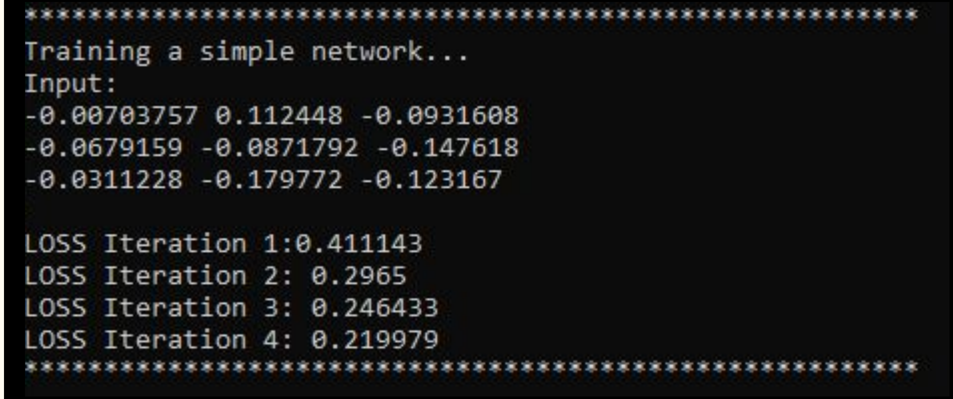
Point Cloud Classification Milestone 2

Disha Jindal, Kushagra Goel and Saket Karve

Milestone 2 Achievements

- Training and Inference
- Forward and Backward
  - Graph Convolution Layer
  - Global Pooling Layer
  - Dropout Layer
- Basic kernels
- Optimized reduction on GPU
- Integration

Training



```
*********************************************************
Training a simple network...
Input:
-0.00703757 0.112448 -0.0931608
-0.0679159 -0.0871792 -0.147618
-0.0311228 -0.179772 -0.123167

LOSS Iteration 1:0.411143
LOSS Iteration 2: 0.2965
LOSS Iteration 3: 0.246433
LOSS Iteration 4: 0.219979
*********************************************************
```

Graph Convolution Layer

Forward Pass

Transforms an input of size (N □ M1) to (N □ M2) where N is the number of points in one sample.

$$Y = \sum_{i=1}^{K} T_i(L) X \theta_i$$

where L is the Symmetric Normalized Laplacian and $T_i(L)$ is the Chebyshev polynomial of order ***i*** and is defined as

$$T_k(L) = 2LT_{k-1}(L) - T_{k-2}(L) \qquad T_0(L) = I_n \quad \text{and} \quad T_1(L) = L$$

Optimizing the forward pass

$$T_k(L) = 2\boxed{LT_{k-1}(L)} - T_{k-2}(L)$$

(1024 x 1024) x (1024 x 1024)

Calculate $T_i(L)X$ directly instead so that we multiply smaller matrices

$$T_k(L)X = 2\boxed{LT_{k-1}(L)X} - T_{k-2}(L)X$$

(1024 x 1024) x (1024 x 3)

Backward Pass

The weight update will be,

$$\frac{\partial L}{\partial \theta} = (I \otimes (T_i(L)X)^T)G_I$$

And the gradient which will be back propagated to the previous layer will be,

$$\frac{\partial L}{\partial X} = (\sum_{i=1}^{K} \theta_i \otimes T_i^T)G_I$$

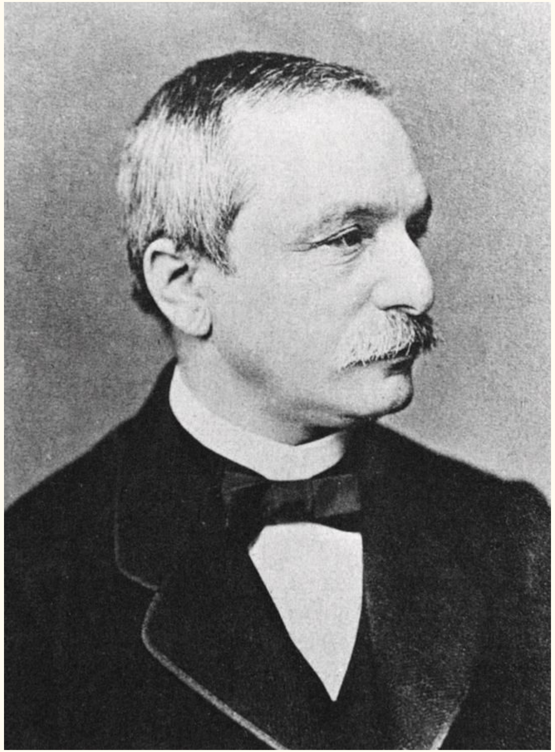
Kronecker Product!

What is Kronecker Product?

$$\begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{bmatrix} \otimes \begin{bmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{bmatrix} = \begin{bmatrix} a_{1,1}\begin{bmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{bmatrix} & a_{1,2}\begin{bmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{bmatrix} \\ \\ a_{2,1}\begin{bmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{bmatrix} & a_{2,2}\begin{bmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{bmatrix} \end{bmatrix}$$

The Kronecker product we needed had dimensions of roughly $10^6 \times 10^6$!!

This was impossible to store even on a CPU! ($\sim 2^{14}$ GB)



https://en.wikipedia.org/wiki/Leopold_Kronecker

An Optimized Backward Pass

We installed Eigen to store the matrices as “Sparse Matrices” (as most of them were).

But it didn’t work for our data size, so we had to revisit the problem and simplified it a lot to a simple, fast and efficient 3 matrix multiplication:

$$\frac{\partial L}{\partial \theta} = (T_i X)^T G_I \qquad \frac{\partial L}{\partial X} = \sum_{i=1}^{K} T_i(L) G_I \theta_i^T$$

Global Pooling Layer

Global Pooling Layer

- Graph convolution layers summarize the features in the point cloud
- But these features are sensitive to the location
- Global Pooling layer:
  - Downsampling → Translation Invariance!
  - Robust to the number of points in the point cloud!

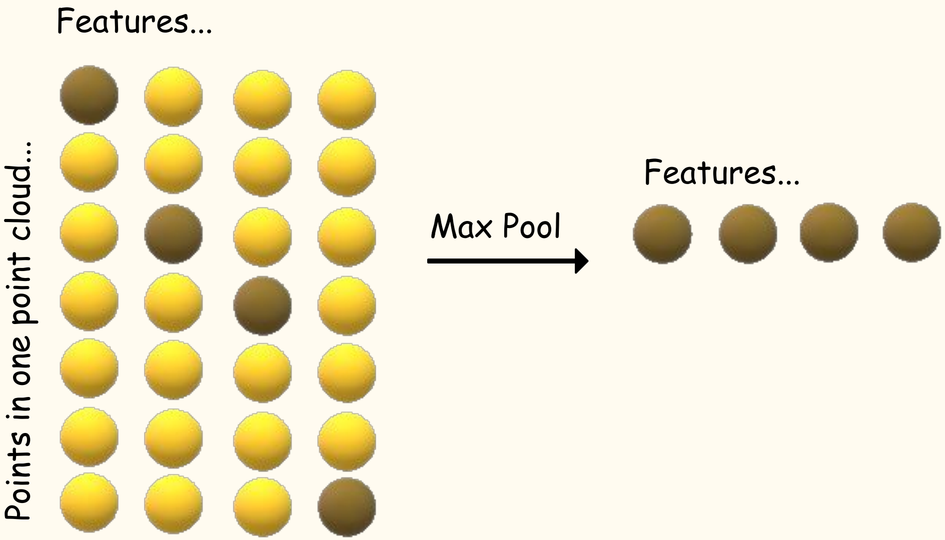
$$\text{Global Pooling} = \text{Global Max Pooling} + \text{Global Variance Pooling}$$

Global Max Pooling

- Down samples all points to one point per dimension

- Transforms N x D to 1 x D
    - N: Number of Points in one point cloud
    - D: Number of Graph Convolution Filters

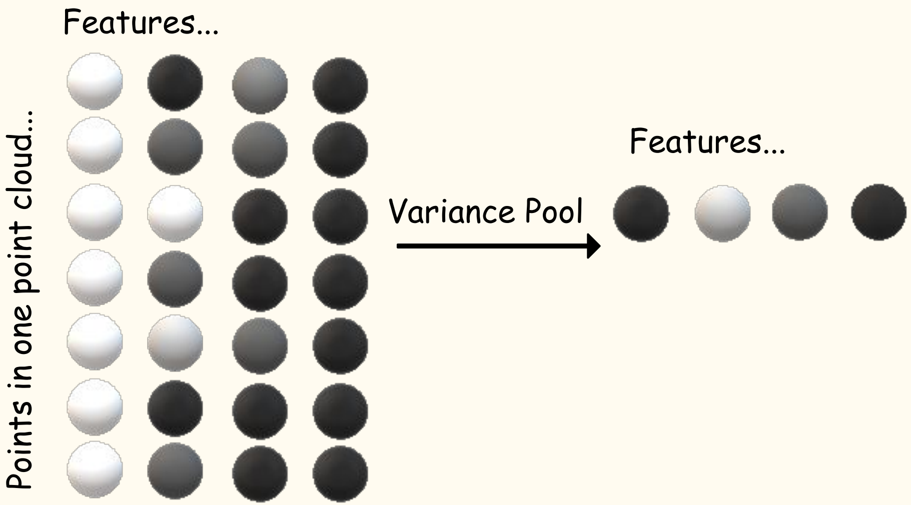
- Maximum value that summarizes the strongest activation/ presence of the feature captured by that filter



Global Variance Pooling

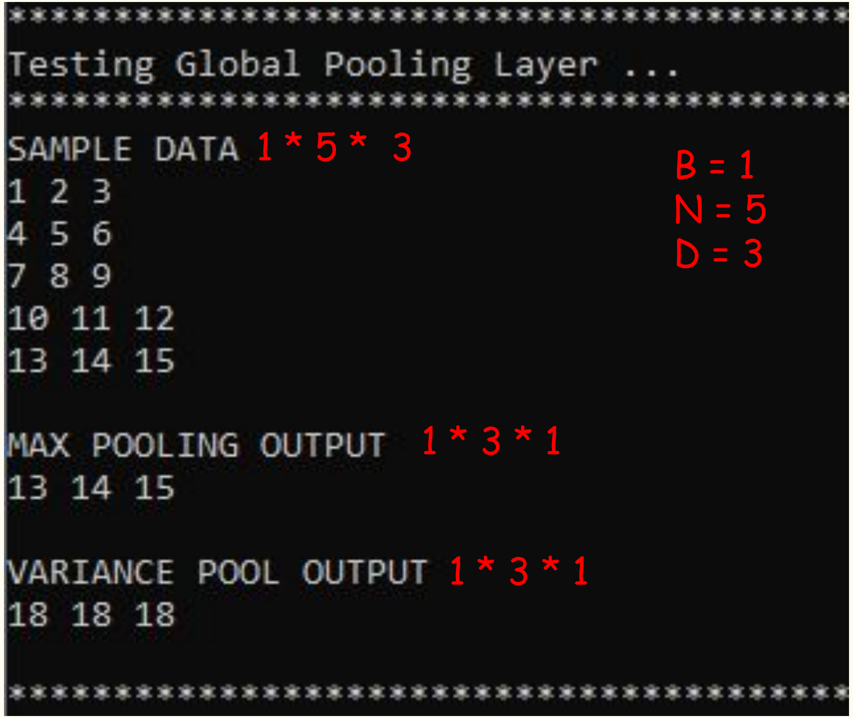
- Down samples all points to one per dimension
- Transforms N x D to 1 x D
  - N: Number of Points in one point cloud
  - D: Number of Graph Convolution Filters
- Single value per feature that summarizes the spread/variance captured by that filter

Black Color - 0
White Color - 1



Forward Pass

- Input: B x N x D
- Output: B x D x 2
- Max Pool:
    - Output Contribution: B x D x 1
    - Selects maximum point per feature dimension
- Variance Pool:
    - Output Contribution: B x D x 1
    - Calculate mean and then variance per feature dimension
- Save state for Backward Pass:
    - Argmax
    - Input
    - Mean

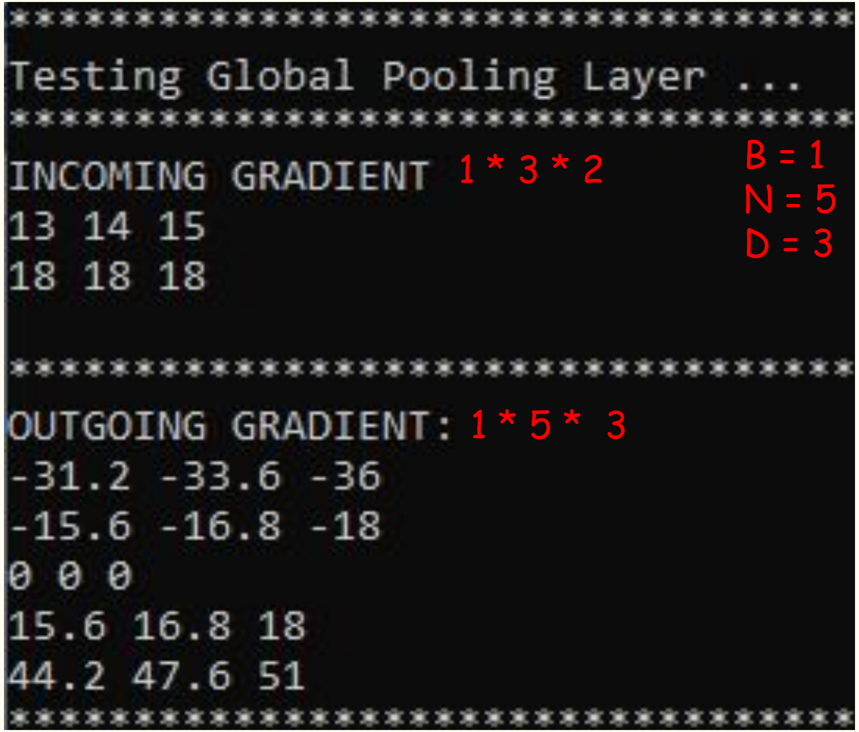


```
****************************************
Testing Global Pooling Layer ...
****************************************
SAMPLE DATA 1 * 5 * 3
1 2 3
4 5 6
7 8 9
10 11 12
13 14 15

MAX POOLING OUTPUT  1 * 3 * 1
13 14 15

VARIANCE POOL OUTPUT 1 * 3 * 1
18 18 18

****************************************
```

B = 1
N = 5
D = 3

Backward Pass

- Incoming Gradient: B x D x 2
- Outgoing Gradient: B x N x D
- Gradient Propagation:
  - Max Pool Gradient is distributed to B x N x D
  - Variance Pool Gradient is distributed B x N x D
  - Both of these are added and propagated backwards



```
****************************************
Testing Global Pooling Layer ...
****************************************
INCOMING GRADIENT
13 14 15
18 18 18

****************************************
OUTGOING GRADIENT:
-31.2 -33.6 -36
-15.6 -16.8 -18
0 0 0
15.6 16.8 18
44.2 47.6 51
****************************************
```

1 * 3 * 2

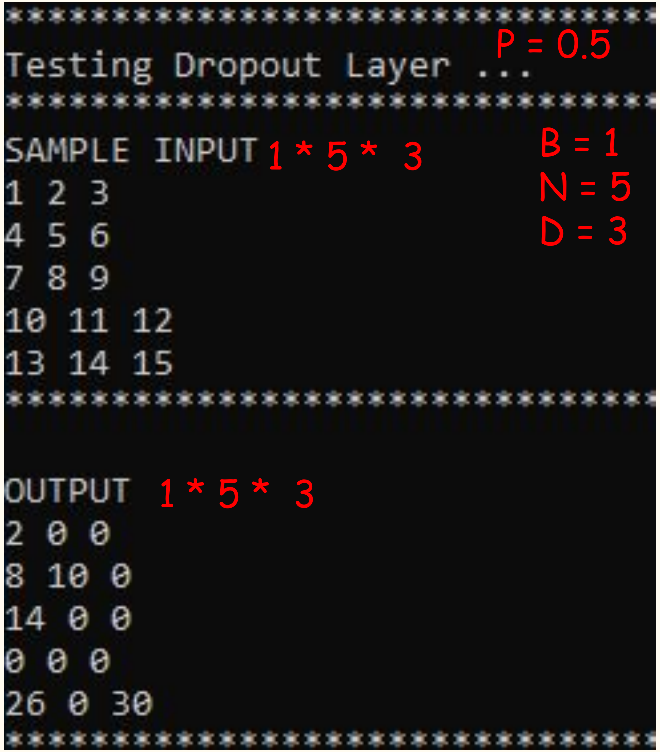
B = 1
N = 5
D = 3

1 * 5 * 3

Dropout Layer

Forward Pass

- Input: B x N x D
- Output: B x N x D
- Dropout Probability: Percentage of nodes to ignore/drop during training
- Zero out some nodes and scale the leftover activations
- Save state for Backward Pass:
  - Track nodes being dropped



```
*******************************
Testing Dropout Layer ...
*******************************
SAMPLE INPUT
1 2 3
4 5 6
7 8 9
10 11 12
13 14 15
*******************************


OUTPUT
2 0 0
8 10 0
14 0 0
0 0 0
26 0 30
*******************************
```

P = 0.5

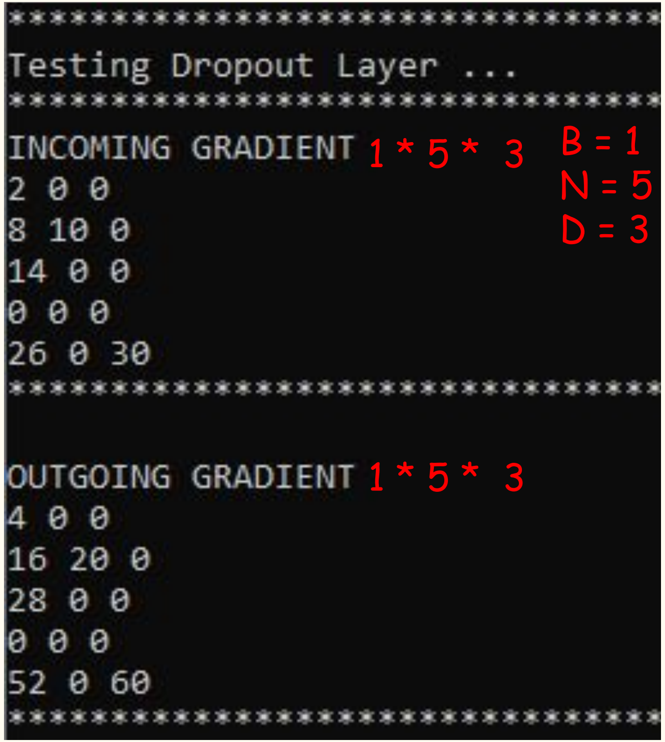
1 * 5 * 3

B = 1
N = 5
D = 3

1 * 5 * 3

Backward Pass

- Incoming Gradient: B x N x D
- Outgoing Gradient: B x N x D
- Gradient Propagation:
  - Very Simple!
  - Multiply the incoming gradient with saved state
  - Propagate the gradient only through the non zero nodes



```
***************************************
Testing Dropout Layer ...
***************************************
INCOMING GRADIENT 1 * 5 * 3   B = 1
2 0 0                         N = 5
8 10 0                        D = 3
14 0 0
0 0 0
26 0 30
***************************************


OUTGOING GRADIENT 1 * 5 * 3
4 0 0
16 20 0
28 0 0
0 0 0
52 0 60
***************************************
```

Basic Algebraic operations on GPU

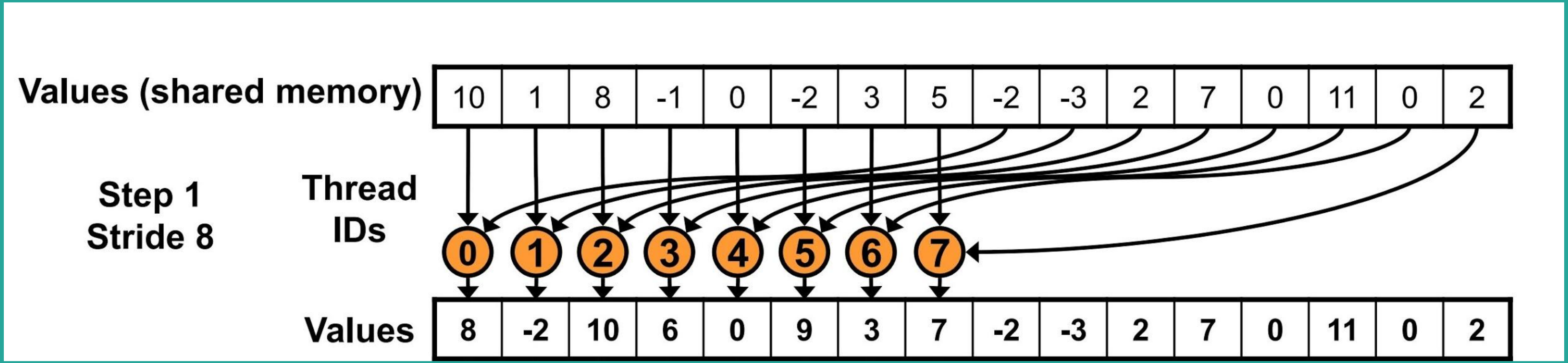
Operations on GPU

We have implemented the layer's basic operations on the GPU.

Given A,B $\in$ R^{mxn} and C $\in$ R^{nxn}

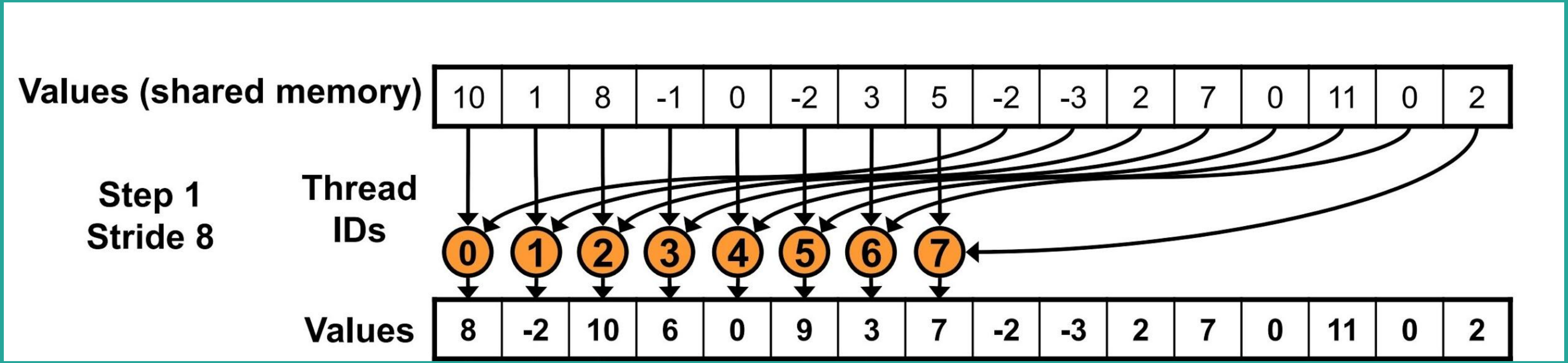
- $A \times B$
- $A \times B^T$
- A^T
- $\alpha A + \beta B$
- $C - I$
- $\max(A) \in R^{1xn}$
- $\mathrm{mean}(A) \in R^{1xn}$
- $\mathrm{var}(A) \in R^{1xn}$

Remember this?



https://developer.download.nvidia.com/assets/cuda/files/reduction.pdf

Remember this?

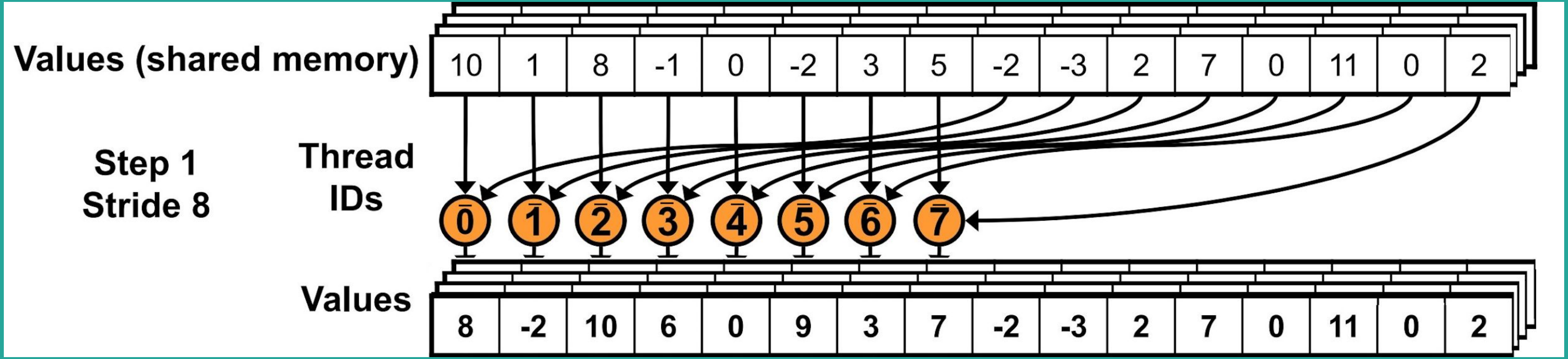


Step 1
Stride 8

Thread IDs

Values (shared memory)	10	1	8	-1	0	-2	3	5	-2	-3	2	7	0	11	0	2
Thread IDs	0	1	2	3	4	5	6	7								
Values	**8**	**-2**	**10**	**6**	**0**	**9**	**3**	**7**	**-2**	**-3**	**2**	**7**	**0**	**11**	**0**	**2**

Optimized 2D reduction



Values (shared memory)	10	1	8	-1	0	-2	3	5	-2	-3	2	7	0	11	0	2
Values	8	-2	10	6	0	9	3	7	-2	-3	2	7	0	11	0	2

What Optimizations?

This kernel works on 2 dimensional matrices and reduces along the columns.

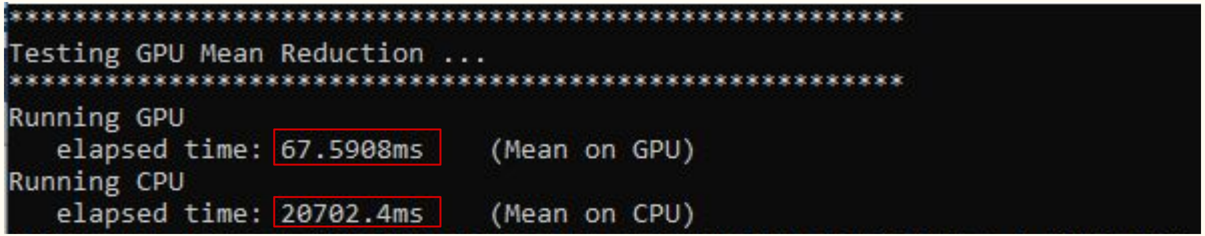
We use this to calculate n-dimensional means, variance and maximum values.

- Sequential Addressing
- Complete Unrolling
- First add during global load
- Multiple elements per thread
- Subsequent reduction also using the same kernel

A little more detail.

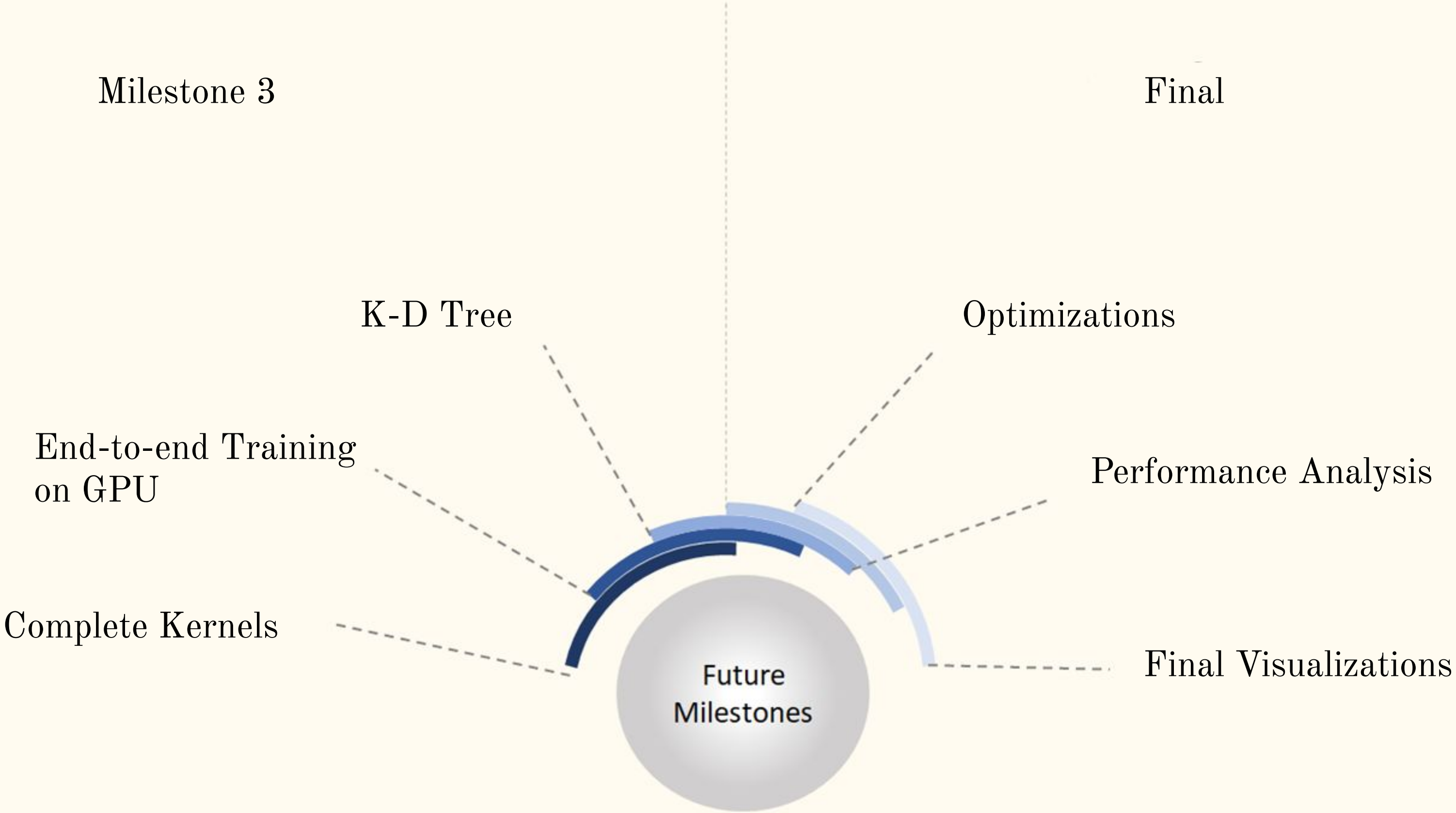
- Using FLOAT values limit us to only 12 features at a time.
- So we loop over 12 features.
  - Row major format greatly benefits this.
- That is all good, but how fast is it?

- In short, crazy fast...



```
***************************************************************
Testing GPU Mean Reduction ...
***************************************************************
Running GPU
   elapsed time: 67.5908ms     (Mean on GPU)
Running CPU
   elapsed time: 20702.4ms     (Mean on CPU)
```

Project Roadmap



Milestone 3

Final

K-D Tree

Optimizations

End-to-end Training on GPU

Performance Analysis

Complete Kernels

Final Visualizations

Future Milestones

Thank You