Performance Analysis of K-Nearest Neighbors on NVIDIA GPU Platforms

Alexander Ingare

Northeastern University

EECE 5640 – High-Performance Computing

Professor David Kaeli

Introduction

The K-Nearest Neighbors (KNN) algorithm is a fundamental instance-based supervised learning method widely employed for classification and regression tasks. Unlike parametric algorithms, KNN does not explicitly build a predictive model but relies entirely on storing the training dataset. When a new instance needs classification, the algorithm searches for the closest data points (neighbors) within the training data, classifying the instance based on these neighbors.

Principle and Functionality

The fundamental principle behind KNN is that data points sharing similar characteristics or features typically belong to the same class or category. The algorithm leverages this assumption by calculating the proximity of data points using distance metrics, primarily the Euclidean distance, although other measures like Manhattan, Minkowski, or Cosine similarity can also be applied depending on the dataset and problem specifics.

Algorithmic Workflow

The KNN algorithm operates in the following stages:

1. **Data Preparation:**
    - All dataset features are normalized to ensure uniform scaling, which is crucial for distance-based computations. For image-based datasets, pixel intensities are typically scaled within a [0,1] range (floating-point normalization).
2. **Distance Computation:**
    - The distance between each test data point (e.g., a test image) and all data points in the training set is computed using the chosen distance metric (Euclidean distance) which involves treating images as flattened vectors where each pixel is a dimension.

$$d(x,\ y)\ =\ \sqrt{\sum_{i=1}^{n}\left(y_{i}-x_{i}\right)^{2}}$$

3. **Sorting and Neighbor Selection:**
    - Once distances are computed, they are sorted in ascending order, and the top k nearest data points are selected as neighbors. The hyperparameter k determines how many neighbors are used for the classification.
4. **Class Assignment (Majority Voting):**
    - The most frequent class label among the k selected neighbors is assigned as the predicted class label for the test instance. If there is a tie, methods such as

weighted voting (giving closer neighbors more influence) can be applied. However, in a tie for this project, the closest neighbor is simply the first label with the maximum count that gets detected.

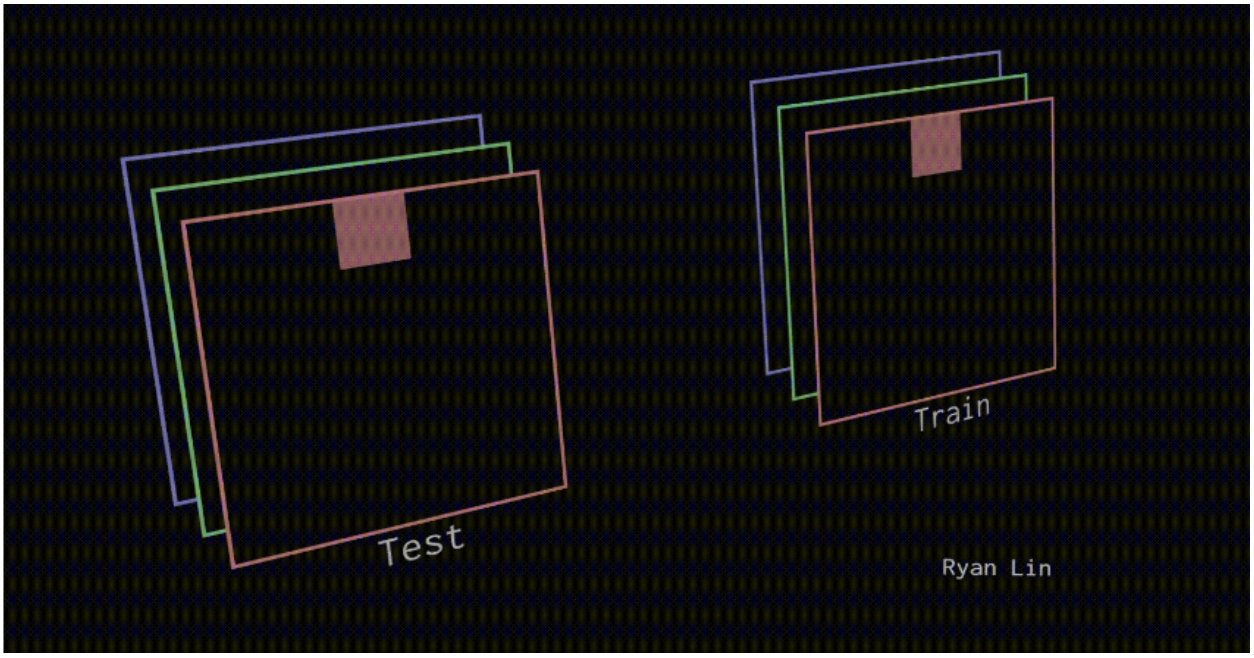

Figure 1. Euclidean distance comparisons for testing and training images [source]

Strengths of KNN

- **Simplicity:** KNN is straightforward to implement, making it accessible for practical use and teaching purposes.
- **Adaptability:** It naturally adapts to multi-class problems and can be used effectively with multi-dimensional data.
- **No Training Phase:** Unlike other machine learning algorithms that require training or fitting, KNN simply stores training data, eliminating the need for explicit model training.

Limitations and Computational Challenges

- **Computational Intensity:** As the dataset size grows, calculating distances between every test instance and each training data point significantly increases computational requirements.
- **Curse of Dimensionality:** Performance degrades as the number of features (dimensions) increases because distances between points tend to become uniform, making neighbor differentiation challenging.
- **Sensitivity to Outliers:** KNN can be heavily influenced by outliers, as the decision boundary directly depends on the closest neighbors.

Datasets

To robustly assess performance, three datasets of varying complexity were employed:

Name	Training Images	Testing Images	Image Size	Characteristics
MNIST	60,000	10,000	28x28 Pixels	Grayscale images, 10 classes
CIFAR-10	50,000	10,000	32x32 Pixels	RGB images, 10 classes
STL-10	5,000	8,000	96x96 Pixels	RGB images, 10 classes

Table 1. Image classification datasets

The STL-10 dataset, while it contains 113,000 images, much of it is unlabeled. As such, this project used just the labeled training and testing images from the dataset (total of 13,000 images).

Originally, the project proposal included the classic Iris Dataset, a small dataset of 150 images of plants with the physical characteristics of the plants (sepal/petal length and width) acting as the high-dimensional feature vectors. However, to maintain consistency with the algorithm being implemented as well as to introduce more computational complexity, it was swapped out with the STL-10 dataset for the final project implementation.

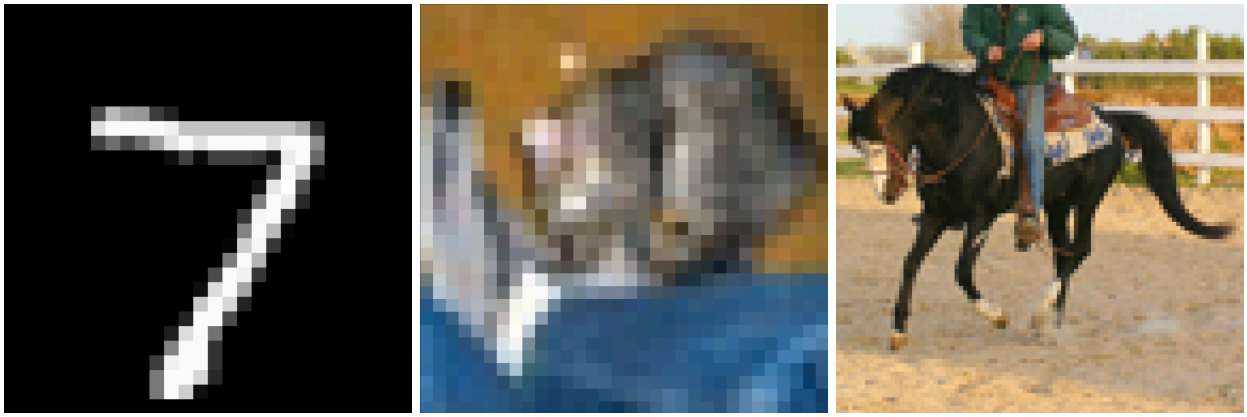

Figure 2. Image dataset example images (from left to right, MNIST, CIFAR-10, STL-10)

GPU Platforms

Experiments were conducted on three GPU architectures:

GPU Model	Characteristics
NVIDIA Tesla P100	Older architecture, 64 KB shared memory per SM
NVIDIA Tesla V100	Improved architecture, 96 KB dynamic shared memory per SM
NVIDIA A100	Latest architecture, 164 KB dynamic shared memory per SM

Table 2. GPU platforms experimented with

Deliverables By Grade

Based on the project proposal, the expected grades and associated criteria are as follows:

Grade	Criteria
A	1 workload evaluated, 3 different inputs used on 2 different platforms, all results reported and analyzed thoroughly in the project writeup
A-	1 workload evaluated, 3 different inputs used on 2 different platforms, all results reported but little analysis of data
B+	1 workload evaluated, 2 different inputs used on 2 different platforms, all results reported and analyzed thoroughly
B	1 workload evaluated, 1 input used on 2 different platforms, all results reported and analyzed thoroughly
C	1 workload evaluated, 1 input used on 1 platform, all results reported and analyzed thoroughly
F	Submission lower than above criteria

Table 3. Grade expectations

Optimization Strategies

Several performance optimizations were implemented to exploit GPU capabilities:

- **Shared Memory Utilization**: Leveraged GPU shared memory to reduce global memory access latency during distance computations. The shared memory loads a singular training image in the GPU kernel.
- **GPU-Optimized Sorting (Thrust)**: Employed NVIDIA's Thrust library for efficient sorting of distances.
- **Batching**: Minimized host-to-device memory transfers through batched image uploads. For this type of batching, multiple testing images are sent to the kernel at the same time to be compared against the singular shared memory training image.

Metrics Evaluated

Performance was analyzed based on:

- Total Execution Time (overall run-time including CPU overhead)
- GPU Execution Time (kernel execution)
- Memory Utilization (GPU memory allocation)
- Classification Accuracy

Experimental Results

A100 GPU Varied Block Size Results

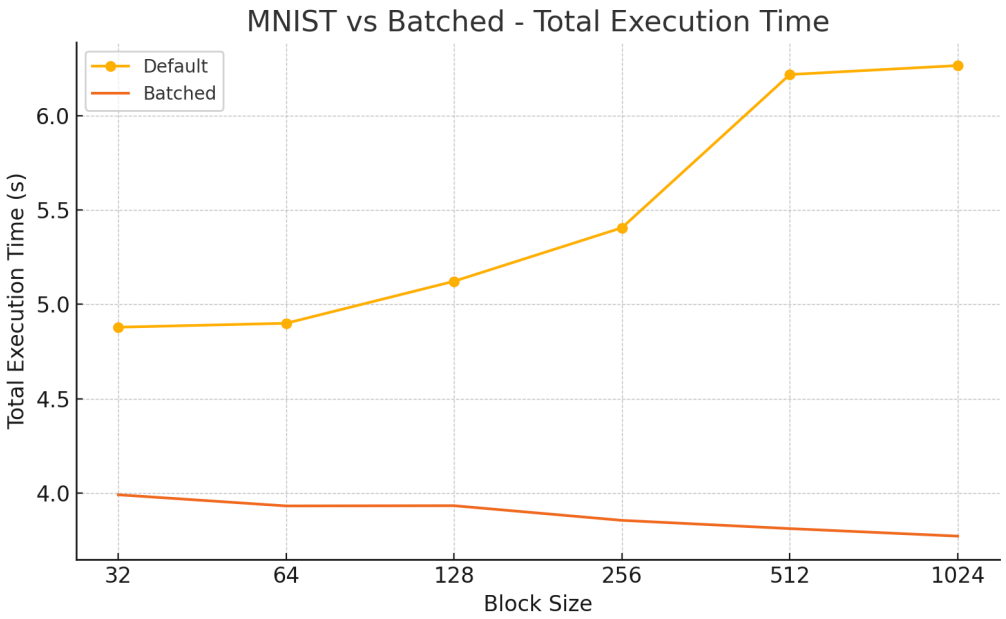

Figure 3. MNIST dataset implementations' total execution time for varying block sizes

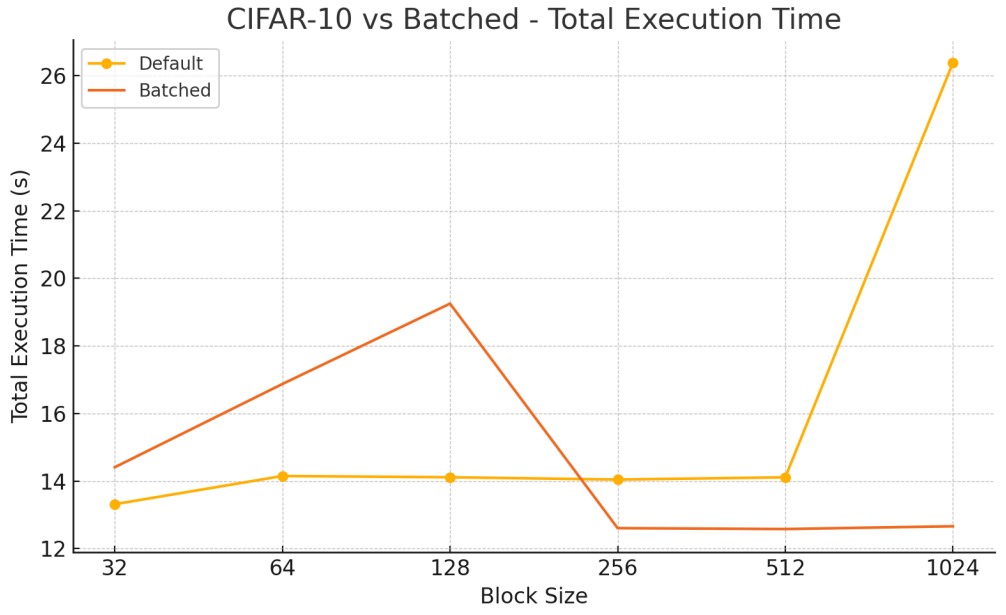

Figure 4. CIFAR-10 dataset implementations' total execution time for varying block sizes

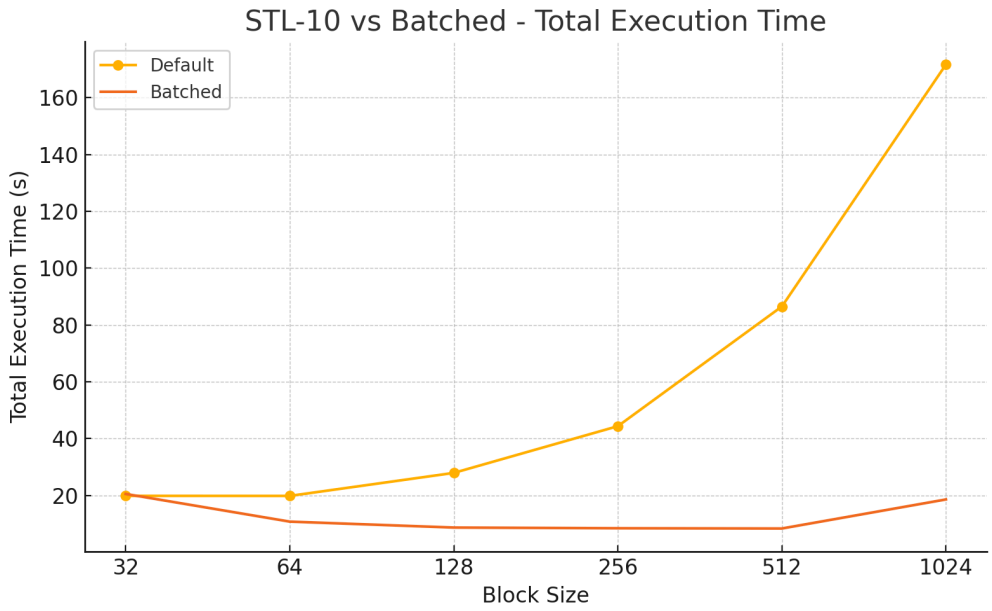

Figure 5. STL-10 dataset implementations' total execution time for varying block sizes

P100 GPU Varied Block Size Results

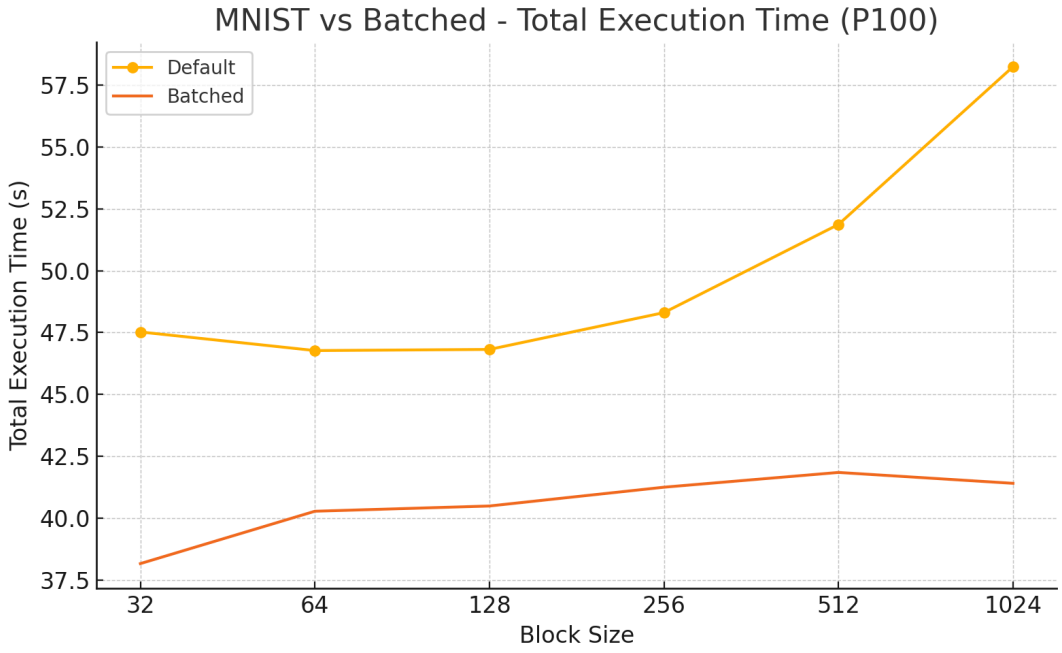

Figure 6. MNIST dataset implementations' total execution time for varying block sizes

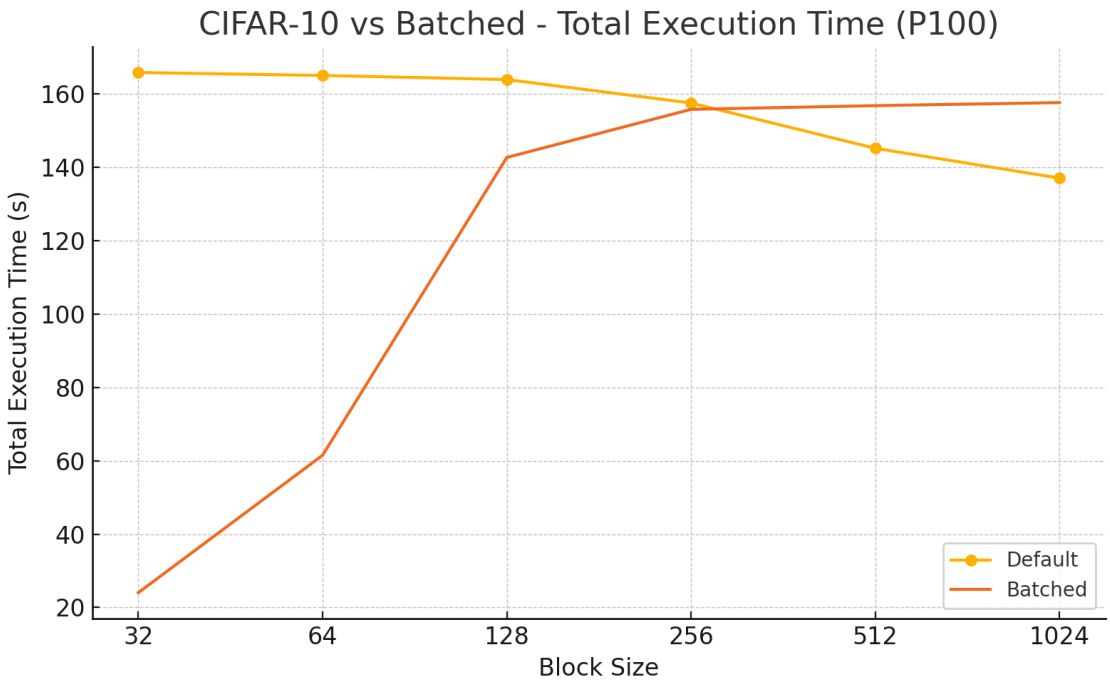

Figure 7. CIFAR-10 dataset implementations' total execution time for varying block sizes

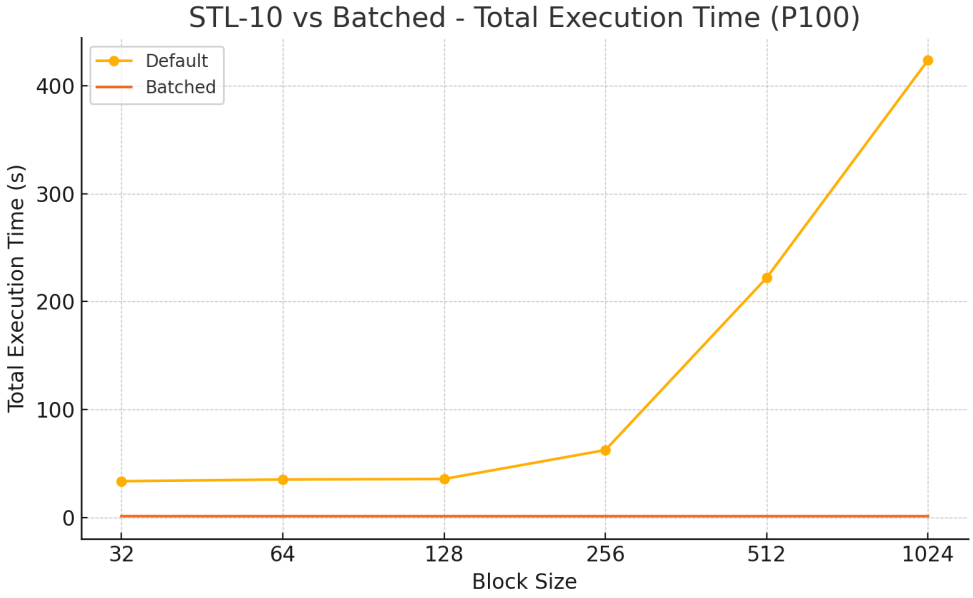

Figure 8. STL-10 dataset implementations' total execution time for varying block sizes

A100/P100 GPU Varied K-Neighbors Results

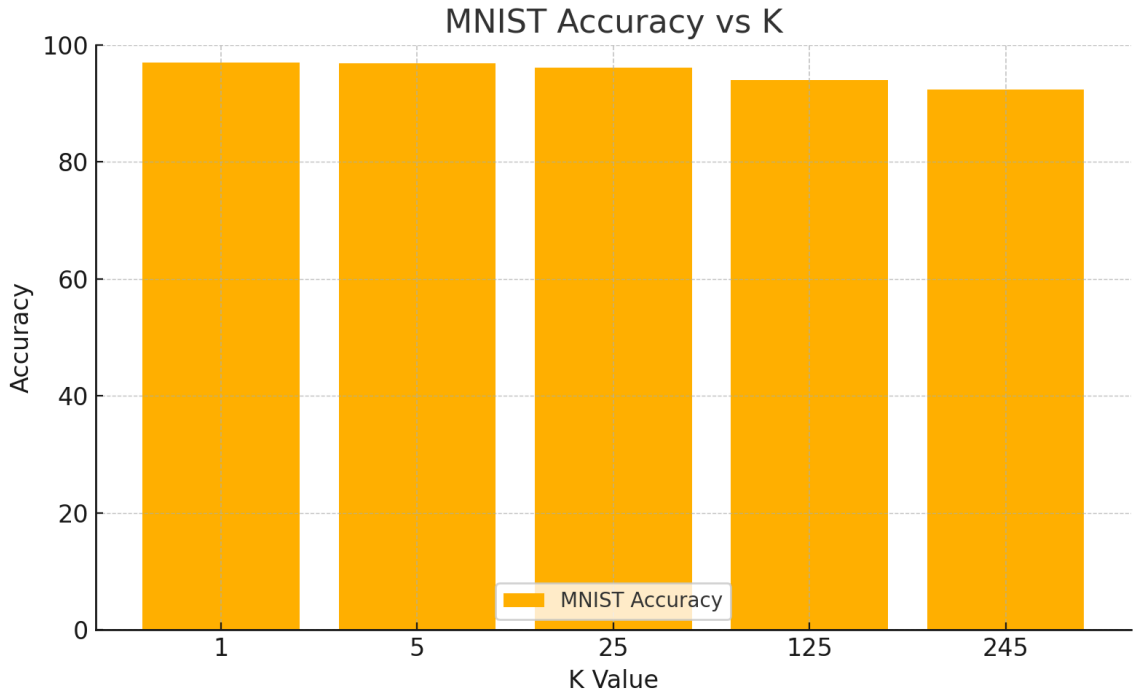

Figure 9. MNIST accuracy as number of neighbors K varies

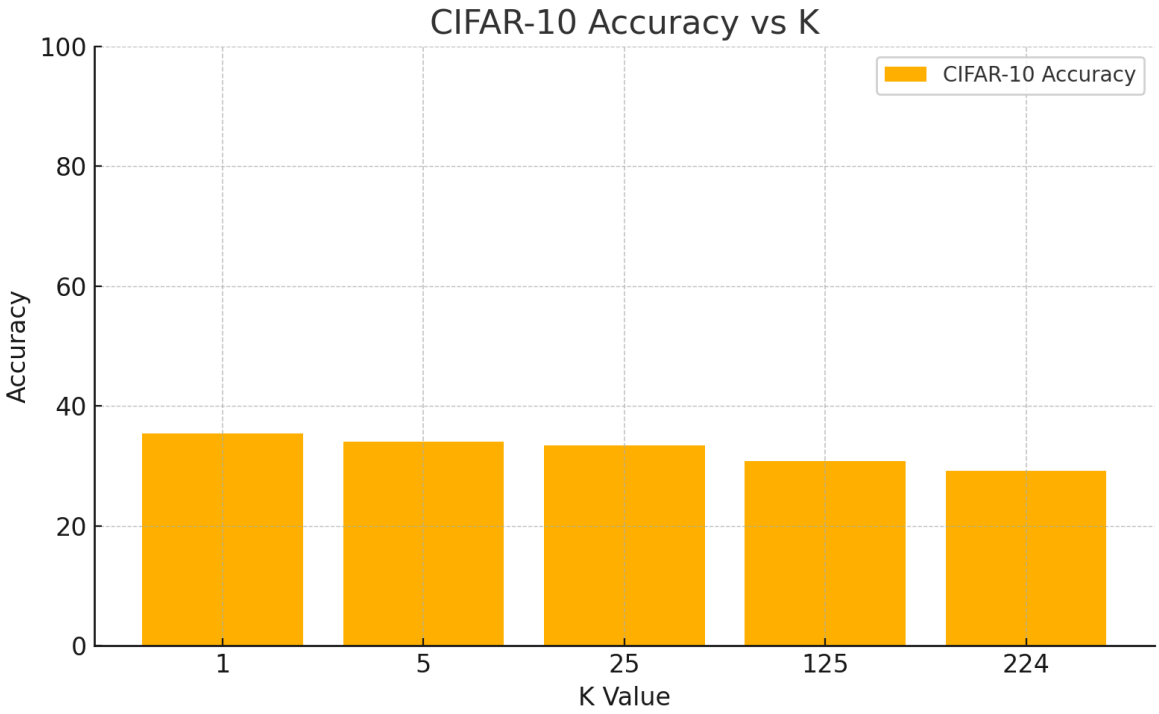

Figure 10. CIFAR-10 accuracy as number of neighbors K varies

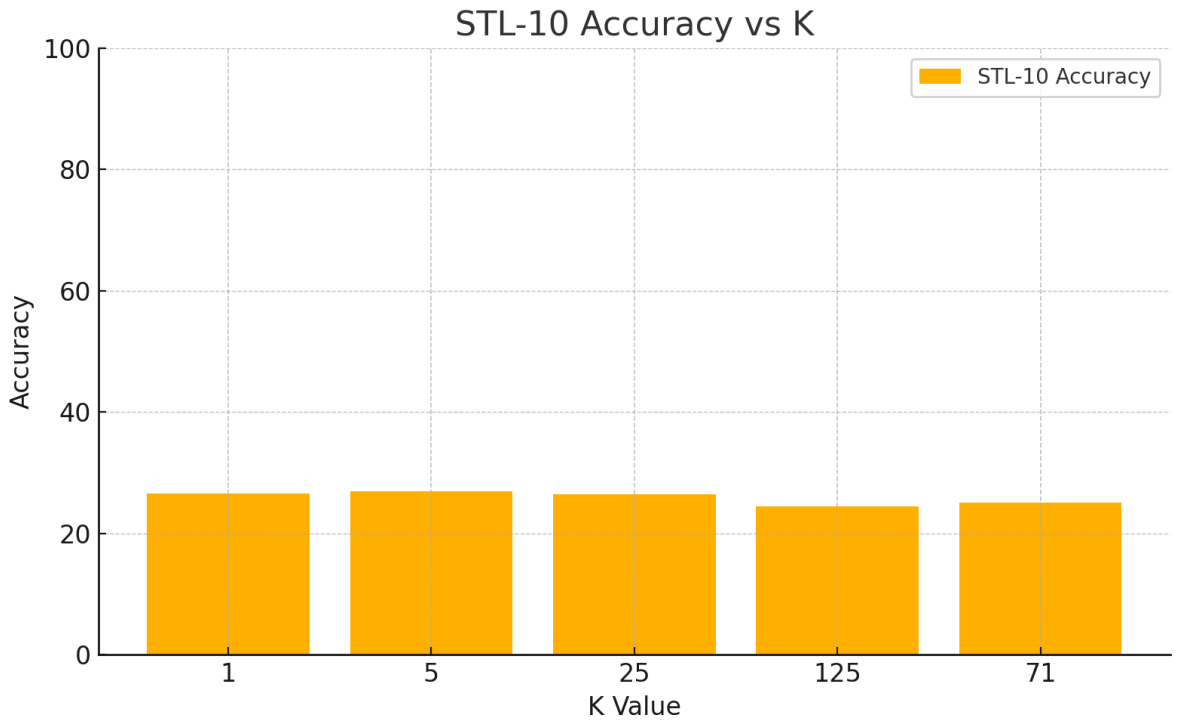

Figure 11. STL-10 accuracy as number of neighbors K varies

See Appendix, Figs [12-15] for the full CSV data results for the A100 and P100 data.

Additional data was gathered for the V100 GPU, but I was not able to gather the full data (compared to the A100 and P100) since I was unable to allocate any V100 GPUs due to other students using them. This is fine as the criteria stipulates two platforms, and I had hoped to collect data for all three.

Dataset	K	GPU Type	GPU Count	Total Execution Time (s)	GPU Execution Time (s)	Memory Usage (MB)	Accuracy (%)
MNIST	5	Tesla V100-SXM2-32GB	1	9.1296	8.4668	182	96.88
CIFAR-10	5	Tesla V100-SXM2-32GB	1	38.3033	35.558	588	33.98
STL-10	5	Tesla V100-SXM2-32GB	1	43.6067	42.5947	530	26.93

Table 4. V100 Default, non-batched performance results

Discussion & Analysis

Early on, this project faced several challenges with implementing these optimizations. One of the first problems arose when the shared memory implementation kernel showed no performance improvement over the baseline kernel. This is because the first implementation ran each test image sequentially with the GPU kernel and also loaded the test image into shared memory. However, this provides no significant data reuse since the global memory data accesses for the entire training image set eclipsed whatever “performance improvements” there might have been

for loading the test image into shared memory. It's also important to note that shared memory is local only to the GPU block it was stored in, so each thread loading the test image into shared memory actually only ended up reading from it once (further proving no data reuse of shared memory data). Similarly, the data batching algorithm ended up not actually batch processing test images at all, and actually was processing the test images sequentially. These findings required a redo of the implementation of these optimizations completely.

The second implementation of shared memory included loading the training images into shared memory instead of the testing images. This is the second obstacle this project ran into regarding performance optimizations, as it was around this time that the STL-10 dataset was used going forward. Due to the large image size of an STL-10 image (~110 KB), a full image could not be loaded into shared memory for P100 and V100 GPUs due to their maximum shared memory size per block (64 KB and 96 KB, respectively. See Table 2). This could be resolved with a tiling implementation to eventually compute the distances of the full images, however it was found that it was vastly slower than the standard baseline kernel implementation (see sample output below for MNIST dataset processing with a tiled-shared memory approach):

```
KNN: Final accuracy: 96.88%
KNN: GPU execution time: 1453.15 seconds
KNN: GPU memory usage: 179.732 MB
MNIST: Total dataset processing time: 1454.17 seconds
MNIST: GPU-only execution time: 1453.15 seconds
MNIST: Non-GPU overhead time: 1.01369 seconds
```

Finally, the third implementation was a collaborative approach that took the idea of batching test images and loading training images into shared memory to exploit data reuse. In this implementation, the kernel loads one training image into shared memory and batches all testing images against the training image stored in shared memory. Since each block now handles one training image, the data reuse happens when each thread of the block is allocated to its own test image to directly compare the distances of the images. This was the final implementation for the respective optimizations in this project, but important to note is that the batched-shared memory implementation would not work for the STL-10 dataset on P100 and V100 GPUs since it is still loading a full training image into each block's shared memory.

With regards to the thrust sorting optimization, the baseline vs batched implementation both use it in their execution. This GPU sorting algorithm really can't be improved upon and as such is close to being perfectly optimized for the sorting portion of KNN.

Taking a look at the experimental results, there was a trend for all datasets, albeit very slight, where the accuracy of the classification decreased as the value of K - the number of nearest neighbors to consider - increased. I tested the values ranging from [1, 125] as well as a square root heuristic based on the number of training images in the dataset for values of K. More

importantly, however, is the effect that the shared memory and batching combination had on the performance of the algorithm. Overwhelmingly, save for a few instances for CIFAR-10, batching provided a substantial improvement to the program execution time over the default baseline. Important to note is that the batching does not work for the STL-10 dataset for P100 and V100 GPUs (as seen in Figure 8 by the zero line) due to the fact that loading a singular STL-10 image is too large for the blocks' shared memory. I also believe this is the reason why STL-10 performed so well on the A100 for batching because it was able to load the full training image into shared memory and utilize as many threads as possible to compute the distances between the batched testing images and the training image.

Conclusion

This project successfully demonstrated the implementation and optimization of the K-Nearest Neighbors (KNN) algorithm on GPU platforms with increasing architectural capabilities. By experimenting with three datasets of varying complexity and three GPU models (P100, V100, and A100), the performance impact of shared memory size, image dimensionality, and batching strategies was systematically analyzed. Optimizations using shared memory and batched computation significantly improved execution time on more modern architectures, particularly when working with smaller datasets like MNIST. However, for higher-resolution RGB datasets like STL-10, memory limitations became a constraining factor even on advanced GPUs, highlighting the need for more advanced memory tiling strategies. This project underscores the importance of tailoring GPU resource usage to both dataset characteristics and hardware constraints, and it provides a strong foundation for further exploration of GPU-accelerated distance-based learning algorithms catered to their respective hardware platforms.

All source code used in this project is available at my GitHub repository

Appendix

	A	B	C	D	E	F	G
1	Dataset	K	BlockSize	TotalExecutionTime(s)	GPUExecutionTime(s)	MemoryUsage(MB)	Accuracy(%)
2	MNIST	5	32	4.879	4.1234	182	96.88
3	MNIST	5	64	4.8998	4.1213	182	96.88
4	MNIST	5	128	5.1227	4.296	182	96.88
5	MNIST	5	256	5.4066	4.6918	182	96.88
6	MNIST	5	512	6.2193	5.5042	182	96.88
7	MNIST	5	1024	6.2669	5.3991	182	96.88
8	MNIST-Batched	5	32	3.9901	3.8071	2502	96.88
9	MNIST-Batched	5	64	3.9309	3.7482	2502	96.88
10	MNIST-Batched	5	128	3.9321	3.7473	2502	96.88
11	MNIST-Batched	5	256	3.8544	3.6738	2502	96.88
12	MNIST-Batched	5	512	3.8107	3.6269	2502	96.88
13	MNIST-Batched	5	1024	3.7708	3.5844	2502	96.88
14	CIFAR-10	5	32	13.3131	12.5874	588	33.98
15	CIFAR-10	5	64	14.1479	13.2948	588	33.98
16	CIFAR-10	5	128	14.1109	13.2795	588	33.98
17	CIFAR-10	5	256	14.0427	13.1705	588	33.98
18	CIFAR-10	5	512	14.1089	13.3997	588	33.98
19	CIFAR-10	5	1024	26.3773	25.4972	588	33.98
20	CIFAR-10-Batched	5	32	14.407	13.6961	2614	33.98
21	CIFAR-10-Batched	5	64	16.8714	16.1566	2614	33.98
22	CIFAR-10-Batched	5	128	19.2503	18.4129	2614	33.98
23	CIFAR-10-Batched	5	256	12.6046	11.7335	2614	33.98
24	CIFAR-10-Batched	5	512	12.5782	11.7808	2614	33.98
25	CIFAR-10-Batched	5	1024	12.6601	11.722	2614	33.98
26	STL-10	5	32	19.8621	19.1822	530	26.925
27	STL-10	5	64	19.845	19.1557	530	26.925
28	STL-10	5	128	27.9416	27.2716	530	26.925
29	STL-10	5	256	44.3848	43.702	530	26.925
30	STL-10	5	512	86.522	85.8188	530	26.925
31	STL-10	5	1024	171.6768	170.9781	530	26.925
32	STL-10-Batched	5	32	20.5277	19.8106	1528	26.925
33	STL-10-Batched	5	64	10.7904	10.0615	1528	26.925
34	STL-10-Batched	5	128	8.7008	7.9855	1528	26.925
35	STL-10-Batched	5	256	8.4617	7.7422	1528	26.925
36	STL-10-Batched	5	512	8.3733	7.6438	1528	26.925
37	STL-10-Batched	5	1024	18.5979	17.8743	1528	26.925

Figure 12. A100 Full CSV for Varied Block Sizes

	A	B	C	D	E	F	G
1	Dataset	K	BlockSize	TotalExecutionTime(s)	GPUExecutionTime(s)	MemoryUsage(MB)	Accuracy(%)
2	MNIST	1	256	5.4654	4.6812	182	96.91
3	MNIST	5	256	5.3674	4.6966	182	96.88
4	MNIST	25	256	5.6911	4.7048	182	96.09
5	MNIST	125	256	5.4167	4.685	182	93.92
6	MNIST	245	256	5.593	4.7003	182	92.38
7	MNIST-Batched	1	256	3.8975	3.7158	2502	96.91
8	MNIST-Batched	5	256	3.8464	3.6658	2502	96.88
9	MNIST-Batched	25	256	3.9379	3.7596	2502	96.09
10	MNIST-Batched	125	256	4.4169	4.2357	2508	93.92
11	MNIST-Batched	245	256	4.9837	4.7996	2512	92.38
12	CIFAR-10	1	256	13.8542	13.1462	588	35.39
13	CIFAR-10	5	256	13.8752	13.1516	588	33.98
14	CIFAR-10	25	256	14.0603	13.1486	588	33.47
15	CIFAR-10	125	256	13.8564	13.1731	588	30.78
16	CIFAR-10	224	256	14.135	13.1641	588	29.23
17	CIFAR-10-Batched	1	256	12.4434	11.729	2614	35.39
18	CIFAR-10-Batched	5	256	12.5231	11.7326	2614	33.98
19	CIFAR-10-Batched	25	256	12.5471	11.8331	2614	33.47
20	CIFAR-10-Batched	125	256	13.2289	12.2961	2620	30.78
21	CIFAR-10-Batched	224	256	13.6795	12.7449	2624	29.23
22	STL-10	1	256	44.3654	43.691	530	26.5125
23	STL-10	5	256	44.4023	43.7106	530	26.925
24	STL-10	25	256	44.3876	43.7064	530	26.425
25	STL-10	125	256	44.3738	43.711	530	24.4125
26	STL-10	71	256	44.4355	43.6988	530	25.1
27	STL-10-Batched	1	256	8.4599	7.7363	1528	26.5125
28	STL-10-Batched	5	256	8.4616	7.7434	1528	26.925
29	STL-10-Batched	25	256	8.5146	7.7923	1528	26.425
30	STL-10-Batched	125	256	8.762	8.0386	1532	24.4125
31	STL-10-Batched	71	256	8.6323	7.9096	1532	25.1

Figure 13. A100 Full CSV for Varied K Values

	A	B	C	D	E	F	G
1	Dataset	K	BlockSize	TotalExecutionTime(s)	GPUExecutionTime(s)	MemoryUsage(MB)	Accuracy(%)
2	MNIST	5	32	47.5241	46.7054	182	96.88
3	MNIST	5	64	46.7733	46.0816	182	96.88
4	MNIST	5	128	46.819	46.2993	182	96.88
5	MNIST	5	256	48.3054	47.8363	182	96.88
6	MNIST	5	512	51.8654	50.89	182	96.88
7	MNIST	5	1024	58.2333	57.7306	182	96.88
8	MNIST-Batched	5	32	38.1705	37.8225	2502	96.88
9	MNIST-Batched	5	64	40.2848	39.9414	2502	96.88
10	MNIST-Batched	5	128	40.4965	40.1522	2502	96.88
11	MNIST-Batched	5	256	41.2553	40.9132	2502	96.88
12	MNIST-Batched	5	512	41.8516	41.3396	2502	96.88
13	MNIST-Batched	5	1024	41.4125	41.0711	2502	96.88
14	CIFAR-10	5	32	165.8591	162.8515	588	33.98
15	CIFAR-10	5	64	165.0601	163.6848	588	33.98
16	CIFAR-10	5	128	163.9482	162.5604	588	33.98
17	CIFAR-10	5	256	157.5499	156.1884	588	33.98
18	CIFAR-10	5	512	145.2004	143.6001	588	33.98
19	CIFAR-10	5	1024	137.0615	135.7013	588	33.98
20	CIFAR-10-Batched	5	32	24.0371	22.5856	2614	33.98
21	CIFAR-10-Batched	5	64	61.5055	60.0635	2614	33.98
22	CIFAR-10-Batched	5	128	142.7008	141.2654	2614	33.98
23	CIFAR-10-Batched	5	256	155.8199	154.3791	2614	33.98
24	CIFAR-10-Batched	5	512	156.8298	155.3897	2614	33.98
25	CIFAR-10-Batched	5	1024	157.6587	156.2169	2614	33.98
26	STL-10	5	32	33.8042	30.0282	530	26.925
27	STL-10	5	64	35.4356	34.0811	530	26.925
28	STL-10	5	128	35.8706	34.5391	530	26.925
29	STL-10	5	256	62.6496	61.3231	530	26.925
30	STL-10	5	512	222.0915	220.7385	530	26.925
31	STL-10	5	1024	423.4384	422.0667	530	26.925
32	STL-10-Batched	5	32	1.4715	0	1528	0
33	STL-10-Batched	5	64	1.4469	0	1528	0
34	STL-10-Batched	5	128	1.4459	0	1528	0
35	STL-10-Batched	5	256	1.4483	0	1528	0
36	STL-10-Batched	5	512	1.4513	0	1528	0
37	STL-10-Batched	5	1024	1.4504	0	1528	0

Figure 14. P100 Full CSV for Varied Block Sizes

	A	B	C	D	E	F	G
1	Dataset	K	BlockSize	TotalExecutionTime(s)	GPUExecutionTime(s)	MemoryUsage(MB)	Accuracy(%)
2	MNIST	1	256	48.3231	47.8217	182	96.91
3	MNIST	5	256	48.3483	47.8319	182	96.88
4	MNIST	25	256	48.2906	47.8206	182	96.09
5	MNIST	125	256	48.3394	47.8381	182	93.92
6	MNIST	245	256	48.3885	47.8477	182	92.38
7	MNIST-Batched	1	256	40.9517	40.6105	2502	96.91
8	MNIST-Batched	5	256	40.9712	40.6291	2502	96.88
9	MNIST-Batched	25	256	41.1086	40.77	2502	96.09
10	MNIST-Batched	125	256	41.6201	41.2796	2508	93.92
11	MNIST-Batched	245	256	42.283	41.9365	2512	92.38
12	CIFAR-10	1	256	157.6026	156.2164	588	35.39
13	CIFAR-10	5	256	157.5693	156.211	588	33.98
14	CIFAR-10	25	256	157.5654	156.206	588	33.47
15	CIFAR-10	125	256	157.6528	156.2903	588	30.78
16	CIFAR-10	224	256	157.701	156.314	588	29.23
17	CIFAR-10-Batched	1	256	155.8257	154.3873	2614	35.39
18	CIFAR-10-Batched	5	256	156.0516	154.6097	2614	33.98
19	CIFAR-10-Batched	25	256	156.0625	154.6212	2614	33.47
20	CIFAR-10-Batched	125	256	156.8999	155.4592	2620	30.78
21	CIFAR-10-Batched	224	256	157.6106	156.1743	2624	29.23
22	STL-10	1	256	62.6393	61.3148	530	26.5125
23	STL-10	5	256	62.6934	61.3231	530	26.925
24	STL-10	25	256	62.7023	61.3473	530	26.425
25	STL-10	125	256	62.6708	61.3023	530	24.4125
26	STL-10	71	256	62.6636	61.3142	530	25.1
27	STL-10-Batched	1	256	1.4478	0	1528	0
28	STL-10-Batched	5	256	1.4488	0	1528	0
29	STL-10-Batched	25	256	1.4486	0	1528	0
30	STL-10-Batched	125	256	1.444	0	1532	0
31	STL-10-Batched	71	256	1.4529	0	1532	0

Figure 15. P100 Full CSV for Varied K Values