Результаты эксперимента

Артем Демченко, Александра Лановая, Артем Ржанков

Эксперимент

Цель: Сравнение средней работы различных реализаций алгоритмов Борувки и Прима

Шаги:

- Провести 20 запусков алгоритмов
- Построить доверительные интервалы
- Проанализировать результаты

Вопросы:

- На какой из трех библиотек (**spla**, **gunrock, graphblas(lagraph)**) быстрее реализация алгоритма Борувки?
- На какой из двух библиотек (**spla** и **gunrock**) быстрее реализация алгоритма Прима?

Характеристики машины

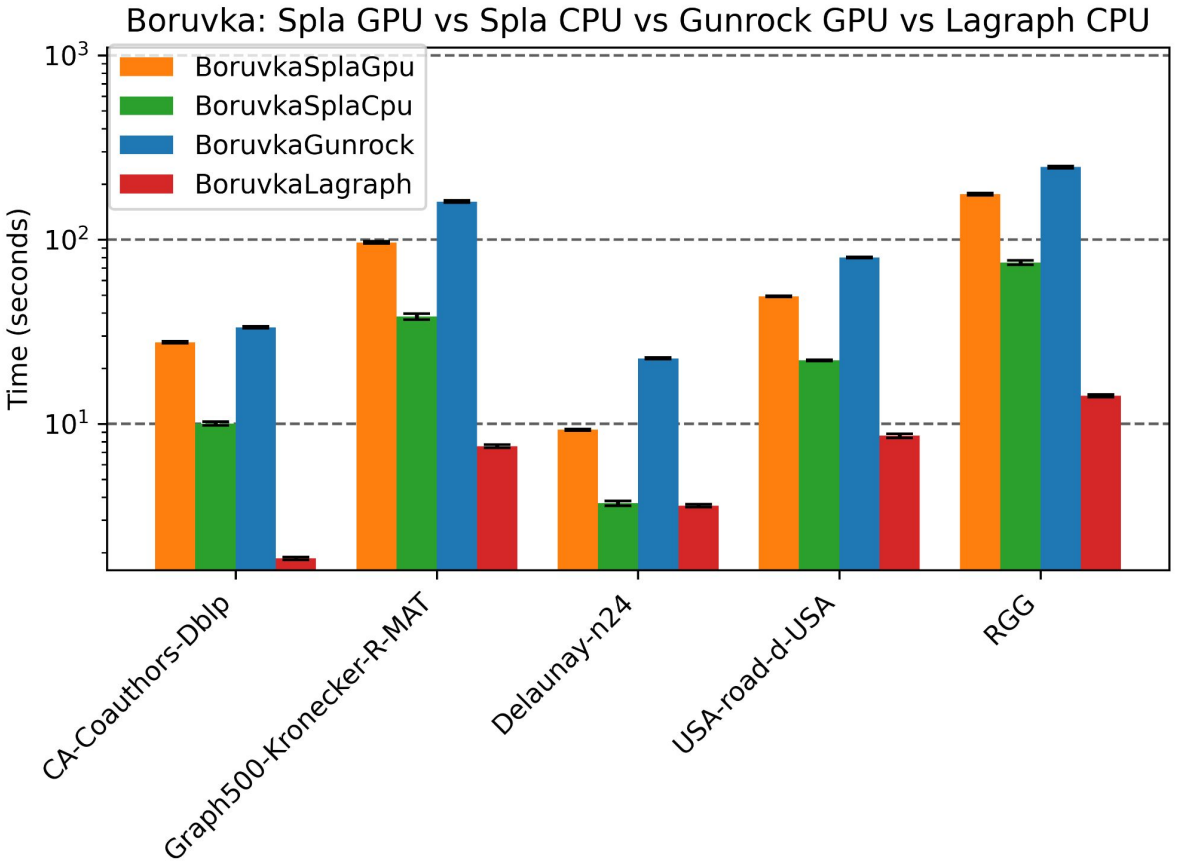
- ОС Ubuntu 24.04 LTS
- Видеокарта NVIDIA GeForce RTX 3050 Ti Laptop GPU
  - 4Gb VRAM
  - Драйвер версии 32.0.15.5597
  - CUDA Toolkit 12.3
  - 2560 CUDA-ядер
- Процессор 11th Gen Intel Core i7-11800H @ 2.30GHz
  - Кэш L1 – 640Kb, Кэш L2 – 10Mb, Кэш L3 – 24Mb
- 16 GB RAM
- OpenCL:
  - Версия: OpenCL 3.0
  - PoCL 7.0

Датасет

Граф	**Количество вершин, ×10⁶**	**Количество ребер, ×10⁶**
CA-Coauthors-Dblp	0.54	15
Graph500-Kronecker-R-MAT	2	32
Delaunay-n24	4.1	3.1
USA-road-d-USA	4.1	8.5
RGG	4.1	30

“A High-Performance MST Implementation for GPUs” 2023

На какой из трех библиотек быстрее реализация алгоритма Борувки?



Результаты

Борувка на всех тестовых графах отрабатывает быстрее на GraphBLAS, чем на остальных библиотеках. На втором месте по производительности реализация Spla на CPU.

Алгоритм Прима на spla с OpenCl Backend

- На самом маленьком графе из датасета (CA-Coauthors-Dblp) реализации с **OpenCl CPU** и **OpenCl GPU** работали более 2-х часов, поэтому для этих бэкендов сравнение на основном датасете не проводилось
- Для выяснения причин долгой работы:
  - Использовался профилировщик **perf**
  - Визуализация результатов профилирования - CLion
  - Запуски проводились на уменьшенном графе CA-Coauthors-Dblp
  - 50K вершин, 2.3M ребер

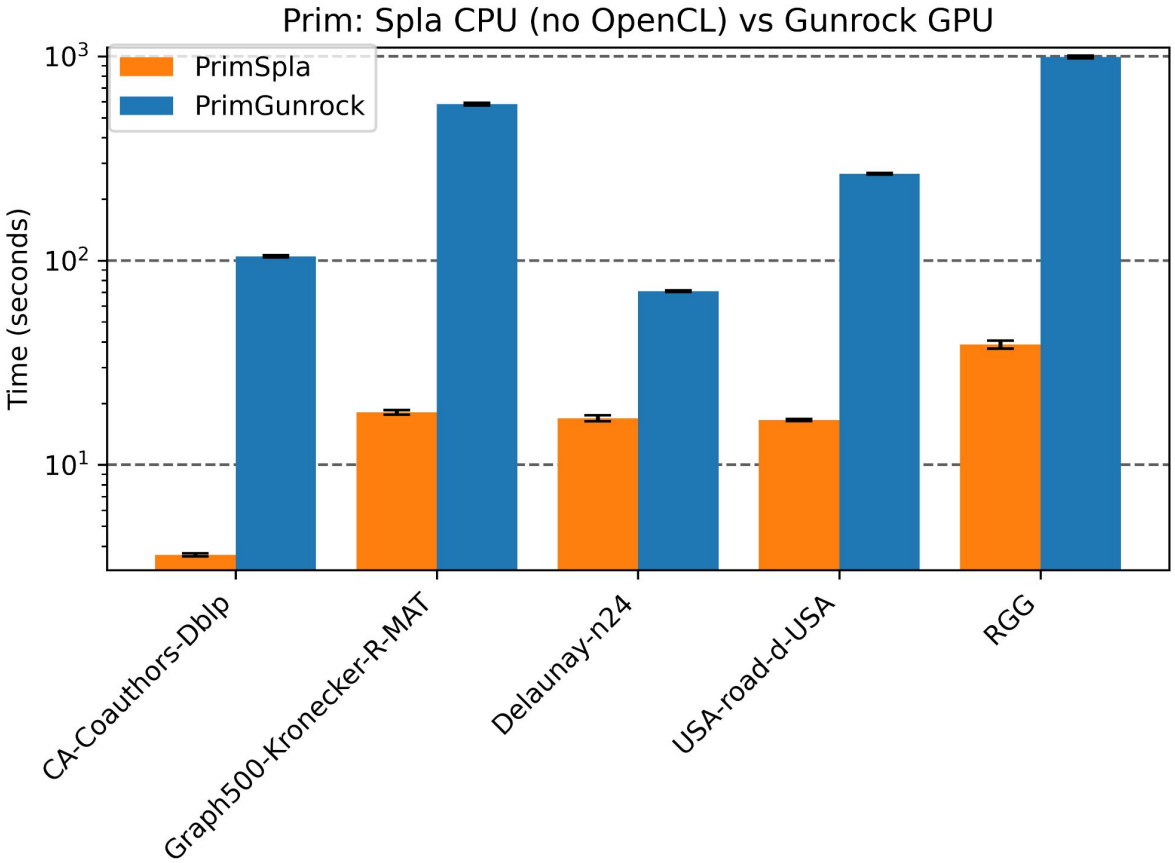
Причины долгой работы Prim spla с OpenCl

OpenCl CPU	
`Status TVector<T>::set_uint(row_id, value)`	**68%**
`Status exec_v_eadd_fdb(r, v, fdb, op, desc, task_hnd)`	**27%**
`Суммарно 2 метода`	**95%**

OpenCl GPU	
`Status TVector<T>::set_uint(row_id, value)`	**54%**
`Status exec_v_eadd_fdb(r, v, fdb, op, desc, task_hnd)`	**34%**
`Суммарно 2 метода`	**88%**

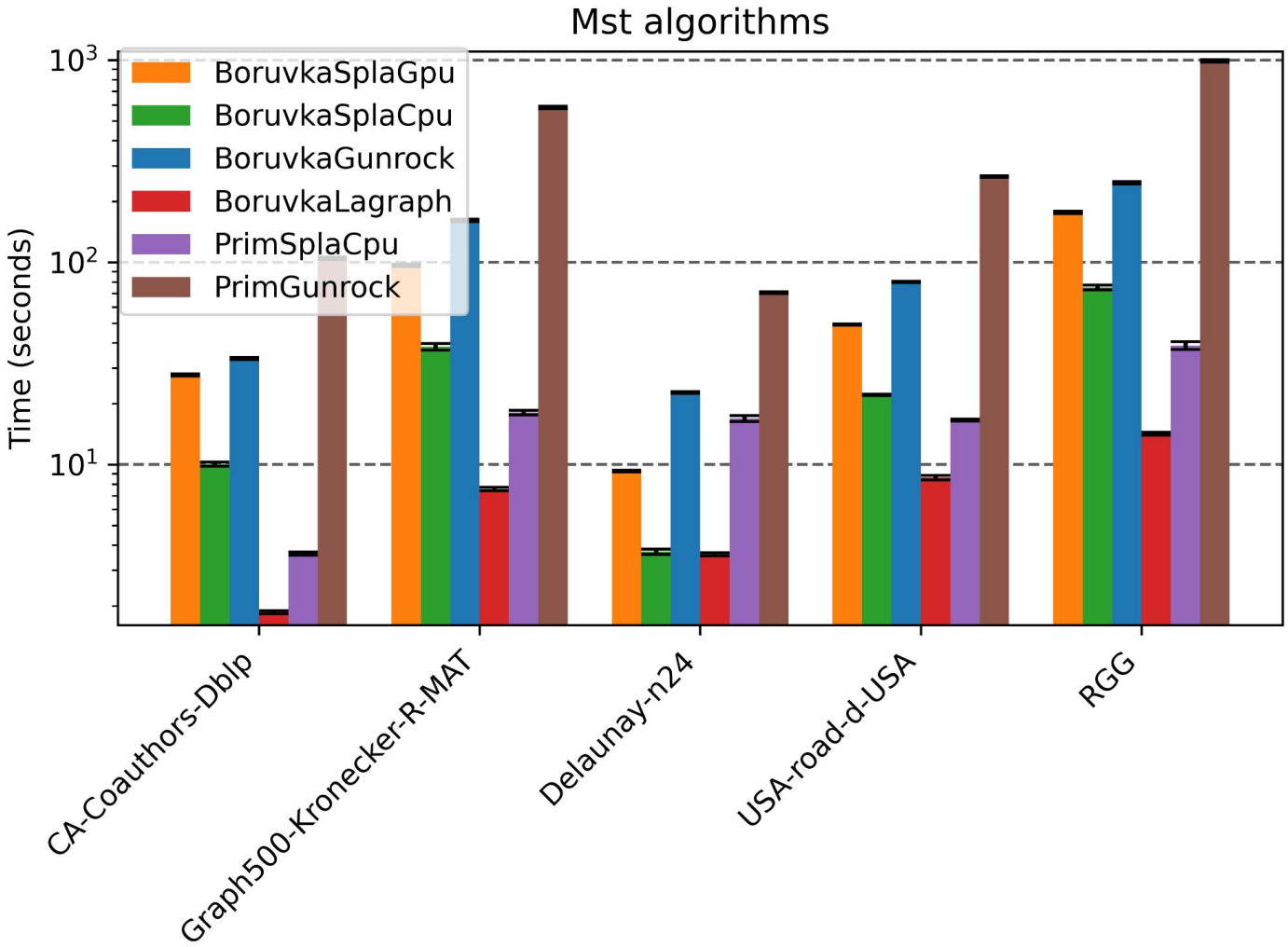
Для методов указан процент от общего времени выполнения

На какой из двух библиотек быстрее реализация алгоритма Прима?



Результаты

Прим во всех случаях быстрее на Spla CPU, чем на Gunrock GPU: разница в среднем почти на порядок в пользу Spla CPU


Mst algorithms
BoruvkaSplaGpu
BoruvkaSplaCpu
BoruvkaGunrock
BoruvkaLagraph
PrimSplaCpu
PrimGunrock
Time (seconds)
10^3
10^2
10^1
CA-Coauthors-Dblp
Graph500-Kronecker-R-MAT
Delaunay-n24
USA-road-d-USA
RGG

Эксперимент 2

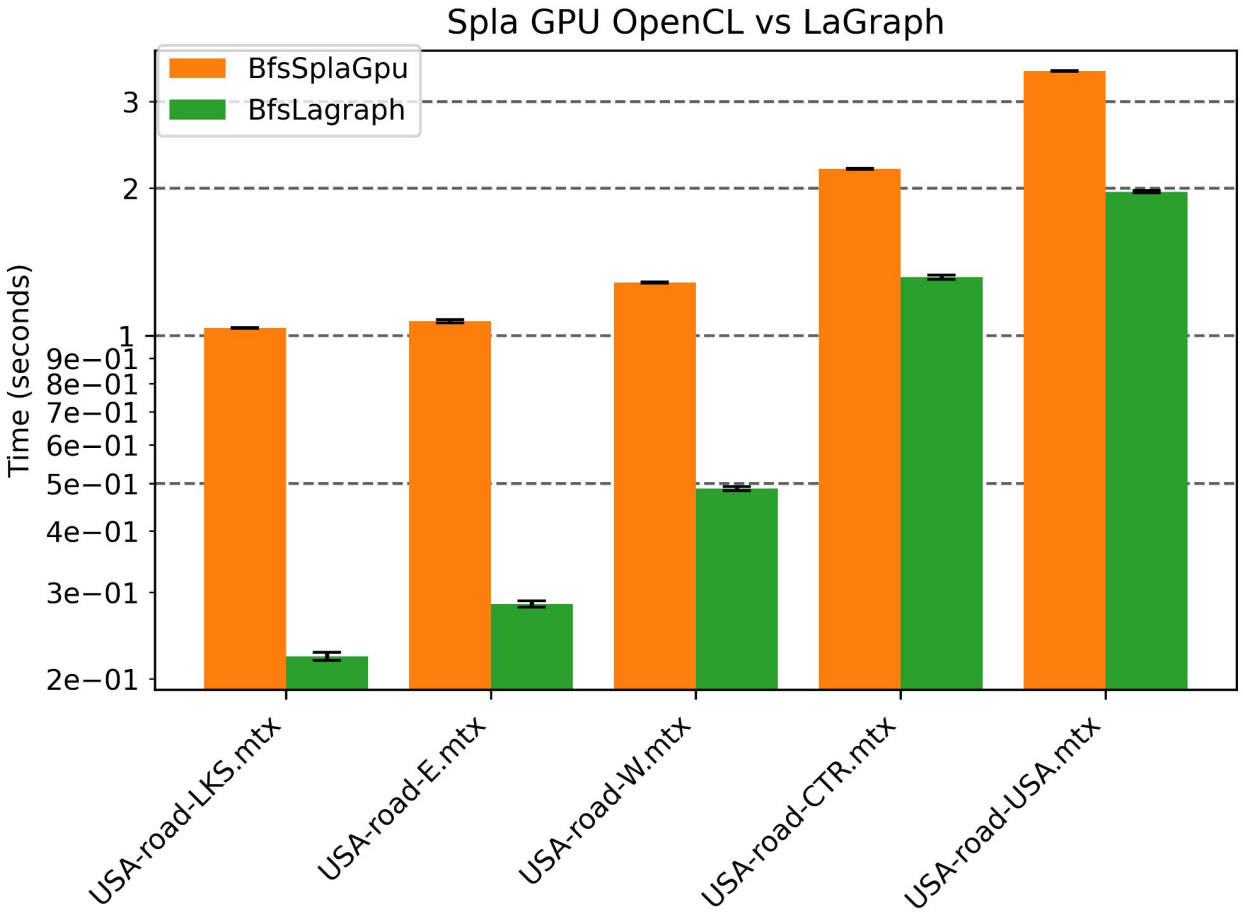
На какой из двух библиотек (**spla, LaGraph**) быстрее реализация алгоритма Single-source Parent BFS?

Граф	**Количество вершин, $\times 10^6$**	**Количество ребер, $\times 10^6$**
USA-road-LKS	2.7	6.8
USA-road-E	3.6	8.8
USA-road-W	6.3	15.2
USA-road-CTR	14.1	34.3
USA-road-USA	24	58.3

An effective GPU implementation of breadth-first search 2010
Датасет

Характеристики машины 2

- ОС Ubuntu 22.04.5 LTS
- Процессор 13th Gen Intel Core i7-13700H @ 2.4GHz
  - L1d – 544KiB; L1i – 704KiB; L2 – 11.5MiB; L3 – 24MiB
- 32 GB RAM
- 14 физических ядер, 20 логических
- Гипертрейдинг включен
- Видеокарта Mesa Intel® Graphics (RPL-P)
- OpenCL 3.0
  - Intel(R) OpenCL Graphics 25.13.33276.16

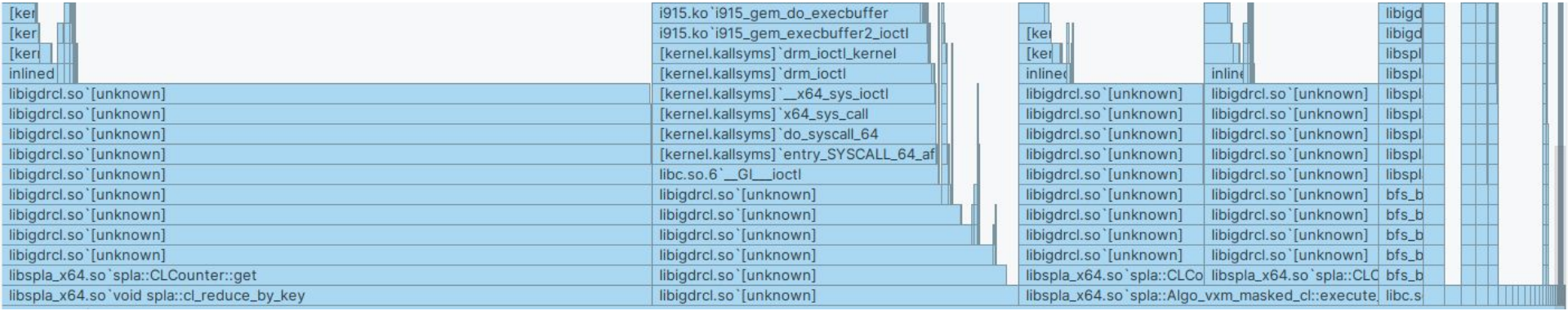


Глубина дерева обхода Bfs

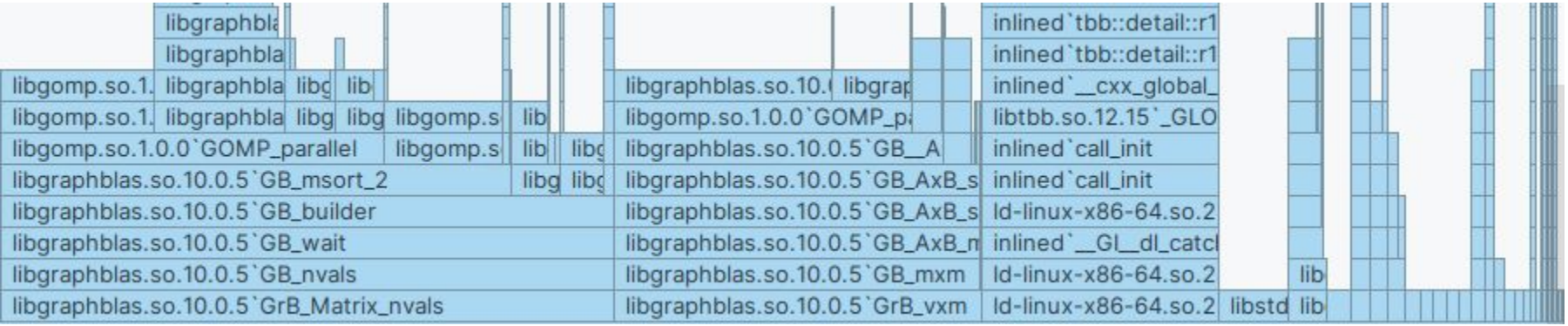
road-LKS	road-E	road-W
3241	2879	3138

SS Parent Bfs на всех тестовых графах отрабатывает быстрее на Lagraph.

Сравнение результатов профилирования (граф LKS)



Spla



LaGraph