Neurokernel Core Architecture

Current Design, Limitations, and Future Aims

Lev Givon

Bionet Group
Department of Electrical Engineering
Columbia University

March 17, 2016

Outline

1. Interfacing LPUs
2. Communication
3. Neurodriver
4. Future Aims

Port Identifiers and Selectors

- Each port must be assigned a unique *identifier*.
- Mandatory path-like format: `/level0/level1/...`
- *Selectors* represent multiple identifiers: `/level0[0:5]`
- Syntax goodies: ranges, lists, concatenation, outer products, elementwise products.
- `Selector` class adds support for set operations.

`nk.plsel`

- `Selector` - validates/expands/caches selectors
- `SelectorParser` - selector grammar
- `SelectorMethods` - selector manipulation routines

- Selectors expanded by parser into lists of tuples. Expensive!
- Adding ports to existing interface also expensive - avoid by creating with all required ports.

Selector Syntax Features

Identifier/Selector	Comments
`/med/L1[0]`	selects a single port
`/med/L1/0`	equivalent to `/med/L1[0]`
`/med+/L1[0]`	equivalent to `/med/L1[0]`
`/med/[L1,L2][0]`	selects two ports
`/med/L1[0,1]`	another example of two ports
`/med/L1[0],/med/L1[1]`	equivalent to `/med/L1[0,1]`
`/med/L1[0:10]`	selects ten ports
`/med/L1/*`	selects all ports starting with `/med/L1`
`(/med/L1,/med/L2)+[0]`	equivalent to `/med/[L1,L2][0]`
`/med/[L1,L2].+[0:2]`	equivalent to `/med/L1[0],/med/L2[1]`

LPU Interface Design

- *Interface* = port identifiers + port attributes + port ↔ transmitted data array mapping.
- **`Interface`** class can encapsulate multiple interfaces.
- Required attributes: interface identifier, I/O direction (input or output), type (spiking or graded potential).
- Stored in Pandas **`DataFrame`**: attribs → data, (expanded) ports → index.
- Interface compatibility: ports in both interfaces must have same type and inverse I/O attributes.

`nk.pattern`

- **`Interface`** - interface class

- Compatibility checking is expensive (requires equiv. of inner join)!

Port Maps

- Goal: use selectors to access array of data transmitted to/from ports.
- Solution: *map* identifiers to array indices: **`/label0[a,b,c]`** → **`[0,1,2]`**
- Stored in Pandas **`DataFrame`**: array indices → data, ports → frame index.
- Port data array must have atomic dtype (**`int32`**, **`float64`**, etc.).

`nk.pm`, `nk.pm_gpu`

- **`PortMapper`** - maps ports to host memory array
- **`GPUPortMapper`** - maps ports to GPU memory array

- Selector-based access is expensive! Solution: use indices rather than selectors during model execution.

Inter-LPU Connectivity Patterns

- Each **`Pattern`** connects two LPUs - N patterns required to connect one LPU to N other LPUs.
- Each pattern contains two interfaces (stored in one **`Interface`** instance).
- Stored in Pandas **`DataFrame`**: index contains connected source/destination ports (expanded).
- If source/dest pair $\notin$ pattern, they are not connected.
- Selectors may be used to set/select connections, but as before..

`nk.pattern`

- **`Pattern`** - pattern class

- .. selector-based access/adding new connections are expensive!
- Use special classmethods to create patterns from selectors.

More Areas for Improvement

- Selector expansion is done to enable storage in Pandas **`MultiIndex`**.
- The good: can facilitate selection of groups of ports using hierarchical indexing.
- The bad: handling of multilevel identifiers is complicated and partially broken (see issue #52), expensive selection operations.
- Possibility: store identifiers as Pandas **`Index`** of string labels?
- Possibility: ensure that parsing/expansion never occurs during model execution?

Defining LPUs

- LPU models are implemented as Python classes with an `Interface` attribute and `run_step()` method.
- `run_step()` is invoked at each execution step.
- Incoming/outgoing port data must be read/updated from/to the interface port mapper within `run_step()`.
- After invocation of `run_step()`, Neurokernel *synchronizes* the LPU's ports with those of other connected LPUs.

`nk.core`, `nk.core_gpu`

- `Module` - parent LPU class

- GPU and non-GPU parent classes are separate - could be conflated.

Multi-LPU Emulation Setup

- LPU classes + params and patterns must be added to an emulation by a **`Manager`** class instance.
- NK uses MPI-2 *dynamic process creation* to automatically spawn sufficient # of processes to run emulation.
- Processes started/stopped by control messages via interprocess communicator.
- Run loop timing info also collected by manager via messages.

`nk.core`, `nk.core_gpu`

- **`Manager`** - emulation manager

More Areas for Improvement

- Process spawning requires active MPI env.
- Solution: relaunch main script via **`mpi_relaunch`** module and then spawn.
- Downside: can’t develop/debug interactively.
- Possibility: decouple manager process from spawning process, but let former steer latter (cf. IPython Parallel).
- Objects required by LPU class need to be present in spawned proc namespace.
- Solution: recursively find/serialize params/globals accessed by class, transmit to spawned proc.
- Downside: some objects can’t be serialized, transmission of large/many params/globals is time-consuming.

Inter-LPU Communication

- Communication between LPUs automatically performed by NK after each execution step:
  1. Source: output port data → transmission buffer.
  2. Source: buffer → MPI.
  3. Destination: MPI → buffer.
  4. Destination: buffer → input port data.
- Data copied to/from contiguous buffers to enable CUDA-enabled MPI to use GPUDirect.
- Transmissions are launched asynchronously, but each LPU waits until initiated send operations complete before next exec step.

- Noncontiguous copy to/from transmission buffers is inefficient.
- Possibility: use multiple sends per LPU?
- Possibility: fully asynchronous transmission (with deadlock breaking algorithm)?

Neurodriver I

- Neurodriver: configurable LPU implementation.
- Supports circuits comprising several point neuron models and associated synapse models.
- New models may be defined by subclassing **`BaseNeuron`**, **`BaseSynapse`**.
- Class may be instantiated with model parameters:

```
{'ModelName': {'Param0': [val0, val1, ..],
               'Param1': [val4, val5, ..]}, ...}}
```

- Can specify circuit as graph (GEXF, NetworkX).
- # of neurons and synapses only constrained by memory.
- All spiking model port data internally stored consecutively in one array, as is graded potential model port data.

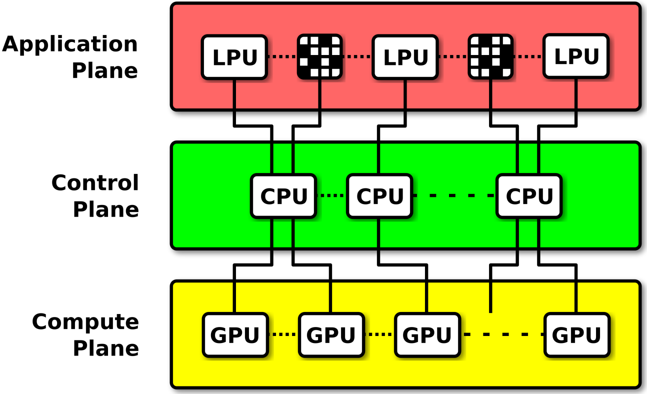
Neurodriver II

`nk.LPU.LPU`, `nk.LPU.neurons`, `nk.LPU.synapses`

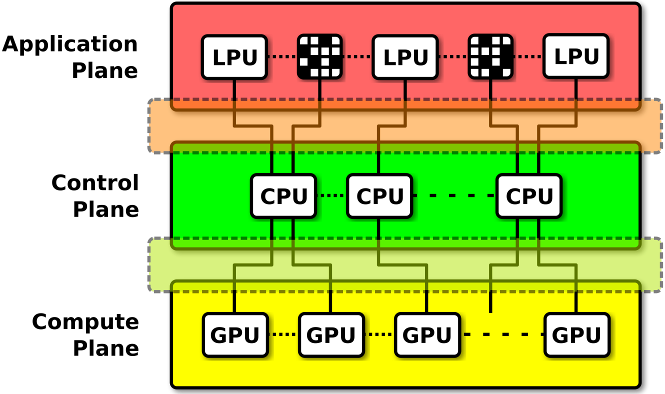
- **`BaseNeuron`**, **`BaseSynapse`**, etc. - neuron/synapse model implementations.
- **`LPU`** - subclass of **`Module`**

- Legacy design: input ports must be explicitly specified, output ports specified implicitly by setting **`public`** model param.
- Legacy design: synapses must be explicitly marked as belonging to different classes (spiking → graded potential, graded potential → spiking, etc.)
- Extra noncontiguous GPU memory copies to/from port mappers.
- Hard to add models with complex input/output relationships.
- Possibility: make neurons/synapses subclasses of a more general *component* class (#2).

Realizing Vertical APIs



Realizing Vertical APIs



Improving GPU Resource Utilization

- Each LPU knows about one GPU (but multiple LPUs can use the same GPU).
- Direct access to GPUs by LPU implementation precludes efficient resource use.

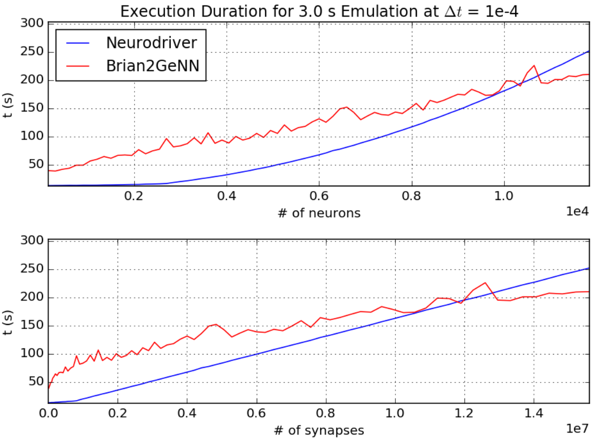
- Restrict direct access to GPUs to compute plane.
- Add mechanism for mapping circuit components to resources (simple prototype using METIS already used in benchmarks).
- Devise resource alloc policies that optimize over available GPUs, bus bandwidth, model component cost, etc.
- Utilize structural data in NeuroArch for resource allocation.
- Possibility: parameterize generation of CUDA code based model resources reqs.

Improving Model Component Support

- Neurodriver plugin system sufficient for point neurons/simple synapses, but hard to extend to more complex models.
- Plugin system complicates efficient resource utilization.

- Generalize to support models with more complex I/O.
- Only the application plane must know what an LPU is.
- Add new primitives/components to compute plane - only exposed through vertical API.
- Convert Neurodriver to compute plane engine.
- Possibility: enforce declarative definition of LPUs?

Improving Emulation Performance



- **https://github.com/neurokernel/neurodriver-benchmark**
- Should we base compute plane on something like Brian2GeNN?

Hacking Notes

- Control logging with **`nk.tools.logging.setup_logger`**.
- Logs may be written to screen, file, or both.
- Logging can be turned off to prevent I/O slowdown.
- Debugging spawned Python processes - see ripdb.
- Tip: when printing **`var`** contents to log,
  1. set **`multiline=True`** in **`setup_logger()`** params;
  2. make sure that **`repr(var)`** returns something sane!
- If **`debug=False`** in **`Module`** constructor, exceptions in **`run_step`** are *not* fatal!

- Logged messages can get lost/overwritten due to MPI-IO sync issues.
- Crude solution: pipe stdout to file.
- Possibility: send logs to aggregator process rather than disk to avoid I/O slow-down.