain	⊕	⊗	0
min-first	$\mathbb{N}$	min	first	0

$$\text{first}(x, y) = x$$

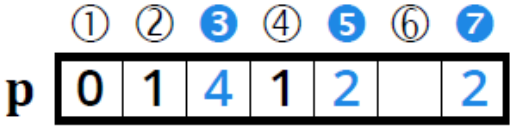






$$\mathbf{f}\langle\neg\mathbf{p}\rangle = \mathbf{f} \text{ min . first } \mathbf{A}$$

$$\mathbf{p}\langle\mathbf{f}\rangle = \mathbf{f} \qquad \mathbf{f}\langle\mathbf{f}\rangle = \mathbf{id}$$

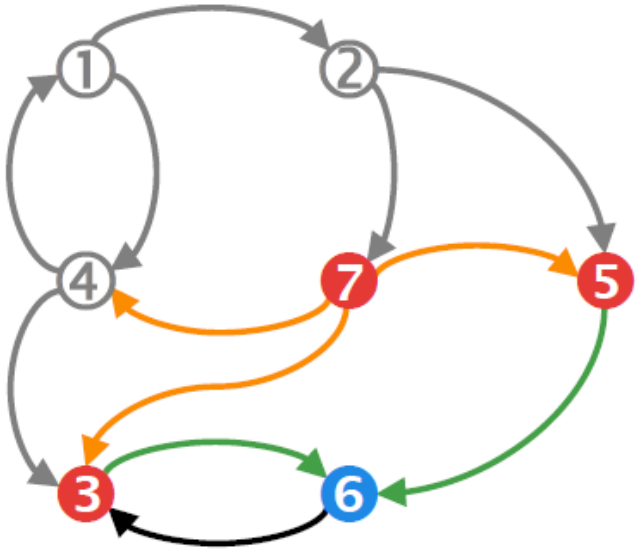


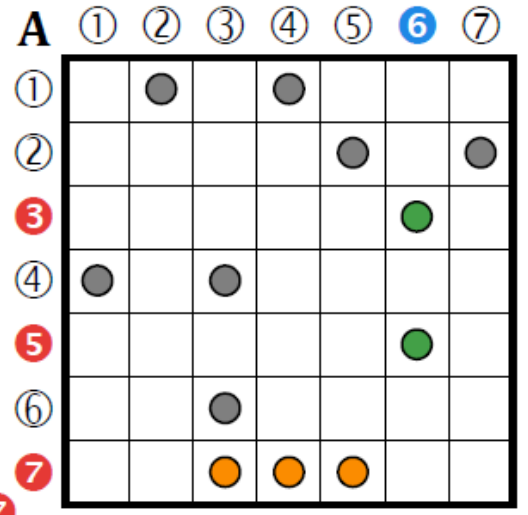
Пример

BFS – PARENTS

semiring	domain	⊕	⊗	0
min-first	$\mathbb{N}$	min	first	0

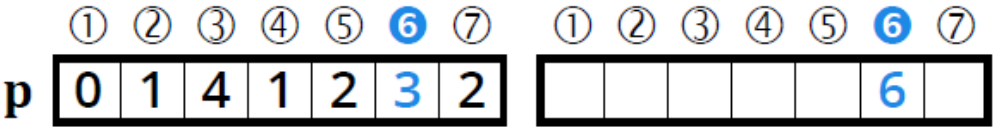
$$\text{first}(x, y) = x$$





$$\mathbf{f}\langle\neg\mathbf{p}\rangle = \mathbf{f}\ \text{min}.\ \text{first}\ \mathbf{A}$$

$$\mathbf{p}\langle\mathbf{f}\rangle = \mathbf{f} \qquad \mathbf{f}\langle\mathbf{f}\rangle = \mathbf{id}$$

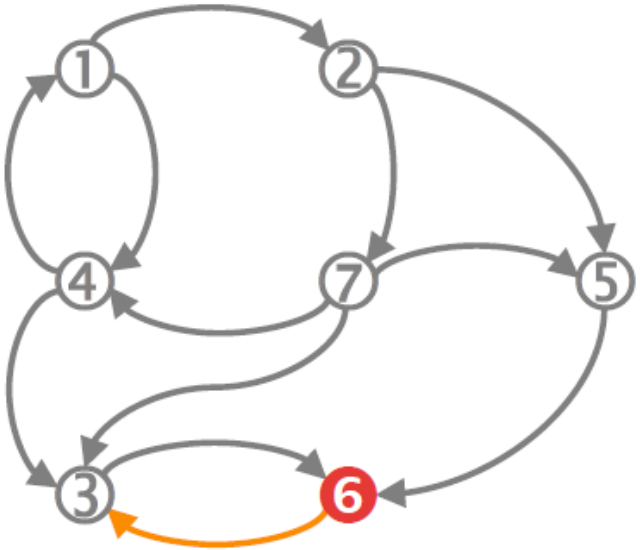


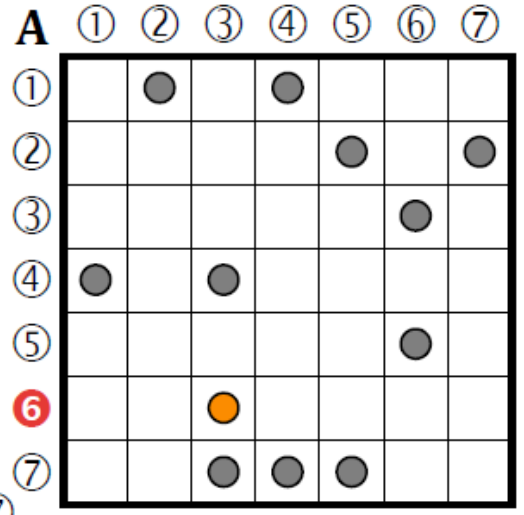
Пример

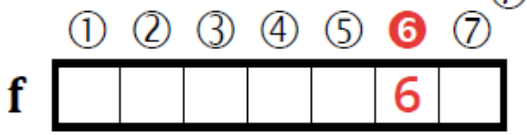
BFS – PARENTS

semiring	domain	⊕	⊗	0
min-first	$\mathbb{N}$	min	first	0

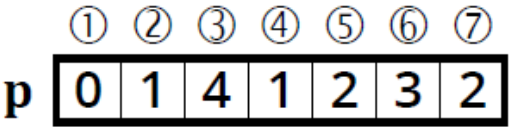
$$\text{first}(x, y) = x$$







$$\mathbf{f}\langle\neg\mathbf{p}\rangle = \mathbf{f} \text{ min . first } \mathbf{A}$$



f is empty
→ terminate

Поиск в ширину с родителем. Код

```
7  /*
8   * Given a binary n x n adjacency matrix A and a source vertex s, performs a BFS
9   * traversal of the graph and sets parents[i] to the index of vertex i's parent.
10  * The parent of the root vertex, s, will be set to itself (parents[s] == s). If
11  * vertex i is not reachable from s, parents[i] will not contain a stored value.
12  */
13 GrB_Info BFS(GrB_Vector *parents, const GrB_Matrix A, GrB_Index s)
14 {
15   GrB_Index N;
16   GrB_Matrix_nrows(&N, A);                    // N = # vertices
17
18   GrB_Vector_new(parents, GrB_UINT64, N);
19   GrB_Vector_setElement(*parents, s, s);      // parents[s] = s
20
21   GrB_Vector wavefront;
22   GrB_Vector_new(&wavefront, GrB_UINT64, N);
23   GrB_Vector_setElement(wavefront, 1UL, s);   // wavefront[s] = 1
24
25   /*
26    * BFS traversal and label the vertices.
27    */
28   GrB_Index nvals;
29   GrB_Vector_nvals(&nvals, wavefront);
30
31   while (nvals > 0)
32   {
33     // convert all stored values in wavefront to their 0-based index
34     GrB_apply(wavefront, GrB_NULL, GrB_NULL, GrB_ROWINDEX_INT64,
35               wavefront, 0UL, GrB_NULL);
36
37     // "FIRST" because left-multiplying wavefront rows. Masking out the parent
38     // list ensures wavefront values do not overwrite parents already stored.
39     GrB_vxm(wavefront, *parents, GrB_NULL, GrB_MIN_FIRST_SEMIRING_UINT64,
40             wavefront, A, GrB_DESC_RSC);
41
42     // Don't need to mask here since we did it in mxm. Merges new parents in
43     // current wavefront with existing parents: parents += wavefront
44     GrB_apply(*parents, GrB_NULL, GrB_PLUS_UINT64,
45               GrB_IDENTITY_UINT64, wavefront, GrB_NULL);
46
47     GrB_Vector_nvals(&nvals, wavefront);
48   }
49
50   GrB_free(&wavefront);
51
52   return GrB_SUCCESS;
53 }
```

Поиск в ширину. Модификации

- Комбинированный алгоритм (обход сверху вниз + обход снизу вверх) https://github.com/GraphBLAS/LAGraph/blob/stable/src/algorithm/LG_BreadthFirstSearch_SSGrB.c
  - Обход сверху вниз: $q^{\mathrm{T}}\langle !\, v\rangle = q^T A$
  - Обход снизу вверх: $q\langle !\, v\rangle = A^T q$
- Yang C., Buluç A., Owens J. D. Implementing push-pull efficiently in GraphBLAS //Proceedings of the 47th International Conference on Parallel Processing. – 2018. – С. 1-11. [pdf]

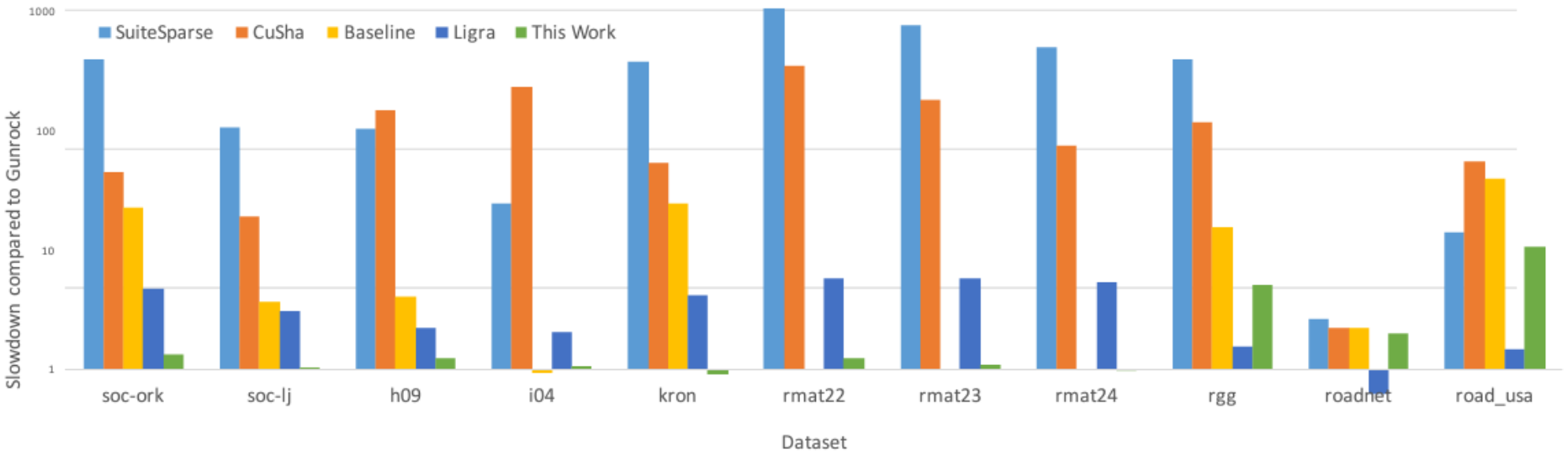
Результаты экспериментов

- Источник: Yang C., Buluç A., Owens J. D. Implementing push-pull efficiently in GraphBLAS //Proceedings of the 47th International Conference on Parallel Processing. – 2018. – С. 1-11.
- HW: a Linux workstation with 2×3.50 GHz Intel 4-core E5-2637v2 Xeon CPUs, 556 GB of main memory, and an NVIDIA K40c GPU with 12 GB on-board memory.

	Runtime (ms) [lower is better]						Edge throughput (MTEPS) [higher is better]					
Dataset	SuiteSparse	CuSha	Baseline	Ligra	Gunrock	This Work	SuiteSparse	CuSha	Baseline	Ligra	Gunrock	This Work
soc-ork	2165	244.9	122.4	26.1	**5.573**	7.280	98.24	868.3	1722	8149	**38165**	29217
soc-lj	1483	263.6	51.32	42.4	**14.05**	14.16	57.76	519.5	1669	2021	**6097**	6049
h09	596.7	855.2	23.39	12.8	**5.835**	7.138	188.7	131.8	4814	8798	**19299**	15775
i04	1866	17609	**71.81**	157	77.21	80.37	159.8	22.45	**4151**	1899	3861	3709
kron	1694	237.9	108.7	18.5	4.546	**4.088**	107.5	765.5	1675	9844	40061	**44550**
rmat-22	4226	1354	OOM	22.6	**3.943**	4.781	114.3	369.1	OOM	21374	**122516**	101038
rmat-23	6033	1423	OOM	45.6	**7.997**	8.655	83.81	362.7	OOM	11089	**63227**	58417
rmat-24	8193	1234	OOM	89.6	16.74	**16.59**	63.42	426.4	OOM	5800	31042	**31327**
rgg	230602	68202	9147	918	**593.9**	2991	1.201	3.887	30.28	288.8	**466.4**	92.59
roadnet	342	288.5	284.9	**82.1**	130.9	214.4	16.14	14.99	19.38	**67.25**	42.18	25.75
road_usa	9413	36194	26594	978	**676.2**	7155	6.131	7.944	2.17	59.01	**85.34**	8.065

Результаты экспериментов

- Источник: Yang C., Buluç A., Owens J. D. Implementing push-pull efficiently in GraphBLAS //Proceedings of the 47th International Conference on Parallel Processing. – 2018. – С. 1-11.
- HW: a Linux workstation with 2×3.50 GHz Intel 4-core E5-2637v2 Xeon CPUs, 556 GB of main memory, and an NVIDIA K40c GPU with 12 GB on-board memory.



Алгоритм Беллмана-Форда

- Задача: в направленном взвешенном графе найти кратчайшие пути из вершины-источника ко всем остальным вершинам
- Алгоритм базовый:

Пусть d – вектор расстояний, A – матрица расстояний

Для $i = 0, \dots, n-1$:

Для всех $(v,u) \in E$ $d(v) = \min\{d(v), d(u) + A_{u,v}\}$

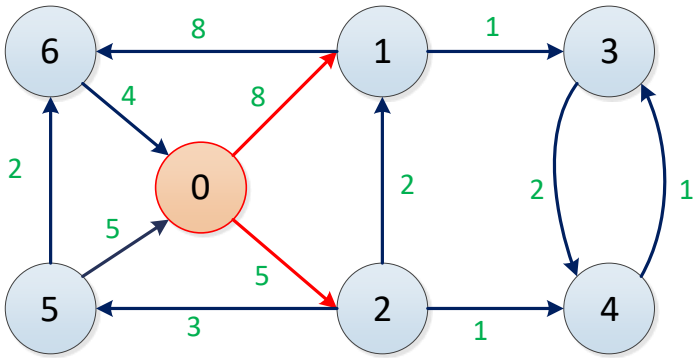
- Алгоритм в терминах линейной алгебры:

Для $i = 0, \dots, n-1$:

$d = A^T d$ (`GrB_mxv(A^T,d)`)

- Основная операция – умножение матрицы на вектор `GrB_vxm(d, A)` или `GrB_mxv(B,d)`
- Полукольцо: $\boldsymbol{R}, \oplus - \min, \otimes - +$, нулевой элемент $+\infty$.

Пример



					5	4
8		2				
5						
	1			1		
	5	1	2			
		3				
	8				2	

A^T

$+$
$\otimes$

0

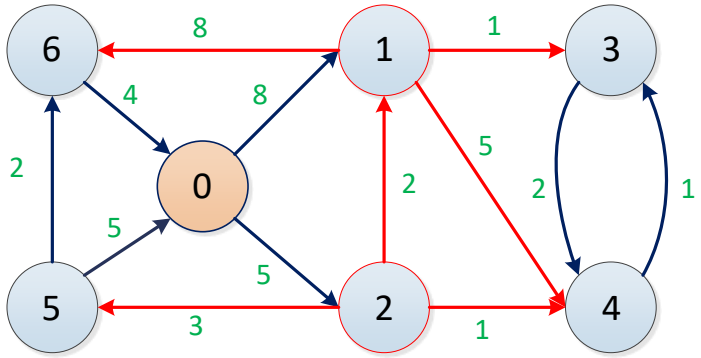
d

$=$

0
8
5

d

Пример



					5	4
8		2				
5						
	1			1		
	5	1	2			
		3				
	8				2	

A^T

$\otimes$

0
8
5

d

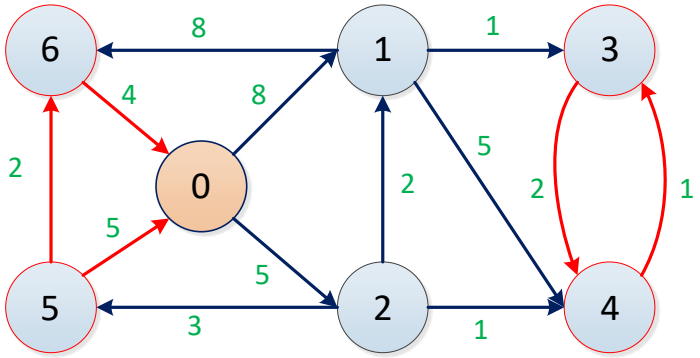
$=$

0
7
5
9
6
8
16

d

Min(8, 5+2)
5+0
1+8
Min(8+5,5+1)
3+5
8+8

Пример



					5	4
8		2				
5						
	1			1		
	5	1	2			
		3				
	8				2	

A^T

$\otimes$

0
7
5
9
6
8
16

d

$=$

0
7
5
7
6
8
10

d

Min(8, 7)
Min(12, 6, 11)
Min(15, 10)

Примеры. Подсчет числа треугольников

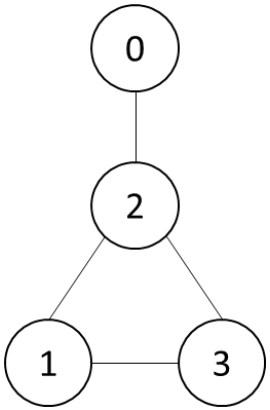
- Задача: найти в графе все клики размера 3.

- Алгоритм Коэна:

1. Разделить матрицу A на верхний U и нижний L треугольник, U и L (`GxB_select`)
2. Найти $C\langle L\rangle = LU^T$ (`GrB_mxm`)
3. $k = \frac{1}{2}\sum_i \sum_j C_{ij}$ - число треугольников (`GrB_reduce`)

- Основная операция – матричное умножение
- Полукольцо: $\boldsymbol{R}$, $\oplus - +, \otimes - \times$, нулевой элемент 0.

Подсчет числа треугольников



$$A = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix}$$

$$L = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{pmatrix} \qquad U = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$B = L * U = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 1 \\ 0 & 0 & 1 & 2 \end{pmatrix} \qquad C = AB = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

$k = 1$

Подсчет числа треугольников. Код

```
1  #include <stdlib.h>
2  #include <stdio.h>
3  #include <stdint.h>
4  #include <stdbool.h>
5  #include "GraphBLAS.h"
6
7  /*
8   * Given an n x n boolean adjacency matrix, A, of an undirected graph, computes
9   * the number of triangles in the graph.
10  */
11 uint64_t triangle_count(GrB_Matrix A)
12 {
13   GrB_Index n;
14   GrB_Matrix_nrows(&n, A);                          // n = # of vertices
15
16   // L: NxN, lower-triangular, bool
17   GrB_Matrix L;
18   GrB_Matrix_new(&L, GrB_BOOL, n, n);
19   GrB_select(L, GrB_NULL, GrB_NULL, GrB_TRIL, A, 0UL, GrB_NULL);
20
21   GrB_Matrix C;
22   GrB_Matrix_new(&C, GrB_UINT64, n, n);
23
24   GrB_mxm(C, L, GrB_NULL, GrB_PLUS_TIMES_SEMIRING_UINT64, L, L, GrB_NULL); // C<L> = L +.* L
25
26   uint64_t count;
27   GrB_reduce(&count, GrB_NULL, GrB_PLUS_MONOID_UINT64, C, GrB_NULL);      // 1-norm of C
28
29   GrB_free(&C);
30   GrB_free(&L);
31
32   return count;
33 }
```

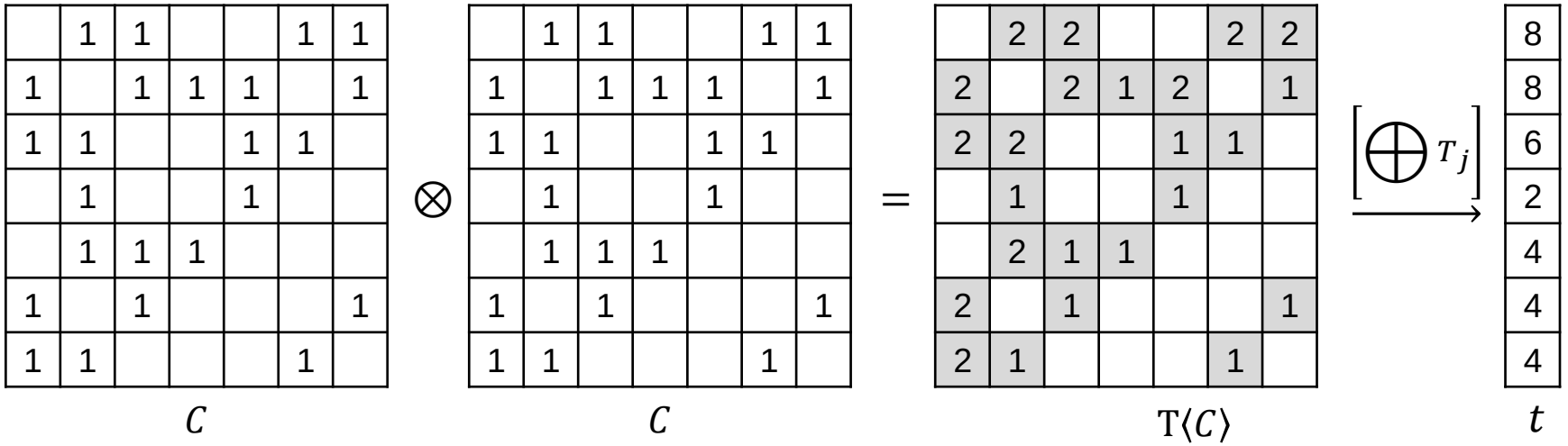
Локальный коэффициент кластеризации

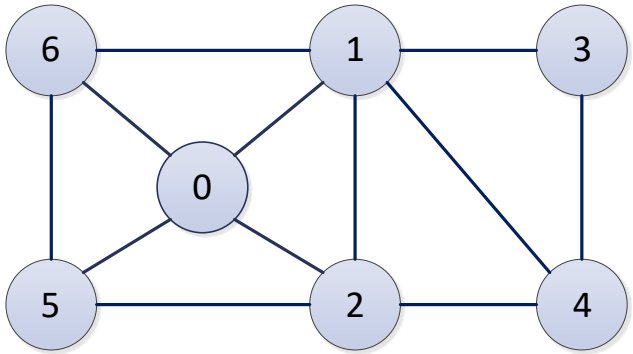
- Задача: для каждой вершины v найти характеристику

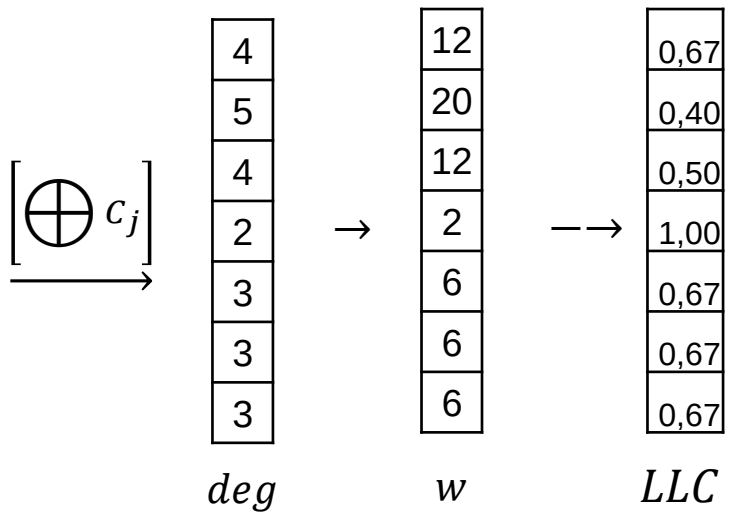
$$LLC(v) = \frac{\text{число треугольников, в которые входит } v}{\text{число возможных ребер между соседями } v}$$

- Алгоритм:
  - Пусть A – матрица смежности, C – симметризованная матрица смежности ($C = A$ или $C = A \vee A^T$)
  1. $T\langle C \rangle = CC$ (`GrB_mxm`) – матрица числа треугольников
  2. $t = [\oplus A[:, j]]$ - число треугольников в каждой строке
  3. $deg = [\oplus C[:, j]]$ - вектор степеней вершин
  4. $w = \deg(\deg - 1)$ - вектор
  5. $LLC = t/w$
- Основная операция – матричное умножение
- Полукольцо: $\boldsymbol{R}$, $\oplus - +, \otimes - \times$, нулевой элемент 0.

Локальный коэффициент кластеризации. Пример







k-Truss

- Задача: найти k-truss графа A – подграф с тем же числом вершин, что и A, в котором каждое ребро входит по крайней мере в $k-2$ треугольников в A.
- Алгоритм итерационный:

  Пусть C_{ij} – *поддержка* ребра (i,j) – число треугольников, содержащих ребро (i,j).

  1. На каждом шаге:
     1. найти $\mathrm{C}\langle\mathrm{C}\rangle = C^2 = CC^T$ (`Gr_mxm`)
     2. Удалить ребра с поддержкой менее $k-2$ (`GxB_select`).
     3. Продолжить, если граф изменился

- Основная операция – матричное умножение

Вычислительные эксперименты SuiteSparse:GraphBLAS

- Источник: Aznaveh M. et al. Parallel GraphBLAS with OpenMP∗ //2020 Proceedings of the SIAM Workshop on Combinatorial Scientific Computing. – Society for Industrial and Applied Mathematics, 2020. – С. 138-148.
- Тестовая инфраструктура: Intel® Xeon® E5- 2698 v4 CPU system with 20 cores running at 2.2 Ghz, 256 GB of RAM and the Intel® icc compiler (19.0.3.199).
- Тестовые матрицы:

Matrix Name	Nodes	Edges	Description
datagen-8_9-fb	10,572,901	848,681,908	LDBC Datagen
datagen-9_2-zf	12,857,671	1,049,527,225	LDBC Datagen
cit-Patents	3,774,768	33,037,894	Citation network among US patents
g-1073643522-268435456	1,073,643,522	268,435,456	Synthetic image grid benchmark matrix
graph500-scale25-ef16	33,554,432	536,870,912	Synthetic graph500 network of scale 25
MAWI/201512020330	226,196,185	480,047,894	Packet trace from WIDE internet backbone

Вычислительные эксперименты SuiteSparse:GraphBLAS

- Задачи:
  - Подсчет числа треугольников
  - 4-Truss – поиск мостов графа
  - Поиск в ширину
  - Алгоритм Беллмана-Форда
  - LCC – подсчет локального коэффициента кластеризации

	40-Thread Speedup Relative to 1 Thread					40-Thread Edge Computation Rate (10^6 edges/s)				
Matrix	Triangle Counting	4-Truss	BFS	Bellman-Ford	LCC	Triangle Counting	4-Truss	BFS	Bellman-Ford	LCC
datagen-8_9-fb	26.6	27.7	3.5	11.6	25.8	15.3	0.6	285.0	114.1	6.7
datagen-9_2-zf	16.9	19.6	*	*	7.9	63.0	3.2	*	*	12.5
cit-Patents	16.1	19.7	2.6	9.1	11.7	87.8	11.7	25.4	23.8	23.0
g-1073643522-268435456	11.2	16.6	3.6	5.2	8.4	268.3	252.4	10.5	0.04	86.0
graph500-scale25-ef16	30.5	*	3.9	9.5	30.2	1.8	0.1	186.6	98.6	0.6
MAWI/201512020330	5.8	13.4	9.7	2.4	5.7	104.5	86.2	48.7	21.7	23.4

Вычислительные эксперименты SuiteSparse:GraphBLAS

- Выводы:
  - Лучшее масштабирование у алгоритмов подсчета треугольников и k-truss, в которых использовано умножение матриц.
  - Ускорение алгоритмов BFS и Беллмана-Форда в 2-4 раза меньше, чем подсчета треугольников. Они используют умножение матрицы на вектор
  - Масштабируемость алгоритма LCC близка к алгоритму подсчета треугольников, но MTEPS меньше, т.к. в алгоритме выполняется больше вычислительных операций.

Заключение

- Большинство алгоритмов на графах можно представить в виде последовательности матрично-векторных операций.
- GraphBLAS – набор стандартных операций над матрицами и векторами, используемый для реализации задач на графах.
- Использование ограниченного набора операций позволяет упростить реализацию новых алгоритмов, облегчить перенос существующих реализаций на новые вычислительные архитектуры, увеличить масштабируемость приложений за счет использования оптимизированных базовых операций.
- Среди реализации стандарта GraphBLAS отметим SuiteSparse::GraphBLAS (параллелизм с технологией OpenMP), CombinatorialBLAS (MPI + OpenMP), GraphBLAST (CUDA).

Заключение

- Для реализации подхода необходимо задать операции умножения и сложения над элементами разреженной матрицы (матрицы смежности графа).
- В число базовых операций, используемых для алгоритмов на графах, входят умножение разреженной матрицы на плотный или разреженный вектор, умножение двух разреженных матриц, поэлементное сложение и умножение, наложение маски.
- Результаты вычислительных экспериментов показывают, что алгоритмы, реализованные с использованием умножения двух матриц масштабируются лучше, чем алгоритмы, реализованные с помощью умножения матрицы на вектор.

Литература

1. Kepner J., Gilbert J. (ed.). Graph algorithms in the language of linear algebra. – Society for Industrial and Applied Mathematics, 2011.
2. Kepner J. et al. Graphs, matrices, and the GraphBLAS: Seven good reasons //Procedia Computer Science. – 2015. – Т. 51. – С. 2453-2462. [pdf]
3. Kepner J. et al. Mathematical foundations of the GraphBLAS //2016 IEEE High Performance Extreme Computing Conference (HPEC). – IEEE, 2016. – С. 1-9. [pdf]
4. Kepner J. GraphBLAS Mathematics-Provisional Release 1.0 //GraphBLAS. org, Tech. Rep. – 2017. [pdf]
5. Davis T. A. Algorithm 1000: SuiteSparse: GraphBLAS: Graph Algorithms in the Language of Sparse Linear Algebra //ACM Trans. on Math. Software (TOMS). – 2019. – Т. 45. – №. 4. – С. 1-25.
6. Aznaveh M. et al. Parallel GraphBLAS with OpenMP // 2020 Proceedings of the SIAM Workshop on Combinatorial Scientific Computing. – Soc. for Industrial and Applied Mathematics, 2020. – С. 138-148. [pdf]

Литература

7. Szarnyas G. GraphBLAS: A linear algebraic approach for high-performance graph algorithms // FOSDEM, 2020. [pdf]
8. Mattson T. et al. LAGraph: A community effort to collect graph algorithms built on top of the GraphBLAS //2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW). – IEEE, 2019. – С. 276-284. [pdf]
9. Yang C., Buluc A., Owens J. D. GraphBLAST: A high-performance linear algebra-based graph framework on the GPU //arXiv preprint arXiv:1908.01407. – 2019. [pdf]

Максимальное независимое множество

- Задача: найти в графе множество максимального размера, в котором никакие две вершины не смежны.

- Основные операции: умножение матрицы на вектор, объединение и пересечение множеств
- Полукольцо: $\boldsymbol{R} \cup \{+\infty\}$, $\oplus - \min, \otimes - selectFirst$, нулевой элемент $+\infty$.
- Код: https://github.com/DrTimothyAldenDavis/GraphBLAS/blob/stable/Demo/Source/mis.c

Максимальное независимое множество

- Алгоритм Луби:
  - Множество независимых вершин $I = \emptyset$, множество вершин-кандидатов $C = \{0,1,..n\}$ (вектор-маска)
  - Пока есть вершины-кандидаты:
  1. Найти вектор случайных чисел $P = rand() \odot C$, отмасштабировать по обратной степени (`GrB_apply`)
  2. Вычислить максимальную вероятность для всех соседей C $N_{max} = C \odot (A \otimes P)$ (`GrB_mxv`)
  3. Выбрать вершины New_mem с вероятностью большей, чем у соседей. (`GrB_eWiseAdd`)
  4. Добавить New_mem в I: $I = I \oplus New\_mem$ (`GrB_eWiseAdd`)
  5. Удалить New_mem из множества кандидатов: $C = C \,\&\, !New\_mem$ (`GrB_eWiseMult`)
  6. Удалить соседей New_mem из множества кандидатов $New_{neigb} = C \odot A \otimes New\_mem$, $C = C \,\&\, !New\_neib$ (`GrB_mxv, GrB_eWiseMult`)