



Efficient Memory-access for Out-of-memory Graph-traversal In GPUs

Егор Орачев

JetBrains Research, Лаборатория языковых инструментов
Санкт-Петербургский Государственный университет

20 октября 2020

Мотивация

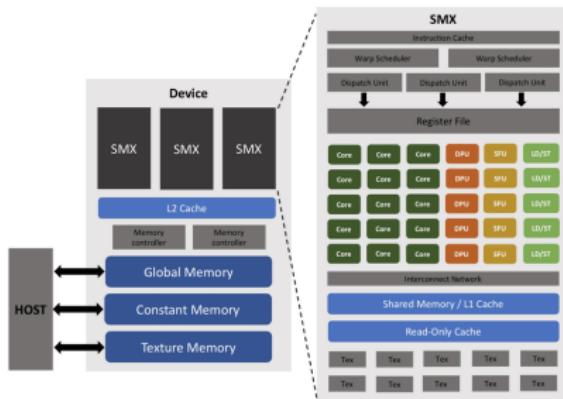


Figure: Generic GPU Architecture

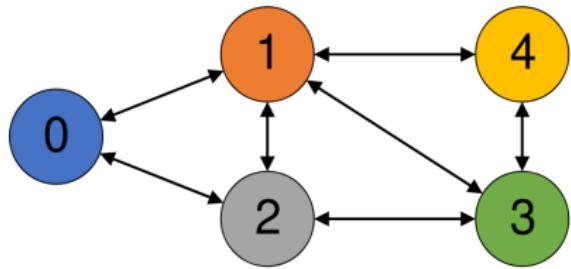
- Современные системы аналитики и рекомендаций все чаще строятся на основе анализа графовых данных
- Реальные графовые данные имеют невероятно большой размер
- Возможность для распараллеливания обработки
- Использование GPU для решения подобного рода задач

Взгляд с высоты

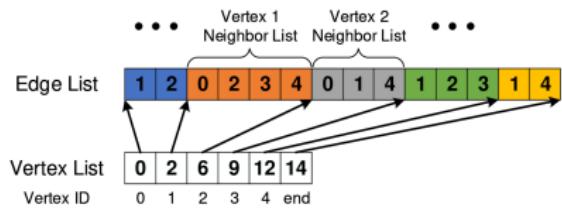
```
set_initial_active_vertex()
for all vertices  $v_1$  in Graph  $G$  do
    if  $v_1$  is active then
        set  $v_1$  as inactive
        for all neighbors  $v_2$  of  $v_1$  do
            application_dependant_workload()
            if application_dependant_condition() then
                | set  $v_2$  as active
            end
        end
    end
end
end
```

Figure: High-level graph traversal flow

Пример графа и матрицы



(a) Graph



(b) CSR matrix

Figure: Sample directed graph and its CSR adjacency matrix

Проблема

- Большинство реальных данных не помещается полностью в видеопамять GPU
- Данные сильно разрежены, средняя степень вершины 71
- UVM влечет избыточное количество I/O операций
 - ▶ Слабая локальность реальных данных
 - ▶ Размер страницы 4КБ
 - ▶ Копирование большого количества данных при сравнительно небольших запросах
- Необходима реструктуризация данных
 - ▶ Дополнительное время и ресурсы на выполнение
 - ▶ Дополнительные усилия со стороны программиста

Решение

- Zero-copy, или *direct access*
- Не требуется предварительная обработка данных
- Граф представляется в виде CSR матрицы смежности
- $|E| \ll |V|^2$
- Размер графа много больше доступной видеопамяти

Предварительные знания

- Peripheral Component Interconnect Express (PCIe) — стандарт высокоскоростной шины расширения для последовательной передачи данных
- Физическая топология — звезда
- Программная топология — шина
- Независимая передача данных в обе стороны
- Распространенный интерфейс на материнских платах для подключения графических адаптеров, внешних накопителей, Wi-Fi и Ethernet компонент

Предварительные знания

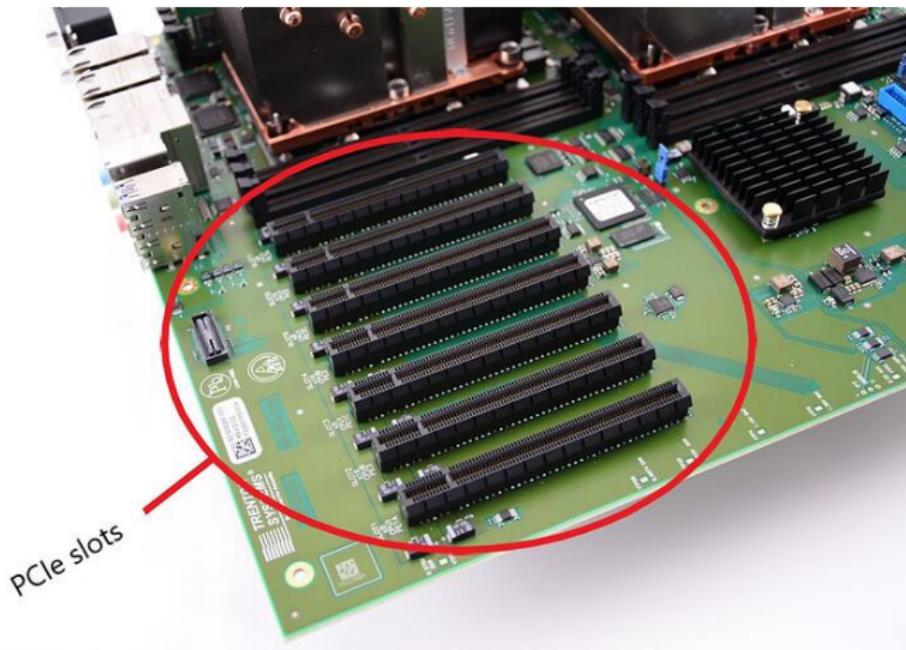


Figure: PCIe slots located on the motherboard

Zero-copy: концепция

- Доступ к zero-copy региону памяти осуществляется так, словно это глобальный регион памяти GPU
- GPU под капотом преобразует запрос к этому региону и направляет его через внешний интерфейс, например PCIe, в систему
- На запрос может ответить любое устройство, которое способно отображать свои регионы памяти (*memory-map*) в пространство адресов транспортной шины, т.е. PCIe интерфейса, например

Zero-copy: пример

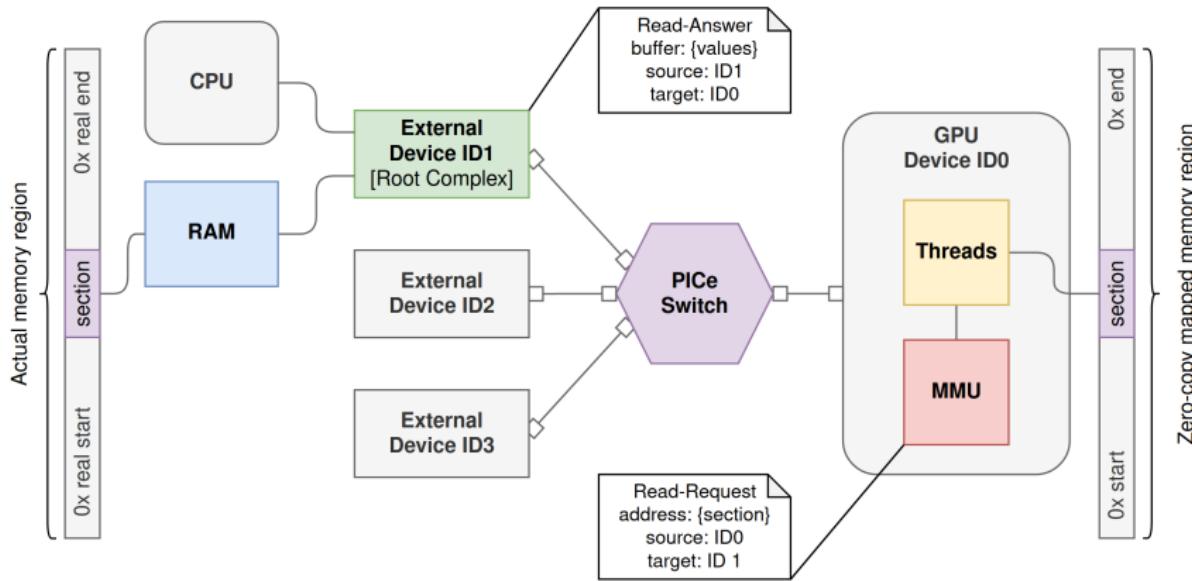


Figure: Zero-copy mechanism with PCIe

Zero-copy: механизм

- С точки зрения системы
 - ▶ Выделить требуемый буфер в оперативной (*host*) памяти
 - ▶ Закрепить (*pinning*) буфер в памяти, т.е. запретить перемещение или вытеснение страниц этого региона
 - ▶ Отобразить адреса буфера в таблицу страниц GPU
 - ▶ Получить адрес буфера для использования в GPU ядрах
- С точки зрения CUDA API
 - ▶ Выделить память с помощью
`cudaError_t cudaMallocHost(void** ptr, size_t size)`
 - ▶ Описание функции: allocates *size* bytes of host memory that is page-locked and accessible to the device. Since the memory can be accessed directly by the device, it can be read or written with much higher bandwidth than pageable memory obtained with functions such as `malloc()`.

Монитор трафика

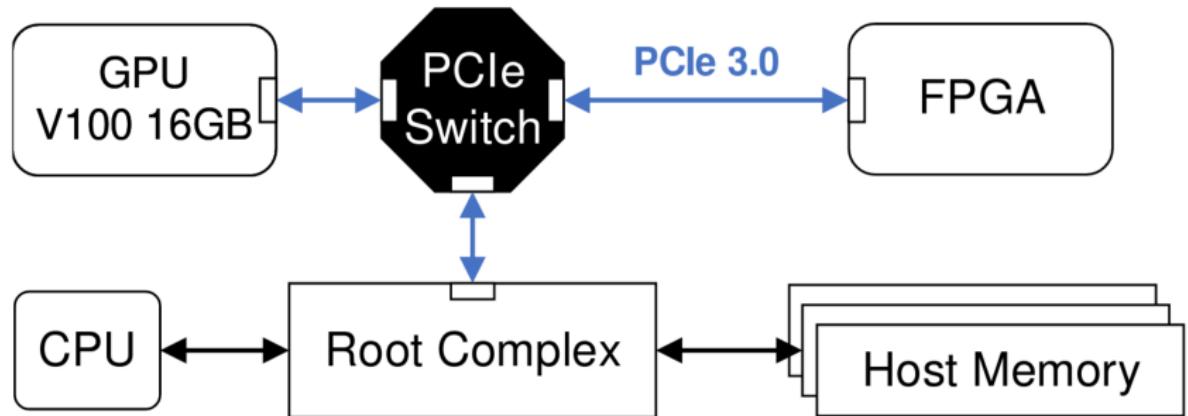


Figure: PCIe traffic monitoring environment. The FPGA is used to characterize the zero-copy memory access pattern from GPU

Паттерны доступа

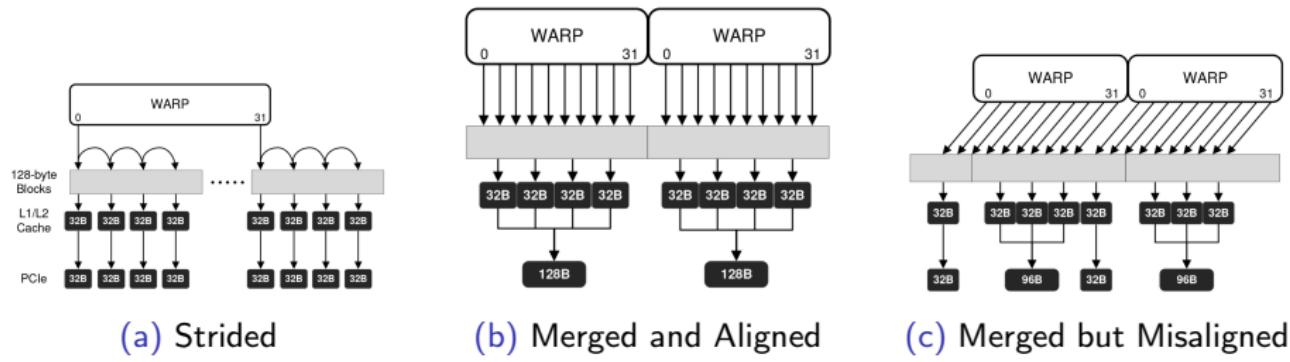


Figure: GPU PCIe memory request patterns observed with FPGA

Исследование

- Strided

- ▶ Запросы размером 32 байта
- ▶ Размер заголовка PCIe 3.0 TLP пакета 18 байт, overhead 36%
- ▶ Максимальная пропускная способность с RTT $1\mu s$: 32 байта
 $/1\mu s * 256 = 7.63GB/s$
- ▶ Пропускная способность с RTT $1.6\mu s$: $4.77GB/s$
- ▶ Минимальный запрос к DDR4 DRAM 64 байта
- ▶ Максимальная скорость DDR4 2400MHz DRAM $19.2GB/s$
- ▶ С учетом размера запроса в 32 байта: $9.6GB/s$

- Merged and Aligned

- ▶ Запросы размером 128 байт
- ▶ Overhead заголовка PCIe TLP пакета: 12.3%
- ▶ Достаточно, чтобы заполнить PCIe 3.0 канал в $16GB/s$

- Merged but Misaligned

Результаты

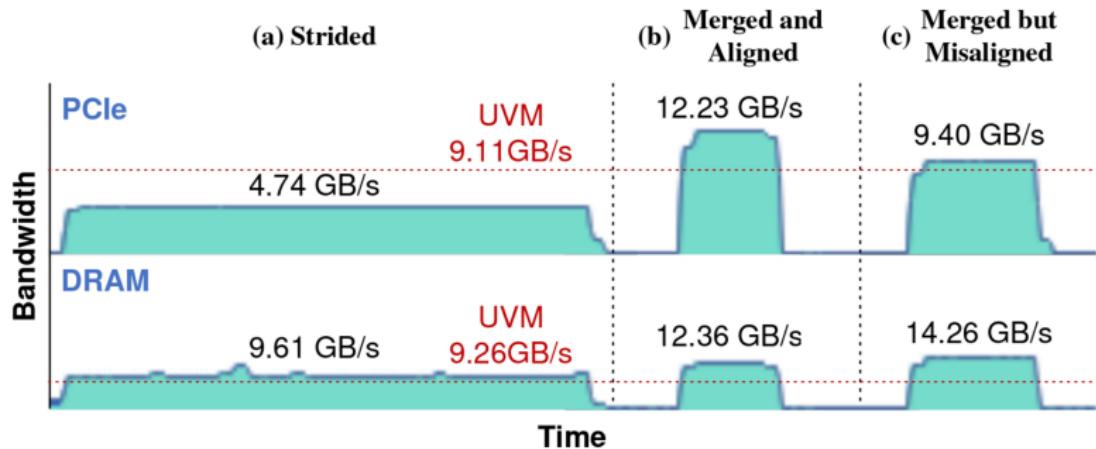


Figure: Average PCIe and DRAM bandwidth utilization for the different zero-copy access patterns

```

1 void naive(*edgeList, *offset, ...) {
2     thread_id = get_thread_id();
3     ...
4     start = offset[thread_id];
5     end = offset[thread_id + 1];
6
7     // Each thread loops over a chunk of edge list
8     for (i = start; i < end; i++) {
9         edgeDst = edgeList[i];
10        ...
11    }
12    ...
13 }
```

(a) Naïve Uncoalesced Memory Access

```

1 #define WARP_SIZE 32
2
3 void aligned(*edgeList, *offset, ...) {
4     thread_id = get_thread_id();
5     lane_id = thread_id % WARP_SIZE;
6     // Group by warp
7     warp_id = thread_id / WARP_SIZE;
8     ...
9     start_org = offset[warp_id];
10    // Align starting index to 128-byte boundary
11    start = start_org & ~0xF; // 8-byte data type
12    end = offset[warp_id + 1];
13
14    // Every thread in a warp goes to the same edgelist
15    for (i = start; i < end; i += WARP_SIZE) {
16        // Prevent underflowed accesses
17        if (i >= start_org) {
18            edgeDst = edgeList[i + lane_id];
19            ...
20        }
21    }
22    ...
23 }
```

(b) EMOGI Coalesced Memory Access
(Merged and Aligned)

Figure: Code listing for solution

Эксперименты: система

Category	Specification
CPU	Dual Socket Intel Xeon Gold 6230 20C/40T
Memory	DDR4 2933MHz 256GB in Quad Channel Mode
GPU	Tesla V100 HBM2 16GB, 5120 CUDA cores
OS	CentOS 8.1.1911 & Linux kernel 5.5.13
S/W	NVIDIA Driver 440.82 & CUDA 10.2.89

Figure: Evaluation system configuration

Эксперименты: данные

Sym.	Graph	Number		Size (GB)	
		V	E	E	w
GK	GAP-kron [10]	134.2M	4.22B	31.5	15.7
GU	GAP-urand [10]	134.2M	4.29B	32.0	16.0
FS	Friendster [52]	65.6M	3.61B	26.9	13.5
ML	MOLIERE_2016 [48]	30.2M	6.67B	49.7	24.8
SK	sk-2005 [12–14]	50.6M	1.95B	14.5	7.3
UK5	uk-2007-05 [13, 14]	105.9M	3.74B	27.8	13.9

Figure: Graph Datasets. V = Vertex, E = Edge, and w = Weight

Эксперименты: данные

- Вершины и ребра индексируются 8 байтными числами
- Веса представляются 4 байтными числами
- Средняя степень вершины 38
- Средняя степень вершины в ML: 222
- Для алгоритмов с одной стартовой вершиной случайным образом выбирались 64 начальные вершины, которые имеют не 0 число исходящих ребер

Эксперименты: результаты

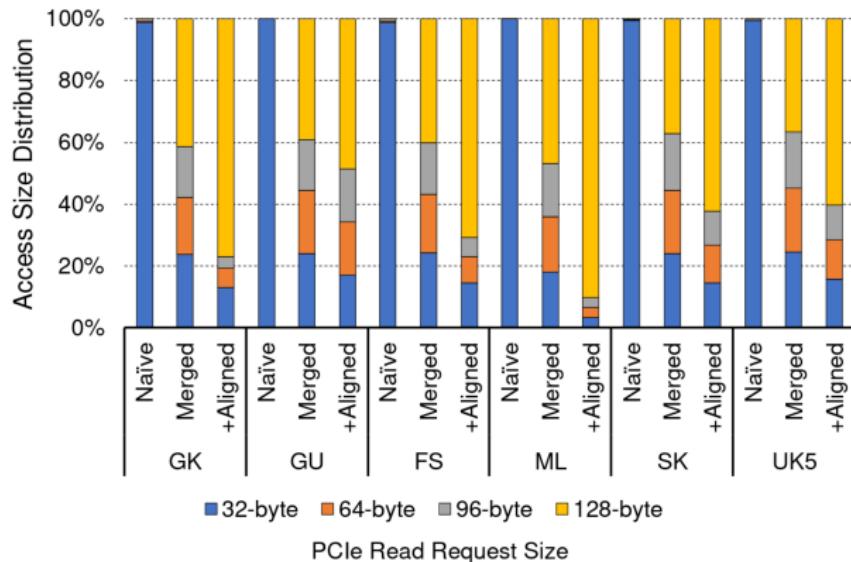


Figure: Distribution of PCIe read request sizes in BFS.+Aligned is abbreviation for Merged+Aligned. As the merged and aligned optimizations are added, the BFS application generates more 128-byte requests for efficient access

Эксперименты: результаты

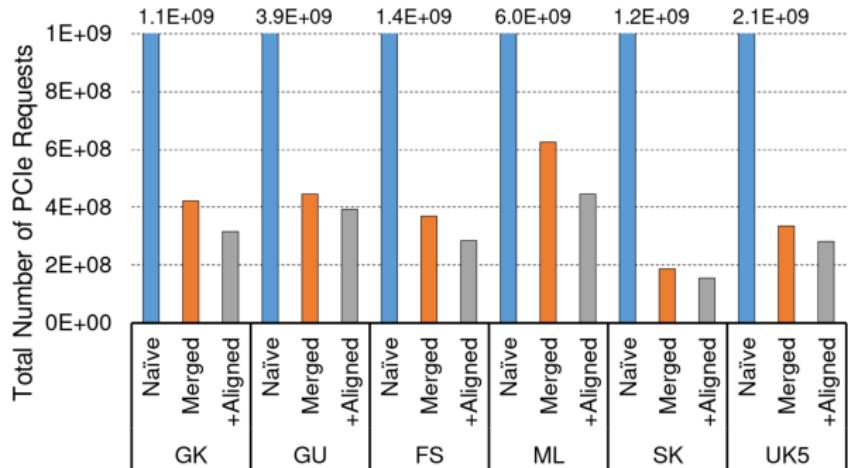


Figure: Number of PCIe requests sent for Naive, Merged and Merged+Aligned implementations while executing BFS on various graph. Collected from FPGA. Merged optimization reduces the PCIe memory requests by up to 83.3% compared to the Na "ive implementation. Merged+Aligned optimization can further reduce the PCIe memory requests by up to 28.8%. +Aligned is abbreviation for Merged+Aligned

Эксперименты: результаты

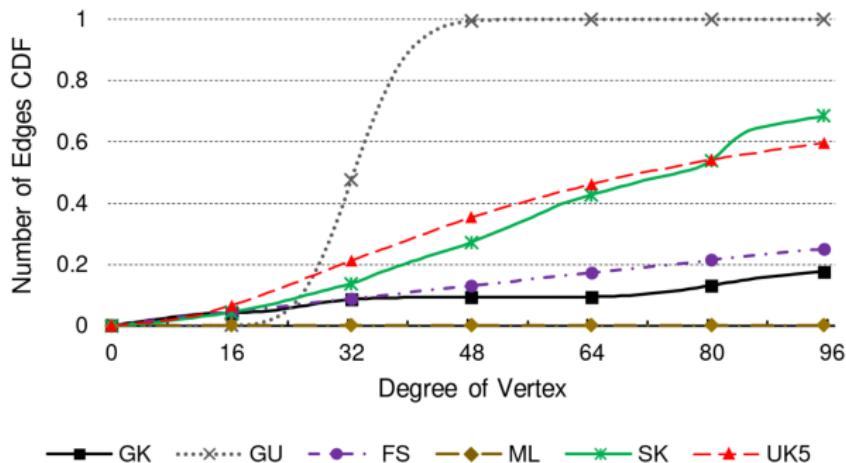


Figure: Number of edges CDF of evaluation graph. This plot provides us a better understanding of the distribution of the neighbor list sizes in the graphs. For example, the GU graph has all of its edges associated with vertices with degree between 16 and 48, meaning the neighbor lists contain at most 48 neighbors

Эксперименты: результаты

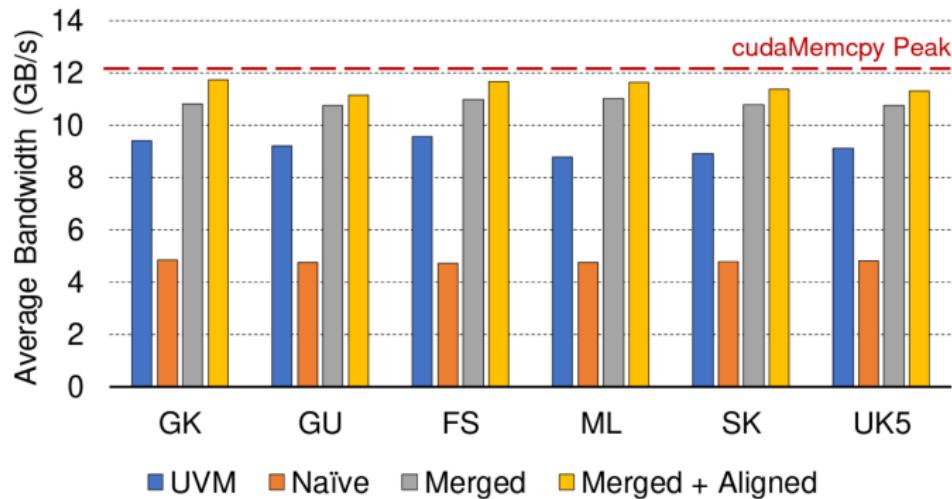


Figure: Average PCIe 3.0 x16 bandwidth utilization of the different implementations executing BFS. The Merged+Aligned implementation can nearly saturate the available PCIe bandwidth

Эксперименты: результаты

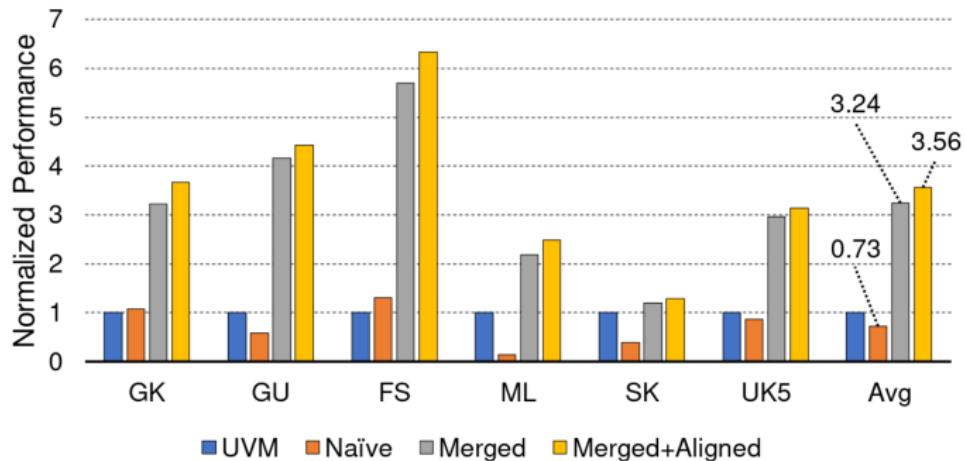


Figure: BFS performance of the Naïve, Merged and Merged+Aligned implementations against the UVM baseline. EMOGI's Merged+Aligned implementation provides the best performance across all graphs

Эксперименты: результаты

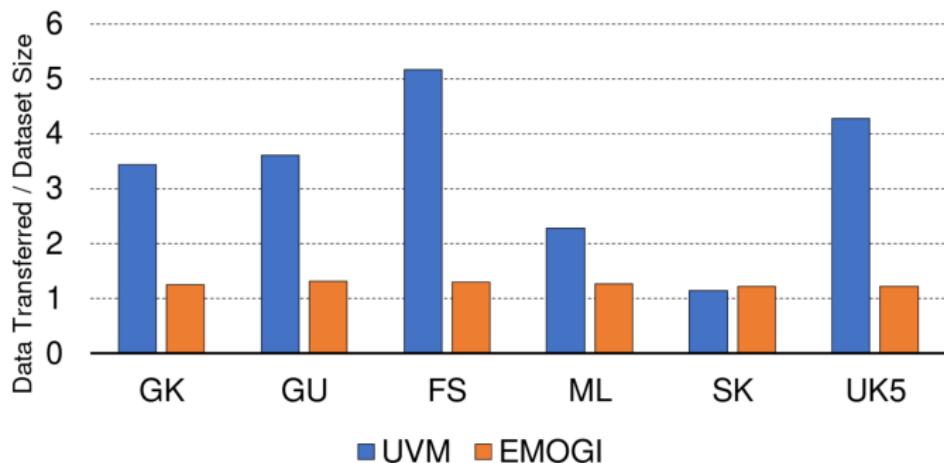


Figure: I/O Read Amplification of EMOGI and the UVM baseline while performing BFS. EMOGI has far less I/O read amplification when the graph sizes are significantly larger than the GPU memory

Эксперименты: результаты

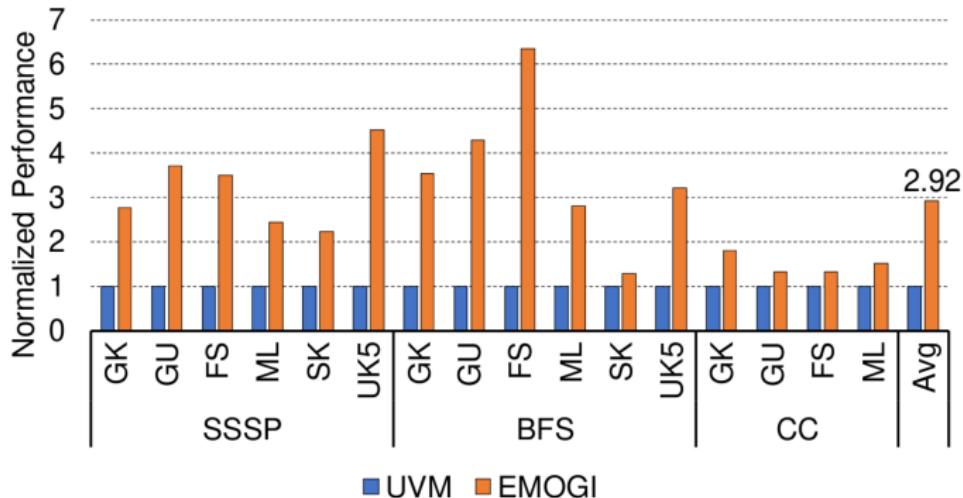


Figure: Performance comparison between UVM and EMOGI with different graph traversal applications. EMOGI is 2.92 \times faster than UVM on average

Эксперименты: результаты

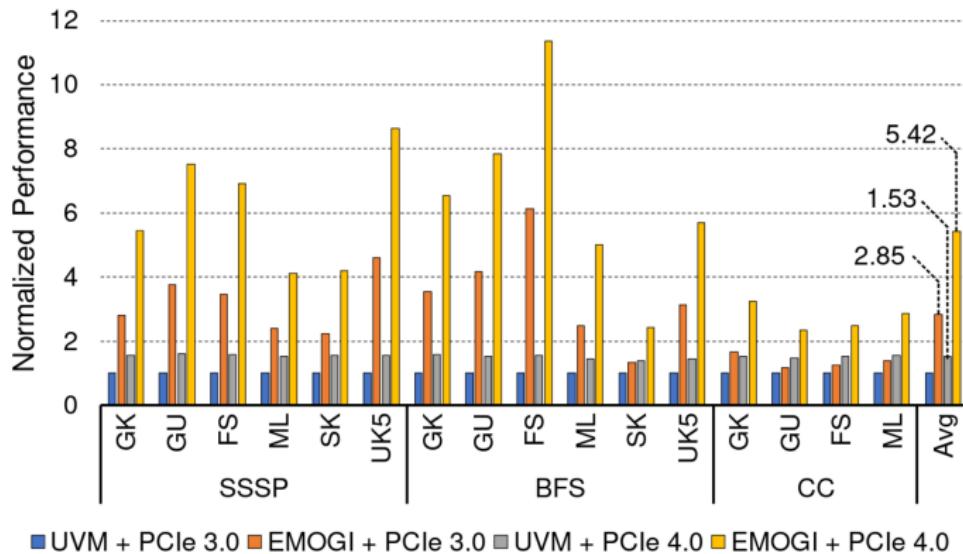


Figure: Performance comparison between UVM and EMOGI using PCIe 3.0 and PCIe 4.0. All results are measured in DGX A100. EMOGI is able to scale almost linearly with the PCIe bandwidth

Дополнительно

- Почта: egororachyov@gmail.com
- Материалы презентации:
 - ▶ EMOGI: Efficient Memory-access for Out-of-memory Graph-traversal In GPUs, Seung Won Min, Vikram Sharma Mailthody, Zaid Qureshi, Jinjun Xiong, Eiman Ebrahimi, Wen-mei Hwu, ссылка: <https://arxiv.org/abs/2006.06890>