A
Seminar Report
On

"SCALING UP MACHINE LEARNING AND DEEP LEARNING : PARALLEL APPROACH WITH CUDA AND OPENMP"

For the degree of

MASTER OF TECHNOLOGY
(Computer Science and Engineering)

Submitted by

Jadhav Veersen Vijay
(2018MTECSCO007)

Under the guidance of
Prof. N. K. Pikle

Department of Computer Science and Engineering

WALCHAND COLLEGE OF ENGINEERING, SANGLI
2018-2019

CERTIFICATE

This is to certify that the dissertation work entitled "SCALING UP MACHINE LEARNING AND DEEP LEARNING : PARALLEL APPROACH WITH CUDA AND OPENMP" submitted by "*Jadhav Veersen Vijay (2018MTECSCO007)*" in partial fulfillment of the requirement for the degree of "*Master of Technology (Computer Science and Engineering*)" is a record of his/her own work carried out under my supervision during the year *2018-19*.

Date:

Prof. Dr. B. F. Momin (Head of Department)	Prof. N. K. Pikle (Seminar Guide)
Prof. M. A. Shah (Panel Member)	Prof. S. S. Solapure (Panel Member)

ACKNOWLEDGMENT

I would like to express my deep gratitude towards my seminar Guide Prof. N. K. Pikle for his constant guidance during my seminar work and to the Computer Science and Engineering Department for making its facilities available to me. I take this opportunity to express my sincere thanks to all the staff members of Computer Science and Engineering Department for their help whenever required. Finally, I express my sincere thanks to all those who helped me directly or indirectly while executing seminar work.

Jadhav Veersen Vijay

(2018MTECSCO007)

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. Title of Paper :

Accelerating nearest neighbour partitioning neural network classifier based on CUDA.

1.1 Research Area :

Applying High Performance Computing / Parallel Computing to Machine Learning Algorithms and Deep Learning.

2. Confidence of Paper :

Authors → Lin Wang, Ajith Abraham, Meihui Li

Publisher → Elsevier

Journal → Engineering Applications of Artificial Intelligence

Year → 2018

3. Abstract

Many algorithms for image process and pattern recognition have recently been enforced on GPU (Graphic Process Unit) for quicker machine times. However, the implementation exploitation GPU encounters 2 issues. First, the engineer ought to master the basics of the graphics shading languages that need the previous data on lighting tricks. Second, in a very job that desires a lot of cooperation between mainframe and GPU, that is common in image processing and pattern recognitions contrary to the graphics space, ought to generate raw feature knowledge for GPU processing the maximum amount as potential to effectively utilize GPU performance. This study based on a lot of fast and economical implementation of neural networks on each GPU and multi-core mainframe. We have a tendency to use CUDA (Compute Unified Device Architecture) that may be simply programmed because of its easy C language-like vogue rather than GPGPU to resolve the primary drawback. Moreover, OpenMP (Open Multi-Processing) is employed to at the same time method multiple knowledge with single instruction on multi-core mainframe, which ends in effectively utilizing the recollections of GPU. Within the experiments, we have a tendency to enforced neural networks-based text detection system exploitation the planned design, and also the machine times showed regarding fifteen times quicker than implementation exploitation mainframe and regarding four times quicker than implementation on solely GPU while not OpenMP.

The Nearest Neighbor Partitioning (NNP) methodology may be a high-performance approach that is employed for rising traditional neural network classifiers. However, the development method of NNP model is extremely long, particularly for giant knowledge sets, therefore limiting its vary of application. During this study, a parallel NNP methodology is projected to accelerate NNP supported Compute Unified Device Architecture (CUDA). During this methodology, blocks and threads are accustomed judge potential neural networks and to perform parallel subtasks, severally. Experimental results manifest that the projected parallel methodology improves performance of NNP neural network classifier.

3.1. Research Attempts

Speedup of Implementing Fuzzy Neural Networks with High-Dimensional Inputs Through Parallel Processing on Graphic Processing Units by Chia-Feng Juang; Teng-Chang Chen; Wei-Yuan Cheng, proposed the implementation of a zero-order Takagi-Sugeno-Kang (TSK)-type fuzzy neural network (FNN) on graphic process units (GPUs) to scale back coaching time. The code platform that this study uses is that the compute unified device architecture (CUDA). The enforced FNN uses structure and parameter learning in a very self-constructing neural fuzzy reasoning network thanks to its admirable learning performance. FNN coaching is conventionally enforced on a single-threaded processor, wherever every input variable and fuzzy rule is serially processed. This sort of coaching is time intense, particularly for a high-dimensional FNN that consists of an oversized variety of rules. The GPU is capable of running an oversized variety of threads in parallel. In a very GPU-implemented FNN (GPU-FNN), blocks of threads area unit partitioned off in line with parallel and freelance properties of fuzzy rules. Giant sets of computer file area unit mapped to parallel threads in every block. For memory management, this analysis fittingly divides the datasets within the GPU-FNN into smaller chunks in line with fuzzy rule structures to share on-chip memory among multiple

thread processors. This study applies the GPU-FNN to totally different issues to verify its potency. The results show that to coach an FNN with GPU implementation achieves a hurrying of quite thirty times that of processor implementation for issues with high-dimensional attributes.

Extending parallelization of the self-organizing map by combining data and network partitioned methods by Trevor Richardson, Eliot Winer proposed High-dimensional information is pervasive in several fields like engineering, geospatial, and medical. It's a relentless challenge to make tools that facilitate individuals in these fields perceive the underlying complexities of their information. Several techniques perform spatial property reduction or alternative "compression" to indicate views of information in either 2 or 3 dimensions, effort the information analyst to infer relationships with remaining freelance and dependent variables. Discourse self-organizing maps provide how to represent and move with all dimensions of an information set at the same time. However, machine times required to get these representations limit their practicability to realistic trade settings. Batch self-organizing maps offer a data-independent technique that permits the coaching method to be parallelized and thus sped up, saving time and cash concerned in process information before analysis. This analysis parallelizes the batch self-organizing map by combining network partitioning and information partitioning ways with CUDA on the graphical process unit to realize important coaching time reductions. Reductions in coaching times of up to 25 times were found whereas exploitation map sizes wherever alternative implementations have shown weakness. The reduced coaching times open up the discourse self-organizing map as viable choice for engineering information image.

3.2. Technology

3.2.1. Particle Swarm Optimization

Particle swarm optimization (PSO) algorithmic program has been wont to solve international optimization issues. This algorithmic program is wide used as an efficient optimization tool in numerous applications. However, ancient PSO consists of solely 2 looking layers and therefore usually leads to premature convergence into the native minima. Thus, multi-layer particle swarm optimization (MLPSO) is projected during this study to enhance the performance of ancient PSO by increasing the 2 layers of swarms to multiple layers. The MLPSO strategy will increase the range of looking swarms to enhance its performance once resolution complicated issues. The experiment indicates that the novel approach improves the ultimate results and also the convergence speed.

3.2.2. Floating Centroid Method

Floating Centroids method (FCM) designed to enhance the performance of a traditional neural network classifier. Partition area may be a area that's wont to categorise information sample when sample is mapped by neural network. Within the partition area, the center of mass may be a purpose, that denotes the middle of a category. During a standard neural network classifier, position of centroids and therefore the relationship between centroids and categories are set manually. Additionally, range of centroids is mounted with relevance the quantity of categories. The planned approach introduces several floating centroids, that are unfold throughout the partition area and obtained by exploitation K-Means rule. Moreover, totally different categories labels ar hooked up to those centroids mechanically. A sample is foretold

as a precise category if the nearest center of mass of its corresponding mapped purpose is labelled by this category. Experimental results illustrate that the planned methodology has favorable performance particularly with regard to the coaching accuracy, generalization accuracy, and average F-measures

3.2.3. Basic GPU Architecture

A unified GPU processor array contains many processor cores, typically organized into multithreaded multiprocessors. Fig. 1 shows a GPU with an array of 112 streaming processor (SP) cores, organized as 14 multithreaded streaming multiprocessors (SM). Each SP core is highly multithreaded, managing 96 concurrent threads and their state in hardware. The processors connect with four 64-bit-wide DRAM partitions via an interconnection network. Each SM has eight SP cores, two special function units (SFUs), instruction and constant caches, a multithreaded instruction unit, and a shared memory. This is the basic Tesla architecture implemented by the NVIDIA GeForce 8800. It has a unified architecture in which the traditional graphics programs for vertex, geometry, and pixel shading run on the unified SMs and their SP cores, and computing programs run on the same processors.

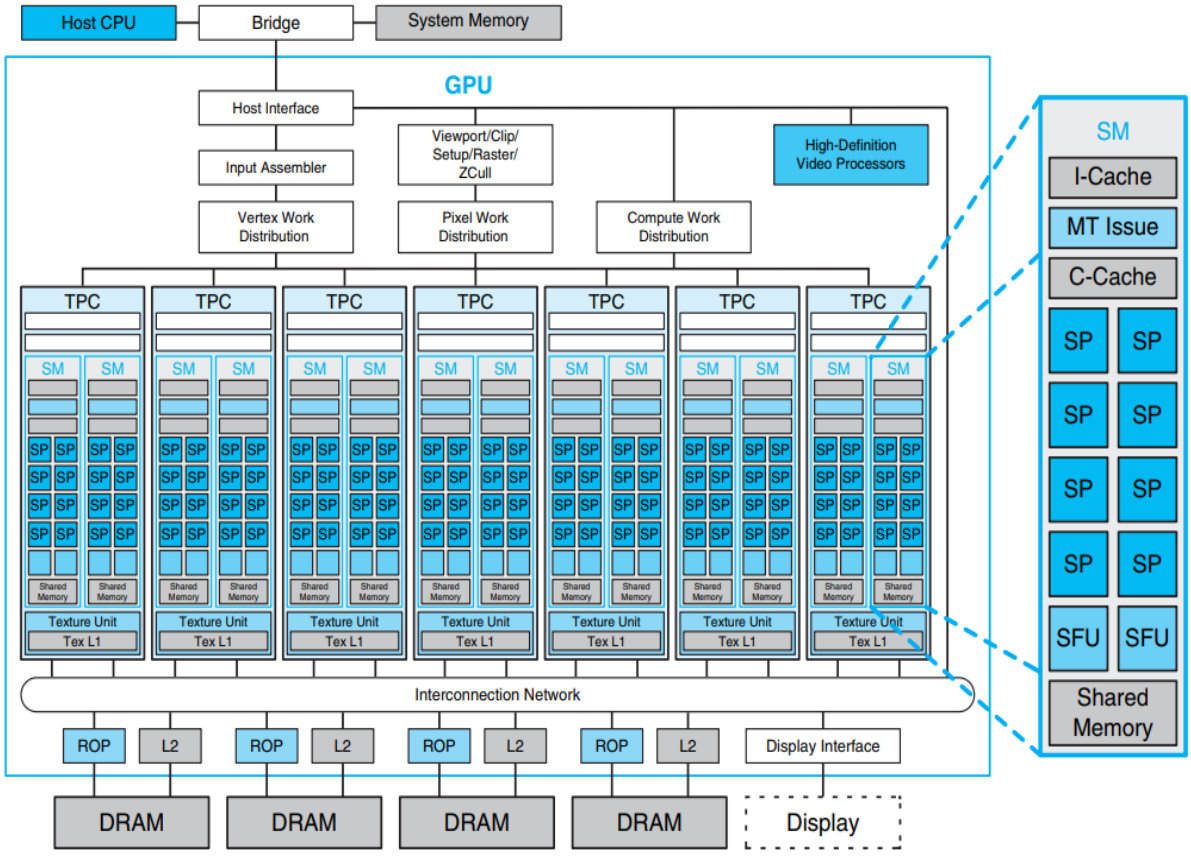

Fig. 1 : Basic unified GPU architecture

The processor array architecture is scalable to smaller and larger GPU configurations by scaling the number of multiprocessors and the number of memory partitions. Fig. 1 shows seven clusters of two SMs sharing a texture unit and a texture L1 cache. The texture unit delivers

filtered results to the SM given a set of coordinates into a texture map. Because filter regions of support often overlap for successive texture requests, a small streaming L1 texture cache is effective to reduce the number of requests to the memory system. The processor array connects with raster operation (ROP) processors, L2 texture caches, external DRAM memories, and system memory via a GPU-wide interconnection network. The number of processors and number of memories can scale to design balanced GPU systems for different performance and market segments.

4. Introduction

Classification is a great tool within the field of knowledge mining and machine learning. It aims to be told a classification operate supported existing knowledge or to construct a classification model, i.e., the ‘‘Classifier’’. When learning, the operate or model not solely maps the knowledge records within the training data to a selected category however is also ready to predict the unknown samples. Within the ancient neural network, the position, number, and labels of centroids are permanent in coaching neural networks. However, within the improvement method, mapping the samples to fixed centroids reduces the chance of finding best neural network. Therefore, the floating centre of mass technique (FCM) was projected to unravel the higher than issues. The goodness of every attainable neural network in NNP is evaluated employing a nearest-neighbour criterion. NNP is ready to simply yield versatile call boundaries and partitions, therefore will increase the likelihood of discovering best answer.

The compute unified device architecture (CUDA), that runs on the Graphics process Unit (GPU) brings the dawn to beat the potency disadvantage of nearest neighbour partitioning on cheap platform. CUDA regards GPU as a high-performance device that implements the management and allocation of tasks. With the utilization of CUDA, it's easier for scientists and engineers to develop general purpose computing intensive applications. Thanks to the high process quality that NNP faces, a CUDA parallel computing primarily based strategy is planned to accelerate NNP’s coaching method. During this technique, blocks and threads work along to perform the parallelization of nearest neighbour partitioning wherever blocks square measure accountable for the analysis of neural networks and threads square measure accountable for the computation of parallel subtasks, severally.

4.1. Neural Network Classifier

A neural network consists of units (neurons), organized in layers, that convert an input vector into some output. Every unit takes an input, applies a (often nonlinear) operate to that then passes the output on to consequent layer. Usually the networks square measure outlined to be feed-forward: a unit feeds its output to all or any the units on consequent layer, however there's no feedback to the previous layer. Weightings square measure applied to the signals passing from one unit to a different, and it's these weightings that square measure tuned within the coaching part to adapt a neural network to the actual drawback at hand. This can be the training part.

Neural Networks is one among the earliest machine learning formula used chiefly for classification functions. It mimics the human brain at an awfully little scale. It's a collection of neurons (kind of supply regression units), organized within the kind of multiple hidden layers, lying between the input layer (input data) and also the output layer (each output nerve cell representing a target variable). The neurons in every layer are connected to the neurons in different layers through weighted connections, wherever the weights are fine-tuned throughout the coaching method. A user identifies the number of hidden layers that a neural network should have, in conjunction with the amount of neurons in every layer.

NNP adopts the particle swarm optimization (PSO) to optimize the weights and bias of neural networks. Every particle is encoded as a vector of weights. In every generation of improvement, once evaluating particles, samples area unit mapped to the partition area by decoded particle, i.e. neural networks, and is normalized mistreatment the Z-score. The

normalized mapped samples (points) area unit more strained into a hypersphere. A nearest neighbour criterion target perform is employed to judge the standard of the distribution of points in partition area. when improvement, a model of best neural network is found. So as to predict the unknown sample, the sample is mapped by best neural network initially and is more categorised by k-nearest neighbour. The general procedure of NNP is shown in Fig. 2.

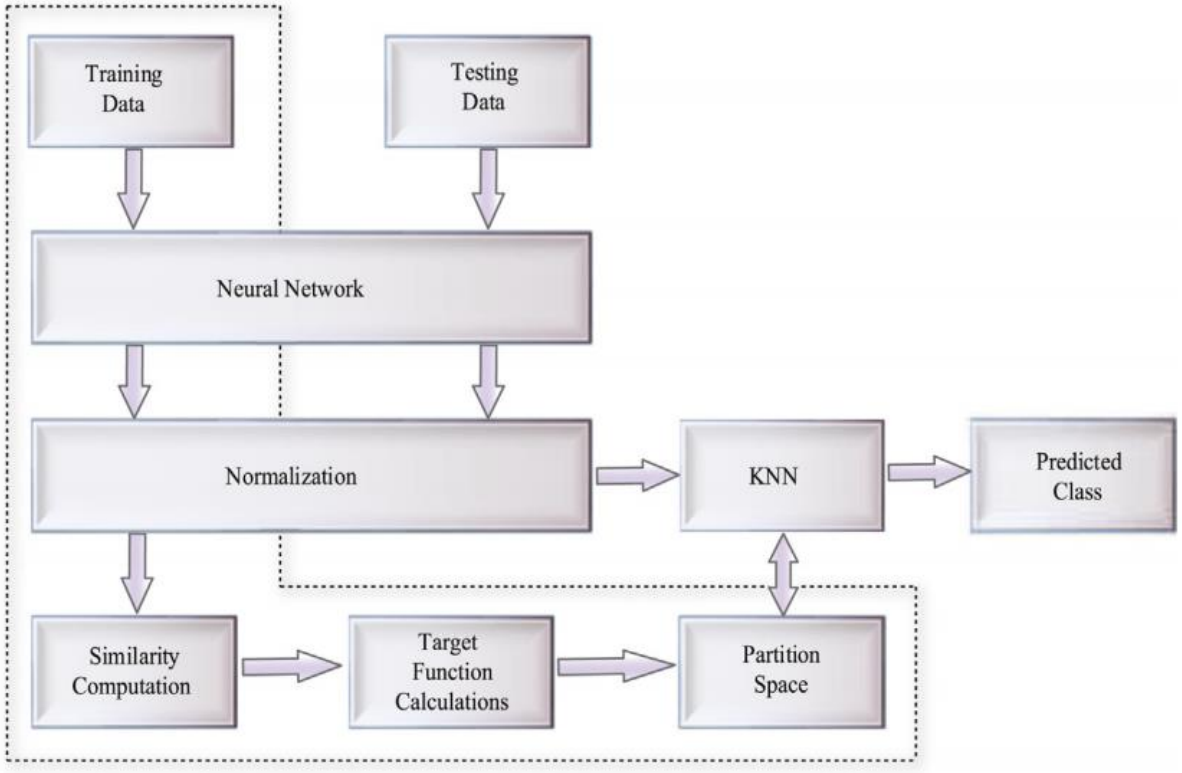

Fig. 2 : The framework of nearest neighbor partitioning.

4.2. Compute Unified Device Architecture (CUDA)

CUDA is NVIDIA's parallel computing design that allows dramatic will increase in computing performance by harnessing the facility of the GPU (graphics processing unit).With uncountable CUDA-enabled GPUs sold up to now, computer code developers, scientists and researchers area unit finding broad-ranging uses for CUDA, as well as image and video process, process biology and chemistry, fluid dynamics simulation, CT image reconstruction, seismic analysis, ray tracing, and far a lot of.

The CUDA parallel computing architecture, that was launched by NVIDIA, may be a revolutionary general purpose computing design in recent years. It contains instruction set architecture(ISA) and parallel computation engine inside the GPU so it allows GPU to resolve advanced process issues. As a technology that supports each hardware and package, CUDA at the same time uses multiple GPU cores to perform general computation process. With the employment of this design, the computing performance are often considerably improved. The program are often run in a very high performance on CUDA processors. The program code supported CUDA is divided into 2 classes in observe, one is that the host code running on the central processor, the opposite one is that the device code running on the GPU. The parallel program runs on the GPU is termed the kernel. Once CUDA deals with parallel computation, the threads constitutes block and blocks constitutes grid. Then an equivalent kernel code

performs on a grid in parallel. The framework of CUDA is shown in Fig. 3. It are often seen from the figure that there are many streaming multiprocessors(SM) in every GPU. Every digital computer includes many streaming processors(SP) and includes a set of native 32-bit registers.

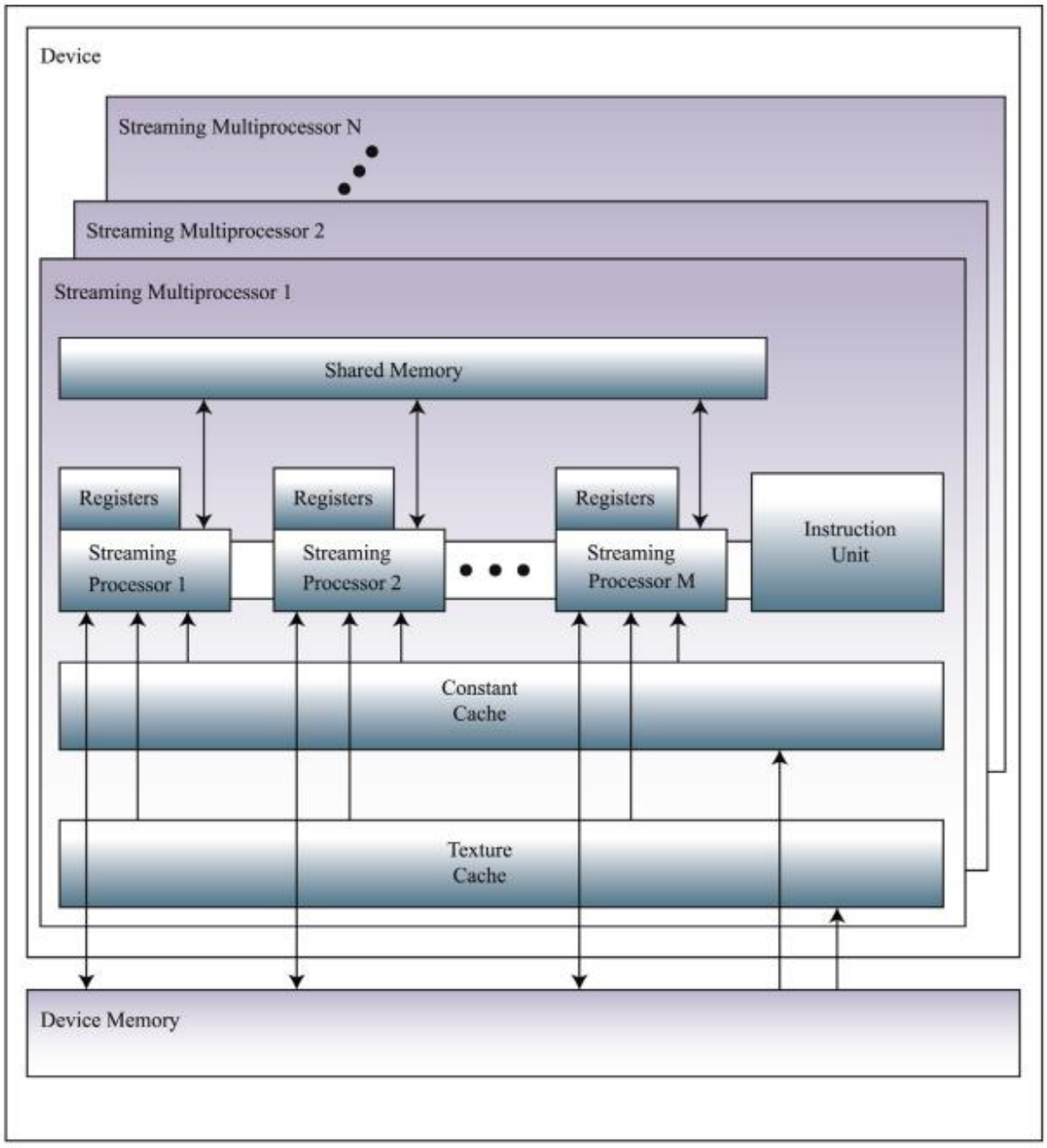

Fig. 3 : The Framework of CUDA.

5. Methodology

Based on CUDA's parallel architecture, the particle swarm optimisation is performed at the CPU facet whereas the classifier is evaluated at the GPU facet. That is, once the load and bias of the neural network area unit provided on the CPU facet, sample mapping, standardisation, similarity computation and target perform calculation area unit performed on the GPU facet. On the one hand, change weights on the CPU needs the GPU to calculate every step in parallel. Thus, CPU has been waiting till the fitness values of the model area unit came by the GPU. On the opposite hand, every block is liable for evaluating many potential models at the GPU facet. Threads in every block management the computation of 4 substages. The parallel relationship of the PNNP is illustrated in Fig. 4.

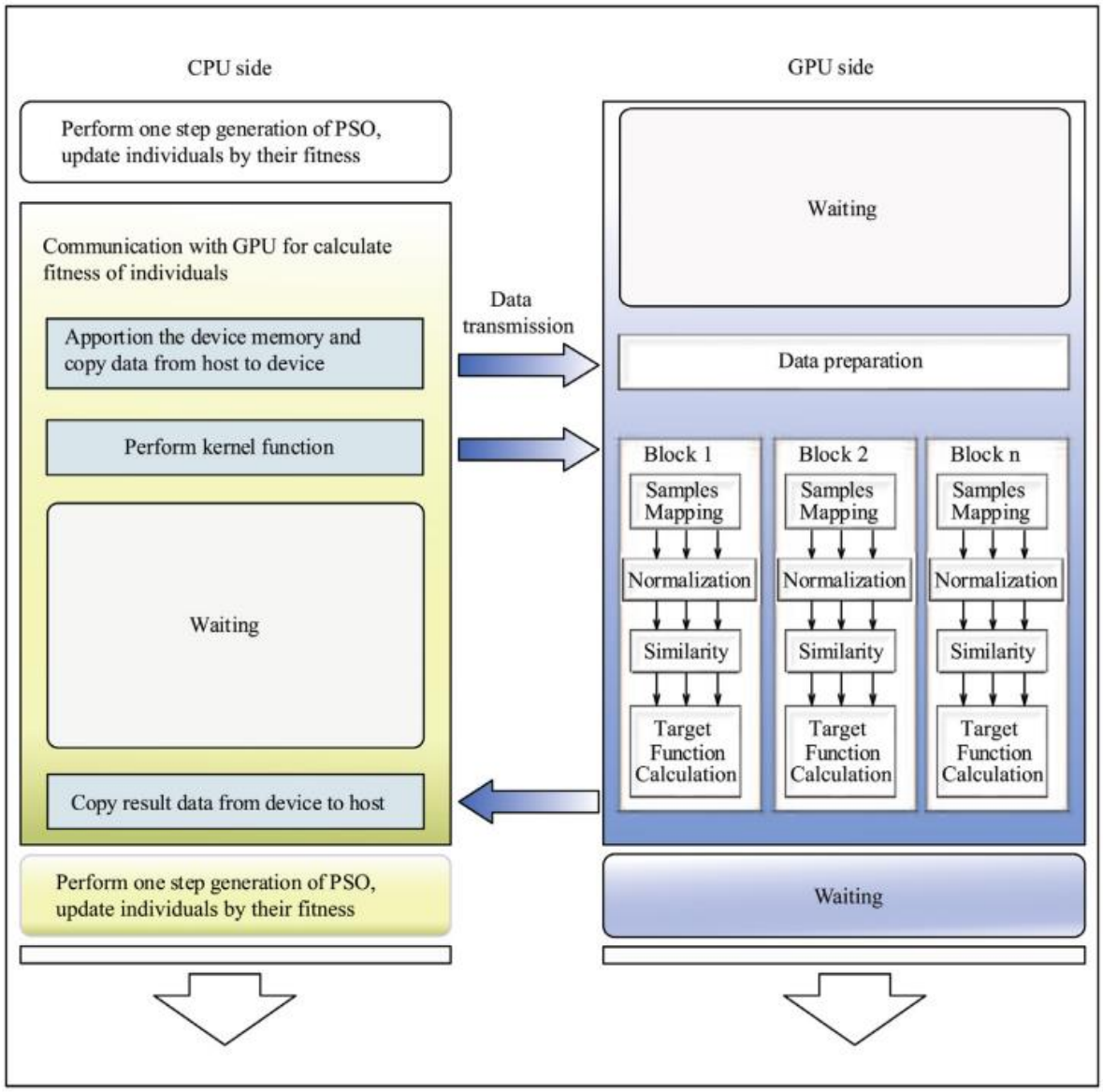

Fig. 4 : The parallel relationship of PNNP

In application, an optimized neural network and its corresponding partition space can be established after training. Therefore, when an unknown sample arrives, it is mapped to the

partition space at first by optimized neural network. Then, the sample **x** is normalized as follows

$$x' = \frac{(x - \mu)}{\sigma} \tag{1}$$

where μ and σ are obtained from training set. Then, the normalized sample is bounded to a hypersphere. Finally, the class of the unknown sample is labelled as a class by the weighted K-nearest neighbours.

5.1. Samples mapping in parallel

In the CUDA parallel computing, the information of neural network and sample data transmit between the GPU and CPU, i.e., the transmission is from the host to the device. The original samples are stored in the global memory and the information of neural networks is stored in the shared memory. Several neural networks are processed in a block at the GPU side, and each neural network corresponds to a particle. When a block has $threadnum$ threads, it will allocate $threadcount$ threads to each neural network. Then the entire data is divided into $threadcount$ groups of subdata $\{S1, S2, \ldots, Sthreadcount\}$ for each neural network, and $S\,i$ (i = 1, 2, …, $threadcount$) contains $|S\,i\,|$ sample where the number of sample is defined as $|S|$. The identity of thread in a block is defined as tx, then the value of the identity of particle is calculated by Eq. (3) and the value of the identity of thread in each neural network is calculated by Eq. (4).

$$|S^i| = f(x) = \begin{cases} \left\lfloor \frac{|S|}{threadcount} \right\rfloor, & i < threadcount \\ |S| \bmod \left\lfloor \frac{|S|}{threadcount} \right\rfloor, & i = threadcount \end{cases} \tag{2}$$

$$particleid = {}^{tx}/_{threadcount} \tag{3}$$

$$threadid = tx \bmod threadcount \tag{4}$$

5.2. Normalization in parallel

After the samples are mapped, the new samples are normalized by the Z-score algorithm. The data are stored in array $Dataout$ and in the global memory. There are several steps involved in the parallel process. First, each thread calculates each dimension of sample and the average in each dimension of all the samples are obtained. The standard deviation of the dimension of each sample is calculated using the average value of data in parallel. Then, each thread is responsible for calculating each sample. The normalized value is obtained using Eq. (1).

Algorithm: *Parallel Normalization in CUDA*

Input: *Sample points $Dataout$, the dimension of partition space m, and thread identification $threadid$*

Output: *Standard value.*

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();

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

__syncthreads();

end

Return standard value.

5.3. Similarity computations in parallel

After the samples are normalized, the new data is saved in array $Dataout$. Each sample is evenly assigned to each thread. The d is resolved using

$$d = |x| \frac{1 - e^{\frac{-|x|}{2}}}{|x| + |x| \cdot e^{\frac{-|x|}{2}}} \tag{5}$$

where **x** represents a normalized sample, and then each subdata is normalized to a hypersphere in parallel. After calculating all the samples, the distance between the two samples is obtained by calculating the new sample point in array $Dataout$, and a fixed value minus the distance to obtain the similarity matrix of samples. The similarity matrix are stored in array $SimilarityArray$ and in global memory.

Algorithm: *Parallel Similarity Computation in CUDA*

Input: *Sample points $Dataout$, the dimension of partition space m, and thread identification $threadid$*

Output: *Similar matrix.*

for i=0 to |S $threadid$| do

for j=0 to m do

Calculate d use formula (5) in subdata $threadid$;

Obtain the values by the normalized points into a hypersphere;

end

end

__syncthreads();

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 SimilarityArray$_{threadid,j}$ is 2;

else

The value of SimilarityArray$_{threadid,j}$ is 2-D;

end

__syncthreads();

end

Return the Similar matrix

5.4. Target function calculation in parallel

After calculating the similarity of all sample pairs, the weight of each sample is calculated at the host side, and the data is transmitted to the device side and stored in array $weight$. There are several steps involved in the parallel process. First of all, each thread calculates each subdata in array $SimilarityArray$. Then the samples have to be listed in descending order according to the value of similarity and the corresponding index numbers of samples are stored in the array $index$. Secondly, each sample is divided into self-class or nonself-class according the similarity of two samples belonging to a definite class. Then each thread calculates the value of $Snonself$ and $Sself$ for each sample in parallel. The value of F is calculated on the basis of $Snonself$, $Sself$ and the weight of the current sample $\omega(\mathbf{x}i)$

$$F = \omega(x_i)\left(S_{nonself}(x_i) - \alpha S_{self}(x_i)\right) \quad (6)$$

where α is a adjustment coefficient. $Snonself$ ($\mathbf{x}i$) represents the sum of similarities between $\mathbf{x}i$ and samples in other classes. $Sself$ ($\mathbf{x}i$) is the sum of similarities between $\mathbf{x}i$ and samples in same class. Finally, the fitness value of the current particle is obtained by adding up all the subdata values.

Algorithm: *Parallel Target Function Calculation in CUDA*

Input: *Similar matrix $SimilarityArray$, array $weight$ of each sample, space neighbor number L and thread identification $threadid$*

Output: *Fitness of each particle.*

Initialize array $index$ for subdata $threadid$;

for i=0 to |S $threadid$| do

for j=0 to L do

if SimilarityArray$j$ < SimilarityArray$j-1$ do

save the index numbers in array index according to the value of similarity

end

end

__syncthreads();

for i=0 to |S threadid | do

Initialize the sum of the weight of each sample point W sum;

for j=0 to W sum < L do

Calculate Sself for point j in subdata threadid;

Calculate Snonself for point j in subdata threadid;

Calculate fitness count for point j using Sself and Snonself ;

end

for k=0 to |S| do

Add the fitness of the current particle in subdata threadid;

end

end

Returns the fitness of each particle.

6. Result and Analysis

6.1. Implementation

The computing platform used in experiments is a GPU-based desktop supercomputer with C++ programming environment and Windows operating system. The platform includes a high-performance GeForce GTX 740 GPU produced by NVIDIA and Intel's Core i7 4th generation CPU. High-speed shared memory and synchronization mechanism are adopted in the SM. There are 3 SMs in the GPU, each of which has 4 SPs. The SP's clock frequency is 0.75 GHz. In each block, there are 65,536 available registers and the total amount of shared memory is 48 KB. Each multiprocessor has 128 CUDA cores. On this platform, the one defined algorithm normalization in parallel method is verified on commonly used data sets as well as heavy data set.

In this implementation, normalization algorithm is applied on the data sets which has 36634 rows and 14 features. In order to normalize data, the min–max normalization method is adopted. This is a linear transformation of the original data, so as to reduce the influence of different scale of attributes. Transformation function is as follows

$$x' = \frac{x - Min}{Max - Min} \tag{7}$$

where Max is maximum value of a attribution in a data set, $M\ in$ is the minimum value of a attribution in a data set, x is the original value of this attribution.

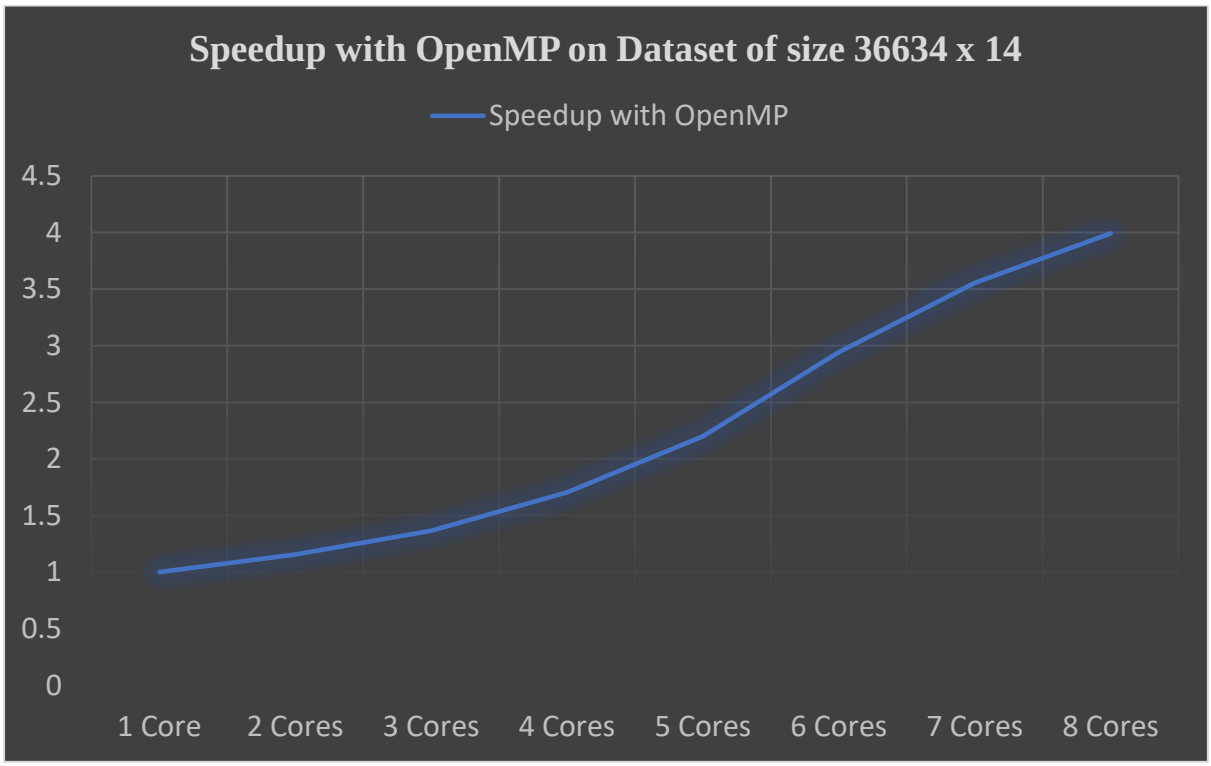

Fig. 5 : Speedup with OpenMP on Dataset of size 36634 x 14

The fig. 5 shows, the OpenMP parallelization achieves 4 time speedup than single core of CPU. For CUDA program, average time taken to CUDA program to run on GPU is 1085 micro seconds. Hence with respect to CPU, GPU got 8 times better speedup than serialized code.

6.2. Experiment Setup

In this subsection, some famous classification data sets are applied to the experiments. All data sets are from the UCI machine learning repository and the web site is http://archive.ics.uci.edu/ml/. Table 1 displays the information of adopted data sets, containing the number of samples, attributes and classes. In order to normalize data, the min–max normalization method is adopted. This is a linear transformation of the original data, so as to reduce the influence of different scale of attributes. Transformation function is as follows

$$x' = \frac{x - Min}{Max - Min} \tag{8}$$

where Max is maximum value of a attribution in a data set, $M\ in$ is the minimum value of a attribution in a data set, x is the original value of this attribution. We take a common test method 10-fold cross-validation. The data set is divided into ten subsets, where one subset is used as testing and the remaining subsets are used as training in turn. Then, the average performance is reported.

Speedup ratio and time consumption in the process of training.

	NNP (s)	PNNP (s)	Speedup ratio
IONOSPHERE	1 283	23	55.78
UKM	1 158	15	77.2
VEHICLE	6 230	66	94.39
CMC	19 583	219	89.42
SEGMENT	71 198	783	90.93
SATELLITE	578 905	5 178	111.8
MAGIC	8 177 290	49 773	164.29

Table 1 : Time consumption in the process of training.

A three layers feedforward neural network is chosen for the comparison of all methods, the parameters of the hidden layer and the dimension of the partition space are adjusted according to the requirements of the different data sets. According to Ref. Wang et al. (2017), the number of nearest neighbours L is chosen from 1 to 50, the discrimination weight α is chosen from 10−2 to 101 and the dimension of partition space m is chosen from 1 to 30. In the test, the number of nearest neighbours k is chosen from {1, 3, …, 25}. In PSO, the population size is set to 40, Max Generation is set to 1000, $\varphi 1$ and $\varphi 2$ are set to 1.8, and $V\ max$ is set to 0.4.

Compared with the NNP, the time consumption of the PNNP is reduced by parallel computation. By setting the same parameters for same data set, the results in Table 1 indicate that the proposed method is able to accelerate NNP's speed, and the speedup ratio is more remarkable for the large data sets. Fig. 6 reveals the relationship between speedup ratio and number of samples. Although the speedup ratio is slightly lower on CMC and SEGMENT than on VEHICLE, speedup ratio is on the rise with the increase of sample size. Therefore, this phenomenon reflects that the proposed PNNP is more favorable to large data sets because, with the use of adopted strategy, the utilization rate of resources increases with the increase of samples.

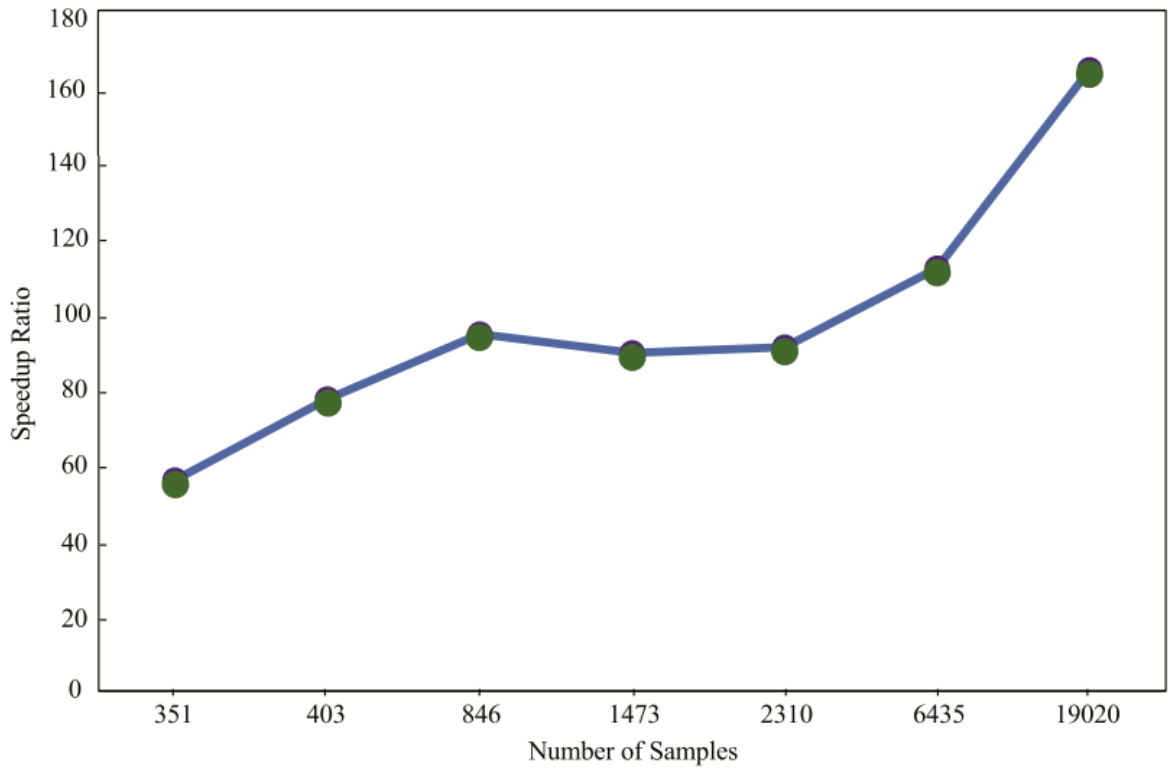

Fig. 6 : Number of samples vs. PNNP speedup ratio.

Table 2 shows the comparison of accuracies between the PNNP and other methods. It can be observed from the table that PNNP has higher accuracy than other methods in the experiment. Although the PNNP's accuracy is lower than accuracy of the FCM on HABERMAN and CMC, it is the best one on most of the data sets, such as PARKINSONS, UKM, VEHICLE, SEGMENT, VC, SEEDS and WINE. The accuracy of PNNP and FCM are tied for first on IONOSPHERE. In addition, the smallest standard deviation is generated by the PNNP method except HABERMAN and VC, which means the PNNP is more stable. Therefore, PNNP has higher performance than Traditional, SoftMax and ECOC in classification. This means that the introduction of PNNP not only speeds up the learning process for the training set, but also improves the average accuracy.

	TRADITIONAL	**SOFTMAX**	**ECOC**	**FCM**	**PNNP**
Ionosphere	88.06(±5.41)	N/A	N/A	94.17(±4.80)	94.17(±1.58)
UKM	92.38(±5.12)	91.67(±6.66)	93.33(±2.46)	95.95(±1.96)	97.62(±1.94)
Vehicle	81.74(±1.98)	83.26(±2.14)	79.88(±3.05)	79.19(±3.78)	83.49(±1.88)
VC	77.74(±3.86)	84.19(±5.15)	79.35(±7.78)	85.16(±6.31)	85.81(±6.12)
Seeds	93.33(±6.02)	92.86(±7.19)	94.29(±7.38)	95.24(±5.02)	96.19(±3.01)

Table 2 : Accuracy on some datasets

7. Conclusion

To speedup the training process of NNP, particularly for large dataset, parallel NNP based on NVIDIA's CUDA framework is used. At the CPU side, the main optimization algorithm performs in serial. At the GPU side, each potential NNP neural network classifier is evaluated on blocks in parallel and its subtasks are also performed in parallel on 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.

8. References

1. Accelerating nearest neighbor partitioning neural network classifier based on CUDA by Lin Wang, Xuehui Zhu, et al. 2018, Elsevier, Engineering Applications of Artificial Intelligence 68 (2018) 53–62.
2. Improving particle swarm optimization using multi-layer searching strategy by Lin Wang, Bo Yang, Yuehui Chen. 2014, Elsevier, Information Sciences 274 (2014) 70–94.
3. Improvement of neural network classifier using floating centroids by L. Wang B. Yang, et al. 2011, Springer, Knowl Inf Syst (2012) 31:433–454.
4. Neural Network Implementation using CUDA and OpenMP by Honghoon Jang, Anjin Park, Keechul Jung. 2008, IEEE, Digital Image Computing: Techniques and Applications.
5. Parallelisation of Fuzzy Inference on a Graphics Processor Unit Using the Compute Unified Device Architecture by Derek Anderson, Simon Coupland. 2008, Proceedings of the 2008 UK Workshop on Computational Intelligence.
6. Book, Graphics and Computing GPUs by John Nickolls, Director of Architecture, NVIDIA & David Kirk, Chief Scientist, NVIDIA.
7. NVIDIA Whitepaper, NVIDIA GeForce GTX 980, Featuring Maxwell, The Most Advanced GPU Ever Made.
8. NVIDIA Whitepaper, NVIDIA TURING GPU ARCHITECTURE Graphics Reinvented.