Walchand College of Engineering, Sangli
Department of CSE

Seminar on "Scaling Up Machine Learning and Deep Learning : Parallel Approach with CUDA and OpenMP"

Roll No.: 2018MTECSCO007

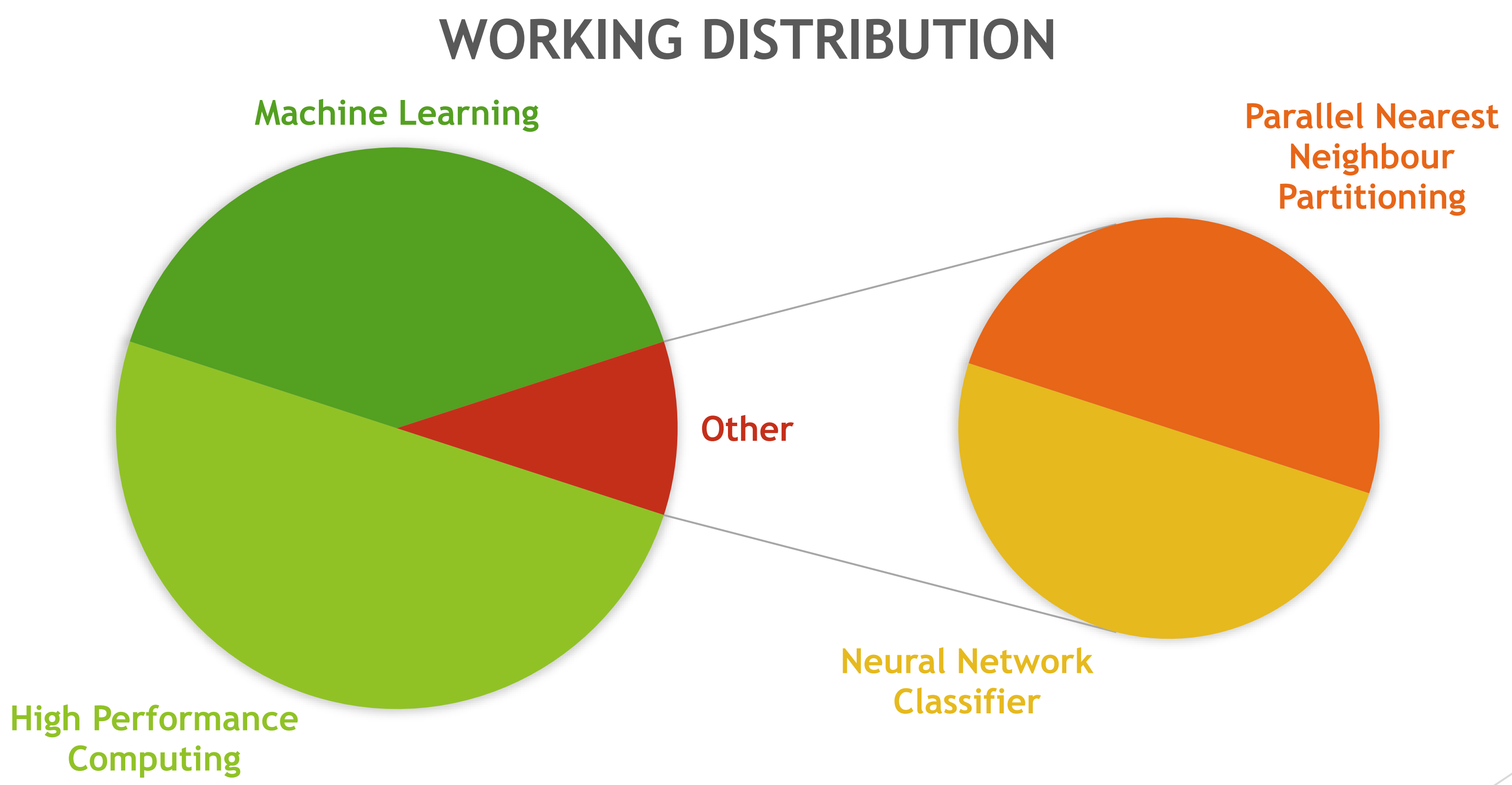
Name : Veersen Vijay Jadhav

Contents

Research Area

- Research Area : Applying High Performance Computing / Parallel Computing to Machine Learning Algorithms.
- Paper Title : Accelerating nearest neighbour partitioning neural network classifier based on CUDA.
- Authors : Lin Wang, Ajith Abraham, Meihui Li
- Publisher : Elsevier
- Journal : Engineering Applications of Artificial Intelligence
- Publication Year : 2018

Technology


WORKING DISTRIBUTION
Machine Learning
Parallel Nearest Neighbour Partitioning
Other
Neural Network Classifier
High Performance Computing

Why Machine Learning with High Performance Computing ?

- Numerical analysis formed backbone for supercomputing devices over the decades.
- Recently scientists have begun experimenting with understanding complex systems using
  - Machine Learning predictive models
  - Deep Neural Network
  - With Parallel Computing

  Trained by virtually unlimited datasets produced globally.
- HPC can improve accuracy, accelerate time to solution and significantly reduce costs.

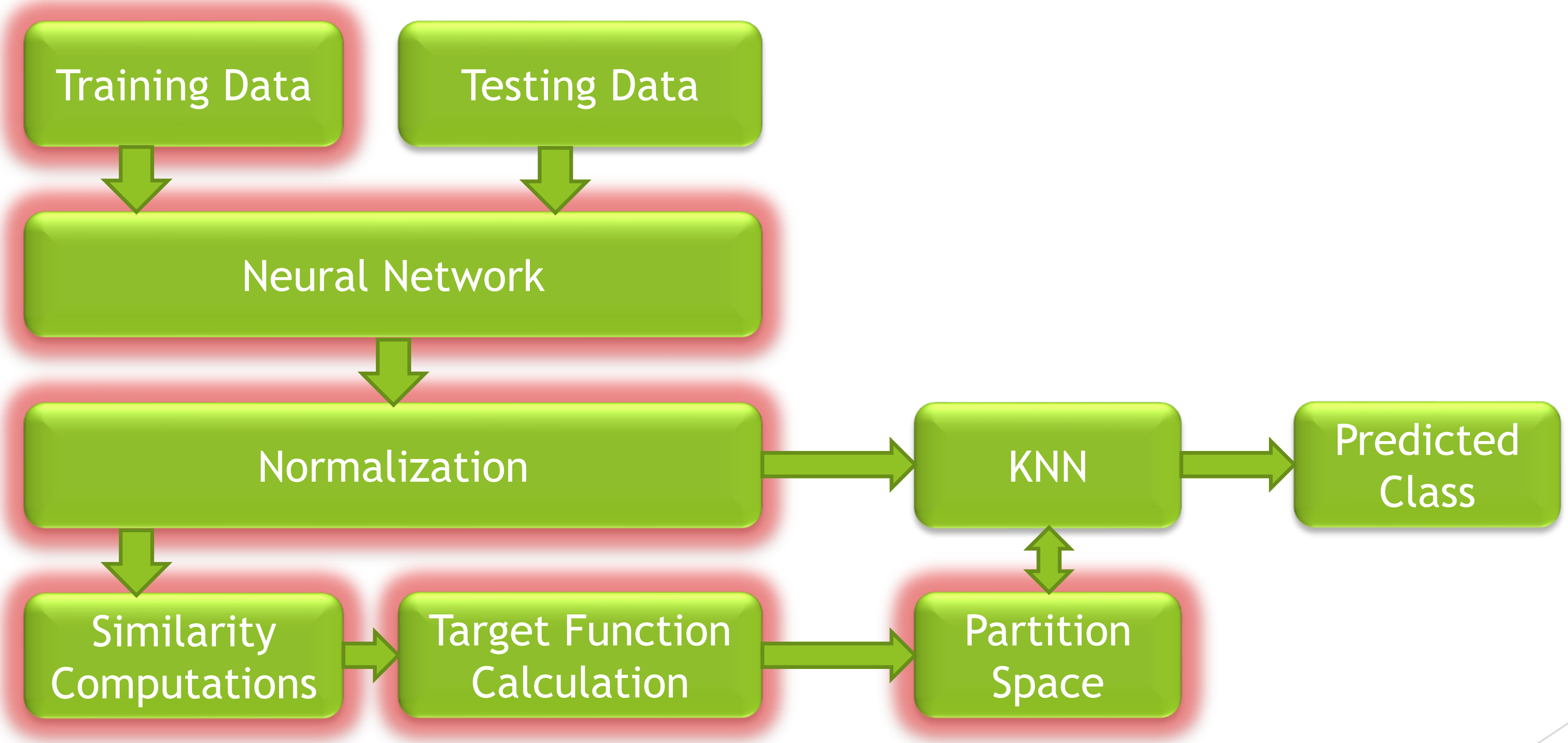
Literature Survey

- Anderson, D., Coupland, S., **Parallelisation of fuzzy inference on a graphics processor unit using the compute unified device architecture.** Recent. Progr. Med. 85 (3), 160–165, 2008.
- Jang, H., Park, A., Jung, K., **Neural network implementation using cuda and openmp. In: Digital Image Computing: Techniques and Applications.** pp. 155–161, 2008.
- Wang, L., Yang, B., Chen, Y., **Improving particle swarm optimization using multilayer searching strategy.** Inform. Sci. 274 (8), 70–94, 2014.
- Wang, L., Yang, B., Chen, Y., Abraham, A., Sun, H., Chen, Z., Wang, H., **Improvement of neural network classifier using floating centroids.** Knowl. Inf. Syst. 31 (3), 433–454, 2012.

Literature Survey : Floating Centroid Method

Fixed Centroids	• In traditional Neural Network, features of centroids are fixed. • In mapping of samples, fixed centroids reduces possibilities of finding optimal solution.
Floating Centroid Method	• In partition space, centroid is point. The proposed approach introduces many floating centroids. • FCMs are spread throughout partition space, and obtained by using K-Means algorithms.
Limitations of FCM	• Cannot yield flexible decision boundaries. • Problem overcomes by Neural Network Partitioning.

Literature Survey : Particle Swarm Optimization



Methodology

- Sample Mapping
- Normalization
- Similarity Computations
- Target Function Calculation

Sample Mapping in Parallel

In CUDA, data transmission takes place between CPU and GPU

Original samples → Global Memory

Information of Neural Network → Shared Memory

Data Divided as →

$$|S^i| = \begin{cases} \left\lceil \dfrac{|S|}{threadcount} \right\rceil, & i < threadcount \\ |S| mod \left\lceil \dfrac{|S|}{threadcount} \right\rceil, & i = threadcount \end{cases}$$

Sample Mapping in Parallel

tx → identity of thread block

Value of identity of particle →

$$\boldsymbol{particleid = {}^{tx}/_{threadcount}}$$

The value of identity of thread in each neural network →

$$\boldsymbol{threadid = tx \bmod threadcount}$$

Evaluation of sample is executed parallelly by tasks ***{ T^1, T^2, …, $T^{threadnum}$ }*** are allocated to the ***threadnum*** parallel thread in one block. Then mapped samples stored in global memory.

Normalization in Parallel

Set the value of mean and standard deviation of subdata *threadid* to 0

for i=0 to |S *threadid* | do
    for j=0 to |S| do
        Calculate the average value and standard deviation of each sample in subdata *threadid*;
    end
end

__syncthreads();

Normalization in Parallel

```
for i=0 to |S threadid | do
    for k=0 to m do
        Calculate the standard value of the sample
        points in subdata threadid;
    end
end
```

__syncthreads();

Return standard value

Similarity Computation in Parallel

$$d = |x|\frac{1 - e^{-|x|/2}}{|x| + |x| \cdot e^{-|x|/2}}$$

for i=0 to |S $threadid$ | do
    for j=0 to m do
        Calculate d use formula in subdata $threadid$; Obtain the values by the normalized points into a hypersphere;
    end
end

__syncthreads();

Similarity Computation in Parallel

for i=0 to |S $threadid$ | do
    for j=0 to |S| do
        Calculate the distance D between two points with Euclidean Distance in subdata $threadid$;
        if threadid:=j then
            The value of $\text{SimilarityArray}_{threadid,j}$ is 2;
        else
            The value of $\text{SimilarityArray}_{threadid,j}$ is 2-D;
    end
end

__syncthreads();

Return standard value

Target Function Calculation in Parallel

$$F = \omega(x_i)(S_{nonself}(x_i) - \alpha S_{self}(x_i))$$

α → adjustment coefficient

$S_{nonself}(x_i)$ → Sum of similarities between x_i and samples in other classes

$S_{self}(x_i)$ → sum of similarities between x_i and samples in same class

Fitness value → adding up all subdata values

- Host side → weight of each sample is calculated
- Device side → data is transmitted to and stored in array *weight*
- Each thread calculates each subdata in array *SimilarityArray*
- Each sample is divided into **self-class** and **nonself-class** according to similarity
- Each thread calculates value of S_{self} and $S_{nonself}$ in parallel

Implementation : Environment

CPU : Intel i7 4th Gen.

- CPU Cores → 8
- CPU Clock → 3.40 GHz
- L3 Cache → 8 MB

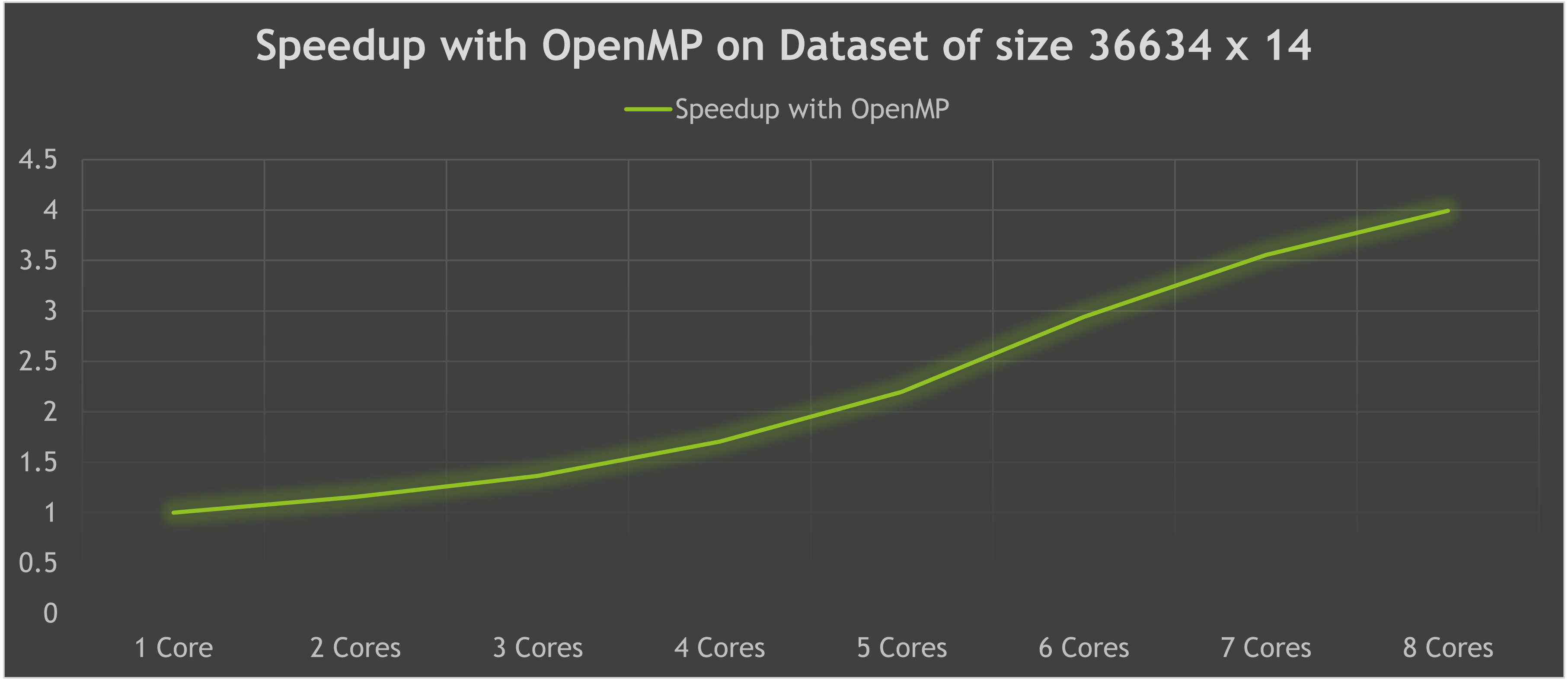
GPU : NVIDIA GeForce GT740

- CUDA Cores → 384
- Graphics Clock → 993 MHz
- Memory → 2048 MB DDR3

Implementation

- C++ Serial program for Normalization of Data
- C++ Parallel program for Normalization of Data using OpenMP
- C++ Parallel program for Normalization of Data using CUDA on NVIDIA's GPU

Implementation : Speedup vs. Cores



- ❖ Average Time Required to CUDA Program for Execution → 1085 microsec.
- ❖ Speedup GPU vs CPU → 7.9253

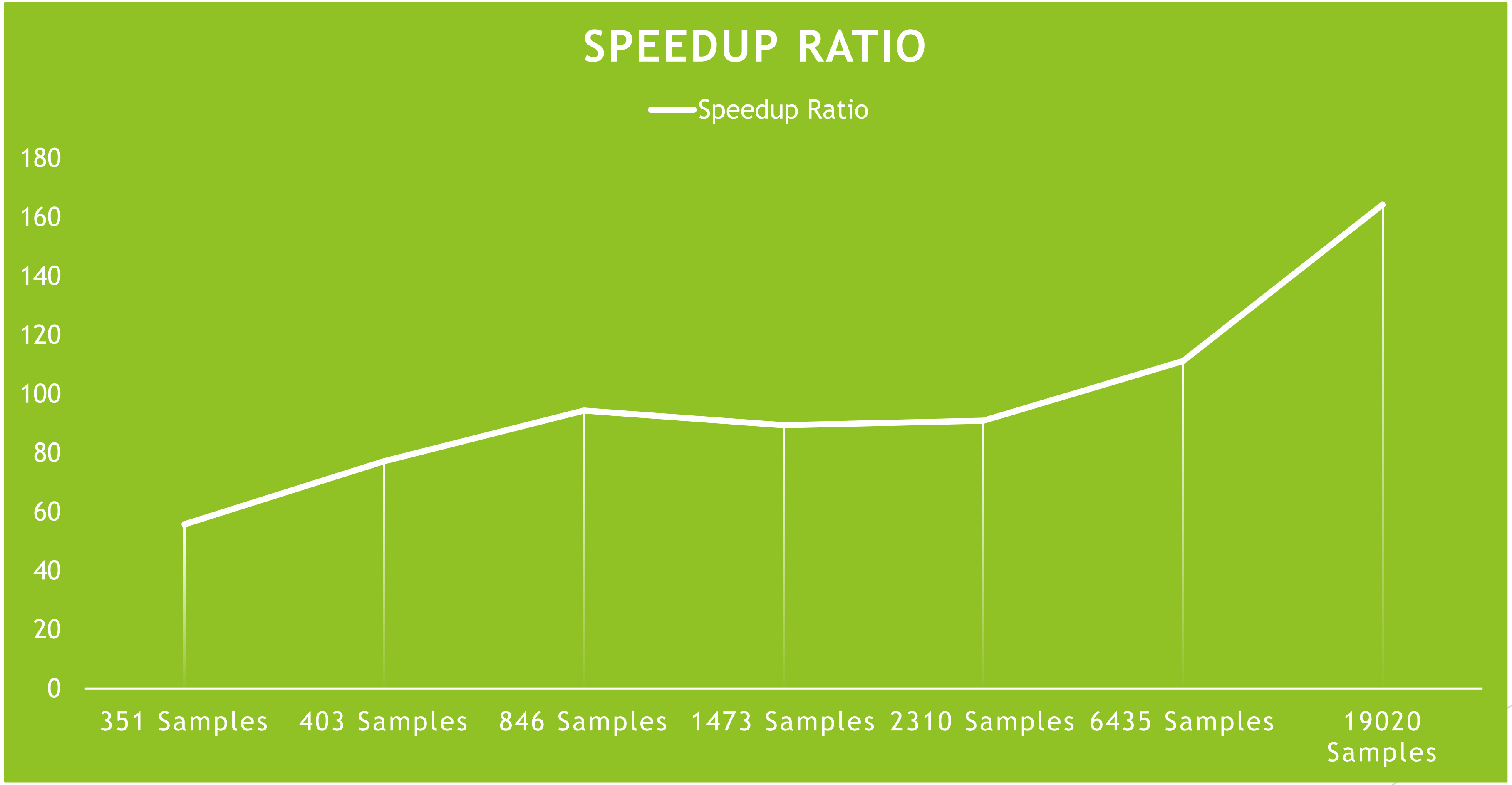
Result & Analysis

- Experiment
  - C / C++ programing environment and Linux Operating System
  - NVIDIA's high performance Tesla K10 GPU
    - 8 SMs each consisting 192 SPs
    - SP's clock frequency → 750 MHz
    - In each block → 65,536 registers
    - Each Multiprocessor → Thousands of CUDA cores
- 10-Fold Cross-Validation
  - Dataset divided into 10 subsets
  - One is used for testing
  - Remaining are used for training
  - Then average performance reported

Result & Analysis

Dataset	Size	NNP $_{\text{Seconds}}$	PNNP $_{\text{Seconds}}$	Speedup Ratio
IONOSPHERE	351 x 34	1283	23	55.78
UKM	403 x 5	1158	15	77.2
VEHICLE	846 x 18	6230	66	94.39
CMC	1473 x 9	19583	219	89.42
SEGMENT	2310 x 19	71198	783	90.93
SATELLITE	6435 x 36	578905	5178	111.8
MAGIC	19020 x 10	8177290	49773	164.29

Result & Analysis



Result & Analysis

	Traditional (Wang et al., 2017)	SoftMax (Wang et al., 2017)	ECOC (Wang et al., 2017)	FCM (Wang et al., 2017)	PNNP
IONOSPHERE	86.18(±6.42)	N/A	N/A	93.44(±5.43)	93.59(±1.63)
PARKINSONS	83.12(±16.57)	N/A	N/A	76.75(±17.39)	85.52(±9.07)
HABERMAN	59.42(±8.02)	N/A	N/A	55.93(±9.61)	66.79(±5.73)
UKM	92.73(±5.03)	85.22(±14.79)	93.26(±2.98)	96.11(±2.03)	97.64(±2.01)
VEHICLE	81.88(±2.14)	83.28(±2.19)	80.10(±3.12)	78.54(±4.17)	83.19(±2.24)
CMC	50.84(±4.88)	50.49(±2.89)	49.28(±3.16)	52.83(±2.55)	53.08(±2.5)
SEGMENT	96.66(±1.79)	96.55(±1.40)	96.16(±1.37)	95.72(±1.69)	96.92(±0.90)
VC	72.50(±5.72)	79.98(±6.31)	73.59(±9.61)	81.21(±7.77)	83.02(±6.68)
SEEDS	93.32(±6.01)	92.85(±7.15)	94.26(±7.43)	95.18(±5.08)	96.16(±3.06)
WINE	97.85(±3.69)	98.33(±3.77)	97.78(±2.88)	98.88(±2.36)	99.49(±1.62)

Conclusion

- To speedup the training process of NNP, particularly for large dataset, we proposed parallel NNP based on NVIDIA's CUDA framework.
- At CPU Side → Main optimization algorithm performed in serial.
- At GPU Side → NNP Neural Network Classifier is evaluated parallelly on multiple blocks and all subtasks are performed parallelly using threads.
- PNNP not only increases speed but also improves measurements.
- PNNP yields promising that it is able to solve real world problems.
- In the future, PNNP can be used in medical image analysis and bioinformatics with referenced to big data.

References

- Anderson, D., Coupland, S., **Parallelisation of fuzzy inference on a graphics processor unit using the compute unified device architecture.** Recent. Progr. Med. 85 (3), 160–165, 2008.
- Jang, H., Park, A., Jung, K., **Neural network implementation using cuda and openmp. In: Digital Image Computing: Techniques and Applications.** pp. 155–161, 2008.
- Wang, L., Yang, B., Chen, Y., **Improving particle swarm optimization using multilayer searching strategy.** Inform. Sci. 274 (8), 70–94, 2014.
- Wang, L., Yang, B., Chen, Y., Abraham, A., Sun, H., Chen, Z., Wang, H., **Improvement of neural network classifier using floating centroids.** Knowl. Inf. Syst. 31 (3), 433–454, 2012.

Thank You !