POSTER P4169 | CONTACT NAME Pedro Rittner: pr273@cornell.edu

The MYRIAD Simulator: Densely Coupled Realistic Neural Networks on GPU

Pedro Rittner and Thomas A. Cleland
Cornell University

We gratefully acknowledge the support of NVIDIA Corporation for donation of the GTX Titan GPU used for this work.

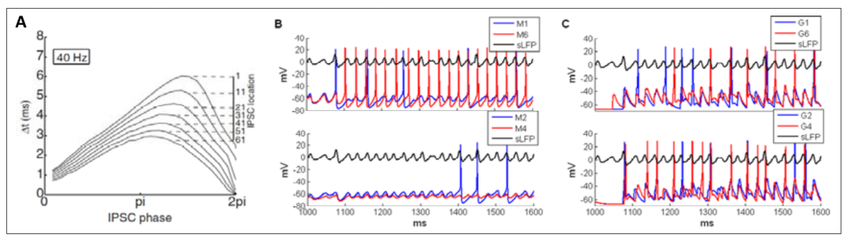
WHAT IS MYRIAD?

- **A framework for developing scalable, biophysically-accurate, and sophisticated neural simulations for computational neuroscientists**
- Extensible object-oriented C99 with on-GPU type checking
- Allows for single simulations to be run both in CPU or in GPU
- Ease-of-use for less-technical users with Python bindings

BACKGROUND: NEURAL MODELING

- Biological neural simulations are diverse, but larger simulations remain computationally limited, especially for *highly-coupled systems*
- Existing parallel simulators rely on hand-coded, user-side, MPI-based, cluster-centric architectures
- Spike-coupled networks dominate computational neuroscience due to their compatibility with the limitations of cluster architectures
- Realistic network models that can directly simulate brain circuits have no reliable, easy-to-use frameworks built for large, complex networks

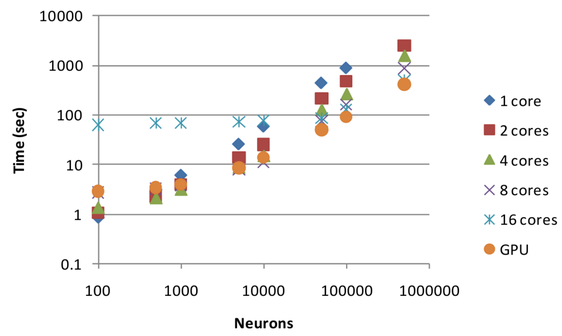
Fig 1: An Example of a Highly-Coupled Olfactory Bulb Network



CLUSTER VS. GPU

- For highly-coupled systems, MPI-based clusters get bogged down due to Ethernet contention; constant updates over network cause thrashing
- Result: Simulators ignore the problem, prioritize development of simplistic spiking networks, which can work well with MPI clusters
- Fast GPU device memory has been shown to perform better under very high update loads; memory contention is less of a problem

Fig 2: Spiking Network Performance of $500k Cluster vs. GTX 260



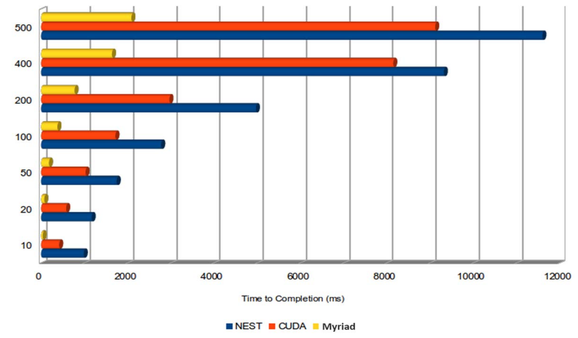
Time needed to complete a simulation for a neural population in a ring topology. The number of synapses equals the number of neurons.

Galbraith B (2010) Computational Modeling of Biological Neural Networks on GPUs: Strategies and Performance. *Master's Theses (2009 -)*, Paper 61. http://epublications.marquette.edu/theses_open/61

THE MYRIAD ADVANTAGE

- Removes unnecessary hierarchy by flattening biological structure into
  - **Compartments –** self-describing structs with neuronal state
  - **Mechanisms –** unilateral point-to-point interaction definitions
- Result: All interactions are reduced to shared memory updates on GPU
  - Synchronization is faster than Ethernet for highly-coupled systems
- Can use large matrix computations with minimal CUDA core execution overhead; most computation is centered on simple integrators/solvers

Fig 3: Neural Simulator Performance, 560Ti vs. Octacore CPU



MINIMAL OBJECT-ORIENTED DESIGN

- Extensibility requirements of neuroscience community require some form of OOP, but CUDA doesn't allow on-the-card type inference
- Solution: Create bare-minimal OOP system with low overhead
  - **Extensibility –** Nest "super-class" struct at head of subclass
  - **Methods –** Embed device function pointers in structs in CPU
  - **Type Inference –** Simple static pointer comparison tree traversal

THE MYRIAD ARCHITECTURE

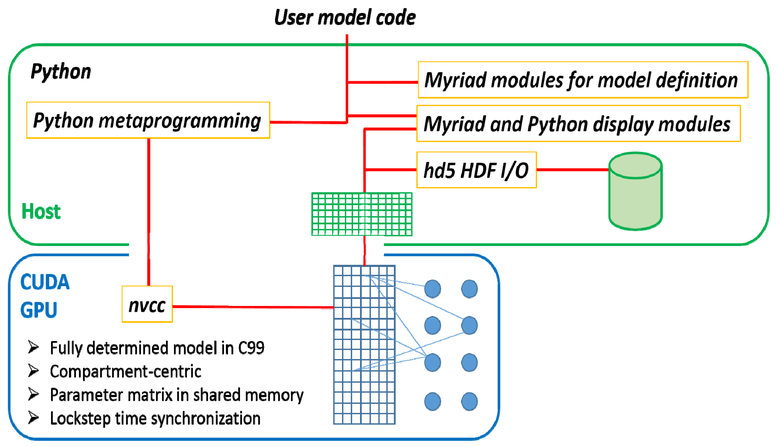


Fig 4: High-Level Overview of the Myriad Architecture

- Python-based user-level model code with simple API
- Language-independent model definitions based on NeuroML
- Python meta-programming enables JIT optimizations, flexibility, and portability; works well with autotools for deployment
- Take advantage of CPython memory pass-through/pointer sharing from C to Python for zero-memcpy Numpy arrays
- C99 code optimized to work on multicore CPUs as well as GPUs
- Utilizes fast, efficient, reliable HDF5 framework for data management; excellent scalability properties for Big Data

FUTURE DIRECTIONS

- Perfecting thread/block synchronization in single-GPU systems
- Multi-card utilization for more complex simulations
  - **SLI –** Leverage shared memory connection
  - **Coupled Tesla cards** within single system
  - **GPU Clusters –** Networked GPU computers
- Large space for parameter-searching applications
  - Manage multiple GPUs to optimize parameterization
  - Use host-pinned memory for data analysis