Solving Hierarchical Neuroscience Problems With Parsl
Matthew Madany, madany@ucsd.edu
Challenges in Application Design

- Defining a parallelization scheme for efficient GPU and CPU utility
- Defining a data structure scheme that is scalable and portable

Design Criterion:
3D volume has to be factored into windows (tensors) compatible to GPU specs, while this process must also compress and store data compatible to storage location, (balancing shared filesystem capacity, node storage capacity, compression, inodes)

Scalable Volume ML/GPU Applications must therefore define:
Workers to import and format data, workers to port this data through GPUs, workers to compile outputs back to original volume while dealing with complex multidimensional processing pipelines that wrap around simplified machine-learning experiments.
Solution: Combine a parallel execution engine with a concurrent data structure scheme

Build tensor processing applications in parsl apps which can port into any kind of local, cloud, or supercomputer application

Define data concurrency and compression schemes that are compatible with the parsl app and A.I. workflow

Building Scalable volumetric A.I. applications becomes much easier

```python
from torch.utils.data import DataLoader
import z5py, parsl
#Define Configuration (local, local-ssh-cluster, slurm, lsf, kubernetes)
class N5Dataset(DataLoader):
    def load(self, image_data):
        # Use z5py to define concurrency and compression dataset
        # z5py.create_dataset('image', shape=image_shape, chunks=(1,512,512), dtype='uint8', compression options)
        # define parsl applications
    @python_app
    def factor(iteration):
        # Large image data -> tensor field -> single tensor
```

Data achieves very high dimensionalities and efficiencies
Dataset x Batch x Channels x Z-Depth x X-Dimension x Y-Dimension x Tensor-iteration

Configuration options passed to Parsl and z5py will naturally load-balance these needs:
- Efficiency in GPU/CPU processing
- Wrangling large data into deep learning frameworks
- Easy design of high-dimensionality processing pipelines
- Data compression cost and benefit
- Porting of applications to heterogonous node definitions

Altogether enable the testing of applications at scale that we wouldn’t normally even consider building in the first-place due to parallelization challenges.
Electron Microscopy Superresolution

**Generator**
- Input Layer
  - Layers: 1
  - Type: 3D Convolution
  - Filters: 96
  - Kernel: 3x7x7
  - Stride: 2x2x2

- Body
  - Layers: 6
  - Type: 3D ResNet Blocks
  - Filters: 384
  - Kernel: 3x3x3
  - Stride: 2x2x2

- Upsampling Pool
  - Layers: 2
  - Type: Transposed 3D Convolution
  - Input Filters: 96
  - Output Filters: 384
  - Kernel: 3x3x3
  - Stride: 2x2x2

**Discriminator**
- Input Layer
  - Layers: 1
  - Type: 3D Convolution
  - Filters: 84
  - Kernel: 3x3x3
  - Stride: 2x2x2

- Convolution Pool 1
  - Layers: 2
  - Type: 3D Convolution
  - Filters: 168
  - Kernel: 3x3x3
  - Stride: 2x2x2

- Convolution Pool 2
  - Layers: 2
  - Type: 3D Convolution
  - Filters: 336
  - Kernel: 3x3x3
  - Stride: 2x2x2

- Output Layer
  - Layers: 1
  - Type: 3D Convolution
  - Filters: 84
  - Kernel: 3x3x3
  - Stride: 2x2x2

**Training Preprocess**
- Random 3D Tile Extraction
- Set A Tensor
- Set B Tensor

**Inference**
- Rolling 3D Window
- Blank Upsampling
- Original
- Resolved
- Scaffold
- Generator B

**Training Setup**
- Set A
- G-D Pair A
- Step 1
- Step 2
- Step 3
- Step 4
- Set B
- G-D Pair B
- Step 1
- Step 2
- Step 3
- Step 4

**Objective Pairs**
- Step 1
- Step 2
- Step 3
- Step 4
Examples of Applications

Superresolution

MultiLabel Voxel Classification

Content-Aware Image Restoration

Microscopy Domain Translation
Examples of Reconstructions, Analyses, and Composable Workflows