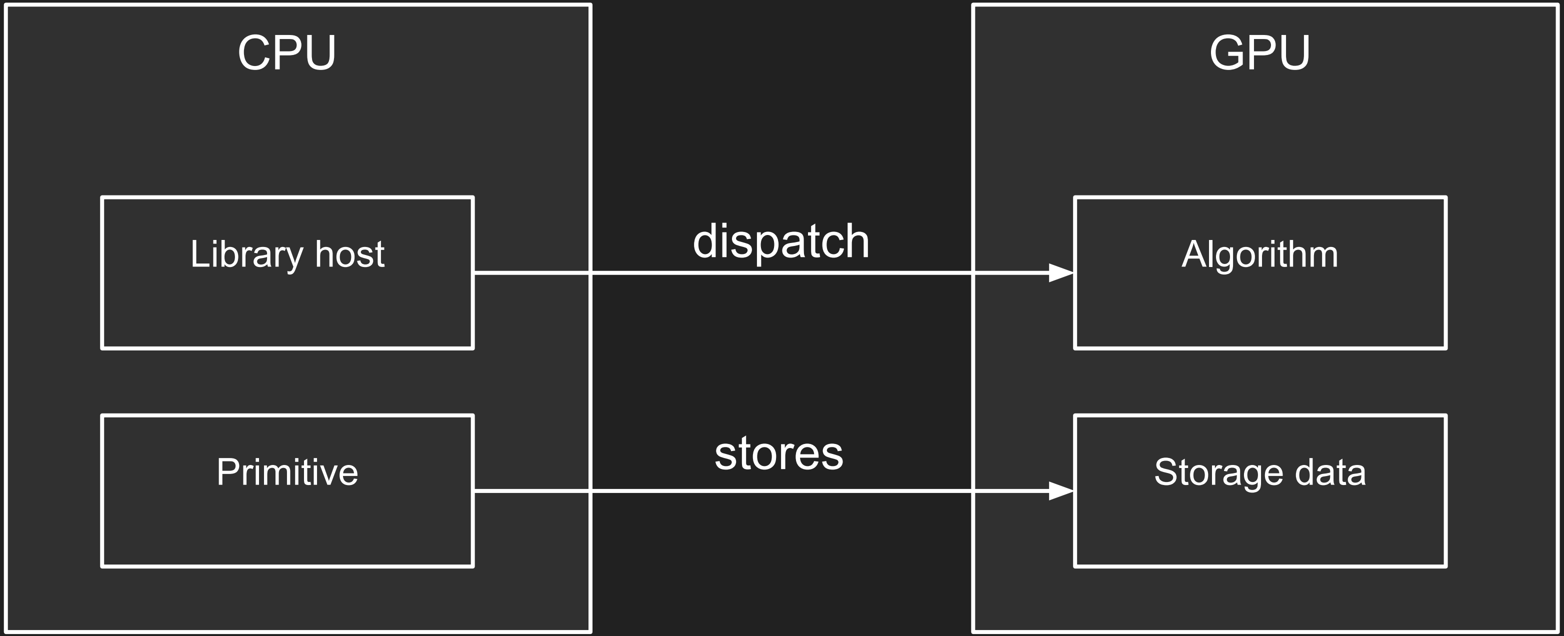
GENERALIZED SPARSE LINEAR ALGEBRA FRAMEWORK WITH GPUs ACCELERATION

New project vision & development strategy

Previous concept



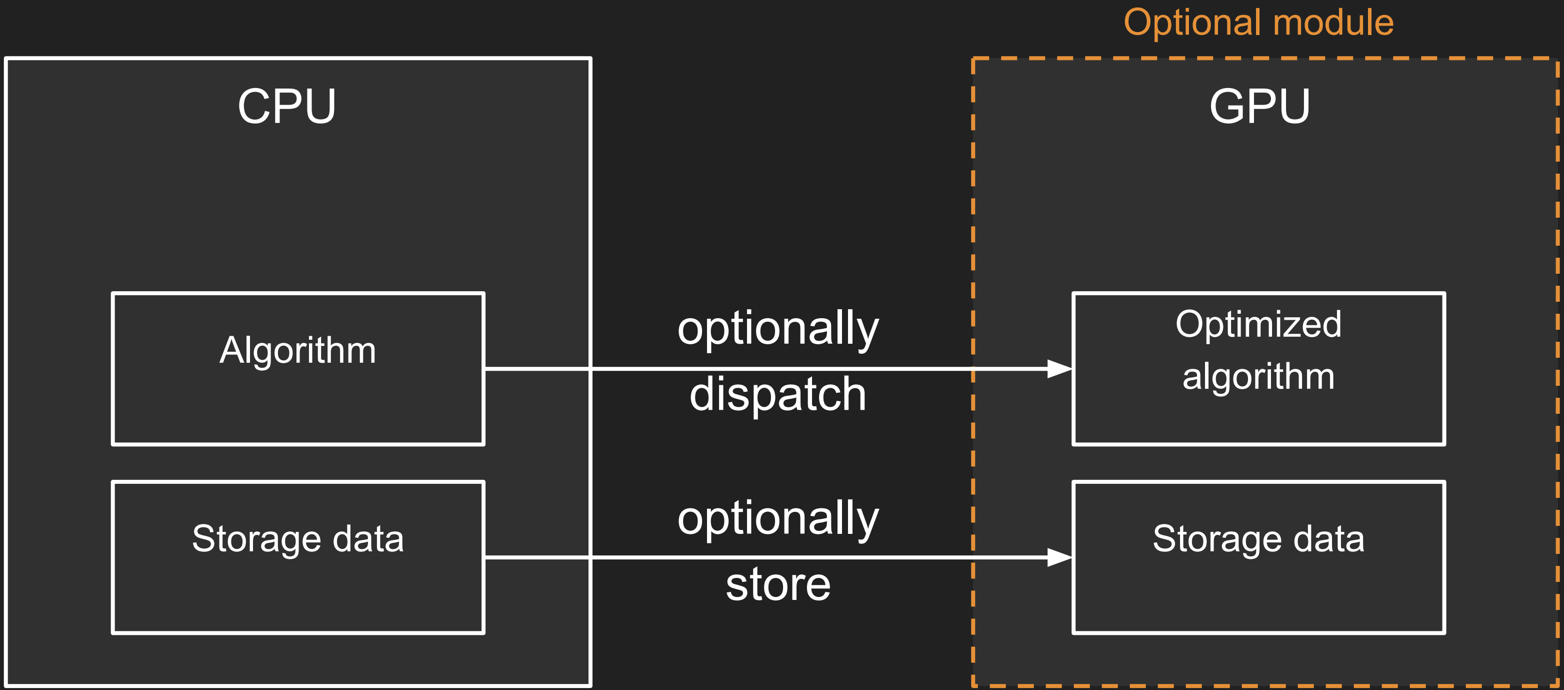
Previous design ideas

- Automated blocked storage for data → Limits implementation flexibility
- Automated work scheduling → Not enough info to properly schedule
- DAG for expression declaration → Too high-level
- OpenCL for computations → GPU cannot be used for all operations
- User-defined types → GPU has strict data limitations
- User-defined functions → Limits ad-hoc optimizations

Things to consider

- RAM much larger than VRAM (64 GB vs 8 GB)
- RAM and VRAM data may be duplicated
- GPU may run out of memory
- GPU has own optimized data structures
- GPU may require different data layouts
- GPU outperforms CPU in tasks with high compute intensity
- GPU not always presented in the system
- GPU has limited number of hardware queues
- GPU requires precise memory flags for allocations (especially on AMD)

New concept



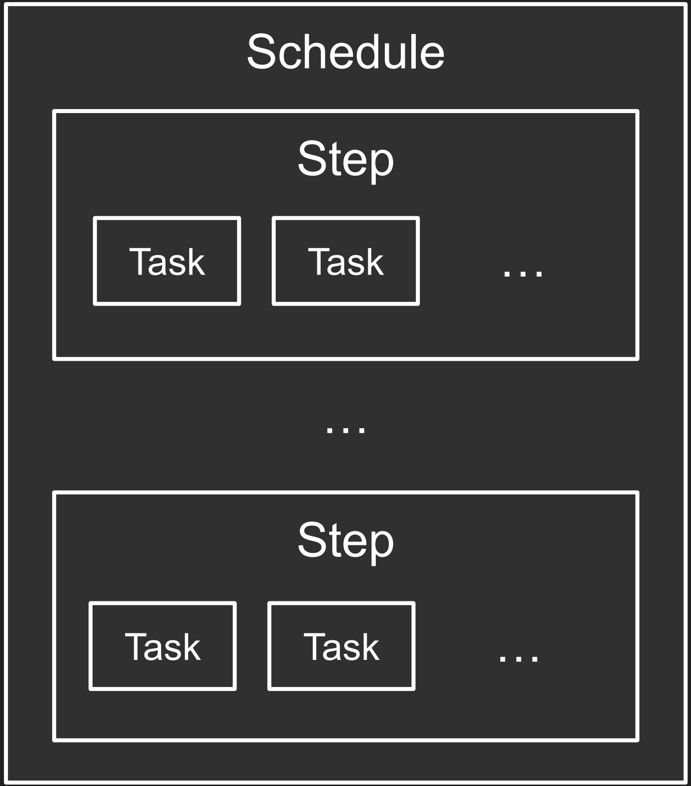
Optional module
CPU
GPU
Algorithm
optionally dispatch
Optimized algorithm
Storage data
optionally store
Storage data

New design ideas

- Multiple storage formats → Best format for given task
- Manual work scheduling → Computation defined by op impl
- Schedule for work submission → Submit user-defined order of execution
- Optional GPU acceleration → Better flexibility and more features
- Built-in types → GPU friendly layout
- Built-in functions → GPU kernels ad-hoc optimizations

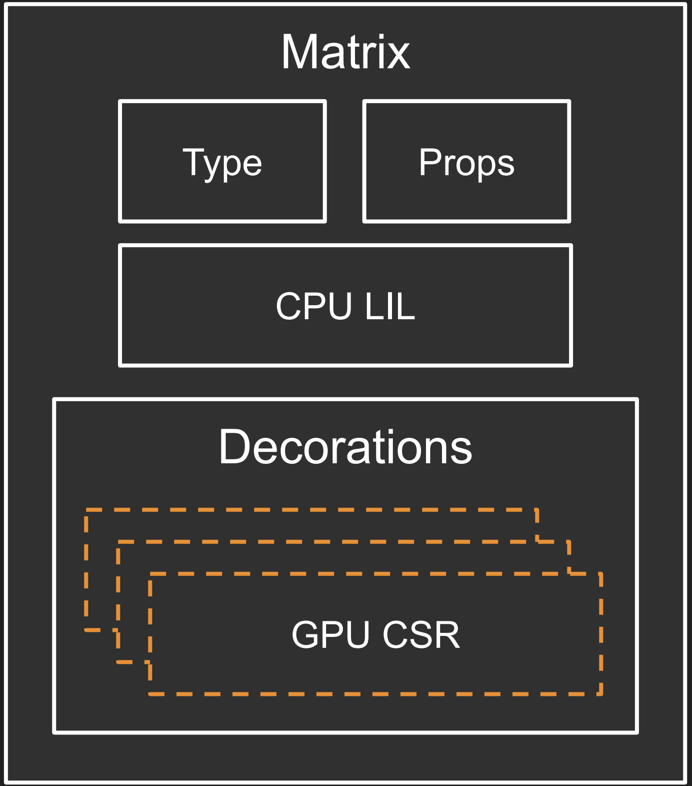
Schedule

- Schedule defines list of steps
- On each step has one or more tasks
- Task in step ordered
- Order defines priority
- Steps serialized
- Tasks inside a step are parallelized



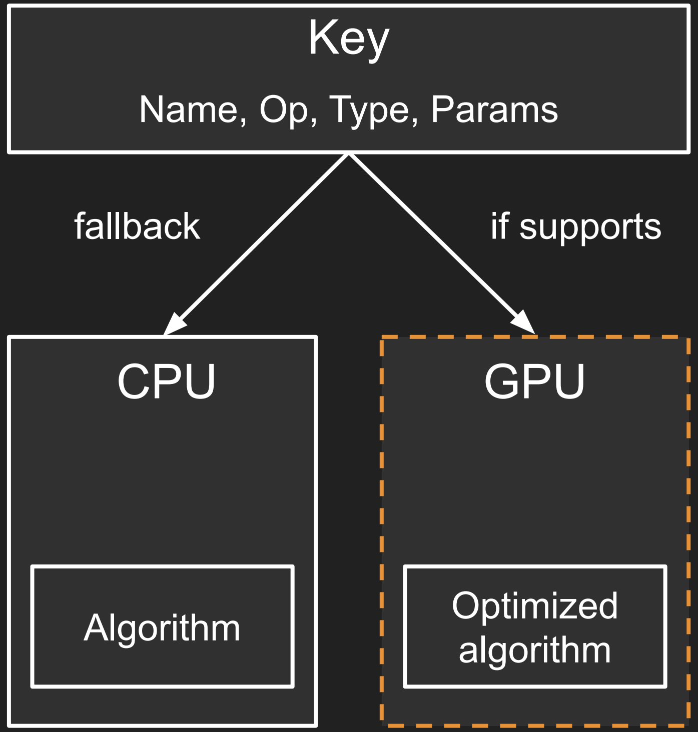
Storage scheme

- Primary storage in RAM
- Select best format for an op
- Conversion on CPU
- Store many formats
- Decorations to add new formats
- GPU copy data on-demand
- GPU stash data on-demand
- GPU keep required data



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



Key

Name, Op, Type, Params

fallback

if supports

CPU

Algorithm

GPU

Optimized algorithm

Technical details

- Boost compute → Khronos `OpenCL API C++ bindings`
- Boost `kernel cache` → Manual kernel compilation and storage
- Boost meta-programming → Kernel is effectively a `std::string`
- Boost `vector type` → `cl::Buffer` with custom flags
- Taskflow → `std::thread` and manual control

OpenCL

- 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

Package

Download package and test now on Windows, Linux and MacOS

```
$ pip install pyspla
```