SPLA: GENERALIZED SPARSE LINEAR ALGEBRA LIBRARY WITH VENDOR-AGNOSTIC GPUs ACCELERATED COMPUTATIONS

Speaker: Egor Orachev
Supervisor: Semyon Grigorev

SPBU, Saint Petersburg
April 10, 2023

Background

- Graph model to structure the data
  natural way to model the domain
- Practical tasks for analysis
  social networks, bioinformatics, databases, code analysis
- High-performance analysis
- CPU and GPU utilization
- Linear-algebra based solutions

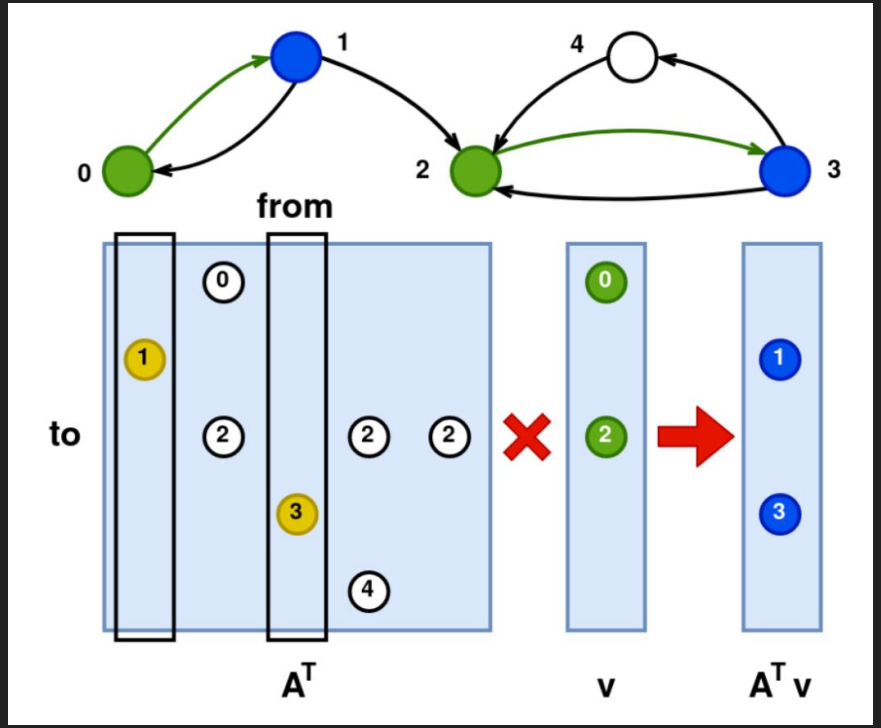
Related work

- **GraphBLAS:SuiteSparse** (GraphBLAS, CPU, GPU in progress)
- **GBTL** (GraphBLAS-like, GPU)
- **Gunrock** (Vertex-centric, edge-centric, GPU)
- **CuSha** (GAS, GPU)
- **nvGRAPH** (Linear-algebra based)
- **GraphBLAST** (GraphBLAS-like, GPU)

No complete GraphBLAS library with portable GPU support

GraphBLAS concepts

- Data containers
  matrices, vectors, scalars
- Algebraic structure
  monoid, semiring
- Operations
  mxm, mxv, ewiseadd, reduce
- Programming constructs
  mask, descriptor
- Algorithms
  bfs, sssp, pr, tc, cc



GraphBLAS shortcomings and limitations

According to a talk given by John R. Gilbert

- *Lack of interoperability* → How to integrate and extend
- *Little introspection* → How to inspect and interact
- *Implicit zeros* → Semantic abuse
- *Masking* → Semantic abuse
- *Templates usage* → Inflexible and too complex
- *GPU support* → No-support, partial Nvidia GPU support
- *…*

Proposed solution

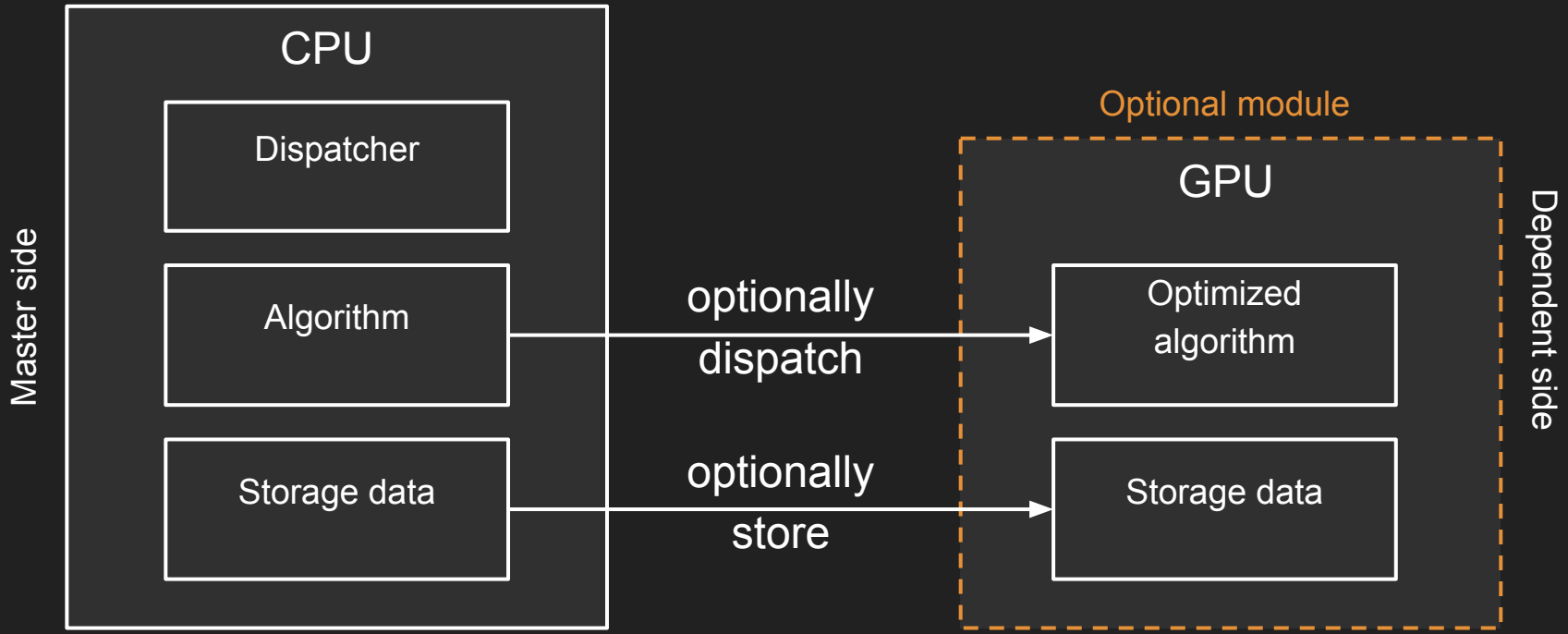
Spla: Generalized sparse linear algebra library with vendor-agnostic GPUs accelerated computations

- Linear-algebra based model
- GraphBLAS-inspired interface
- OpenCL for GPUs computations
- Portability across different GPU vendors
- Support for multiple API, such as C++, C, and Python

Design principles

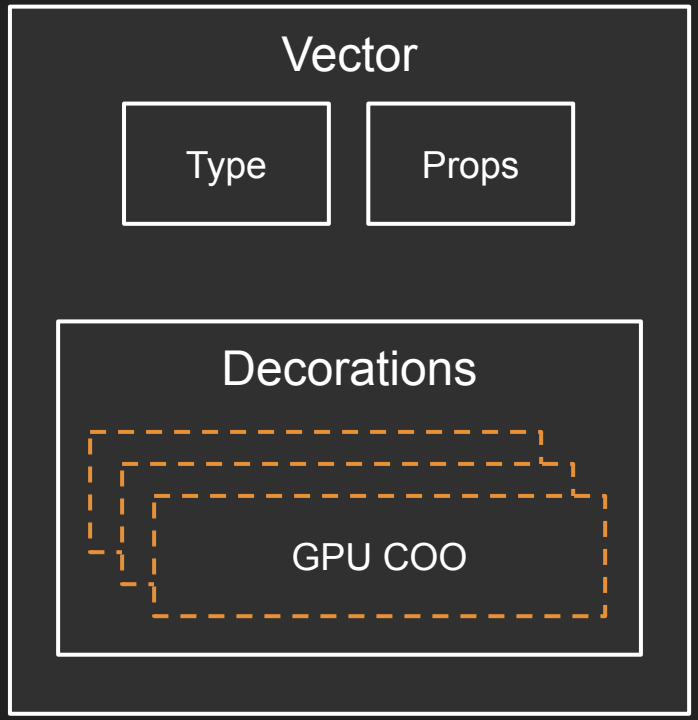
- *Optional acceleration* → Flexibility in implementation and exec
- *User-defined functions* → For both CPU and GPU
- *Predefined scalar types* → Native 1:1 mapping to GPU types
- *Hybrid storage format* → Format for a particular task
- *Exportable interface* → Bindings to other languages
- *Introspection* → First-class objects

Architecture overview

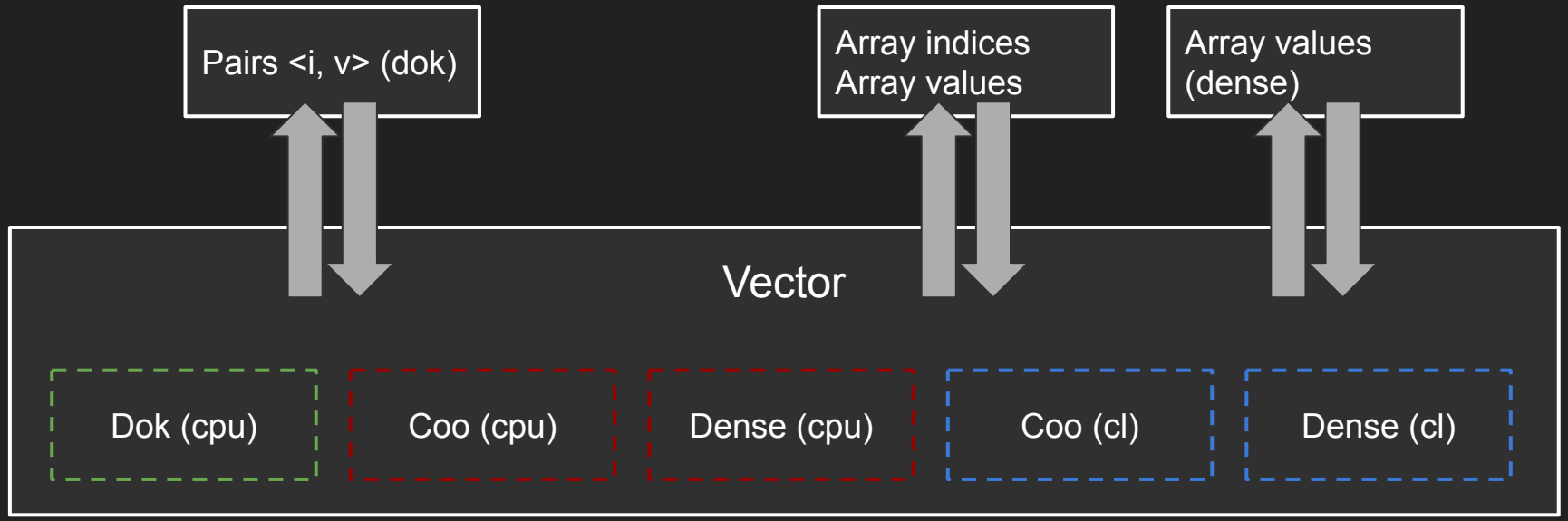


Storage overview

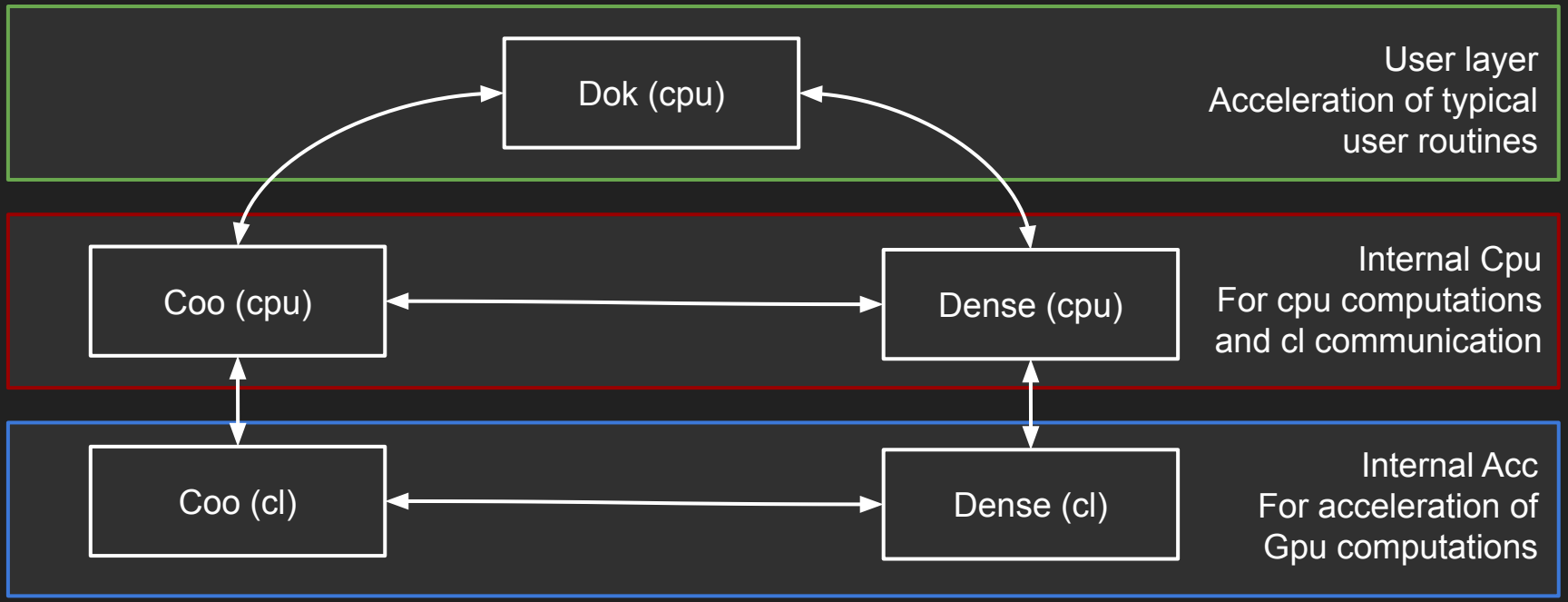
- Primitives
  - *matrix*, *vector*, *array*, *scalar*
- Decorations
  - Select best format for an operation
  - Store many formats
  - Decorations to add new formats
- Primary storage in RAM
  - GPU copy data on-demand
  - GPU stash data on-demand
  - GPU keep required data



Storage layout



Storage transformation

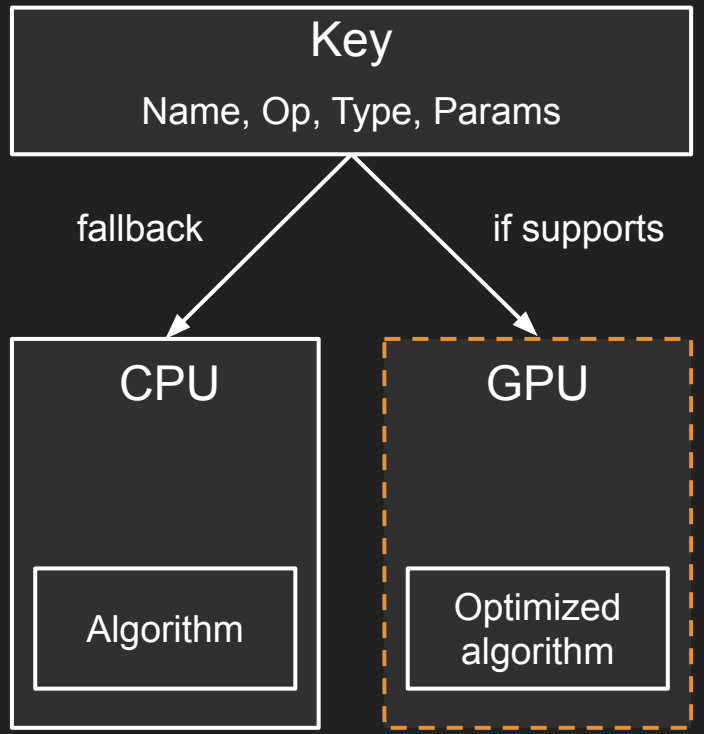


Data exchange

- Typed flat dense sequence of scalar elements
  `int`, `uint`, `float`, `complex`, etc.
- Agnostic for CPU and GPU data
- Hold data or hold view to data
- Pass as input to setup container
- Get as an output from container
- Get raw pointer to buffer
  `void* ptr`, `cl::Buffer* ptr`

Operations implementation

- Divided operation invocation and implementation (`command pattern`)
- Operation key is a `std::string`
- Key: `name, type, options`
- Operations stored in registry
- Acceleration backend may implement operation
- Redirects execution to accelerator if possible



Differences with GraphBLAS standard

Proposed solution is GraphBLAS-inspired, but not a strict implementation..

- *Fixed size POD-data* → Treated as sequence of bytes
- *Explicit zeros* → Pass `identity` where required
- *Mask selector* → `Predicate` to filter result
- *Fill value* → `filler to transform storage`
- *Storage hints* → Additional control over storage

OpenCL setup

- Khronos OpenCL C API header files bundled
  - No external installation of OpenCL SDK required
- Khronos OpenCL C++ API bindings bundled
  - High-level RAII safe API
- Khronos OpenCL ICD loader bundled
  - Build as `STATIC` library
  - Required sources modification to enable `-fPIC`
  - Enables arbitrary runtime loading on target machine
- On package import `ICD loader` finds OpenCL runtime
  - Possible to load at Runtime Intel, Nvidia, AMD implementations
- Only `spla library` shared object shipped within package
  - Self-contained PyPI package

OpenCL user functions and kernels

- User defined functions
  - Effectively are strings
  - Operations implemented in .cl files with generalization
  - Simple text preprocessing options required variation
- Kernel compilation
  - On demand
  - Runtime kernel cache
  - Cache key includes parameters, defines, user functions
  - Key composition in O(1) with inline allocators
  - Robin hood hashing based hashmap for cache

OpenCL memory management

- General data management
  - Use standard `cl::Buffer`
  - Strict memory flags: *host no access*
- Temporary small allocations
  - High overhead on some OpenCL runtimes
  - Use custom linear/stack allocator
  - Scope of a single function or operation
  - Subbufer mechanism for allocation
  - Request less than 1 MiB size

Operations implementation

- Masked sparse vector sparse matrix product (*vxm*)
  - k-way merge
  - custom gather
  - radix sort
  - key-value reduction based on prefix scan
- Masked sparse matrix vector product (*mxv*)
  - Scalar and vector version
  - No precise load balance
- Masked sparse matrix sparse matrix product (*mxmT*)
  - Single pass
  - Memory allocation using mask structure
  - Row-major matrix column-major matrix product

Algorithms implementation

- BFS
  - Masking to filter already discovered vertices
  - Change of direction (push-pull)
  - Switch from sparse front to dense and vice versa
  - Early exit in pull phase
- SSSP
  - Change of direction and format switch
  - Filter unproductive vertices which do not relax distances
- PR
  - Heavy use of *mxv*
  - Custom element-wise function for error estimation
- TC
  - Use *mxm* operation
  - Second argument is not actually transposed

Evaluation

Research questions

- **RQ1**. *What is the performance of the proposed solution relative to existing tools for GPU analysis?*
- **RQ2**. *What is the performance of the proposed solution on various devices vendors and OpenCL runtimes?*
- **RQ3**. *What is the performance of the proposed solution on integrated GPU compared to existing CPU tool for analysis?*

Tools:
Gunrock, GraphBLAST, LaGraph, Spla (proposed)

Algorithms:
BFS, SSSP, PR, TC

Setup

For RQ1

- PC1 with Ubuntu 20.04
- 3.40Hz Intel Core i7-6700 4-core CPU
- DDR4 64Gb RAM
- Intel HD Graphics 530 integrated GPU
- Nvidia GeForce GTX 1070 dedicated GPU with 8Gb on-board VRAM

For RQ2

- PC2 with Ubuntu 22.04
- 4.70Hz AMD Ryzen 9 7900x 12-core CPU
- DDR4 128 GB RAM
- AMD GFX1036 integrated GPU
- either Intel Arc A770 flux dedicated GPU with 8GB on-board VRAM
- or AMD Radeon Vega Frontier Edition dedicated GPU with 16GB on-board VRAM

For RQ3

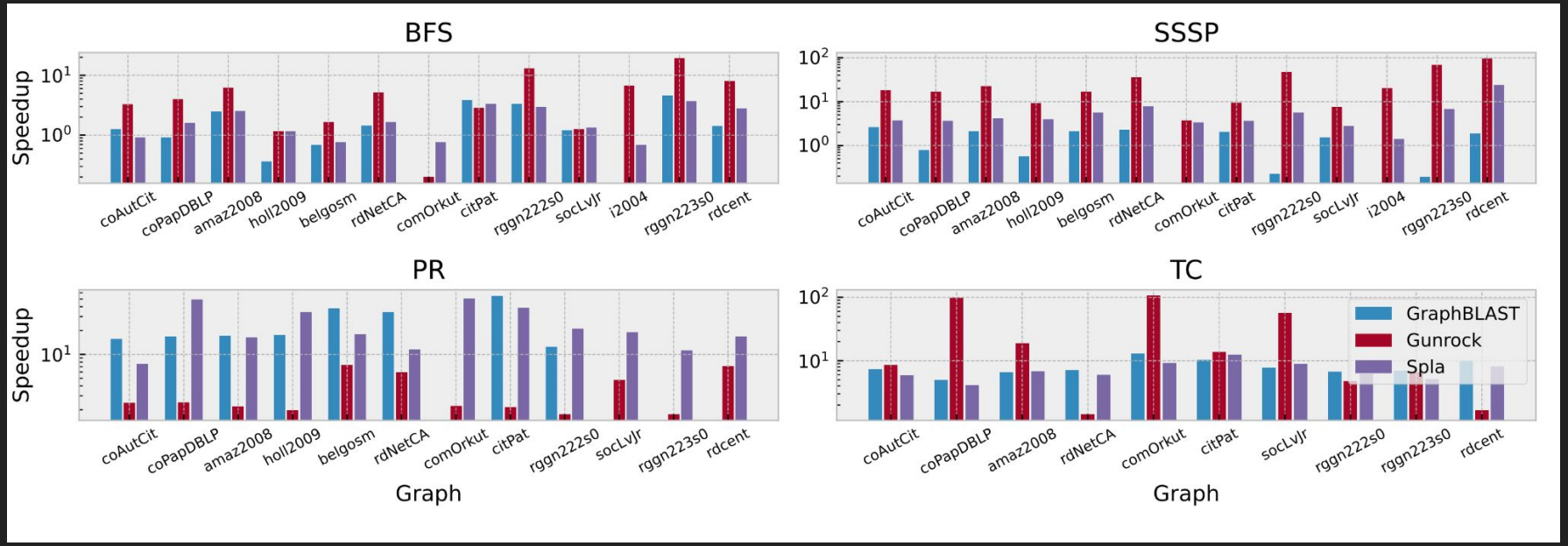
- First PC with Intel CPU and integrated GPU and
- Second PC with AMD CPU and integrated GPU are used

Dataset

- 13 graphs
- Suite Sparse matrix collection
- Two categories
- Made undirected
- Self-loops removed
- Uniform weights

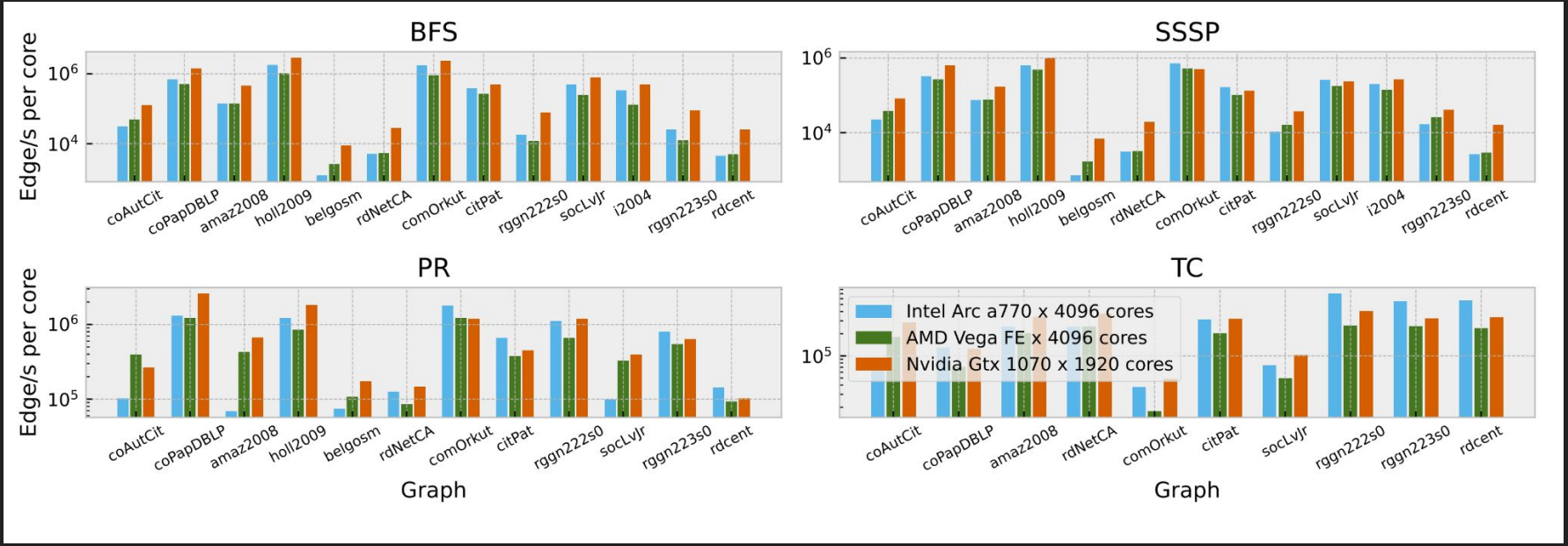
Graph	Vertices	Edges	Out Degree		
			Avg	Sd	Max
coAuthorsCit	227.3K	1.6M	7.2	10.6	1.4K
coPapersDBLP	540.5K	30.5M	56.4	66.2	3.3K
amazon2008	735.3K	7.0M	9.6	7.6	1.1K
hollywood2009	1.1M	112.8M	98.9	271.9	11.5K
comOrkut	3.1M	234.4M	76.3	154.8	33.3K
citPatents	3.8M	33.0M	8.8	10.5	793.0
socLiveJournal	4.8M	85.7M	17.7	52.0	20.3K
indochina2004	7.4M	302.0M	40.7	329.6	256.4K
belgiumosm	1.4M	3.1M	2.2	0.5	10.0
roadNetCA	2.0M	5.5M	2.8	1.0	12.0
rggn222s0	4.2M	60.7M	14.5	3.8	36.0
rggn223s0	8.4M	127.0M	15.1	3.9	40.0
roadcentral	14.1M	33.9M	2.4	0.9	8.0

Results: RQ1



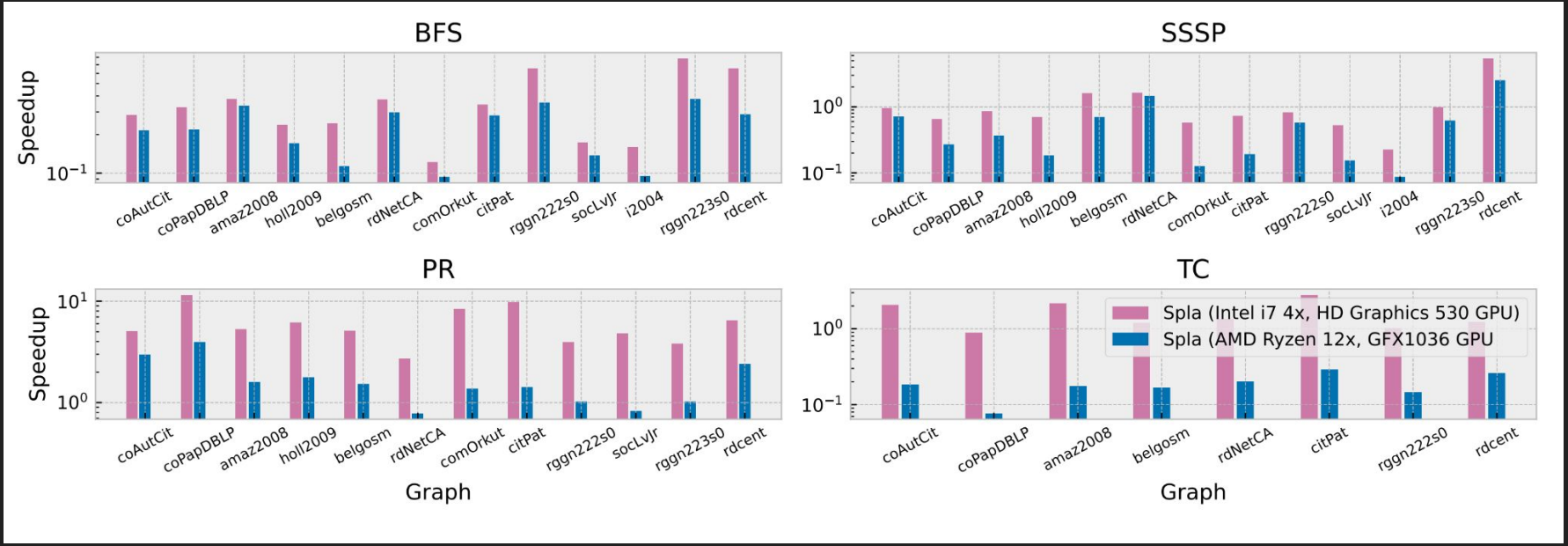
Performance of Spla library and GPU tools on the same device compared to LaGraph. Logarithmic scale is used

Results: RQ2



Performance of Spla library on different devices relative to the number of compute cores. Logarithmic scale is used

Results: RQ3



Performance of Spla library on integrated GPU compared to LaGraph on the same chip. Logarithmic scale is used

Further research

- Performance tuning
  - Vendor specific optimization
  - HIgh-performance SpGEMM implementation
- New operations
  - Reductions
  - Filtering
  - Union, intersection
- Graph streaming
  - Fit twitter 3 billions edges into 8 GiBs of memory
- Multi-GPU support
  - Scale up to 4 GPUs in a single board

Package

Download package and test now on Windows, Linux and MacOS

```
$ pip install pyspla
```