Progress Report

Kuei-Yueh (Clint) Ko

Quick summary of my current progress

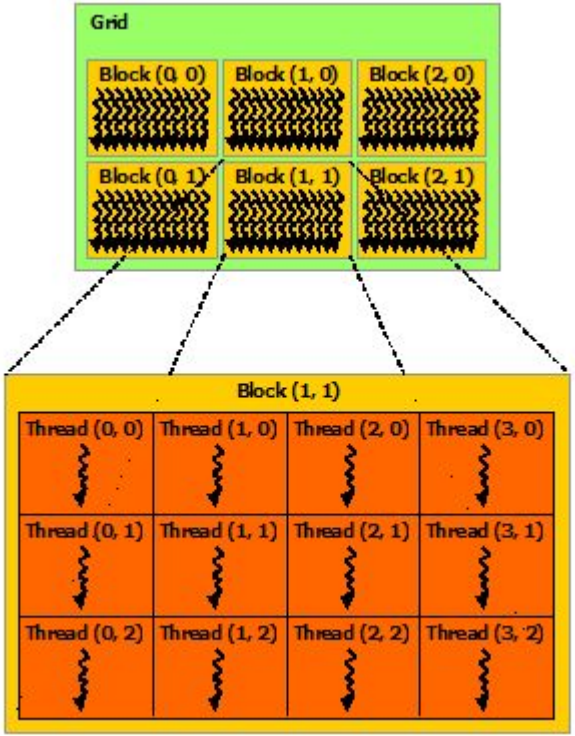
- **Learn Numba CUDA**
  - To-Do: Learn to run python with GPU acceleration (Numba cuda)
  - To-Do: Try to improve the interpolation function using Numba cuda.
  - ----------
  - Progress: After reading different examples and some documents, currently I have succeeded in writing a simple interpolation function with Numba cuda.

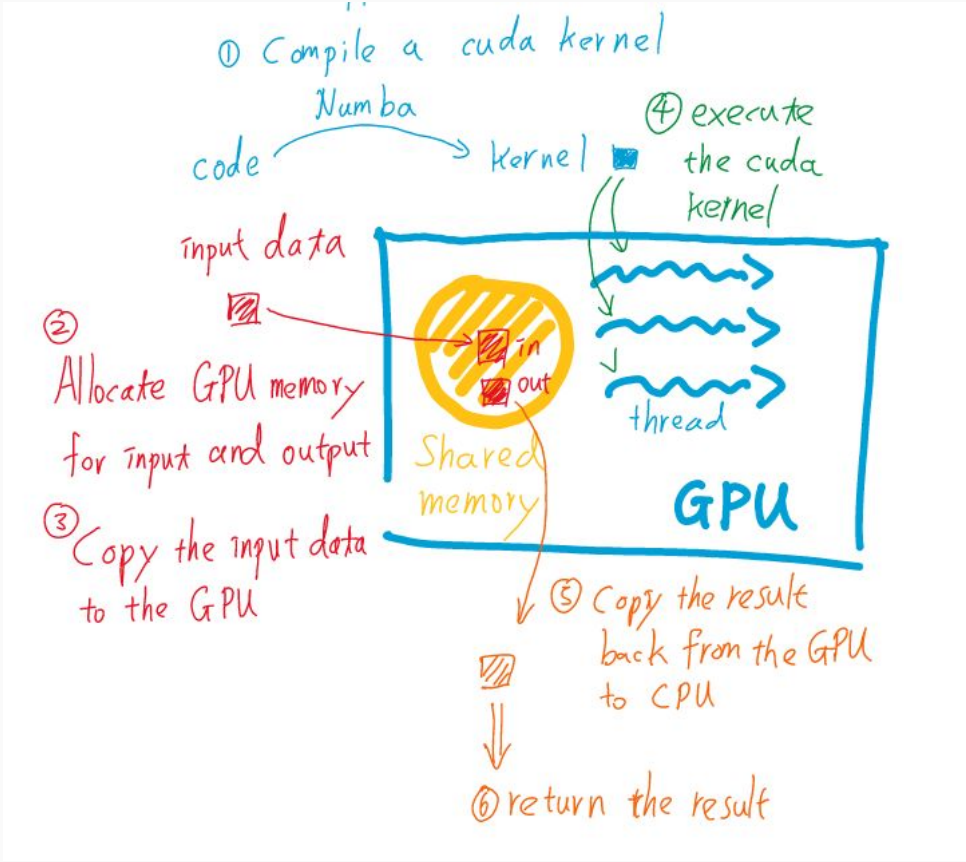
- **Data+**
  - To-Do: FastQ -> Count
  - To-Do: Read paper Myoepithelial Cells of Submucosal Glands Can Function as Reserve Stem Cells to Regenerate Airways after Injury
  - ----------
  - Progress: I have started to read the paper, but haven't understood the scripts that Yoshi ran. Scott is installing the tools on the bubbles.

- **Approximate UMAP by NN**
  - To-Do: Try to use more events / cells to approximate the UMAP function
  - ----------
  - Progress: Currently no progress yet

Progress on Trying Numba.cuda

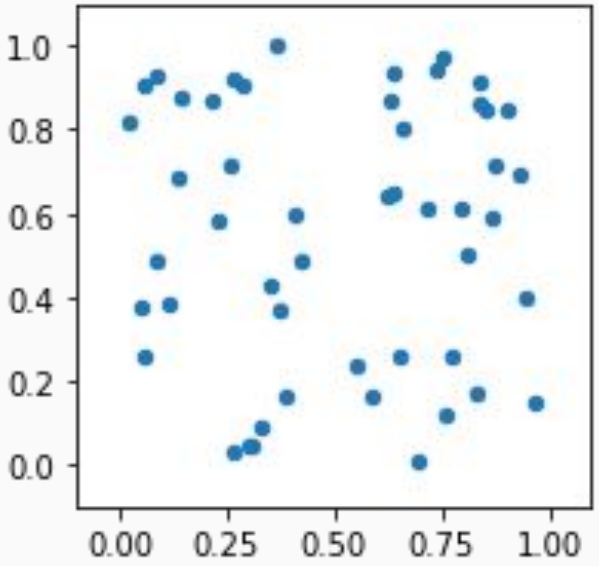
The CUDA Programming Model

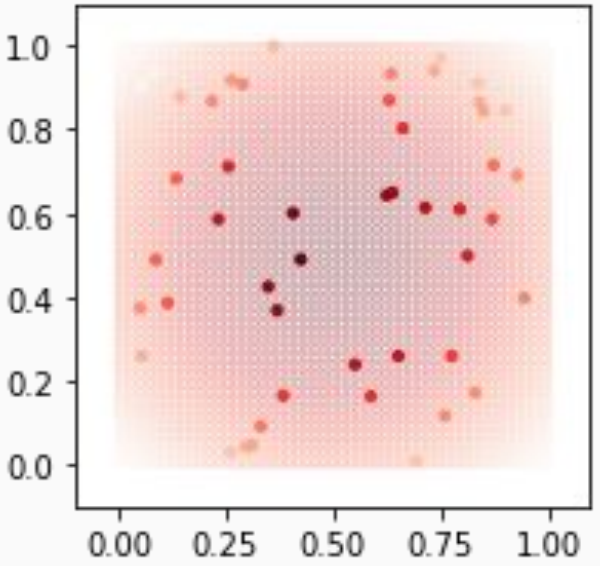




Prepare Points for Interpolation

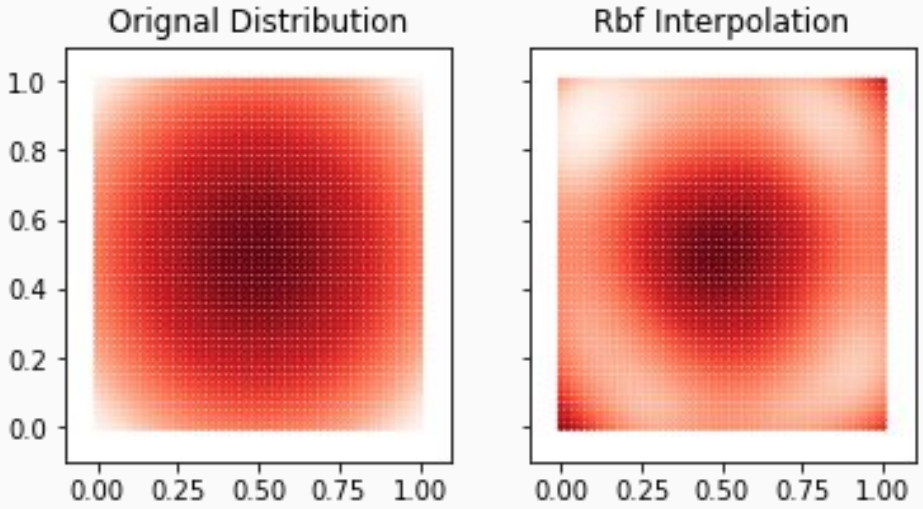
```
dat_pts = np.array([rand.random(50), rand.random(50)])

distr = stat.multivariate_normal(mean = [0.5, 0.5], cov=np.eye(2))
```





Radial basis function Interpolation

```
rbfi = Rbf(
    dat_pts[:, 0],
    dat_pts[:, 1],
    value,
    function='multiquadric',
    smooth=1)
```



ufunc to calculate pairwise Distance

```
def dist2(p_x, p_y, x, y):
    """calculate the distance between the point (x, y) and input point"""
    return np.power(p_x - x, 2) + np.power(p_y - y, 2)


@vectorize(['float32(float32, float32, float32, float32)'], target='cuda')
def dist2_ufunc(p_x, p_y, x, y):
    """calculate the distance between the point (x, y) and input point"""
    return math.pow(p_x - x, 2) + math.pow(p_y - y, 2)
    #return math.pow(point["x"] - x, 2) + math.pow(point["y"] - y, 2)
```

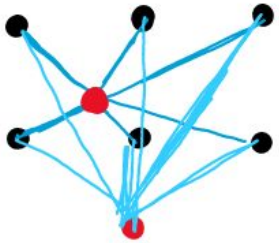
Benchmark of Distance ufunc

```
# Check if the function work correctly

x = dat_pts[:, 0].astype(np.float32)
y = dat_pts[:, 1].astype(np.float32)

np.allclose(
     dist2(x, y, 0.0, 0.0),
     dist2_ufunc(x, y, 0.0, 0.0))
=> True
```

```
#  Try Larger test data to time function

n = 1000000
# ----------------------
a = np.random.random(n).astype(np.float32)
b = np.random.random(n).astype(np.float32)
# ----------------------
a_device   = cuda.to_device(a)
b_device   = cuda.to_device(b)
out_device = cuda.device_array(
     shape=(n,), dtype=np.float32)
```

```
#  Benchmark Results

Numpy
13.2 ms ± 1.1 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
GPU (CPU input/output arrays)
5.02 ms ± 46 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
GPU (GPU input/output arrays)
2.03 ms ± 5.83 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

Interpolation of one marker (weight := 1 / exp{dist2})



$$\exp\left\{-\frac{(x-\mu)^2}{2\sigma^2}\right\}$$

```
foreach grid point in grids:
    res = 0
    foreach data point in points:
        dist  = d(grid, point)
        value = val(point)
        res  += value * 1 / exp{dist^2}
    out[grid] = res
```

Interpolation of one marker (weight := 1 / exp{dist2})

```
def interpolation(pts_x, pts_y, value, grids_x, grids_y, grids_z):
    """interpolation"""
    dim = grids_x.shape

    for idx_x in range(dim[0]):
        for idx_y in range(dim[1]):
            distance2 = dist2(pts_x, pts_x, grids_x[idx_x, idx_y], grids_y[idx_x, idx_y])
            weight = np.exp(-distance2)
            grids_z[idx_x, idx_y] = np.sum(value * weight)


@cuda.jit('float32(float32, float32, float32, float32)', device = True)
def dist2_gpu(p_x, p_y, x, y):
    """calculate the distance between the point (x, y) and input point"""
    return math.pow(p_x - x, 2) + math.pow(p_y - y, 2)
    #return math.pow(point["x"] - x, 2) + math.pow(point["y"] - y, 2)

@cuda.jit
def interpolation_gpu(pts_x, pts_y, value, grids_x, grids_y, grids_z):
    """addition of exponential matrix"""
    # thread coordinate system
    idx_x, idx_y = cuda.grid(2)

    #
    grid_value = 0
    for idx in range(len(value)):
        distance2 = dist2_gpu(pts_x[idx], pts_x[idx], grids_x[idx_x, idx_y], grids_y[idx_x, idx_y])
        weight = math.exp(-distance2)
        grid_value += (value[idx] * weight)

    grids_z[idx_x, idx_y] = grid_value
```

Benchmark of Interpolation

"Numpy"

```
%%timeit
interpolation(dat_pts[:, 0], dat_pts[:, 1], value, x_c, y_c, z_test)
```

```
27.8 ms ± 1.3 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

```
%%timeit
interpolation_gpu[blockspergrid, threadsperblock](
    dat_pts[:, 0].astype(np.float32),
    dat_pts[:, 1].astype(np.float32),
    value.astype(np.float32),
    x_c.astype(np.float32),
    y_c.astype(np.float32), z_test_gpu)
```

```
2.44 ms ± 23.6 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

```
%%timeit
interpolation_gpu[blockspergrid, threadsperblock](
    x_device,
    y_device,
    z_device,
    xc_device,
    yc_device,
    out_device)

out_host = out_device.copy_to_host()
```

```
1.36 ms ± 526 ns per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

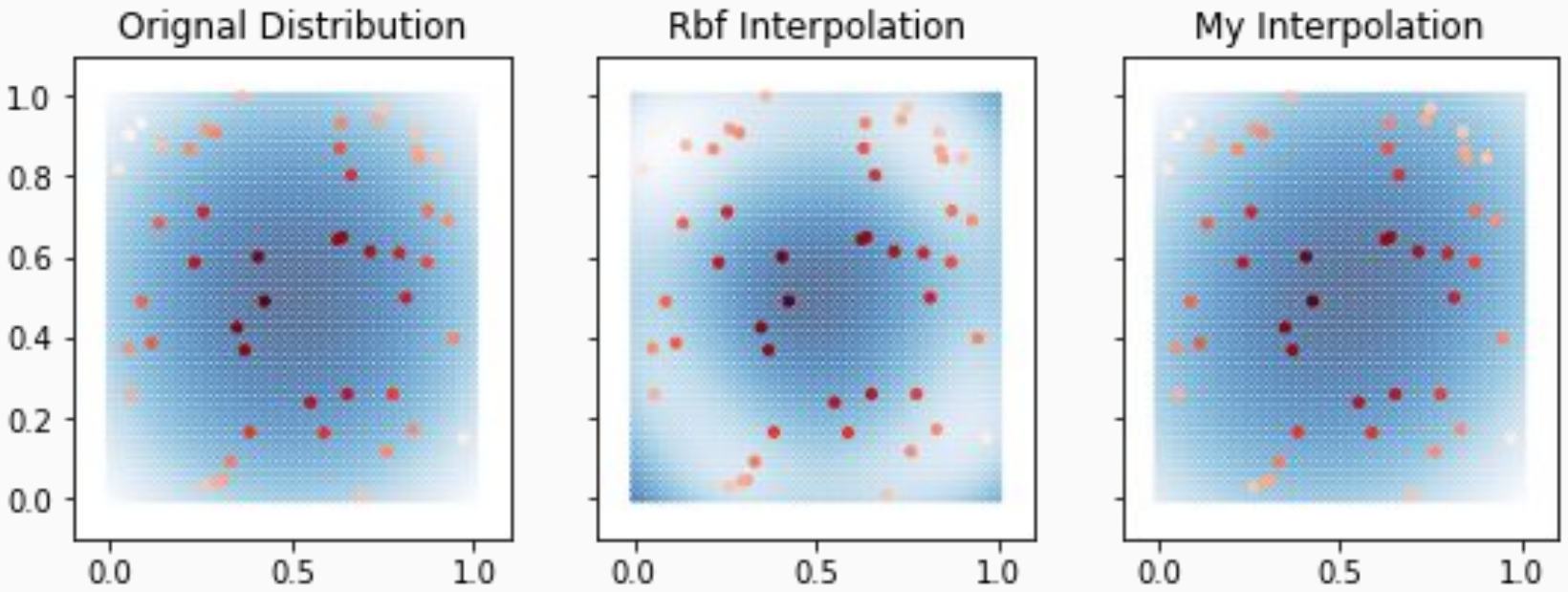
Benchmark of Interpolation

```
%%timeit
rbfi = Rbf(dat_pts[:, 0], dat_pts[:, 1], value, function='multiquadric', smooth=1)
z_rbf = rbfi(x_c, y_c)
```

```
1.57 ms ± 6.42 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

However, in this small dataset, there is not much improvement yet. Therefore, I will try to see the difference when the input contains more data points and more grid points in the interpolation.

Compare Output



Next Step

- Try different examples for interpolation
- Interpolation for multiple markers (I am still thinking how to write it.)