Семинар

Dynamic Graphs on the GPU

Егор Орачев

JetBrains Research, Лаборатория языковых инструментов
Санкт-Петербургский Государственный университет

2 Октября 2020

Мотивация

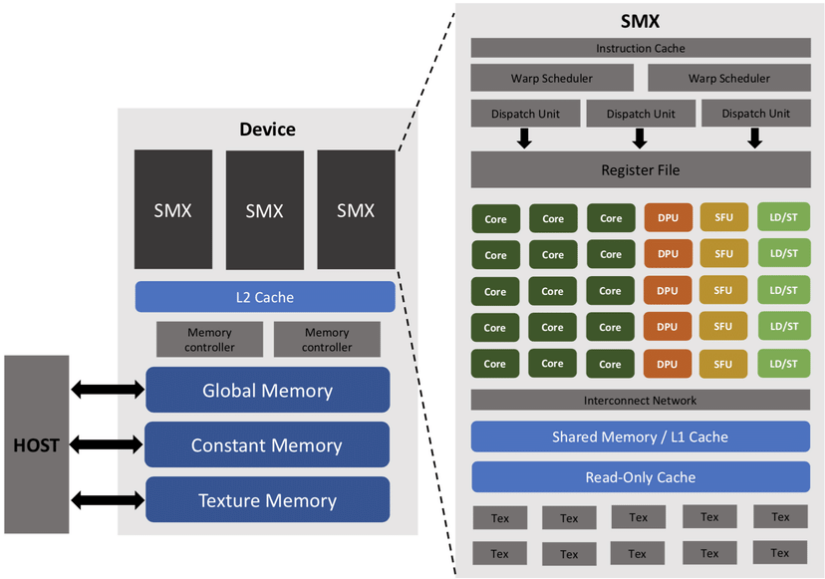


Figure: Generic GPU Architecture

- Развитие GPU как платформы для вычислений
- Увеличение количества обрабатываемых данных
- Все больше и больше вычислительно сложных операций переходят на GPU (особенно, если мы хотим оставаться в пределах одного кластера)

Идея

- На данный момент большинство алгоритмов обработки графов на GPU предполагают неизменяемость данных
- Большинство вычислений выглядят как: загрузить в VRAM, посчитать, выгрузить обратно в RAM
- Вопрос: можно ли поддерживать структура графа в VRAM, и осуществлять доступ к ней посредством операций, которые будут выполняться на GPU

Динамические графы на GPU

- В свежей статье[1] приведены результаты, показывающие, что теоретически возможно поддерживать граф на GPU
- Цель: поддерживать *истинно* динамические сценарии обновления, т.е. непрерывное добавление, удаление вершин и ребер, а также высокую скорость поиска ребер
- Допущения: $|E| \ll |V|^2$
- Реализация: CUDA C

[1] Dynamic Graphs on the GPU, Muhammad A. Awad, Saman Ashkiani, Serban D. Porumbescu et al., ссылка: https://escholarship.org/uc/item/48j4k7np

Структура графа

- Граф: $G = (V, E, W)$
- Ребро графа: $e = (u, v, w)$
- Все ребра $e = (u, v, *)$ уникальны
- Ребра индексируются *uint32*: до 4294967296 вершин
- Веса индексируются *uint32*: до 4294967296 уникальных весов
- Специальное значение $\perp$ для пустых ячеек

Структура графа

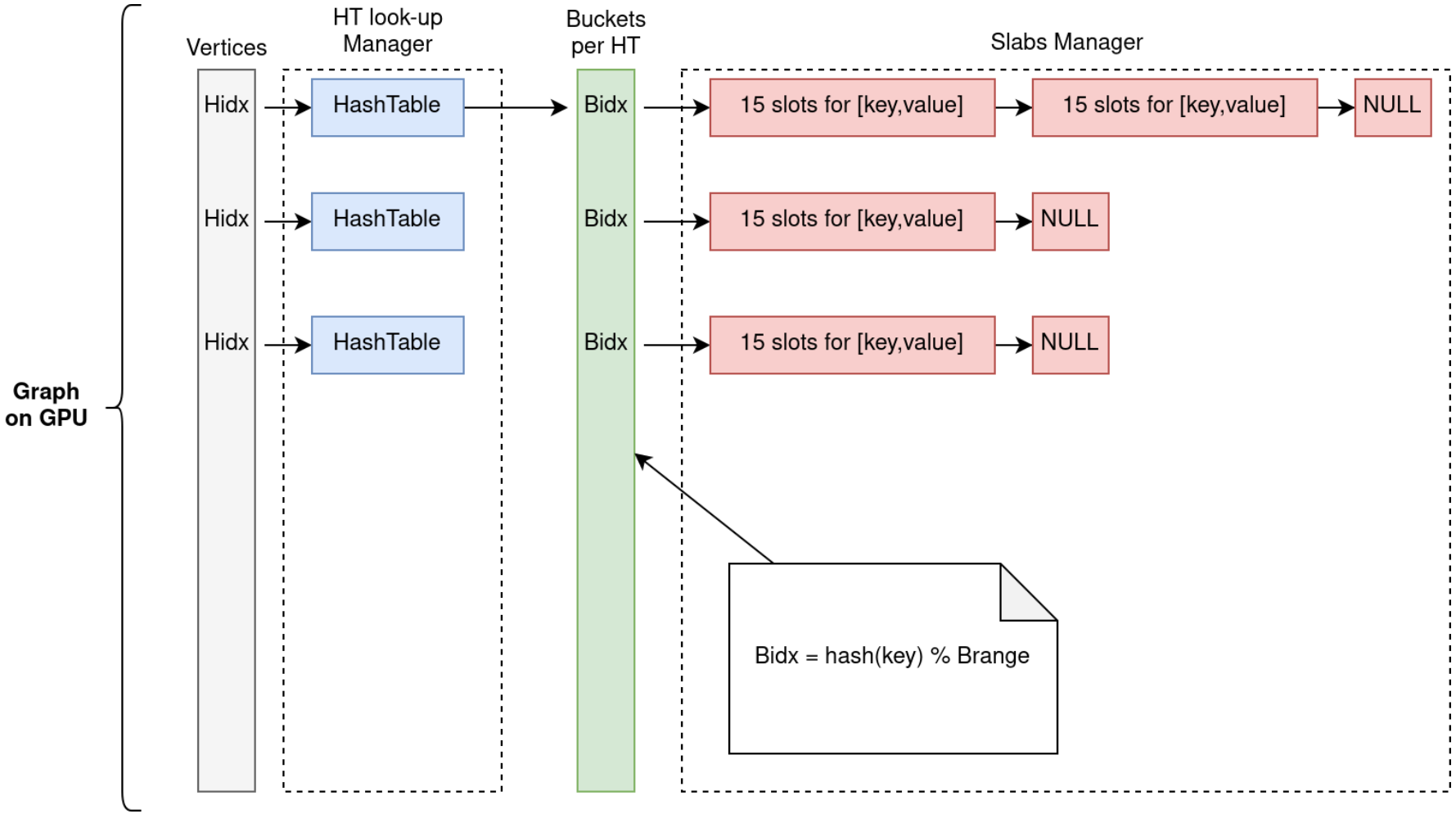


Figure: Graph Structure on the GPU

Операции над графом

- Добавление ребра (u, v, w)
- Удаление ребра (u, v)
- Добавление вершины u
- Удаление вершины u
- Поиск ребра вида (u, v)
- **В данной реализации дублирование ребер с разными весами запрещено: при вставке ребра, если до этого существовало такое-же ребро, но с другим весом, то оно будет заменено**

Реализация

- Как это устроено?
- Warp Cooperative Work Sharing (WCWS)
- GPU Slab Hash Table[2]

[2] A Dynamic Hash Table for the GPU, Saman Ashkiani, Martin Farach-Colton, John D. Owens, ссылка: https://arxiv.org/abs/1710.11246

WCWS: Идея

- GPU состоит из стриминговых мульти-процессоров (SMs)
- В каждом таком SM фиксированное количество потоков, которые сгруппированы в SIMD блоки размером 32 потока, называемые warp
- Каждый такой warp может исполнять только один **путь** программы на своих потоках
- Сильное ветвление $\Longrightarrow$ замедление warp до 32 раз
- Решение: писать программу таким образом, что на warp почти всегда будет один путь ветвления, но на 32 вариациях данных

WCWS: Реализация

- Каждый поток внутри warp имеет свою задачу
- Пока у потоков есть задачи:
  - *Голосуют*, чья задача выполняется раньше
  - Получают все одну задачу
  - Читают непрерывный блок памяти, кратный 32
  - Определяют свой статус
  - Выбирают того, кто преуспел. Тот, кто преуспел, записывает результат
  - Если никто не преуспел, переходят к следующему блоку памяти, иначе - задача решена

GPU Slab Hash Table

- Разрешение коллизий - метод цепочек
- В качестве узлов - блоки памяти на 128 байт
- Один узел вмещает M = 15 пар (key,value), 4 байта на вспомогательную информацию, и 4 байта на индекс следующего узла в списке
- Максимальная эффективности по памяти 94%
- Среднее количество узлов в цепочке $\beta = \lceil N\ (M * B) \rceil$
- Эффективность по памяти: $\frac{N*x}{(M*x+y)*\Sigma k_i} \leq \frac{M*x}{M*x+y}$, где x - размер пары, y - размер индекса следующего узла, k_i - длина i-ой цепочки

GPU Slab Hash Table

```
1:  __device__ void warp_operation(bool &is_active, uint32_t &myKey, uint32_t &myValue) {
2:  next ← BASE_SLAB;
3:  work_queue ← __ballot(is_active);
4:  while (work_queue != 0) do
5:      next ← (if work_queue is changed) ? (BASE_SLAB) : next;
6:      src_lane ← next_prior(work_queue); src_key ← __shfl(myKey, src_lane);
7:      src_bucket ← hash(src_key); read_data ← ReadSlab(next, laneId);
8:      warp_search_macro() OR warp_replace_macro() OR warp_delete_macro()
9:      work_queue ← __ballot(is_active);
10: end while
11: }
```

Figure: Slab Hash Table Algo Pseudocode

GPU Slab Hash Table

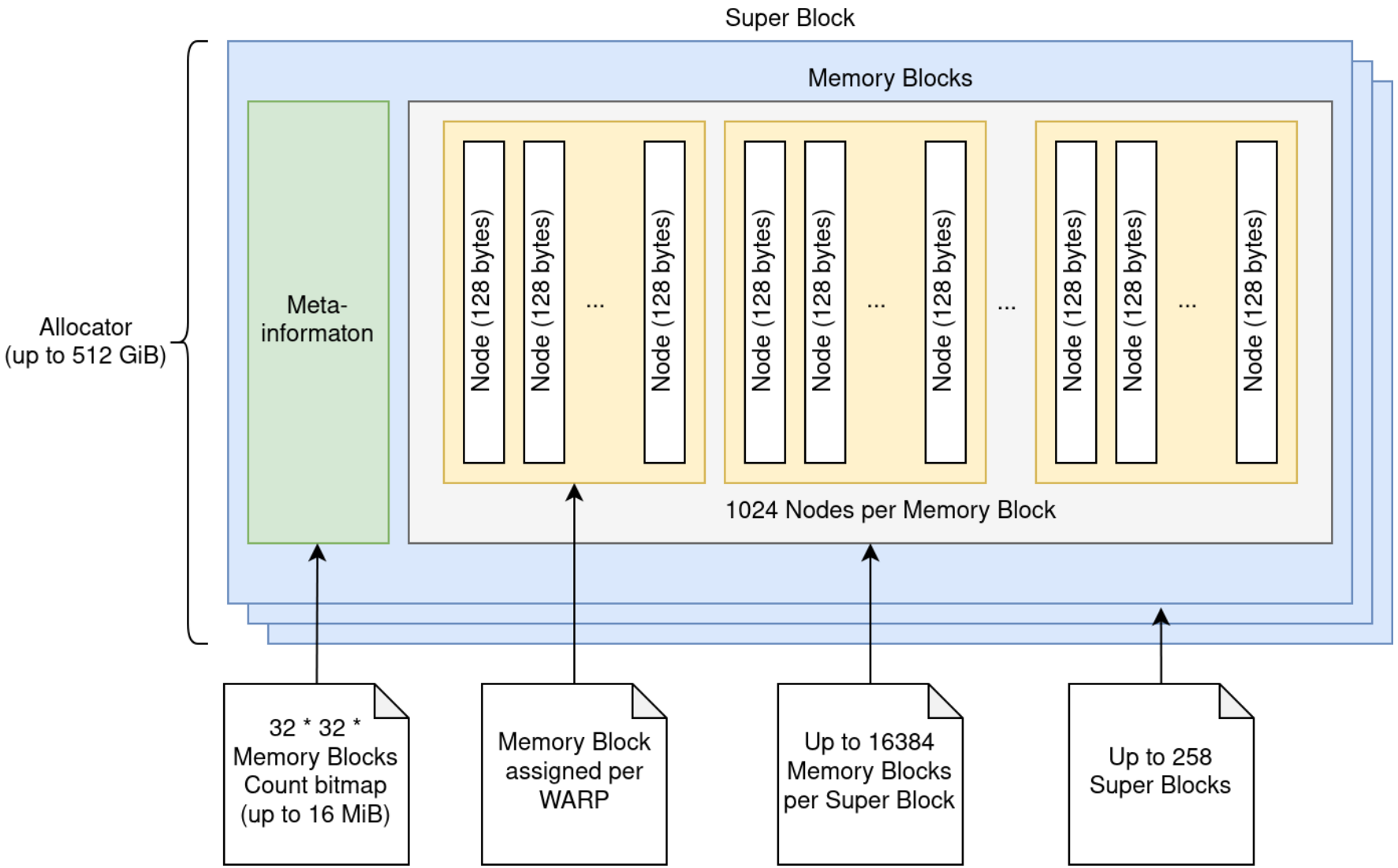


Figure: VRAM Memory Allocator for Slab Hash Table

Динамические графы на GPU: Реализация

- Помним про WCWS стратегию
- Вставка/удаление ребер:
  - Очередь ребер на warp
  - Выбираем ребра с одинаковым src голосованием
  - Получаем *src* вершину
  - $table \leftarrow graph[src]$
  - $table.replace(dst, weight)$ или $table.delete(dst)$
  - Уменьшаем или увеличиваем число вершин, смежных с *src*

Динамические графы на GPU: Реализация

- Удаление вершин немного сложнее:
  - Очередь вершин на warp
  - Выбираем вершину голосованием
  - Итерируемся по ее ребрам
  - Для каждой такой *dst* вершины исходящего ребра
  - $table \leftarrow graph[dst]$
  - $table.delete(src)$
  - Освобождаем память при необходимости

Динамические графы на GPU: Датасет

Dataset	Vertices	Edges	Degree			
			Min.	Max.	Avg.	σ
luxembourg_osm	114K	239K	1	6	2.1	0.41
germany_osm	11.5M	24.7M	1	13	2.1	0.51
road_usa	23.9M	57.71M	1	9	2.4	0.85
delaunay_n23	8.4M	50.3M	3	28	6.0	1.33
delaunay_n20	1M	6.3M	3	23	6.0	1.33
rgg_n_2_20_s0	1M	13.8M	0	36	13.1	3.62
rgg_n_2_24_s0	16.8M	265.1M	0	40	16.0	3.99
coAuthorsDBLP	299K	1.9M	1	336	6.4	9.80
ldoor	952K	45.5M	27	76	47.7	11.97
soc-LiveJournal1	4.8M	85.7M	0	20K	17.2	50.65
soc-orkut	3M	212.7M	1	27K	70.9	139.72
hollywood-2009	1.1M	112.8M	0	11K	98.9	271.70

Figure: Graphs Datasets

Динамические графы на GPU: Замеры

Batch size	Hornet	faimGraph	Ours
2^{16}	33.67	92.47	501.33
2^{17}	44.71	133.97	513.56
2^{18}	51.43	157.15	591.06
2^{19}	70.81	188.98	641.25
2^{20}	83.54	—	664.52
2^{21}	97.41	—	658.09
2^{22}	110.89	—	646.01

(a) Insertions

Batch size	Hornet	faimGraph	Ours
2^{16}	91.73	111.71	640.63
2^{17}	159.69	112.96	886.92
2^{18}	259.31	171.75	947.60
2^{19}	377.79	257.66	939.98
2^{20}	537.73	—	988.85
2^{21}	739.82	—	1,007.16
2^{22}	1,024.87	—	1,015.47

(b) Deletions

Figure: Edge operations rate in MEdges/s

Динамические графы на GPU: Замеры

Dataset	Hornet	Ours
luxembourg_osm	5.562	0.184
germany_osm	330.311	12.407
road_usa	644.308	27.910
delaunay_n23	273.532	19.590
delaunay_n20	37.68	2.494
rgg_n_2_20_s0	37.084	5.053
rgg_n_2_24_s0	—	0.697
coAuthorsDBLP	11.672	0.835
ldoor	46.486	15.936
soc-LiveJournal1	179.879	26.176
soc-orkut	—	39.907
hollywood-2009	90.705	42.387

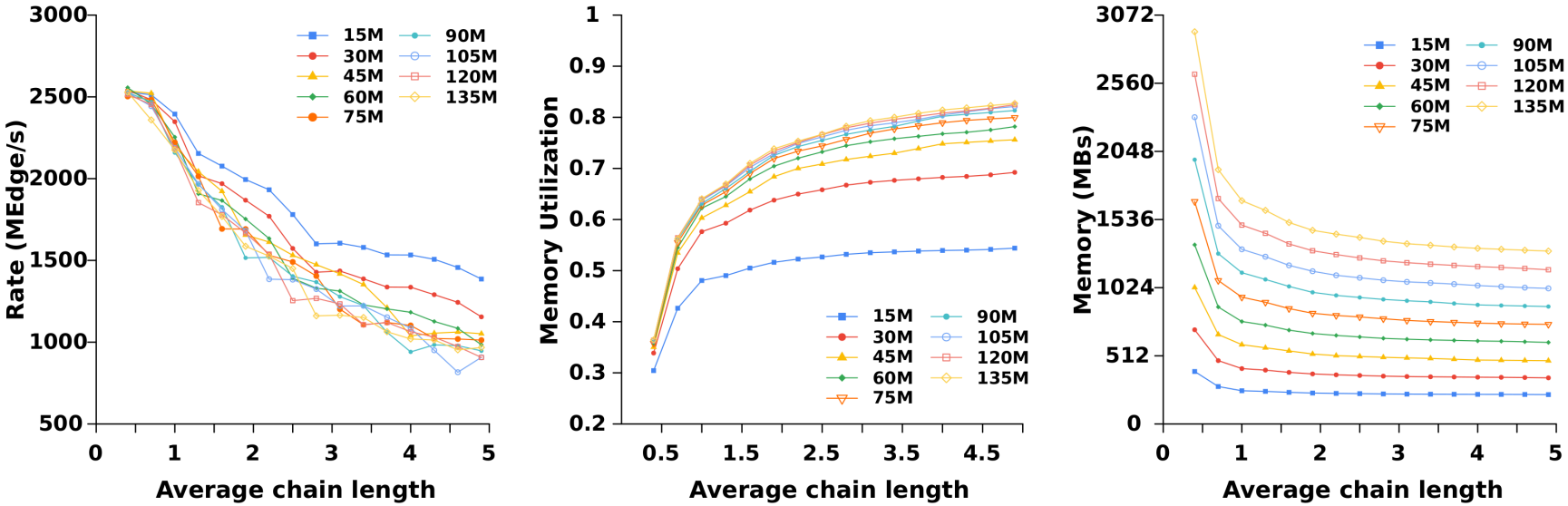
(a) Bulk

Batch size	Hornet	Ours
2^{20}	164.44	841.31
2^{21}	176.96	945.64
2^{22}	184.75	993.82

(b) Incremental

Figure: Graph building

Динамические графы на GPU: Замеры



(a) Insertion Rate

(b) Memory Utilization

(c) Memory Usage

Figure: Loadfactor scalability

Итоги

- Операции группируются *per-batch*
- Только одна метка на ребро
- Скорость vs. Количество расходуемой памяти
- Хороший пример того, как использовать WCWS
- Решит ли unified-memory все проблемы?

Дополнительно

- Почта: egororachyov@gmail.com
- Материалы презентации:
  - Dynamic Graphs on the GPU, Muhammad A. Awad, Saman Ashkiani, Serban D. Porumbescu et al., ссылка: https://escholarship.org/uc/item/48j4k7np
  - A Dynamic Hash Table for the GPU, Saman Ashkiani, Martin Farach-Colton, John D. Owens, ссылка: https://arxiv.org/abs/1710.11246