Neurokernel: An Open Source Platform for Emulating the Fruit Fly Brain

Neurokernel RFC #4 v1.0

Lev E. Givon and Aurel A. Lazar
Bionet Group
Department of Electrical Engineering
Columbia University

October 9, 2015

Abstract

We have developed an open software platform called Neurokernel for collaborative development of comprehensive models of the brain of the fruit fly *Drosophila melanogaster* and their execution and testing on multiple Graphics Processing Units (GPUs). Neurokernel provides a programming model that capitalizes upon the structural organization of the fly brain into a fixed number of functional modules to distinguish between these modules' local information processing capabilities and the connectivity patterns that link them. By defining mandatory communication interfaces that specify how data is transmitted between models of each of these modules regardless of their internal design, Neurokernel explicitly enables multiple researchers to collaboratively model the fly's entire brain by integration of their independently developed models of its constituent processing units. We demonstrate the power of Neurokernel's model integration by combining independently developed models of the retina and lamina neuropils in the fly's visual system and by demonstrating their neuroinformation processing capability. We also illustrate Neurokernel's ability to take advantage of direct GPU-to-GPU data transfers with benchmarks that demonstrate scaling of Neurokernel's communication performance both over the number of interface ports exposed by an emulation's constituent modules and the total number of modules comprised by an emulation.

This RFC obsoletes all versions of NK-RFC #1.

 DOI:10.5281/zenodo.31947

Contents

 DOI:10.5281/zenodo.31947

1 Introduction

Reverse engineering the information processing functions of the brain is an engineering grand challenge of immense interest that has the potential to drive important advances in computer architecture, artificial intelligence, and medicine. The human brain is an obvious and tantalizing target of this effort; however, its structural and architectural complexity place severe limitations upon the extent to which models built and executed with currently available computational technology can relate its biological structure to its information processing capabilities. Successful development of human brain models must therefore be preceded by an increased understanding of the structural/ architectural complexity of the more tractable brains of simpler organisms and how they implement specific information processing functions and govern behavior [25].

The nervous system of the fruit fly *Drosophila melanogaster* possesses a range of features that recommend it as a model organism of choice for relating brain structure to function. Despite the obvious differences in size and complexity between the mammalian and fly brains, researchers dating back to Cajal have observed common design principles in the structure of their sensory subsystems [56]. Many of the genes and proteins expressed in the mammalian brain are also conserved in the genome of *Drosophila* [1]. These features strongly suggest that valuable insight into the workings of the mammalian brain can be obtained by focusing on that of *Drosophila*.

Remarkably, the fruit fly is capable of a host of complex nonreactive behaviors that are governed by a brain containing only $\sim 10^5$ neurons and $\sim 10^7$ synapses organized into fewer than 50 distinct functional units, many of which are known to be directly involved in functions such as sensory processing, locomotion, and control [7]. The relationship between the fly's brain and its behaviors can be experimentally probed using a powerful toolkit of genetic techniques for manipulation of the fly's neural circuitry such as the GAL4 driver system [13, 55, 61, 66, 40], recent advances in experimental methods for precise recordings of the fly's neuronal responses to stimuli [27, 69, 28], techniques for analyzing the fly's behavioral responses to stimuli [5, 39, 8], and progress in reconstruction of the fly connectome, or neural connectivity map [9, 64]. These techniques have provided access to an immense amount of valuable structural and behavioral data that can be used to model how the fly brain's neural circuitry implements processing of sensory stimuli [16, 42, 7, 24, 43, 57].

Despite considerable progress in mapping the fly's connectome and elucidating the patterns of information flow in its brain, the complexity of the fly brain's structure and the still-incomplete state of knowledge regarding its neural circuitry pose challenges that go beyond satisfying the current computational resource requirements of fly brain models. These include (1) the need to explicitly target the information processing capabilities of functional units in the fly brain, (2) the need for fly brain model implementations to efficiently scale over additional hardware resources as they advance in complexity, and (3) the need for brain modeling to be approached as an explicitly open and collaborative process of iterative refinement by multiple parties similar to that successfully employed in the design of the Internet [3] and large open source projects such as the Python programming language [67].

To address these challenges, we have developed an open source platform called Neuroker-

 DOI:10.5281/zenodo.31947

nel for implementing connectome-based fly brain models and executing them upon multiple Graphics Processing Units (GPUs). In order to achieve scaling over multiple computational resources while providing the programmability required to model the constituent functional modules in the fly brain, the Neurokernel architecture provides features similar to that of an operating system kernel. In contrast to general-purpose neural simulators, the design of Neurokernel and brain models built upon it is driven by publicly available proposals called Requests for Comments (RFCs).

Neurokernel's design is predicated upon the organization of the fly brain into a fixed number of functional modules characterized by local neural circuitry. Neurokernel explicitly enforces a programming model for implementing models of these functional modules called Local Processing Units (LPUs) that separates between their internal design and the connectivity patterns that link their external communication interfaces (APIs) independently of the internal design of models designed by other researchers and of the connectivity patterns that link them. This modular architecture facilitates collaboration between researchers focusing on different functional modules in the fly brain by enabling models independently developed by different researchers to be integrated into a single whole brain model irrespective of their internal designs.

This paper is organized as follows. We review the anatomy of the fly brain that motivate Neurokernel's design in § 2.1 and describe its architecture and support for GPU resources and programmability in § 2.2. We present Neurokernel's programming model in § 2.3 and detail its API in § 2.4. To illustrate the use of Neurokernel's API, we use it to integrate independently developed models of the retina and lamina neuropils in the fly's visual system; this integration is described in § 3.1. To assess Neurokernel's ability to exploit technology for accelerated data transmission between multiple GPUs, we provide benchmarks of its module communication services in § 3.2. Finally, we compare Neurokernel to other computational projects directed at reverse engineering the function of neural circuits in § 4 and discuss the project's long-term goals in § 5.

2 Framework Design and Features

2.1 Modeling the Fruit Fly Brain

Analysis of the *Drosophila* connectome has revealed that its brain can be decomposed into fewer than 50 distinct neural circuits, most of which correspond to anatomically distinct regions in the fly brain [7]. These regions, or neuropils, include sensory circuits such as the olfactory system's antennal lobe and the visual system's lamina and medulla, as well as control and integration neuropils such as the protocerebral bridge and ellipsoid body (Fig. 1). Most of these modules are referred to as local processing units (LPUs) because they are characterized by unique populations of local neurons whose processes are restricted to specific neuropils.

The axons of an LPU's local neurons and the synaptic connections between them and other neurons in the LPU constitute an internal pattern of connectivity that is distinct from

 DOI:10.5281/zenodo.31947

the bundles, or tracts, of projection neuron processes that transmit data to neurons in other LPUs (Fig. 1); this suggests that an LPU's local neuron population and synaptic connections largely determine its functional properties. The fly brain also comprises modules referred to as hubs that contain no local neurons; they appear to serve as communication relays between different LPUs.

In contrast to a purely anatomical subdivision, the decomposition of the brain into functional modules casts the problem of reverse engineering the brain as one of discovering the information processing performed by each individual LPU and determining how specific patterns of axonal connectivity between these LPUs integrates them into functional subsystems. Modeling both these functional modules and the connectivity patterns that link them independent of the internal design of each module is a fundamental requirement of Neurokernel's architecture.

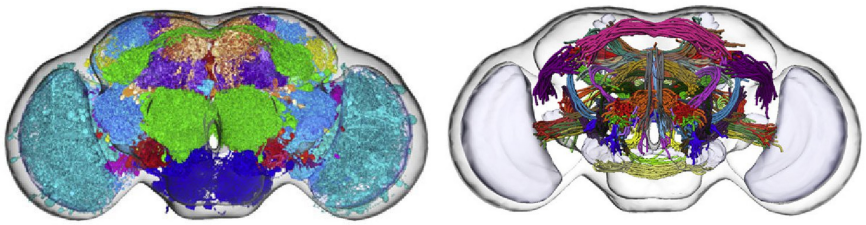

Figure 1: Modular structure of fly brain. Individual LPUs, hubs, and tracts are identified by different colors; for example, the central green structures in the left-hand figure are the antennal lobes, while the large peripheral cyan structures are the medullae. Most LPUs are paired across the fly's two hemispheres. Tracts depicted in the right-hand figure may connect pairs of LPUs located in each hemisphere or within a single hemisphere ([7], reproduced with permission).

2.2 Architecture of the Neurokernel

We refer to our software framework for fly brain emulation as a *kernel* because it aims to provide two classes of functions associated with traditional computer operating systems [31]: it must serve as a *resource allocator* that enables the scalable use of parallel computing resources to accelerate the execution of an emulation, and it must serve as an *extended machine* that provides software services and interfaces that can be programmed to emulate and integrate functional modules in the fly brain.

Neurokernel's architectural design consists of three planes that separate between the time scales of a model's representation and its execution on multiple parallel processors (Fig. 2). This enables the design of vertical APIs that permit development of new features within one plane while minimizing the need to modify code associated with the other planes. Services that implement the computational primitives and numerical methods required to execute supported models on parallel processors are provided by the framework's *compute plane*. Translation or mapping of a models' specified components to the methods provided by the

compute plane and management of the parallel hardware and data communication resources required to efficiently execute a model is performed by Neurokernel's *control plane*. Finally, the framework's *application plane* provides support for specification of neural circuit models, connectivity patterns, and interfaces that enable independently developed models of the fly brain's functional subsystems to be interconnected; we describe these interfaces in greater detail in § 2.4.

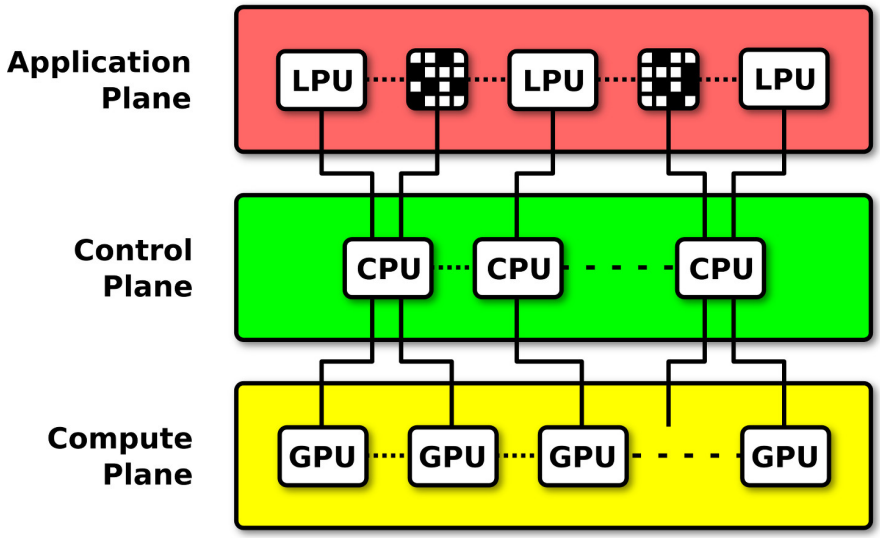

Figure 2: The three-plane structure of the Neurokernel architecture is based on the principle of separation of time scales. The application plane provides support for hardware-independent specification of LPUs and their interconnects. Services that implement the neural primitives and computing methods required to execute neural circuit model instantiations on GPUs are provided by the compute plane. Translation or mapping of specified model components to the methods provided by the compute plane and management of multiple GPUs and communication resources is performed by the control plane operating on a cluster of CPUs.

2.3 Neurokernel Programming Model

2.3.1 Interface Configuration

A key aspect of Neurokernel's design is the separation it imposes between the internal processing performed by an LPU model and how that model communicates with other models (Fig. 3). Neurokernel's programming model requires that one specify how an LPU's interface is configured and connected to those of other LPUs. The interface of an LPU must be described exclusively in terms of communication ports that either transmit data to or receive data from ports exposed by other LPUs. Each port must be configured either to receive input or emit output, and must be configured to either accept spike data represented as boolean

values or graded potential data represented as floating point values (Fig. 4). Both of these settings are mutually exclusive; a single port may not both receive input and emit output, nor may it accept both spike and graded potential data. Ports are uniquely specified relative to other ports within an interface using a path-like identifier syntax to facilitate hierarchical organization of large numbers of ports (Tab. 1).

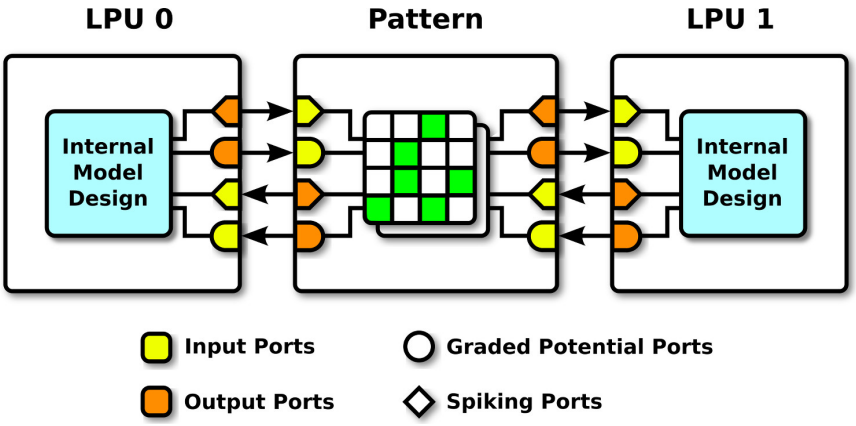

Figure 3: Neurokernel programming model. An LPU model's internal components (cyan) are exposed via input and output ports (yellow and orange). Connections between LPUs are described by patterns (green) that link the ports of one LPU to those of another. Connections may only be defined between ports of the same transmission type (circles, diamonds).

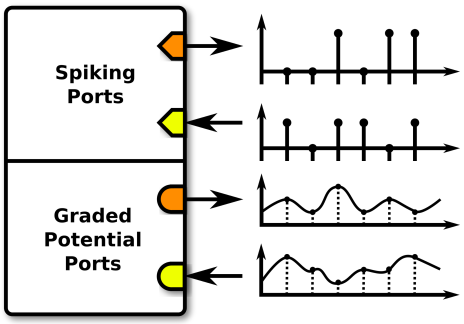

Figure 4: LPU interface. Each communication port must either receive input (yellow) or emit output (orange), and must either transmit spikes (diamonds) or graded potentials (circles).

2.3.2 Pattern Configuration

A single LPU may potentially be connected to many other LPUs; these connections must be expressed as patterns between pairs of LPUs (Fig. 3). Each pattern must be expressed in terms of (1) two interfaces - each comprising a set of ports - between which connections may

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]`

Table 1: Path-like port identifier and selector syntax examples.

be defined, and (2) the actual connections between individual ports in the two interfaces (Tab. 2).

(a) Port Attributes

Port	Interface	I/O	Port Type
`/lam[0]`	0	in	graded potential
`/lam[1]`	0	in	graded potential
`/lam[2]`	0	out	graded potential
`/lam[3]`	0	out	spiking
`/lam[4]`	0	out	spiking
`/lam[5]`	0	out	spiking
`/med[0]`	1	out	graded potential
`/med[1]`	1	out	graded potential
`/med[2]`	1	out	graded potential
`/med[3]`	1	in	spiking
`/med[4]`	1	in	spiking

(b) Connections

From	To
`/lam[0]`	`/med[0]`
`/lam[0]`	`/med[1]`
`/lam[1]`	`/med[2]`
`/med[3]`	`/lam[3]`
`/med[4]`	`/lam[4]`
`/med[4]`	`/lam[5]`

Table 2: Inter-LPU connectivity pattern example. An instance of the `Pattern` class comprises the attributes associated with each port in the pattern's two interfaces (2a) and the connections between ports (2b).

Port attributes are used by Neurokernel to determine compatibility between LPU and pattern objects. To provide LPU designers with the freedom to determine how to multiplex input data from multiple sources within an LPU, Neurokernel does not permit multiple input ports in a pattern to be connected to a single output port. Input ports in a pattern may be connected to multiple output ports. It should be noted that the connections defined by an inter-LPU connectivity pattern do not represent synaptic models; any synapses comprised by a brain model must be a part of the design of a constituent LPU and connected to the LPU's ports in order to either receive or transmit data from or to modeling components in

other LPUs.

2.4 Application Programming Interface

In contrast to other currently available GPU-based neural emulation packages [45, 44, 47, 54], Neurokernel is implemented entirely in Python, a high-level language with a rich ecosystem of scientific packages that has enjoyed increasing popularity in neuroscience research. Although GPUs can be directly programmed using frameworks such as NVIDIA CUDA and OpenCL, the difficulty of writing and optimizing code using these frameworks exclusively has led to the development of packages that enable run-time code generation (RTCG) using higher level languages [4]. Neurokernel uses the PyCUDA package to provide RTCG support for NVIDIA's GPU hardware without forgoing the development advantages afforded by Python [30].

To make use of Neurokernel's LPU API, all LPU models must subclass a base Python class called `Module` that provides LPU designers with the freedom to organize the internal structure of their model implementations as they see fit independent of the LPU interface configuration. Implementation of a Neurokernel-compatible LPU requires that (1) the LPU be uniquely identified relative to all other LPUs to which it may be connected in a subsystem or whole-brain emulation, (2) the execution of all operations comprised by a single step of the LPU's emulation be performed by invocation of a single method called `run_step()`, and that (3) the LPU's interface be configured as described in § 2.3.1.

An instantiated LPU's graded potential and spiking ports are respectively associated with GPU data arrays that Neurokernel accesses to transmit data between LPUs during emulation execution; LPU designers are responsible for reading the data elements associated with input ports and populating the elements associated with output ports in the `run_step()` method. Modeling components that do not communicate with other LPUs and the internal connectivity patterns defined between them are not made accessible through the LPU's interface (Fig. 3).

Inter-LPU connectivity patterns correspond to the connections described by the tracts depicted in Fig. 1. These are represented by a tensor-like class called `Pattern` that contains the port and connection data described in § 2.3.2. To conserve memory, only existing connections are stored in a `Pattern` instance. In addition to manually constructing inter-LPU connectivity patterns using the configuration methods provided by the `Pattern` class, Neurokernel also supports loading connectivity patterns from CSV, GEXF, or XML files using a schema similar to NeuroML [19] with components that enable the specification of LPUs, connectivity patterns, and the ports they expose. Inter-LPU connections currently remain static throughout an emulation; future versions of Neurokernel will support dynamic instantiation and removal of connections while a model is being executed.

The designer of an LPU is responsible for associating ports with internal components that either consume input data or emit output data. Neurokernel provides a class called `GPUPortMapper` that maps port identifiers to GPU data arrays; by default, each `Module` instance contains two `GPUPortMapper` instances that respectively associate the LPU's ports with arrays containing graded potential and spike values. After each invocation of the LPU's

`run_step()` method, data within these arrays associated with the LPU's output ports is automatically transmitted to the port data arrays of destination LPUs, while input data from source LPUs is automatically inserted into those elements associated with the LPU's input ports (Fig. 3).

(a) Mapping of graded potential ports.

Port	Array Index	Array Data
`/lam[0]`	0	0.71
`/lam[1]`	1	0.83
`/lam[2]`	2	0.52

(b) Mapping of spiking ports.

Port	Array Index	Array Data
`/lam[3]`	0	1
`/lam[4]`	1	0
`/lam[5]`	2	1

Table 3: Example of input and output data mapped to and from data arrays by the `GPUPortMapper` class for the ports comprised by interface 0 in Tab. 2.

In addition to the classes that represent LPUs and inter-LPU connectivity patterns, Neurokernel provides an emulation manager class called `Manager` that provides services for configuring LPU classes, connecting them with specified connectivity patterns, and determining how to route data between LPUs based upon those patterns. The manager class hides the process and communication management performed by OpenMPI so as to obviate the need for model designers to directly interact with the traditional MPI job launching interface. Once an emulation has been fully configured via the manager class, it may be executed for a specified interval of time or for a specified number of steps.

Apart from the API requirements discussed above, Neurokernel currently places no explicit restrictions upon an LPU model's internal implementation, how it interacts with available GPUs, how LPUs record their output, or the topology of interconnections between different LPUs; compatible LPUs and inter-LPU patterns may be arbitrarily composed to construct subsystems (Fig. 5). It should be noted that the current LPU interface is not intended to be final; we anticipate its gradual extension to support communication between models that more accurately account for the range of interactions that occur within the fly's brain.

Neurokernel's compute plane currently provides GPU-based implementations for several common neuron and synapse models. These modeling components may be used to construct and execute LPUs without writing any Python code by specifying an LPU's design declaratively as a graph stored in GEXF format. Additional modeling components may be added to the compute plane as plugins.

Communication between LPU instances in a running Neurokernel emulation is performed using MPI to enable brain emulations to take advantage of multiple GPUs hosted either on single computer or a computer cluster. Neurokernel uses OpenMPI [17] to provide accelerated access between GPUs that support NVIDIA's GPUDirect technology [48, 49] when the source and destination memory locations of an MPI data transfer are both in GPU memory. Neurokernel-based models are executed in a bulk synchronous fashion; each LPU's execution step is executed asynchronously relative to other LPUs' execution steps, but data associated

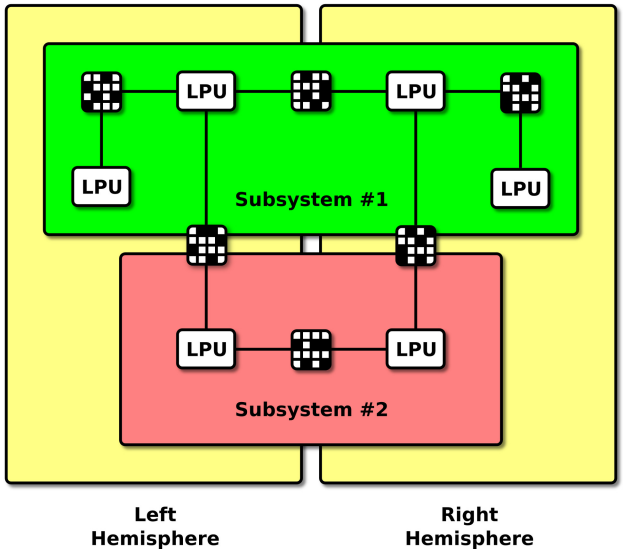

Figure 5: Neurokernel brain modeling architectural hierarchy. Independently developed LPUs and connectivity patterns may be composed into subsystems (red, green) which may in turn be connected to other subsystems to construct a model of the whole brain (yellow).

with the output ports of all connected LPUs must be propagated to their respective destinations before those LPUs can proceed to the next execution step.

2.5 Using the Neurokernel API

This section illustrates how to use the Neurokernel classes described in § 2.4 to construct and execute an emulation consisting of multiple connected LPUs. The section assumes that Neurokernel and its relevant dependencies (including OpenMPI) have already been installed on a system containing multiple GPUs. First, we import several required Python modules; the `mpi_relaunch` module provided by Neurokernel sets up the MPI environment required to enable communication between LPUs.

```
import neurokernel.mpi_relaunch

from mpi4py import MPI
import numpy as np
import pycuda.gpuarray as gpuarray

from neurokernel.mpi import setup_logger
from neurokernel.core_gpu import Module, Manager
from neurokernel.pattern import Pattern
from neurokernel.plsel import Selector, SelectorMethods
```

Next, we create a subclass of `Module` whose `run_step()` method accesses the class instance's port data arrays; the example below generates random graded potential and spiking

output port data.

```
class MyModule(Module):
    """
    Example of derived module class.
    """

    def run_step(self):

        super(MyModule, self).run_step()

        # Log input graded potential data:
        self.log_info('input gpot port data: '+\
                      str(self.pm['gpot'][self.in_gpot_ports]))

        # Log input spike data:
        self.log_info('input spike port data: '+\
                      str(self.pm['spike'][self.in_spike_ports]))

        # Output random graded potential data:
        out_gpot_data = \
            gpuarray.to_gpu(np.random.rand(len(self.out_gpot_ports)))
        self.pm['gpot'][self.out_gpot_ports] = out_gpot_data
        self.log_info('output gpot port data: '+str(out_gpot_data))

        # Output spikes to randomly selected output ports:
        out_spike_data = \
            gpuarray.to_gpu(np.random.randint(0, 2,
                                              len(self.out_spike_ports)))
        self.pm['spike'][self.out_spike_ports] = out_spike_data
        self.log_info('output spike port data: '+str(out_spike_data))
```

The data arrays associated with an LPU's ports may be accessed using their path-like identifiers via two instances of the `GPUPortMapper` class stored in the `self.pm` attribute. Updated data associated with output ports is propagated to the relevant destination LPUs by Neurokernel before the next iteration of the emulation's execution.

To connect two LPUs, we specify the ports to be exposed by each LPU using path-like selectors. The example below describes the interfaces for two LPUs that each expose two graded potential input ports, two graded potential output ports, two spiking input ports, and two spiking output ports. `Selector` is a convenience class that provides methods and overloaded operators for combining and manipulating sets of validated port identifiers. Additional methods for manipulating port identifiers are provided by the `SelectorMethods` class.

```
m1_sel_in_gpot = Selector('/a/in/gpot[0:2]')
m1_sel_out_gpot = Selector('/a/out/gpot[0:2]')
m1_sel_in_spike = Selector('/a/in/spike[0:2]')
m1_sel_out_spike = Selector('/a/out/spike[0:2]')

m2_sel_in_gpot = Selector('/b/in/gpot[0:2]')
m2_sel_out_gpot = Selector('/b/out/gpot[0:2]')
```

```
m2_sel_in_spike = Selector('/b/in/spike[0:2]')
m2_sel_out_spike = Selector('/b/out/spike[0:2]')

m1_sel = m1_sel_in_gpot+m1_sel_out_gpot+\
         m1_sel_in_spike+m1_sel_out_spike
m1_sel_in = m1_sel_in_gpot+m1_sel_in_spike
m1_sel_out = m1_sel_out_gpot+m1_sel_out_spike
m1_sel_gpot = m1_sel_in_gpot+m1_sel_out_gpot
m1_sel_spike = m1_sel_in_spike+m1_sel_out_spike

m2_sel = m2_sel_in_gpot+m2_sel_out_gpot+\
         m2_sel_in_spike+m2_sel_out_spike
m2_sel_in = m2_sel_in_gpot+m2_sel_in_spike
m2_sel_out = m2_sel_out_gpot+m2_sel_out_spike
m2_sel_gpot = m2_sel_in_gpot+m2_sel_out_gpot
m2_sel_spike = m2_sel_in_spike+m2_sel_out_spike

N1_gpot = SelectorMethods.count_ports(m1_sel_gpot)
N1_spike = SelectorMethods.count_ports(m1_sel_spike)

N2_gpot = SelectorMethods.count_ports(m2_sel_gpot)
N2_spike = SelectorMethods.count_ports(m2_sel_spike)
```

Using the above LPU interface data, we construct an inter-LPU connectivity pattern by instantiating the `Pattern` class, setting its port transmission types, and populating it with connections:

```
pat12 = Pattern(m1_sel, m2_sel)
pat12.interface[m1_sel_out_gpot] = [0, 'in', 'gpot']
pat12.interface[m1_sel_in_gpot] = [0, 'out', 'gpot']
pat12.interface[m1_sel_out_spike] = [0, 'in', 'spike']
pat12.interface[m1_sel_in_spike] = [0, 'out', 'spike']
pat12.interface[m2_sel_in_gpot] = [1, 'out', 'gpot']
pat12.interface[m2_sel_out_gpot] = [1, 'in', 'gpot']
pat12.interface[m2_sel_in_spike] = [1, 'out', 'spike']
pat12.interface[m2_sel_out_spike] = [1, 'in', 'spike']
pat12['/a/out/gpot[0]', '/b/in/gpot[0]'] = 1
pat12['/a/out/gpot[1]', '/b/in/gpot[1]'] = 1
pat12['/b/out/gpot[0]', '/a/in/gpot[0]'] = 1
pat12['/b/out/gpot[1]', '/a/in/gpot[1]'] = 1
pat12['/a/out/spike[0]', '/b/in/spike[0]'] = 1
pat12['/a/out/spike[1]', '/b/in/spike[1]'] = 1
pat12['/b/out/spike[0]', '/a/in/spike[0]'] = 1
pat12['/b/out/spike[1]', '/a/in/spike[1]'] = 1
```

We can then pass the defined LPU class and the parameters to be used during instantiation to a `Manager` class instance that connects them together with the above pattern. The `setup_logger` function may be used to enable output of log messages generated during execution:

```
logger = setup_logger(screen=True, file_name='neurokernel.log',
                      mpi_comm=MPI.COMM_WORLD, multiline=True)
```

```
man = Manager()

m1_id = 'm1   '
man.add(MyModule, m1_id, m1_sel, m1_sel_in, m1_sel_out,
        m1_sel_gpot, m1_sel_spike,
        np.zeros(N1_gpot, dtype=np.double),
        np.zeros(N1_spike, dtype=int),
        device=0)
m2_id = 'm2   '
man.add(MyModule, m2_id, m2_sel, m2_sel_in, m2_sel_out,
        m2_sel_gpot, m2_sel_spike,
        np.zeros(N2_gpot, dtype=np.double),
        np.zeros(N2_spike, dtype=int),
        device=1)
man.connect(m1_id, m2_id, pat12, 0, 1)
```

After all LPUs and connectivity patterns are provided to the manager, the emulation may be executed for a specified number of steps as follows. Neurokernel uses the dynamic process creation feature of MPI-2 supported by OpenMPI to automatically spawn as many MPI processes are needed to run the emulation:

```
duration = 10.0
dt = 1e-2
man.spawn()
man.start(int(duration/dt))
man.wait()
```

3 Results

To evaluate Neurokernel's ability to facilitate interfacing of functional brain modules that can be executed on GPUs, we employed Neurokernel's programming model (§ 2.3) to interconnect independently developed LPUs in the fly's early visual system to provide insights into the representation and processing of the visual field by the cascaded LPUs. We also evaluated Neurokernel's scaling of communication performance in simple configurations of the architecture parameterized by numbers of ports and LPUs.

The scope of the effort to reverse engineer the fly brain and the need to support the revision of brain models in light of new data requires a structured means of advancing and documenting the evolution of those models and the framework required to support them. To this end, the Neurokernel project employs Requests for Comments documents (RFCs) as a tool for advancing the designs of both Neurokernel's architecture and the LPU models built to use it. RFCs containing detailed descriptions of the models of the visual system LPUs described below are publicly available on the project website http://neurokernel.github.io/docs.html.

 DOI:10.5281/zenodo.31947

3.1 Integration of Independently Developed LPU Models

The integrated early visual system model we considered consists of models of the fly's retina and lamina. The retina model comprises a hexagonal array of 721 ommatidia, each of which contains 6 photoreceptor neurons. The photoreceptor model employs a stochastic model of how light input (photons) produce a membrane potential output. Each photoreceptor consists of 30,000 microvilli modeled by 15 equations per microvillus, a photon absorption model, and a model of how the aggregate microvilli contributions produce the photoreceptor's membrane potential [32]; the entire retina model employs a total of about 1.95 billion equations. The lamina model consists of 4,326 Morris-Lecar neurons configured to not emit action potentials and about 50,000 conductance-based synapses [37]. The LPUs were linked by 4,326 feed-forward connections from the retina to the lamina; the connections from the retina to the lamina were configured to map output ports exposed by the retina to input ports in the lamina based upon the neural superposition rule [29].

The combined retina and lamina models were executed on up to 4 Tesla K20Xm NVIDIA GPUs with a natural video scene provided as input to the retinal model's photoreceptors. The computed membrane potentials of specific photoreceptors in each retinal ommatidium and of select neurons in each cartridge of the lamina were recorded (Fig. 6); videos of the computed potentials are included in the supporting information. In this example, the observed R1 photoreceptor outputs demonstrate the preservation of visual information received from the retina by the lamina LPU. The L1 and L2 lamina neuron outputs demonstrate the signal inversion taking place in the two pathways shaping the motion detection circuitry of the fly. These initial results illustrate how Neurokernel's API enables LPU model designers to treat their models as neurocomputing modules that may be combined into complex information processing pipelines whose input/output properties may be obtained and evaluated.

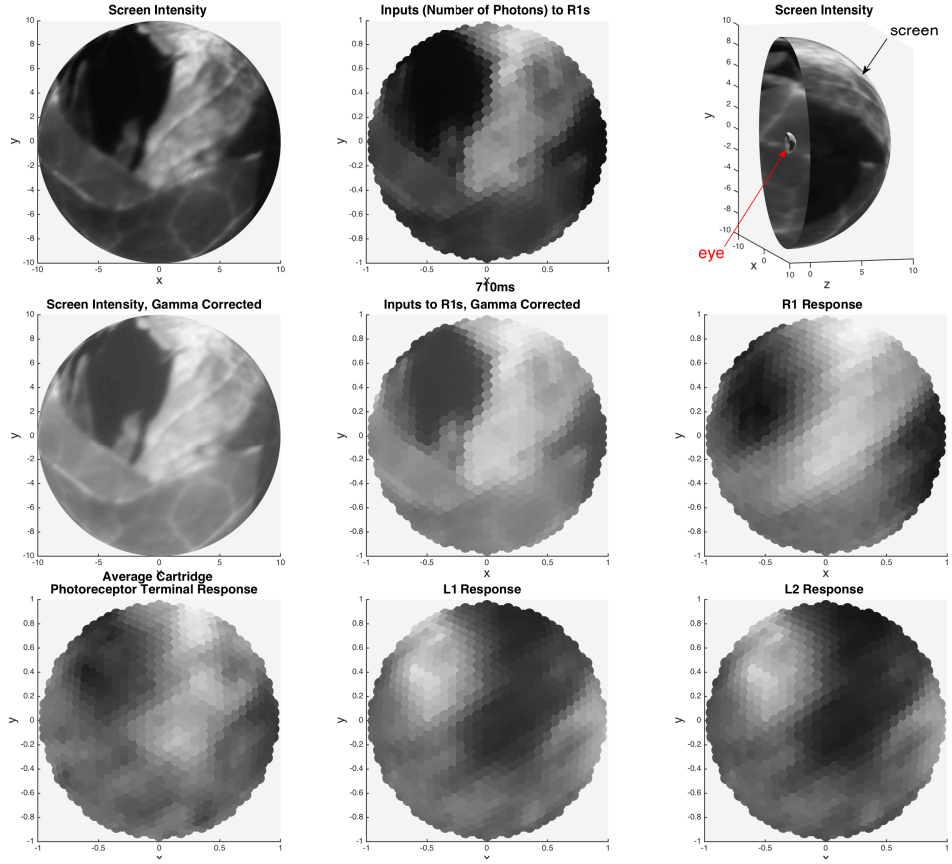

Figure 6: Example of natural input to the combined retina/lamina model. The hexagonal tiling depicts the array of ommatidia in the retina and the corresponding retinotopic cartridges in the lamina. Outputs of select photoreceptors in the retina (R1) that are fed to neurons in the lamina and outputs of specific neurons in the lamina (L1, L2) are also depicted.

3.2 Module Communication Performance

We compared the performance of emulations in which port data stored in GPU memory is copied to and from host memory for traditional network-based transmission by OpenMPI to that of emulations in which port data stored in GPU memory is directly passed to OpenMPI's communication functions. The latter functions enabled direct GPU-to-GPU transmission to take place on supported GPU hardware [49]. All tests discussed below were performed on a host containing 2 Intel Xeon 6-core E5-2620 CPUs, 32 Gb of RAM, and 4 NVIDIA Tesla K20Xm GPUs running Ubuntu Linux 14.04, NVIDIA CUDA 7.0, and OpenMPI 1.8.5 built with CUDA support.

3.2.1 Scaling over Number of LPU Output Ports

To evaluate how well inter-LPU communication scales over the number of ports exposed by an LPU on a multi-GPU machine, we constructed and ran emulations comprising multiple connected instances of an LPU class with an empty `run_step()` method and measured (1) the average time taken per execution step to synchronize the data exposed by the output ports in each of two connected LPUs with their respective destination input ports; (2) the average throughput per execution step (in terms of number of port data elements transmitted per second) of the synchronization, where each port is stored either as a 32-bit integer or double-precision floating point number (both of which occupy 8 bytes).

We initially examined how the above performance metrics scaled over the number of output ports exposed by each LPU in a 2-LPU emulation and over the number of LPUs in an emulation where each LPU is connected to every other LPU and the total number of output ports exposed by each LPU is fixed. The metrics for each set of parameters were averaged over 3 trials; the emulation was executed for 500 steps during each trial.

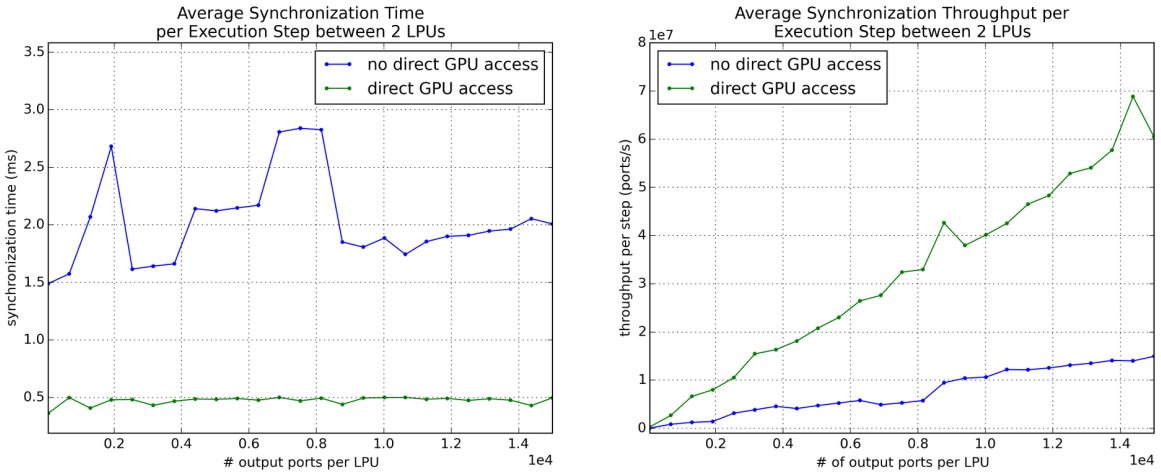

Figure 7: Synchronization performance for an emulation comprising 2 interconnected LPUs accessing 2 different GPUs on the same host scaled over number of output ports exposed by each LPU. The number of output ports was varied over 25 equally spaced values between 50 and 15,000. The plot on the left depicts average synchronization time per execution step, while the plot on the right depicts average synchronization throughput (in number of ports per unit time) per execution step.

The scaling of performance over number of ports depicted in Fig. 7 clearly illustrate the ability of GPU-to-GPU communication between locally hosted GPUs to ensure that increasing the number of ports exposed by an LPU does not increase model execution time for numbers of ports similar to the numbers of neurons in actual LPUs. We also observed noticeable speedups in synchronization time for scenarios using more than 2 GPUs as the number of ports exposed by each LPU is increased (Fig. 8). As the number of GPUs in use reached the maximum available in our test system, overall speedup diminished; this appears

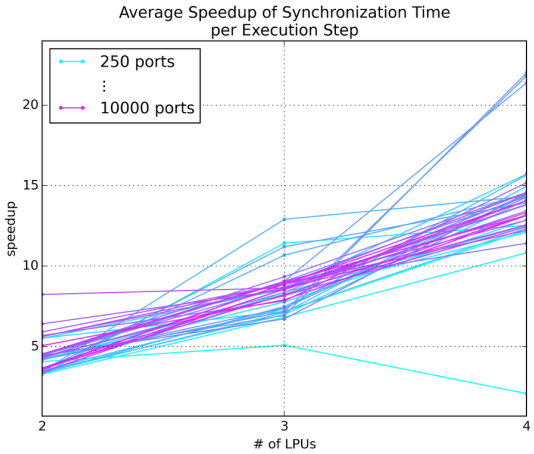

Figure 8: Speedup of average synchronization time per execution step for an emulation scaled over number of LPUs, where each LPU is mapped to a single GPU. The total number of output ports exposed by each LPU was varied between 250 and 10,000 at 250 port intervals.

to be due to gradual saturation of the host's PCI bus.

3.2.2 Scaling over Number of LPUs

Current research on the fly brain is mainly focused on LPUs in the fly's central complex and olfactory and vision systems. Since the interplay between these systems will be key to increasing understanding of multisensory integration and how sensory data might inform behavior mediated by the central complex, we examined how well Neurokernel's communication mechanism performs in scenarios where LPUs from these three systems are successively added to a multi-LPU emulation. Starting with the pair of LPUs with the largest number of inter-LPU connections, we sorted the 19 LPUs in the above three systems in decreasing order of the number of connections contributed with the addition of each successive LPU and measured the average speedup in synchronization time per execution step due to direct GPU-to-GPU data. The number of connections for each LPU was based upon estimates from a mesoscopic reconstruction of the fruit fly connectome; these numbers appear in Document S2 of the supplement of [58]. The LPU class instances were designed to send and receive data only; no other computation was performed or benchmarked during execution. To amortize inter-LPU transmission costs, the LPUs were partitioned across the available GPUs using the METIS graph partitioning package [26] to minimize the total edge cut. The speedup afforded by direct GPU-to-GPU data (Fig. 9) illustrates that current GPU technology can readily power multi-LPU models based upon currently available connectome data.

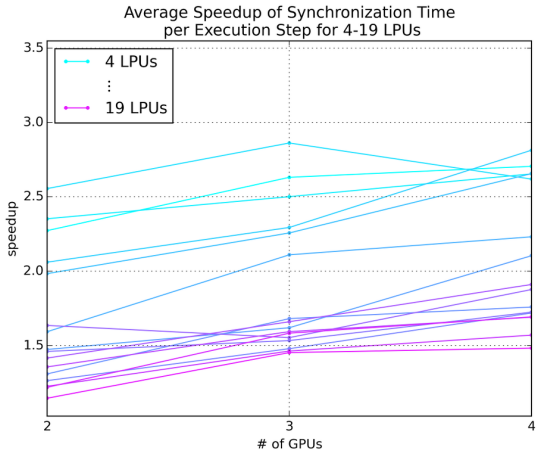

Figure 9: Synchronization performance for an emulation comprising between 4 and 19 interconnected LPUs selected from the central complex, olfactory, and vision systems partitioned over 2 to 4 GPUs on the same host.

4 Discussion

In light of their low costs and rapidly increasing power and availability, there is growing interest in leveraging the power of multiple GPUs to support neural simulations with increasingly high computational demands [65, 46, 41]. When combined with concomitant increases in fly connectomic knowledge and improvements in electrophysiological techniques, the ongoing advance of GPU technology affords an unprecedented opportunity to emulate an entire brain or nervous system of a computationally tractable organism. The OpenWorm project [63], for instance, is capitalizing on the extremely small number of neurons in the nervous system of the nematode *Caenorhabditis elegans* and the full reconstruction of its connectome [68] to develop an emulation of the entire worm on a computer. A recently started effort is the development of a neuromechanical model called Sibernetic [50] that uses GPUs to power simulation of its body and environment. In a similar vein, Neurokernel stands to enable fly researchers to leverage improving GPU technology to take advantage of the increasing amounts of connectome data produced by ongoing advances in our understanding of the fly brain's connectivity [9, 7, 64] for designing and testing fly brain models.

Currently available neural simulation software affords researchers with a range of ways of constructing neural circuit models. These include tools that enable models to be explicitly expressed as systems of differential equations [22], structured documents [19], or explicit calls to a high-level programming API [6, 10, 14]. They also include tools for defining and manipuling neural connectivity patterns [21, 2, 11]. A platform for developing emulations of the entire fly brain, however, must provide programming services for expressing the functional architecture of the whole brain (or its subsystems) in terms of subunits with high-level information processing properties that clearly separate between the internal design of each subunit and how they communicate with each other. Neurokernel's architecture specifically

 DOI:10.5281/zenodo.31947

targets these gaps by providing both the high-level APIs needed to explicitly define and manipulate the architectural elements of brain models as well as the low-level computational substrate required to efficiently execute those models' implementations on multiple GPUs (see Fig. 2).

Existing technologies for interfacing neural models currently provide no native support for the use of GPUs and none of the aforementioned services required to scale over multiple GPU resources. Neurokernel aims to address the problem of model incompatibility in the context of fly brain modeling by ensuring that GPU-based LPU model implementations and inter-LPU connectivity patterns that comply with its APIs are interoperable regardless of their internal implementations.

Despite the impressive performance GPU-based spiking neural network software can currently achieve for simulations comprising increasingly large numbers of neurons and synapses, enabling increasingly detailed fly brain models to efficiently scale over multiple GPUs will require resource allocation and management features that are not yet provided by currently available neural simulation packages. By explicitly providing services and APIs for management of GPU resources, Neurokernel will enable fly brain emulations to benefit from the near-term advantages of scaling over multiple GPUs while leaving the door open to anticipated improvements in GPU technology that can further accelerate the performance of fly brain models.

The challenges of reverse engineering neural systems have spurred a growing number of projects specifically designed to encourage collaborative neuroscience research endeavors. These include technologies for model sharing [23, 19, 20], curation of publicly available electrophysiological data [59], and the construction of comprehensive nervous system models for specific organisms [63]. For collaborative efforts at fly brain modeling to succeed, however, there is a need to both ensure the interoperability of independently developed LPU models without modification of their internal implementations while enforcing a model of the overall brain connectivity architecture. Software packages that enable multiple independently developed neural simulators to execute complex models either by means of communication APIs that simulators must support [12] or through encapsulation of calls to one simulator by a second simulator [51] must be complemented with the flexibility to define and manipulate the emulated connectivity architecture. By imposing mandatory communication interfaces upon models, Neurokernel explicitly ensures that LPU models may be combined with other compatible models to construct subsystem or whole brain emulations.

Neuromorphic platforms whose design is directly inspired by the brain have the potential to execute large-scale neural circuit models at speeds that significantly exceed those achievable with traditional von Neumann computer architectures [60, 62, 53, 52]. Increasing support for high-level software interfaces such as PyNN [10] by such platforms raises the possibility of executing highly detailed LPU models on neuromorphic hardware. As neuromorphic technology matures and becomes available to the wider neurocomputing community, we anticipate Neurokernel's compute plane eventually supporting the use of such hardware alongside and eventually in the place of GPU technology to power whole brain emulations.

 DOI:10.5281/zenodo.31947

5 Future Development

Efforts at reverse engineering the brain must ultimately confront the need to validate hypotheses regarding neural information processing against actual biological systems. In order to achieve biological validation of the Neurokernel, the computational modeling of the fly brain must be tightly integrated with increasingly precise electrophysiological techniques and the recorded data evaluated with novel system identification methods [27, 28, 34, 33, 38, 35, 36]. This will enable direct comparison of the output of models executed by Neurokernel to that of corresponding neurons in the brain regions of interest. Given that recently designed GPU-based systems for emulating neuronal networks of single spiking neuron types have demonstrated near real-time execution performance for networks of up to $\sim 10^5$ spiking neurons and $\sim 10^7$ synapses using single GPUs [45, 15, 54], and in light of advances in the power and accessibility of neuromorphic technology [60, 10, 62, 53, 52], we anticipate that future advances in parallel computing technology will enable Neurokernel's model execution efficiency to advance significantly towards the time scale of the actual fly brain. These advances will enable researchers to validate models of circuits in the live fly's brain within similar time scales and use the observed discrepancies to inform subsequent model improvements (Fig. 10).

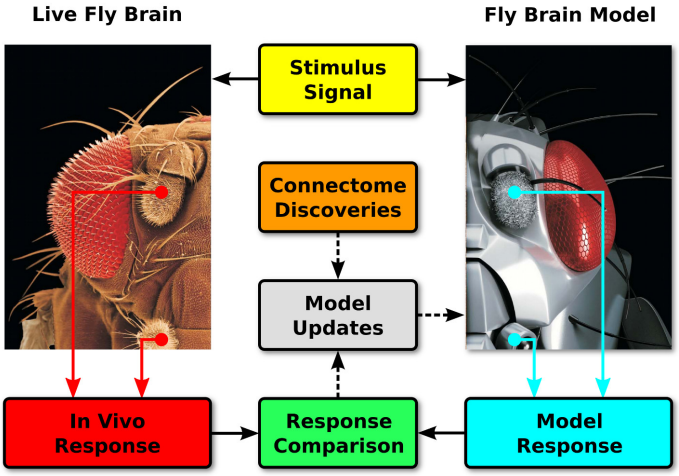

Figure 10: In vivo validation is essential to the development of accurate fly brain models. Neural responses to sensory stimuli are recorded from the live fly brain in real time and compared to the computed responses of the corresponding components in a fly brain model executed on the same time scale. Discrepancies between these responses and new connectome data may be used to improve the model's accuracy (fly photograph adapted from Berger and fly robot image adapted from Vizcano, Benton, Gerber, and Louis, both reproduced with permission).

Although Neurokernel currently permits brain models to make use of multiple GPUs, it requires programmers to explicitly manage the GPU resources used by a model's implementation. Having implemented the API for building and interconnecting LPUs described in

 DOI:10.5281/zenodo.31947

§ 2.4 within Neurokernel's application plane, our next major goal is to implement a prototype GPU resource allocation mechanism within the control plane to automate selection and management of available GPUs used to execute a fly brain model. Direct access to GPUs will also be restricted to modeling components implemented by LPU developers and added to Neurokernel's compute plane; models implemented or defined in the application plane will instantiate and invoke these components. These developments will permit experimentation with different resource allocation policies as LPU models become more complex. Restricting parallel hardware access to modeling components exposed by the compute plane will also facilitate development of future support for other parallel computing technologies such as non-NVIDIA GPUs or neuromorphic hardware.

Neurokernel is a fundamental component of the collaborative workflow needed to accelerate the process of fly brain model development, execution, and refinement by multiple researchers. This workflow, however, also requires a means of efficiently constructing brain models and modifying their structure and parameters in light of output discrepancies observed during validation or to incorporate new experimental data. As noted in § 2.4, Neurokernel currently can execute LPU models declaratively specified as GEXF files that each describe an individual LPU's design as a graph of currently supported neuron and synapse model instances and separately specified inter-LPU connectivity patterns. Since this model representation must either be manually constructed or generated by ad hoc processing of connectome data, modification of LPUs is currently time consuming and significantly slows down the improvement of brain models. LPUs explicitly implemented in Python that do not use supported neuron or synapse models are even less easy to update because of the need to explicitly modify their implementations.

To address these limitations and enable rapid updating and reevaluation of fly brain models, we are building a system based upon graph databases called Neuroarch for the specification and sophisticated manipulation of structural data associated with LPU models and inter-LPU connectivity [18]. Neuroarch will (1) provide LPU developers with a means of defining model components and canonical circuit abstractions using biologically-oriented model-specific labels, (2) enable powerful queries against the data associated with multiple interconnected LPU models via an object-oriented interface similar to that provided by object-relational mapping (ORM) software to web application developers, (3) provide access to model data at different levels of structural abstraction higher than neurons and synapses, (4) enable access to and/or modification of stored data in multiple modes, i.e., as a subgraph (to facilitate graph-based queries) or a table (to facilitate tabular or relational queries), and (5) provide an interface to Neurokernel that enables immediate execution of models defined in Neuroarch.

6 Conclusion

Despite the fly brain's relative numerical tractability, its successful emulation is an ambitious goal that will require the joint efforts of multiple researchers from different disciplines. Neurokernel's open design, support for widely available commodity parallel computing technology,

and ability to integrate independently developed models of the brain's functional subsystems all facilitate this joining of forces. The framework's first release is a step in this direction; we expect and anticipate that aspects of the current design such as connectivity structure and module interfaces will be superseded by newer designs informed by the growing body of knowledge regarding the structure and function of the fly brain. We invite the research community to join this effort on Neurokernel's website (https://neurokernel.github.io/), online code repository (https://github.com/neurokernel/neurokernel), and development mailing list (https://lists.columbia.edu/mailman/listinfo/neurokernel-dev).

7 Acknowledgements

The authors would like to thank Konstantinos Psychas, Nikul H. Ukani, and Yiyin Zhou for developing and integrating the visual system LPU models used to test the software. The authors would also like to thank Juergen Berger for kindly permitting reuse of his fly photograph and thank Nacho Vizcano, Richard Benton, Bertram Gerber, and Matthieu Louis for permitting reuse of the robot fly image they composed for the ESF-EMBO 2010 Conference on Functional Neurobiology in Minibrains.

This work was supported in part by AFOSR under grant #FA9550-12-10232, in part by NSF under grant #1544383, and in part by a Professional Scholarship of the Engineering Graduate Student Council at Columbia University.

8 Supporting Information

S1 Video

Natural video signal input and photoreceptor/neuron outputs of integrated retina/lamina LPU models.

This video depicts a natural video signal input to the photoreceptors in the 721 ommatidia comprised by the retina model, average photoreceptor response per ommatidium, and outputs (membrane potentials) of select photoreceptors (R1) in retina and neurons (L1, L2) in the lamina.

References

[1] J. Douglas Armstrong and Jano I. van Hemert. Towards a virtual fly brain. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 367(1896):2387 –2397, June 2009.

[2] Ulysses Bernardet and Paul F. M. J. Verschure. iqr: A tool for the construction of multi-level simulations of brain and behaviour. *Neuroinformatics*, 8(2):113–134, June 2010.

[3] S. Bradner. The internet standards process - revision 3. *Internet RFCs, ISSN 2070-1721*, RFC 2026, October 1996.

[4] Romain Brette and Dan F. M. Goodman. Simulating spiking neural networks on GPU, December 2012.

[5] Seth A. Budick and Michael H. Dickinson. Free-flight responses of Drosophila melanogaster to attractive odors. *Journal of Experimental Biology*, 209(15):3001 –3017, 2006.

[6] Nicholas T Carnevale and Michael L Hines. *The NEURON Book*. Cambridge University Press, Cambridge; New York, 2006.

[7] Ann-Shyn Chiang, Chih-Yung Lin, Chao-Chun Chuang, Hsiu-Ming Chang, Chang-Huain Hsieh, Chang-Wei Yeh, Chi-Tin Shih, Jian-Jheng Wu, Guo-Tzau Wang, and Yung-Chang Chen. Three-dimensional reconstruction of brain-wide wiring networks in Drosophila at single-cell resolution. *Current Biology*, 21(1):1–11, January 2011.

[8] M. Eugenia Chiappe, Johannes D. Seelig, Michael B. Reiser, and Vivek Jayaraman. Walking modulates speed sensitivity in Drosophila motion vision. *Current Biology*, 20(16):1470–1475, August 2010.

[9] Dmitri B. Chklovskii, Shiv Vitaladevuni, and Louis K. Scheffer. Semi-automated reconstruction of neural circuits using electron microscopy. *Current Opinion in Neurobiology*, 20(5):667–675, October 2010.

[10] Andrew P. Davison, Daniel Brüderle, Jochen Eppler, Jens Kremkow, Eilif Muller, Dejan Pecevski, Laurent Perrinet, and Pierre Yger. PyNN: a common interface for neuronal network simulators. *Frontiers in Neuroinformatics*, 2:11, 2009.

[11] Mikael Djurfeldt. The Connection-Set Algebra - A Novel Formalism for the Representation of Connectivity Structure in Neuronal Network Models. *Neuroinformatics*, 10(3):287–304, July 2012.

[12] Mikael Djurfeldt, Johannes Hjorth, Jochen M. Eppler, Niraj Dudani, Moritz Helias, Tobias C. Potjans, Upinder S. Bhalla, Markus Diesmann, Jeanette Hellgren Kotaleski, and Örjan Ekeberg. Run-time interoperability between neuronal network simulators based on the MUSIC framework. *Neuroinformatics*, 8(1):43–60, March 2010. PMID: 20195795 PMCID: 2846392.

[13] Joseph B. Duffy. GAL4 system in Drosophila: a fly geneticist's Swiss Army knife. *Genesis (New York, N.Y.: 2000)*, 34(1-2):1–15, October 2002. PMID: 12324939.

[14] Chris Eliasmith, Terrence C. Stewart, Xuan Choo, Trevor Bekolay, Travis DeWolf, Yichuan Tang, and Daniel Rasmussen. A large-scale model of the functioning brain. *Science*, 338(6111):1202–1205, November 2012.

[15] A.K. Fidjeland and M.P. Shanahan. Accelerated simulation of spiking neural networks using GPUs. In *Neural Networks (IJCNN), The 2010 International Joint Conference on*, pages 1–8, 2010.

[16] Mark A. Frye and Michael H. Dickinson. Closing the loop between neurobiology and flight behavior in Drosophila. *Current Opinion in Neurobiology*, 14(6):729–736, December 2004.

[17] Edgar Gabriel, Graham E. Fagg, George Bosilca, Thara Angskun, Jack J. Dongarra, Jeffrey M. Squyres, Vishal Sahay, Prabhanjan Kambadur, Brian Barrett, Andrew Lumsdaine, Ralph H. Castain, David J. Daniel, Richard L. Graham, and Timothy S. Woodall. Open MPI: Goals, concept, and design of a next generation MPI implementation. In *Proceedings, 11th European PVM/MPI Users' Group Meeting*, pages 97–104, Budapest, Hungary, September 2004.

[18] Lev E. Givon, Aurel A. Lazar, and Nikul H. Ukani. Neuroarch: A Graph-Based Platform for Constructing and Querying Models of the Fruit Fly Brain Architecture. *Frontiers in Neuroinformatics*, (42), Aug 2014.

[19] Padraig Gleeson, Sharon Crook, Robert C. Cannon, Michael L. Hines, Guy O. Billings, Matteo Farinella, Thomas M. Morse, Andrew P. Davison, Subhasis Ray, Upinder S. Bhalla, Simon R. Barnes, Yoana D. Dimitrova, and R. Angus Silver. NeuroML: a language for describing data driven models of neurons and networks with a high degree of biological detail. *PLoS Comput Biol*, 6(6):e1000815, June 2010.

[20] Padraig Gleeson, Eugenio Piasini, Sharon Crook, Robert Cannon, Volker Steuber, Dieter Jaeger, Sergio Solinas, Egidio D'Angelo, and R. Angus Silver. The Open Source Brain Initiative: enabling collaborative modelling in computational neuroscience. *BMC Neuroscience*, 13(Suppl 1):O7, July 2012.

[21] Padraig Gleeson, Volker Steuber, and R. Angus Silver. neuroConstruct: a tool for modeling networks of neurons in 3D space. *Neuron*, 54(2):219–235, April 2007.

[22] Dan F. M. Goodman and Romain Brette. The Brian simulator. *Frontiers in Neuroscience*, September 2009.

[23] Michael L. Hines, Thomas Morse, Michele Migliore, Nicholas T. Carnevale, and Gordon M. Shepherd. ModelDB: a database to support computational neuroscience. *Journal of Computational Neuroscience*, 17(1):7–11, August 2004. PMID: 15218350.

[24] Stephen J Huston and Vivek Jayaraman. Studying sensorimotor integration in insects. *Current Opinion in Neurobiology*, 21(4):527–534, August 2011.

[25] Eric R. Kandel, Henry Markram, Paul M. Matthews, Rafael Yuste, and Christof Koch. Neuroscience thinks big (and collaboratively). *Nature Reviews Neuroscience*, 14(9):659–664, September 2013.

[26] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1998.

[27] Anmo J. Kim, Aurel A. Lazar, and Yevgeniy B. Slutskiy. System identification of Drosophila olfactory sensory neurons. *Journal of Computational Neuroscience*, 30(1):143–161, August 2011.

[28] Anmo J. Kim, Aurel A. Lazar, and Yevgeniy B. Slutskiy. Projection Neurons in Drosophila Antennal Lobes Signal the Acceleration of Odor Concentrations. *eLife*, page e06651, 2015.

[29] Kuno Kirschfeld. Die projektion der optischen umwelt auf das raster der rhabdomere im komplex auge von musca. *Experimental Brain Research*, 3:248–270, 1967.

[30] Andreas Klöckner, Nicolas Pinto, Yunsup Lee, Bryan Catanzaro, Paul Ivanov, and Ahmed Fasih. PyCUDA and PyOpenCL: a scripting-based approach to GPU run-time code generation. *Parallel Computing*, 38(3):157–174, March 2012.

[31] Aurel A. Lazar. Programming telecommunication networks. *IEEE Network*, 11(5):8–18, October 1997.

[32] Aurel A. Lazar, Konstantinos Psychas, Nikul H. Ukani, and Yiyin Zhou. A Parallel Processing Model of the Drosophila Retina, August 2015. NK RFC #3.

[33] Aurel A. Lazar and Yevgeniy B. Slutskiy. Channel Identification Machines for Multidimensional Receptive Fields. *Frontiers in Computational Neuroscience*, 8, 2014.

[34] Aurel A. Lazar and Yevgeniy B. Slutskiy. Functional Identification of Spike-Processing Neural Circuits. *Neural Computation*, 26(2):264–305, 2014.

[35] Aurel A. Lazar and Yevgeniy B. Slutskiy. Spiking Neural Circuits with Dendritic Stimulus Processors. *Journal of Computational Neuroscience*, 38(1):1–24, 2015.

[36] Aurel A. Lazar, Yevgeniy B. Slutskiy, and Yiyin Zhou. Massively Parallel Neural Circuits for Stereoscopic Color Vision: Encoding, Decoding and Identification. *Neural Networks*, 63:254–271, 2015.

[37] Aurel A. Lazar, Nikul H. Ukani, and Yiyin Zhou. The Cartridge: A Canonical Neural Circuit Abstraction of the Lamina Neuropil - Construction and Composition Rules, January 2014. NK RFC #2.

[38] Aurel A. Lazar and Yiyin Zhou. Volterra Dendritic Stimulus Processors and Biophysical Spike Generators with Intrinsic Noise Sources. *Frontiers in Computational Neuroscience*, 8, 2014.

[39] Gaby Maimon, Andrew D. Straw, and Michael H. Dickinson. A simple vision-based algorithm for decision making in flying Drosophila. *Current Biology*, 18(6):464–470, March 2008.

[40] Matthew S. Maisak, Juergen Haag, Georg Ammer, Etienne Serbe, Matthias Meier, Aljoscha Leonhardt, Tabea Schilling, Armin Bahl, Gerald M. Rubin, Aljoscha Nern, Barry J. Dickson, Dierk F. Reiff, Elisabeth Hopp, and Alexander Borst. A directional tuning map of Drosophila elementary motion detectors. *Nature*, 500(7461):212–216, August 2013.

[41] K. Minkovich, C.M. Thibeault, M.J. O'Brien, A. Nogin, Y. Cho, and N. Srinivasa. HRLSim: a high performance spiking neural network simulator for GPGPU clusters. *IEEE Transactions on Neural Networks and Learning Systems*, 25(2):316–331, 2014.

[42] Javier Morante and Claude Desplan. The color-vision circuit in the medulla of Drosophila. *Current Biology*, 18(8):553–565, April 2008.

[43] Laiyong Mu, Kei Ito, Jonathan P. Bacon, and Nicholas J. Strausfeld. Optic glomeruli and their inputs in drosophila share an organizational ground pattern with the antennal lobes. *The Journal of Neuroscience*, 32(18):6061–6071, May 2012. PMID: 22553013.

[44] Jim Mutch, Ulf Knoblich, and Tomaso Poggio. CNS: a GPU-based framework for simulating cortically-organized networks. Technical Report MIT-CSAIL-TR-2010-013, MIT, 2010.

[45] Jayram Moorkanikara Nageswaran, Nikil Dutt, Jeffrey L. Krichmar, Alex Nicolau, and Alexander V. Veidenbaum. A configurable simulation environment for the efficient simulation of large-scale spiking neural networks on graphics processors. *Neural Networks*, 22(5-6):791–800, July 2009.

[46] Andrew Nere, Sean Franey, Atif Hashmi, and Mikko Lipasti. Simulating cortical networks on heterogeneous multi-GPU systems. *Journal of Parallel and Distributed Computing*, 2012. Article in press.

[47] Thomas Nowotny. Flexible neuronal network simulation framework using code generation for NVidia® CUDA(TM). *BMC Neuroscience*, 12(Suppl 1):P239, July 2011. PMID: null PMCID: PMC3240344.

[48] NVIDIA. CUDA Toolkit 4.0 Readiness for CUDA Applications, March 2011.

[49] NVIDIA. Kepler GK110 whitepaper, 2012.

[50] Andrey Palyanov, Sergey Khayrulin, and Vella Mike. Sibernetic fluid mechanics simulator [internet], 2015.

[51] Dejan Pecevski, Thomas Natschläger, and Klaus Schuch. PCSIM: a parallel simulation environment for neural circuits fully integrated with Python. *Frontiers in Neuroinformatics*, 3:11, 2009.

[52] Robert Preissl, Theodore M. Wong, Pallab Datta, Myron Flickner, Raghavendra Singh, Steven K. Esser, William P. Risk, Horst D. Simon, and Dharmendra S. Modha. Compass: a scalable simulator for an architecture for cognitive computing. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '12, pages 54:1–54:11, Los Alamitos, CA, USA, 2012. IEEE Computer Society Press.

[53] Alexander D. Rast, Xin Jin, Francesco Galluppi, Luis A. Plana, Cameron Patterson, and Steve Furber. Scalable event-driven native parallel processing: the SpiNNaker neuromimetic system. In *Proceedings of the 7th ACM international conference on Computing frontiers*, CF '10, page 21–30, New York, NY, USA, 2010. ACM. ACM ID: 1787279.

[54] Micah Richert, Jayram Moorkanikara Nageswaran, Nikil Dutt, and Jeffrey L. Krichmar. An efficient simulation environment for modeling large-scale cortical processing. *Frontiers in Neuroinformatics*, 5:19, 2011.

[55] Jens Rister, Dennis Pauls, Bettina Schnell, Chun-Yuan Ting, Chi-Hon Lee, Irina Sinakevitch, Javier Morante, Nicholas J. Strausfeld, Kei Ito, and Martin Heisenberg. Dissection of the peripheral motion channel in the visual system of Drosophila melanogaster. *Neuron*, 56(1):155–170, October 2007.

[56] Joshua R. Sanes and S. Lawrence Zipursky. Design principles of insect and vertebrate visual systems. *Neuron*, 66(1):15–36, April 2010.

[57] Johannes D. Seelig and Vivek Jayaraman. Feature detection and orientation tuning in the Drosophila central complex. *Nature*, advance online publication, October 2013.

[58] Chi-Tin Shih, Olaf Sporns, Shou-Li Yuan, Ta-Shun Su, Yen-Jen Lin, Chao-Chun Chuang, Ting-Yuan Wang, Chung-Chuang Lo, Ralph J. Greenspan, and Ann-Shyn Chiang. Connectomics-Based Analysis of Information Flow in the Drosophila Brain. *Current Biology*, 0(0), 2015.

[59] Tripathy Shreejoy, Gerkin Richard, Savitskaya Judy, and Urban Nathaniel. NeuroElectro.org: a community database on the electrophysiological diversity of mammalian neuron types. *Frontiers in Neuroinformatics*, 7, 2013.

[60] Rae Silver, Kwabena Boahen, Sten Grillner, Nancy Kopell, and Kathie L. Olsen. Neurotech for neuroscience: Unifying concepts, organizing principles, and emerging tools. *The Journal of Neuroscience*, 27(44):11807 –11819, October 2007.

[61] Zhuoyi Song, Marten Postma, Stephen A. Billings, Daniel Coca, Roger C. Hardie, and Mikko Juusola. Stochastic, adaptive sampling of information by microvilli in fly photoreceptors. *Current Biology*, 22(15):1371–1380, June 2012.

[62] Terrence C. Stewart, Bryan Tripp, and Chris Eliasmith. Python scripting in the Nengo simulator. *Frontiers in Neuroinformatics*, 3:7, 2009.

[63] Balazs Szigeti, Padraig Gleeson, Michael Vella, Sergey Khayrulin, Andrey Palyanov, Jim Hokanson, Michael Currie, Matteo Cantarelli, Giovanni Idili, and Stephen Larson. OpenWorm: an open-science approach to modelling Caenorhabditis elegans. *Frontiers in Computational Neuroscience*, 8:137, 2014.

[64] Shin-Ya Takemura, Arjun Bharioke, Zhiyuan Lu, Aljoscha Nern, Shiv Vitaladevuni, Patricia K. Rivlin, William T. Katz, Donald J. Olbris, Stephen M. Plaza, Philip Winston, Ting Zhao, Jane Anne Horne, Richard D. Fetter, Satoko Takemura, Katerina Blazek, Lei-Ann Chang, Omotara Ogundeyi, Mathew A. Saunders, Victor Shapiro, Christopher Sigmund, Gerald M. Rubin, Louis K. Scheffer, Ian A. Meinertzhagen, and Dmitri B. Chklovskii. A visual motion detection circuit suggested by Drosophila connectomics. *Nature*, 500(7461):175–181, August 2013.

[65] C.M. Thibeault, R. Hoang, and F.C. Harris, Jr. A novel multi-GPU neural simulator. In *Proceedings of 3rd International Conference on Bioinformatics and Computational Biology 2011*, New Orleans, LA, March 2011.

[66] Trevor J. Wardill, Olivier List, Xiaofeng Li, Sidhartha Dongre, Marie McCulloch, Chun-Yuan Ting, Cahir J. O'Kane, Shiming Tang, Chi-Hon Lee, Roger C. Hardie, and Mikko Juusola. Multiple spectral inputs improve motion discrimination in the Drosophila visual system. *Science*, 336(6083):925–931, May 2012. PMID: 22605779.

[67] Barry Warsaw, Jeremy Hylton, David Goodger, and Nick Coghlan. PEP purpose and guidelines. *Python Enhancement Proposals*, PEP 1, June 2000.

[68] J. G. White, E. Southgate, J. N. Thomson, and S. Brenner. The structure of the nervous system of the nematode Caenorhabditis elegans. *Philosophical Transactions of the Royal Society of London. B, Biological Sciences*, 314(1165):1–340, November 1986. PMID: 22462104.

[69] Rachel I. Wilson. Understanding the functional consequences of synaptic specialization: insight from the Drosophila antennal lobe. *Current Opinion in Neurobiology*, 21(2):254–260, April 2011.