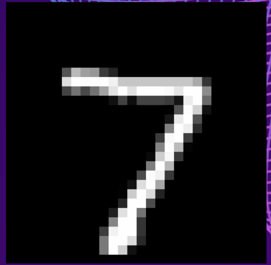
K-Nearest Neighbors (KNN) on GPUs

Alexander Ingare

Datasets

Name	Characteristics
MNIST	• Database of handwritten digits: 60k training, 10k testing • 10 classes, 28x28 pixel, anti-aliased (grayscaled) images
CIFAR-10	• Subset if the 80 Million Tiny Images dataset • 50k training images, 10k testing images • 10 classes, 32x32 pixel, RGB images
STL-10	• Inspired by CIFAR, but used more for unsupervised learning • 5k training images, 8k test images, 100k unlabeled images • 10 classes, 96x96 pixel, RGB images

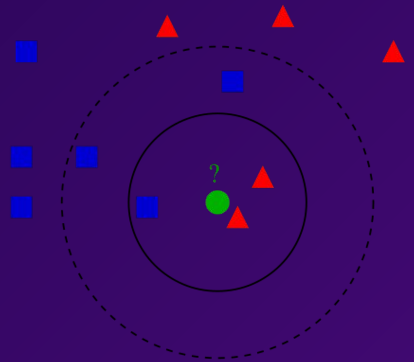


GPU Platforms

Name	Characteristics
NVIDIA Tesla P100	• Oldest and slowest architecture • Max (non-configurable) shared memory per SM (64 KB)
NVIDIA Tesla V100	• Newer architecture • Max dynamic shared memory size per SM (96 KB)
NVIDIA A100	• Newest and fastest architecture • Max dynamic shared memory size per SM (164 KB)

K-Nearest Neighbors - Overview & Features

- Non-parametric supervised learning method most often used for classification with the output being class membership
- Objects are classified by a plurality vote of its neighbors
- Closest neighbor classification is determined by Euclidean-distance calculations
- Training examples are vectors in a multidimensional feature space each with a class label
- The training phase for KNN is simply loading the dataset training data (feature vectors and class labels of the training samples)



High-Level Algorithm

1. Load and normalize image data into a vector with pixel values normalized within [0, 1]
2. Compare each test image to all training images
    a. Compute the Euclidean distances between the two images
3. Identify the K Nearest Neighbors
    a. Sort all training images by their distance to the test image (smallest to largest)
    b. Select the top K closest training images
4. Do a majority vote on the neighbor labels
    a. Return the label which appears most frequently amongst the nearest neighbors

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (y_i - x_i)^2}$$

Optimizations & Metrics

Optimizations
Shared memory
GPU optimized sorting (thrust)
H2D image copy batching

Metrics
Total & GPU Execution Time
Memory Allocation
KNN Accuracy

MNIST Results

K=5	P100	V100	A100
Total Execution Time (s)	48.2293	9.1296	5.7931
GPU Execution Time (s)	47.7577	8.4668	4.6876
Memory Usage (MB)	182	182	182
Accuracy (%)	96.88	96.88	96.88

CIFAR-10 Results

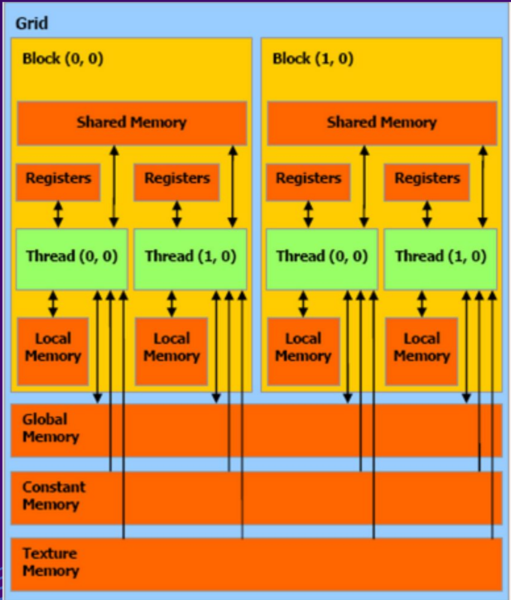
K=5	P100	V100	A100
Total Execution Time (s)	158.4035	38.3033	15.5068
GPU Execution Time (s)	157.2711	35.5580	13.1777
Memory Usage (MB)	588	588	588
Accuracy (%)	33.98	33.98	33.98

STL-10 Results

K=5	P100	V100	A100
Total Execution Time (s)	62.2983	43.6378	44.2068
GPU Execution Time (s)	61.3881	42.6537	43.2905
Memory Usage (MB)	530	530	530
Accuracy (%)	26.925	26.925	26.925

Setbacks

- Using shared memory for each test image, but test images are loaded linearly
  - No performance benefit since no computational reuse
  - Only a benefit if data is reused ***within the same block***
  - Only A100 GPUs can load a full STL-10 image! Tiling is needed
- Memory batching is overshadowed by sequential kernel executions



Future

- Test multi-GPU program execution (if SLURM magically has them available)
- Use shared memory tiling for training images instead of test images
- Use a different image batching implementation
- Compare results for varying K values
- Compare unoptimized results to fully (working) optimized results
- Get more fine grained memory metrics (bandwidth, cache usage, etc...)
- Explore FP16 (instead of FP32) to utilize V100 & A100 Tensor Cores

Q&A