

Heterogeneous computing with graphical processing unit: improvised back-propagation algorithm for water level prediction

Neeru Singh, Supriya Priyabadini Panda

Department of Computer Science, Manav Rachna International Institute of Research and Studies, Faridabad, India

Article Info

Article history:

Received Mar 17, 2021

Revised Mar 24, 2021

Accepted Apr 14, 2022

Keywords:

Back-propagation network

Graphical processing unit

Ground water level

Heterogeneous computing

Unified memory

ABSTRACT

A multitude of research has been rising for predicting the behavior of different real-world problems through machine learning models. An erratic nature occurs due to the augmented behavior and inadequacy of the prerequisite dataset for the prediction of water level over different fundamental models that show flat or low-set accuracy. In this paper, a powerful scaling strategy is proposed for improvised back-propagation algorithm using parallel computing for groundwater level prediction on graphical processing unit (GPU) for the Faridabad region, Haryana, India. This paper aims to propose the new streamlined form of a back-propagation algorithm for heterogeneous computing and to examine the coalescence of artificial neural network (ANN) with GPU for predicting the groundwater level. twenty years of data set from 2001-2020 has been taken into consideration for three input parameters namely, temperature, rainfall, and water level for predicting the groundwater level using parallelized back-propagation algorithm on compute unified device architecture (CUDA). This employs the back-propagation algorithm to be best suited to reinforce learning and performance by providing more accurate and fast results for water level predictions on GPUs as compared to sequential ones on central processing units (CPUs) alone.

This is an open access article under the [CC BY-SA](https://creativecommons.org/licenses/by-sa/4.0/) license.



Corresponding Author:

Neeru Singh

Department of Computer Science, Manav Rachna International Institute of Research and Studies

Faridabad, India

Email: neerusingh123@gmail.com

1. INTRODUCTION

Water is an essential resource for the survival of life on the planet. Enlargement in demands of water due to increasing population, irrelevant usage, and acceleration of new commercial industry, moderately degrade the level of water. To prevent the dearth of water, it is crucial steps for the hydrological researchers to measure the quantity of water available and to act immediately to overcome the forthcoming danger [1]. Due to the enhancement in artificial neural network (ANN), it acts as a powerful machine approach for modeling water-related activity [2]. The deficit in the arbitrary large dataset will tend to fail in prediction with high precision on one core processor i.e., central processing unit (CPU), to improve the efficiency of big data set substantial hardware to team up with the CPU. The graphical processing unit (GPU) structure comprises thousands of cores and each core will act as a computation unit, which will amend the use of parallel structure and proffers very high-level thread parallelism [3]–[5]. The present computing structure of CPUs and GPUs does not promote the adequate improvement of performance over heterogeneous computing. To overcome this issue, a joint approach has been used by combining both CPUs of multi-core environments

and GPUs [6], [7]. Due to the demand for an accelerated high computational environment, an algorithm is required to decrease the execution time and improves performance [8]. Rather than doing shifting and allocating the memory to the host and device allocate a special pointer that can be used by both CPU and GPU, this is the concept of unified memory allocation [9]. According to recent advancements in unified memory employment, a huge extent of features has been added like page fault handling for GPUs, transferring of data when requested, extra memory allotment for GPUs, and counters for accessing the data [10]. In the past, two distinct AutoSwap and SmartPool strategies have been applied to minimize GPU consumption and it prevents any human intervention [11]. In previous work, the different standard algorithm has been tested concerning a parallel version of ANN on compute unified device architecture (CUDA) and results shown before results in favor of parallel implementation on GPUs [12]. Matrix multiplication is the most time-consuming task when training a large dataset. To minimize computing time and to accelerate the processes during preparation, a parallelized matrix multiplication algorithm has been used [13].

In comparison to CPUs, substantial work has been undertaken to take advantage of the GPUs for tremendous computational functions. As GPUs are the most powerful approach to solve complex problems, there is a need to accelerate the hardware for ANN to improve the performance of training. The multicore environment of GPU's structure helps in attaining optimized neural network design for increasing throughput [14]. A GPU-based effective computation has been done for optimizing join-order operation for decreasing the execution time for complicated queries [15]. Stabilizing the assignment of allocated work on both CPU and GPU will improve the efficiency of the static system [16]. Although a massive amount of work has been done in the past to improve the matrix multiplication processing speed, the research association is focusing on implementing new hardware and pushing past the limit [17]. Training of deep recurrent network (DRN) has been evaluated for half-precision floating-point on CUDA [18]. A parallelized version of the back-propagation neural network (BPN) algorithm has been implemented on CUDA for GPU to predict the fluctuation rate for the foreign exchange market and compared with CPU for overall performance improvement [19]. This research work proposed a new parallel BPN algorithm to predict the level of groundwater level for the Faridabad zone on 20 years of data with GPU using CUDA framework.

2. RESEARCH METHOD

Multilayer back-propagation network works in two phases: forward and backward. In the forward phase, inputs are propagated through the input layer to the network and then the resulting vector is produced. Now this actual result is compared to the target result, if the results are distinct then an error is generated. In the Backward phase, the error generated from the feed-forward phase is used to update the values of weights until both the output matches. The machine learning approach provides an assistant to a variety of engineering fields [20]. In a sequential back-propagation network; weight adaptation was contrived to the framework based on a spontaneous deviation of error [21]. BPN algorithms have been applied to many prediction problems and have become a successful tool for engineers [22]. Traditional or sequential BPN algorithms can improve the convergence rate for better training [23].

In parallelized environment multiplication of matrices is executed on GPU to improve its acceleration. A function called kernel is used for defining the code on GPU. A kernel is executed by one or more threads in that kernel, which implies initiates kernel after splitting into different GPU threads [24]. Each thread in the kernel is having its unique id called threadId and it also defines the type of data processed. There are one or more blocks available in each kernel, and each block has one or more threads. But before the backward pass, the delta function kernel is launch so that it can be used to updates weights and bias in simultaneous accessing mode using the multithreaded environment of GPU. Figure 1 shows the parallelism in the GPU grid for artificial neural networks, representing the number of blocks in one grid and the number of threads in one block [25].

2.1. Tiling technique

The tiling technique is used to solve the square matrix multiplication problem, as in standard square multiplication algorithm one thread calculates the one component of the resultant matrix, and both the square matrices are stored on global memory, whereas in the tiling technique all the threads in block work together to replicate the two tile matrices for multiplication from global to shared memory. The structure of matrices is breaking down into tiles, which simplifies the operation of complex matrix multiplication and improves the concurrency rate [26]. Figure 2 portrays an example of tile multiplication.

Here $A_{i,k}$, $B_{k,j}$ are the two given matrices, $C_{i,j}$ is the product matrix and w is the width of one tile. As every cell in matrix C is liberated from each other, the parallel calculation can be done for the value of the cell. While multiplying the two tile matrices $A_{i,j}$ and $B_{j,k}$ `__syncthreads()` function is required to synchronize the threads being executed in separate blocks concurrently. As the overall result relies on computation done on parallel blocks, there must be a synchronism between different blocks of threads. A systematic approach

has been applied to augment the size of tile for matrix multiplication on different kernels i.e., sparse matrix-dense (SpMM) and sampled dense-dense (SDDMM) [27].

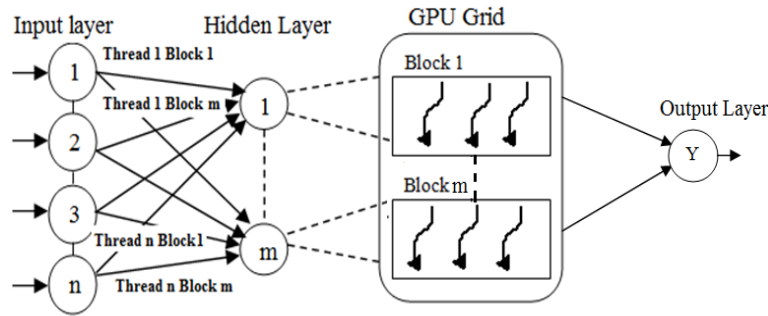


Figure 1. Artificial neural network with GPU parallelism [25]

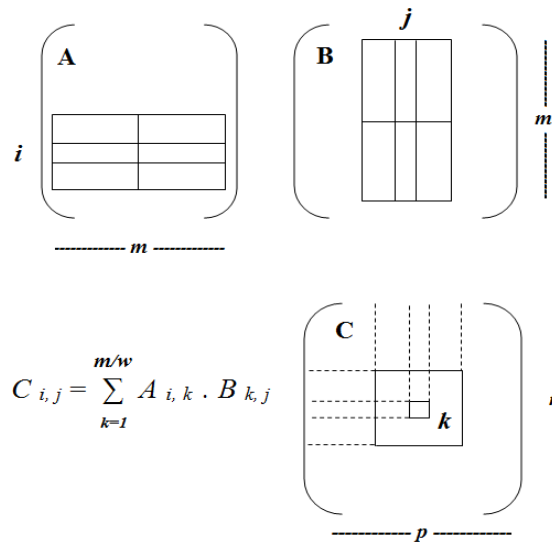


Figure 2. Tile multiplication [28]

2.2. Unified memory prefetching

Another technique used to overcome the overhead of transferring the data from host to device is unified memory prefetching; where, the data is fetched before launching the kernel. `cudaMemPrefetchAsync()` is the function used to prefetch the data from unified memory. While evaluating unified memory efficiency, an average set of delegate members, coextensive utilization is required [29]. Function `cudaMalloc()` used standard memory allocation for GPU, it returns a pointer that points to the starting of GPU memory location. But in unified memory allocation, a new function called `cudaMallocManaged()` is used that will return a pointer and is accessed by both host and device.

2.3. Coalescing technique

While executing or computing in parallel, different threads of the same block access the dynamic random access memory (DRAM) at the same time and taking together all the access and united to achieve the highest memory bandwidth is the work of coalesced technique [30]. In the coalescing technique, the row-wise method and column-wise method are used to access the elements of the matrix, i.e., row after row execution or column after column. The column-wise method is the best-suited format for GPU to provide the maximum usage ratio of 100%, when any associated column is examined, all values will influence to match the access pattern of coalesced memory. Given below coalescing technique is shown in Figure 3. In the coalescing technique address space is breached into small burst segments. When loading instruction for

execution, all threads of a warp are required and if all the thread accessing lies in the same burst segment, then that memory access coalesces as it required only one DRAM, shown in Figure 3, whereas in the un-coalescing technique, accessing the location through thread lies in different burst sections. In this work, in addition to the tiling technique on shared memory, the coalescing technique is also used.

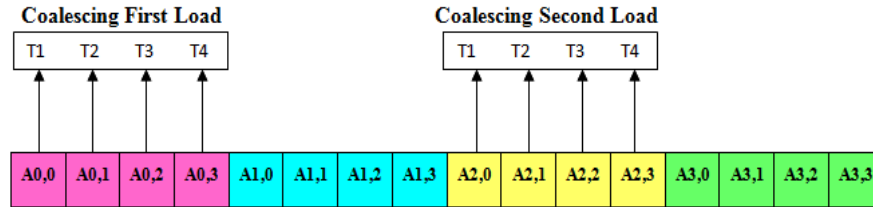


Figure 3. Coalescing technique [31]

3. THE PROPOSED ALGORITHM

Figure 4 shows the flow chart of the proposed parallelized back-propagation algorithm. For adapting the sequential nature of the BPN algorithm, there is a need to parallelize the whole algorithm. A parallelized BPN algorithm was implemented for this work to produce the ground water level prediction. Input variables are the number of substantial parameters that prevails the predicted output parameters i.e. temperature, rainfall, and ground water level has been used for input layer. Generally, network training deploys on one hidden layer. Depth of groundwater has been taken as output and all the parameters are normalized between (0.1-0.9). Activation function used was sigmoid function as it ranges between (0-1) and exclusively helpful for prediction.

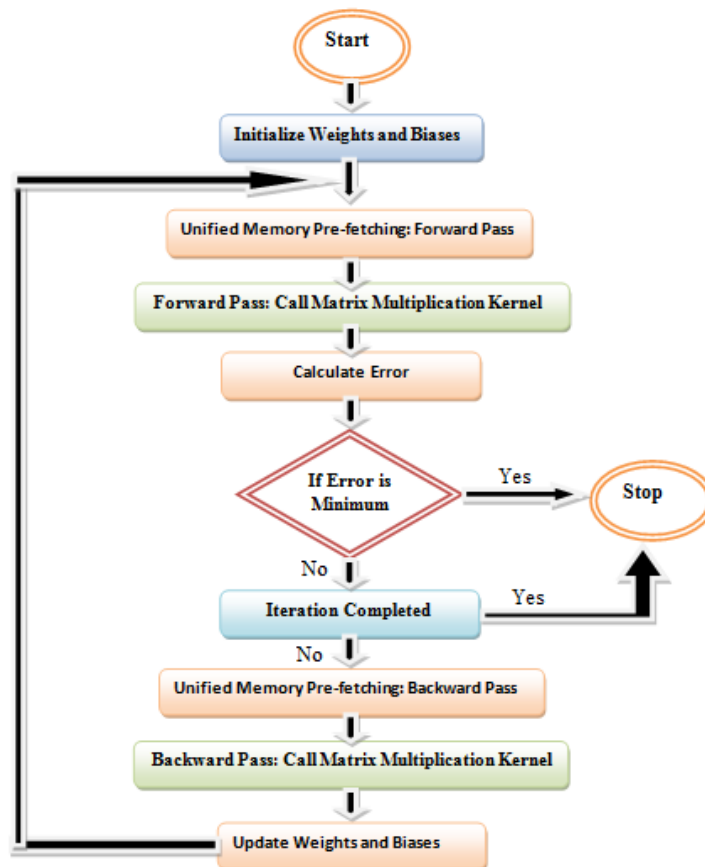


Figure 4. Flow chart for parallelized BPN algorithm

X_i - Input Matrix ()
 $W_{i,j}$ - Connected weights between layers
 T_j - Target Output (Future Groundwater Level)
 O_j - Actual Output
 E_j - Calculated error
 r - Learning rate
 Maximum number of epochs(Iteration) - 100
 θ_j - Threshold

Algorithm: proposed algorithm for parallel backward propagation on GPU

Initialize all weights and bias typically between 0 and 1

```

Step1: for i=1 to no_of_iteration do{ //repeat for every number of iteration
for j=1 to pattern do { // for every pattern in the training set
for each input LayerNetwork j{  $O_j = Net_j$ ;
Step 2: for each hidden/outputLayerNetwork j {
cudaMallocManaged(&X, N*sizeof(float));
cudaMallocManaged(&W, N*sizeof(float));
Step3: initialize data on CPU for input pattern and weights using function
cudaMemAdvise(X,count,advice,CPUdeviceId);
cudaMemAdvise(W,count,advice,CPUdeviceId);
Step 4: unified memory prefetching for forward pass from host to GPU using functions
CudaMemPrefetchAsync(X, N*sizeof(float), device,NULL);
CudaMemPrefetchAsync(W, N*sizeof(float), device,NULL);
Step5: define grid and blocks before calling a kernel
NetSumj=MatrixMultKernel<<<blocks_per_grid, threads_per_block>>> ( $O_j, X_i$ ) ;
//While configuring the blocks, 16 threadsperblock and 100 blockspergrid has been used
Step 6: calculate the weight sum of the inputs to the node by launching MatrixMultKernel( )
to multiply the two matrix using tiling technique with coalescing shared memory;
Step 7: add the threshold to the sum& calculate the activation for the node
 $Net_j = NetSum_j + \theta_j$  ;  $O_j = 1/(1 + e^{Net_j})$  ; }
Step 8: propagate the errors backward through the network
for every node j in the output layer, calculate the error for the output layer
 $E_j = O_j(1 - O_j)(T_j - O_j)$ ;
Step 9: prefetch memory from GPU to hostby using the function
CudaMemPrefetchAsync(X, N*sizeof(float), device,NULL);
CudaMemPrefetchAsync(W, N*sizeof(float), device,NULL);
Step10: Save results on GPU by using function
cudaMemAdvise(E,count,advice,GPUdeviceId);
Step11: repeat step 2 to step 7 for the hidden layer
Step 12: update weights and bias for each weight and bias
for each weight  $W_{i,j}$  and bias  $\theta_j$ 
 $\Delta W_{i,j} = r E_j X_j$ ;
 $W_{i,j} = W_{i,j} + \Delta W_{i,j}$ ;  $\theta_j = r E_j$ ;
 $\theta_j = \theta_j + \Delta \theta_j$ ; } } }
Step 13: calculate Global Error  $E = 1/2 \sum (\sum (T_k - O_k)^2)$ 
Step 14: prefetch Memory from GPU to host and save results back on GPU
CudaMemPrefetchAsync(E, N*sizeof(float), device,NULL);
CudaMemPrefetchAsync(W, N*sizeof(float), device,NULL);
cudaMemAdvise(E,count,advice,GPUdeviceId);
Step 14: while ((maximum no_of_iteration < than specified) AND (E > than specified))
End of Algorithm
  
```

4. RESULTS AND DISCUSSION

Implementation of parallelized back-propagation algorithm has been done on CUDA version 10.1 using Google Collab. Data set has been taken from [32] where total data taken into account comprises 120 rows; from 2001-2020, i.e., six annual readings skipping one month between two readings. The number of rows considered for data training was 90, while the number of rows considered for testing was 30. The prediction has been done for the next seven readings. Google collab is a data science research tool from Google. It is an open source that offers Jupyter Notebook for assessment. Users can access a variety of machine learning libraries as well as stimulating hardware [33]. Google is removing the barriers to entry into deep learning for users. Many researchers who do not have access to a large quantity of GPU resources can benefit from this tool. It allows GPU access for 12 hours at a time.

Perform the following steps in the case of a GPU-enabled notebook backend: Go to Google collab→click on runtime→change the runtime type by clicking on hardware accelerator→change the run time to GPU. An NVIDIA Tesla T4 with 2560 CUDA Cores and CUDA Version of 11.2 was used to investigate the results. The NVIDIA system management interface is depicted in Figure 5.

This segment deals with the different outcomes and the interpretation of various resultant graphs for training execution time, accuracy, error, model loss, and prediction graph over GPU. GPUs deployment is distinguishable over CPUs results. Figure 6 shows the plot for the dataset of 120 readings. Here X-axis represents the observed months concerning Groundwater level in meters at the Y-axis. Where the blue line represents the complete 120 input dataset, the orange line shows the training done by the model on the first 90 readings and the green line represents the predicted test data by model for the last 30 readings. Figures 7(a) and 7(b) shows the execution time and mean squared error (MSE) with the increasing number of epochs for both CPU and GPU. Parallelized algorithm with GPU displays better performance with a minimum error rate and execution time.

```

Wed Jun 9 04:48:23 2021
+-----+
| NVIDIA-SMI 465.27      Driver Version: 460.32.03   CUDA Version: 11.2     |
+-----+-----+
| GPU  Name      Persistence-M| Bus-Id        Disp.A   Volatile Uncorr. ECC  |
| Fan  Temp  Perf  Pwr:Usage/Cap|      Memory-Usage  GPU-Util  Compute M.  |
|                                           MIG M.         |
+-----+-----+
|  0   Tesla T4      Off          | 00000000:00:04:0 Off |    0%        0          |
| N/A   55C    P0     29W / 70W   | 104MiB / 15109MiB |              Default  |
|                                           N/A          |
+-----+-----+

Processes:
+-----+
| GPU  GI  CI      PID  Type  Process name          GPU Memory |
| ID   ID  ID              |              Usage   |
+-----+-----+

```

Figure 5. NVIDIA system management interface



Figure 6. Training and prediction

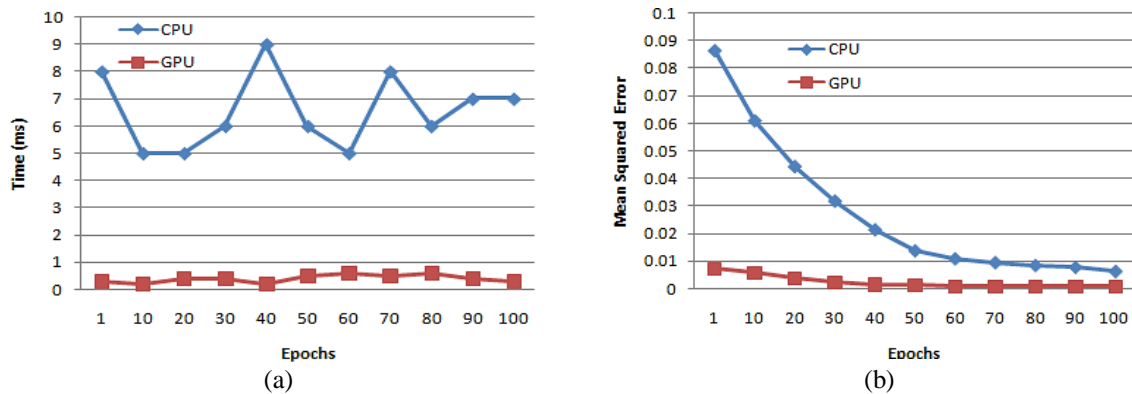


Figure 7. Comparing results for CPU vs. GPU (a) execution time comparison and (b) mean squared error comparison

Table 1 represents the type of error calculated while predicting the value of groundwater level for different parameters to evaluate the performance of different learning algorithms. The value of mean absolute error (MAE) and mean squared error (MSE) is used to check the efficiency of regression value. Whereas root mean square error (RMSE) is the error that shows the standard deviation while predicting based on data set records. To evaluate the efficiency of different standards in weather sciences, predicting atmospheric conditions, RMSE would be the regular analytical method, while MAE is good at the assessment of different models [34].

Table 1. Computational error

Error Type	Value
Mean Absolute Error	0.0696268
Mean Squared Error	0.0051229
Root Mean Squared Error	0.0715743

Figure 8 shows the execution time taken by both CPU and GPU to predict the level of groundwater level for twenty years of a dataset. It is clear from the figure that the time taken by GPU using parallelized BPN algorithm is less than the time taken by CPU alone for the same data set. The comparison between CPU vs. GPU for total execution time, average time per epoch, and memory used has been shown below in Table 2.

```

Epoch 98/100
12/12 [=====] - 0s 7ms/step - loss: 1.8031e-04 - val_loss: 0.0012
Epoch 99/100
12/12 [=====] - 0s 7ms/step - loss: 1.5398e-04 - val_loss: 0.0016
Epoch 100/100
12/12 [=====] - 0s 7ms/step - loss: 2.1770e-04 - val_loss: 0.0012
with CPU time taken in seconds: 10.076101181000013
with GPU time taken in seconds: 0.7781454485542905

```

Figure 8. CPU vs. GPU execution time

Table 2. CPU vs. GPU

Package	CPU	GPU
Total Time [sec]:	10.07610	0.77814
Average Seconds/Step:	0.014	0.006
Memory Used:	0.99 GB	1.54 GB

5. CONCLUSION

Based on the results of the aforesaid research, it can be concluded that the suggested parallelized back-propagation method on GPU predicts groundwater levels in the Faridabad region faster than the CPU alone. It should also be noted that the CPU execution time is approximately 10.08 seconds while training and testing the network and in contrast, GPU execution time reduces to approximately 0.78 seconds, which is approximately a 90.3% improvement. It can be referred from above that parallelized implementation of the GPU produces an improved performance compared to CPUs with a minimum error rate of 0.0696268.

6. FUTURE WORK

Future work includes the extension of parallelized back-propagation algorithm to other real-world problems to boost the acceleration of different hardware for ANN research and for faster GPUs; the power of various algorithms must be increased by parallelization.

REFERENCES

- [1] K. A. N. Adiat, O. F. Ajayi, A. A. Akinlalu, and I. B. Tijani, "Prediction of groundwater level in basement complex terrain using artificial neural network: a case of Ijebu-Jesa, southwestern Nigeria," *Applied Water Science*, vol. 10, no. 1, Nov. 2020, doi: 10.1007/s13201-019-1094-6.
- [2] T. Roshni, M. K. Jha, and J. Drisya, "Neural network modeling for groundwater-level forecasting in coastal aquifers," *Neural Computing and Applications*, vol. 32, no. 16, pp. 12737–12754, Jan. 2020, doi: 10.1007/s00521-020-04722-z.




- [3] Y. Go, M. Jamshed, Y. G. Moon, C. Hwang, and K. S. Park, "Apunet: revitalizing GPU as packet processing accelerator," in *Proceedings of the 14th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2017*, 2017, pp. 83–96.
- [4] Z. Zheng *et al.*, "Gen: A GPU-accelerated elastic framework for NFV," in *ACM International Conference Proceeding Series*, 2018, pp. 57–64., doi: 10.1145/3232565.3234510.
- [5] I. M. Coelho, V. N. Coelho, E. J. d. S. Luz, L. S. Ochi, F. G. Guimarães, and E. Rios, "A GPU deep learning metaheuristic based model for time series forecasting," *Applied Energy*, vol. 201, pp. 412–418, Sep. 2017, doi: 10.1016/j.apenergy.2017.01.003.
- [6] K. Raju and N. N. Chiplunkar, "A survey on techniques for cooperative CPU-GPU computing," *Sustainable Computing: Informatics and Systems*, vol. 19, pp. 72–85, Sep. 2018, doi: 10.1016/j.suscom.2018.07.010.
- [7] M. Dashti and A. Fedorova, "Analyzing memory management methods on integrated CPU-GPU systems," in *International Symposium on Memory Management, ISMM*, Jun. 2017, vol. Part F1286, pp. 59–69., doi: 10.1145/3092255.3092256.
- [8] D. T. V. D. Rao and K. V. Ramana, "Accelerating training of deep neural networks on GPU using CUDA," *International Journal of Intelligent Systems and Applications*, vol. 11, no. 5, pp. 18–26, May 2019, doi: 10.5815/ijisa.2019.05.03.
- [9] J. Jung, D. Park, Y. Do, J. Park, and J. Lee, "Overlapping host-to-device copy and computation using hidden unified memory," in *Proceedings of the ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPOPP*, Feb. 2020, pp. 321–335., doi: 10.1145/3332466.3374531.
- [10] H. Xu, M. Emani, P.-H. Lin, L. Hu, and C. Liao, "Machine learning guided optimal use of GPU unified memory," in *2019 IEEE/ACM Workshop on Memory Centric High Performance Computing (MCHPC)*, Nov. 2019, pp. 64–70., doi: 10.1109/MCHPC49590.2019.00016.
- [11] J. Zhang, S. H. Yeung, Y. Shu, B. He, and W. Wang, "Efficient memory management for GPU-based deep learning systems," *arXiv preprint arXiv:1903.06631*, 2019.
- [12] X. Sierra-Canto, F. Madera-Ramirez, and V. Uc-Cetina, "Parallel training of a back-propagation neural network using CUDA," in *2010 Ninth International Conference on Machine Learning and Applications*, Dec. 2010, pp. 307–312., doi: 10.1109/ICMLA.2010.52.
- [13] A. O. Jimale, F. Ridzuan, and W. M. N. Wan Zainon, "Square matrix multiplication using CUDA on GP-GU," *Procedia Computer Science*, vol. 161, pp. 398–405, 2019, doi: 10.1016/j.procs.2019.11.138.
- [14] S. Shi, Q. Wang, and X. Chu, "Performance modeling and evaluation of distributed deep learning frameworks on GPUs," in *2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech)*, Nov. 2017, pp. 949–957., doi: 10.1109/DASC/PiCom/DataCom/CyberSciTec.2018.000-4.
- [15] A. Meister and G. Saake, *GPU-accelerated dynamic programming for join-order optimization GPU-accelerated dynamic programming for join-order optimization*. Fakultät für Informatik, Otto-von-Guericke-Universität Magdeburg, 2020.
- [16] C. Yang *et al.*, "Adaptive optimization for petascale heterogeneous CPU/GPU computing," in *2010 IEEE International Conference on Cluster Computing*, Sep. 2010, pp. 19–28., doi: 10.1109/CLUSTER.2010.12.
- [17] B. Chen, T. Medini, J. Farwell, C. Tai, and A. Shrivastava, "Slide : in defense of smart algorithms over hardware acceleration for large-scale deep learning systems," in *Proceedings of the 3rd MLSys Conference*, 2020, vol. 91, no. 4., doi: 1903.03129.
- [18] A. Svyatkovskiy, J. Kates-Harbeck, and W. Tang, "Training distributed deep recurrent neural networks with mixed precision on GPU clusters," in *Proceedings of the Machine Learning on HPC Environments*, Nov. 2019, pp. 1–8., doi: 10.1145/3146347.3146358.
- [19] K. Ganeshamoorthy and N. Ratnarajah, "On the performance of parallel back-propagation neural network implementations using CUDA," in *Proceedings of the 32nd International Conference on Computers and Their Applications, CATA 2017*, 2017, pp. 85–92.
- [20] Q. H. Nguyen *et al.*, "A novel hybrid model based on a feedforward neural network and one step secant algorithm for prediction of load-bearing capacity of rectangular concrete-filled steel tube columns," *Molecules*, vol. 25, no. 15, Jul. 2020, doi: 10.3390/molecules25153486.
- [21] V. K. Ojha, P. Dutta, H. Saha, and S. Ghosh, "Detection of proportion of different gas components present in manhole gas mixture using backpropagation neural network," in *International Proceedings Of Computer Science and Information Technology*, 2012, vol. 37, no. Icint, pp. 11–15.
- [22] T.-A. Nguyen, H.-B. Ly, and B. T. Pham, "Backpropagation neural network-based machine learning model for prediction of soil friction angle," *Mathematical Problems in Engineering*, vol. 2020, pp. 1–11, Dec. 2020, doi: 10.1155/2020/8845768.
- [23] F. Izhari, M. Zarlis, and Sutarman, "Analysis of backpropagation neural neural network algorithm on student ability based cognitive aspects," *IOP Conference Series: Materials Science and Engineering*, vol. 725, no. 1, Jan. 2020, doi: 10.1088/1757-899X/725/1/012103.
- [24] M. Gupta *et al.*, "Compiler techniques to reduce the synchronization overhead of GPU redundant multithreading," in *Proceedings of the 54th Annual Design Automation Conference 2017*, Jun. 2017, pp. 1–6., doi: 10.1145/3061639.3062212.
- [25] N. Singh and S. P. Panda, "Enhancing the proficiency of artificial neural network on prediction with GPU," in *Proceedings of the International Conference on Machine Learning, Big Data, Cloud and Parallel Computing: Trends, Perspectives and Prospects, COMITCon 2019*, Feb. 2019, pp. 67–71., doi: 10.1109/COMITCon.2019.8862440.
- [26] G. Bansal, C. J. Newburn, and P. Besl, "Fast matrix computations on heterogeneous streams," in *High Performance Parallelism Pearls*, vol. 2, Elsevier, 2015, pp. 271–304., doi: 10.1016/B978-0-12-803819-2.00011-2.
- [27] S. E. Kurt, A. Sukumaran-Rajam, F. Rastello, and P. Sadayappan, "Efficient tiled sparse matrix multiplication through matrix signatures," in *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, Nov. 2020, pp. 1–14., doi: 10.1109/SC41405.2020.00091.
- [28] T. Athil, R. Christian, and Y. B. Reddy, "CUDA memory techniques for matrix multiplication on quadro 4000," in *2014 11th International Conference on Information Technology: New Generations*, Apr. 2014, pp. 419–425., doi: 10.1109/ITNG.2014.24.
- [29] M. Knap and P. Czarul, "Performance evaluation of unified memory with prefetching and oversubscription for selected parallel CUDA applications on NVIDIA pascal and volta GPUs," *The Journal of Supercomputing*, vol. 75, no. 11, pp. 7625–7645, Nov. 2019, doi: 10.1007/s11227-019-02966-8.
- [30] S. Ashkiani, A. Davidson, U. Meyer, and J. D. Owens, "GPU multisplit: an extended study of a parallel algorithm," *ACM Transactions on Parallel Computing*, vol. 4, no. 1, pp. 1–44, Jan. 2017, doi: 10.1145/3108139.
- [31] X. Sun, L.-F. Lai, P. Chou, L.-R. Chen, and C.-C. Wu, "On GPU Implementation of the island model genetic algorithm for solving the unequal area facility layout problem," *Applied Sciences*, vol. 8, no. 9, Sep. 2018, doi: 10.3390/app8091604.
- [32] IWRIS, "India water resources information system," 2022. Accessed: Sep. 06 2021 [Online]. Available: <https://indiawris.gov.in/wris/#/groundWater>.
- [33] T. Carneiro, R. V. M. Da Nobrega, T. Nepomuceno, G. Bin Bian, V. H. C. De Albuquerque, and P. P. R. Filho, "Performance

analysis of google colab as a tool for accelerating deep learning applications,” *IEEE Access*, vol. 6, pp. 61677–61685, 2018, doi: 10.1109/ACCESS.2018.2874767.




- [34] T. Chai and R. R. Draxler, “Root mean square error (RMSE) or mean absolute error (MAE)? – arguments against avoiding RMSE in the literature,” *Geoscientific Model Development*, vol. 7, no. 3, pp. 1247–1250, Jun. 2014, doi: 10.5194/gmd-7-1247-2014.

BIOGRAPHIES OF THE AUTHOR



Neeru Singh    currently pursuing a Ph.D. in the Department of computer science from Manav Rachna International Institute of Research and Studies (MRIIRS), Faridabad, India. She is working as an assistant professor at Rawal Institute of Engineering and Technology, Faridabad. She has 5 years of teaching experience. She has done her M.Tech. from Maharishi Dayanand University, Rohtak. Her areas are artificial neural network and heterogeneous computing. Contact her at email: neeruksingh123@gmail.com.



Supriya Priyabadini Panda    she is working as a professor and Head of the Department of Computer Science and engineering at Manav Rachna International Institute of Research and Studies (MRIIRS), Faridabad, India. She is having 35+ years of experience in the teaching field. She guides M.Tech. and Ph.D. Students in a variety of fields. She has done her Ph.D. from Ohio University, USA. Contact her at email: supriya.fet@mriu.edu.in.