Course Project, Parallel Scientific Computing, January-May 2024
Indian Institute of Technology Madras
Chennai

PARALLELIZATION OF DEEP NEURAL NETWORKS FOR STEADY-STATE HEAT CONDUCTION PROBLEMS

Ruthwik Chivukula
B.Tech., Department of Mechanical Engineering
Indian Institute of Technology Madras
Email: me21b166@smail.iitm.ac.in

Sriram Pillutla
B.Tech., Department of Mechanical Engineering
Indian Institute of Technology Madras
Email: me21b196@smail.iitm.ac.in

ABSTRACT

Conventional methods for steady-state heat conduction problems often rely on computationally expensive Computational Fluid Dynamics (CFD) solvers. This work proposes a novel approach that leverages deep neural networks (DNNs) to achieve significantly faster and thus efficient solutions. System-specific details, encompassing thermal properties and boundary conditions, serve as inputs to the network, with the output being the temperature distribution across the domain of interest. While DNNs offer a clear advantage regarding computational efficiency, the training process can be resource-intensive. Our proposed implementation leverages OpenACC and MPI together: OpenACC achieves the incremental parallelism of the gradient and loss computations in the training process, while data generation using finite-difference methods to create diverse heat conduction problems utilizes OpenMPI. This comprehensive parallelization strategy promises to significantly expedite DNN training, paving the way for its wider adoption in solving complex heat conduction problems.

NOMENCLATURE

T	Absolute temperature
i	Index of the discretized x-coordinate
j	Index of the discretized y-coordinate
$\Phi_{i,j}^{k}$	Normalized temperature as a function of position in the kth iteration
L	Half the length or breadth of the plate
$u_{i,j}^{(k)}$	Predicted normalized temperature value as a function of position for the kth data point
$\Phi_{i,j}^{(k)}$	Normalized temperature value as a function of position for the kth data point
$L(\theta)$	Loss as a function of model parameters

INTRODUCTION

Steady-state heat conduction problems are a powerful representation of more complicated transient heat transfer problems. Traditionally, the solutions to these problems are computed using various finite difference schemes. However, recent research has also moved towards using modern learning algorithms to solve them, considering the latency associated with large problem sizes. The major motivation is the boom of deep learning in the past decade, which has shown its potential in solving complex problems with high accuracy in various fields of science and engineering. These offer a non-linear approximation to the solution function, often providing an acceptable error margin. There have been many attempts at utilizing state-of-the-art deep learning techniques to solve these problems (Deng & Hwang, 2006; Kharazmi et al., 2021). These include forward and inverse problems (Jagtap et al., 2020; Deng & Hwang, 2006). This work focuses on the forward problem, where the temperature distribution is predicted given the thermal properties and boundary conditions.

The major challenge of the training bottleneck of deep networks is addressed with parallelization techniques, which have also received the limelight (Shukla et al., 2021; Yagawa & Aoki, 1995). The training process is inherently parallelizable, as the gradient computation for each layer can be done independently. This has been exploited in various works, including the use of GPUs (Shukla et al., 2021). Here, we demonstrate the speed-up in training using OpenACC. Alongside this, the data generation process is also parallelized using OpenMPI, which is used to generate diverse heat conduction problems. Finite difference schemes such as the Jacobi iterative method are used to generate the data, with parallelization providing speed-up in this process.

Thus, to summarize, we aim to solve the following problems:

1. Speed-up of the data generation process for large problem sizes.
2. Implementing DNNs to solve steady-state heat conduction problems.
3. Parallelization of the training process to improve the latency associated with the solution.

PROBLEM FORMULATION

The steady-state heat conduction setup is initialized by four boundary conditions on a solid uniform plate, as shown in Fig. 1, and is governed by the Laplace equation in two dimensions,

$$\nabla^2 \Phi(x, y) = 0,\ |x|,\ |y| < L$$

with Φ(x,y) representing the normalized temperature distribution across the domain. That is,

$$\Phi = \frac{T - T_{min}}{T_{max} - T_{min}}$$

where T_max and T_min are the universal maximum and minimum temperatures over all the data points. The boundary conditions shown in Fig. 1 are the normalized values.

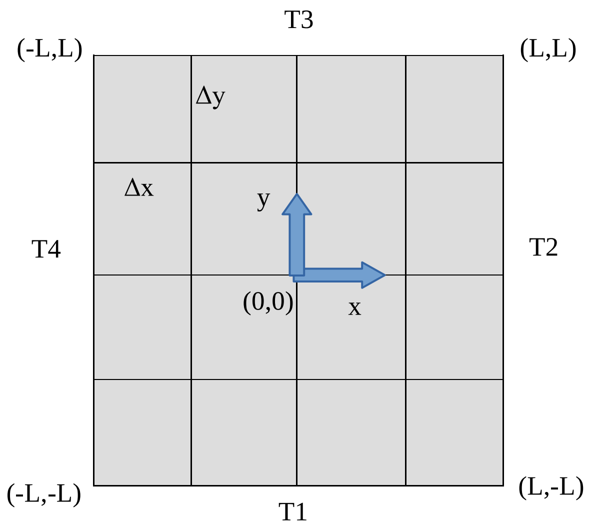


Figure 1. Discretization of the domain with boundary conditions. (Δx = Δy)

The desired neural network must take in as input the four boundary conditions for the plate, and for each node in the entire domain, predict the normalized temperature at steady state. Thus, the network approximates the solution for Φ(x,y).

DATASET GENERATION PROCESS

The steady-state solution is first obtained from the Jacobi iterative method, with the update step for $\Phi_{i,j}$ i.e., the (i, j) node, being as follows.

$$\Phi_{i,j}^{k+1} = \frac{1}{4}\left(\Phi_{i+1,j}^{k} + \Phi_{i,j+1}^{k} + \Phi_{i-1,j}^{k} + \Phi_{i,j-1}^{k}\right)$$

All the values of Φ(x,y), except the boundaries, are initialized to zero before the first iteration. In each iteration of updates across the domain, only the interior points have their values modified according to the rule.

Parallelization.

This process is parallelized using OpenMPI using row-wise block decomposition. The rows of the domain are divided among multiple processors, as shown in Fig. 2, which update the values in only those particular nodes.

Notice that since the update step requires only values from the previous iteration at every node, data parallelization can be achieved with row-wise block decomposition of the domain.

The domain consists of 441 nodes, with the axes being discretized into 21 points each, with $\Delta x = \Delta y = 0.1$m.

The dataset comprises of 1296 such temperature distributions. This includes all combinations of (T1, T2, T3, T4) for T1, T2, T3, T4 ϵ {0, 0.2, 0.4, 0.6, 0.8, 1}. For each combination, the value of Φ(x,y) is calculated at each node using the Jacobi iterative method, forming the entire dataset.

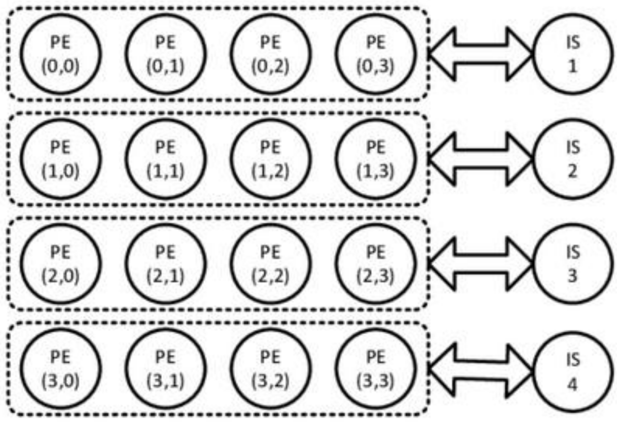


Figure 2. Row-wise block decomposition of the domain using OpenMPI. "IS" represents a processor.

DEEP NEURAL NETWORK

A neural network is effectively a series of matrix multiplications with some activation functions to induce nonlinearity. This allows it to approximate nonlinear functions with a high accuracy. It comprises layers, each of which have nodes and associated weights and biases. As a matrix flows through the layers, the weights get multiplied to it, and the biases get added. After each layer, a nonlinear activation function acts on the matrix.

In this work, the input to the neural network are the four boundary conditions, (T1, T2, T3, T4), and thus the input layer has four nodes. As the output of the network is the prediction of all the temperature values across the domain, it consists of 441 nodes. The network has a single hidden layer with 20 nodes. A schematic of the network is shown in Fig. 3.

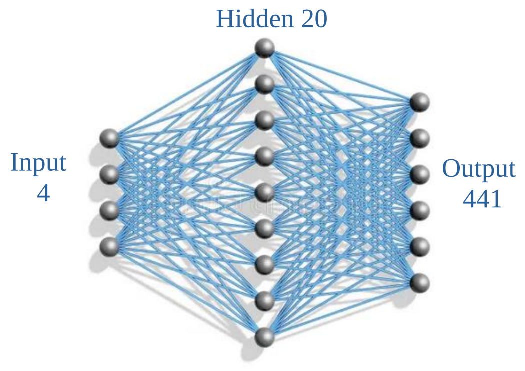


Figure 3. Neural network architecture schematic.
Image credits: Eugenesergeev, dreamstime.com

For the weight matrix w_{ih} between the input and hidden layers, the dimension would be (4, 20). With a bias b, the hidden layer is calculated by,

$$h = g(w_{ih}^{T} * x + b)$$

where h is the hidden layer matrix (20, 1) and x is the input matrix (4, 1), and *g(z)* is the activation function. In this work, the sigmoid activation is used.

$$g(z) = \frac{1}{1 + e^{-z}}$$

The output layer with 441 nodes predicts $\Phi(x,y)$ at each element in the domain. This is the **forward propagation** process.

The loss function chosen is the mean-squared error (MSE), which finds the mean square difference between the predicted values u(x,y) and the actual values $\Phi(x,y)$.

The loss is calculated as follows for N data points.

$$L(\theta) = \frac{1}{N}\sum_{k=1}^{N}\left[\sum_{i=1}^{n}\sum_{j=1}^{n}\left(u_{i,j}^{(k)}(\theta) - \Phi_{i,j}^{(k)}\right)^{2}\right]$$

This is the average of the norm of the difference between the predicted and actual values. The goal is to optimize over the parameters *theta,* which includes the weights and biases of the model, such that this loss is minimized.

The optimization is done via gradient descent, where the weights are updated by moving in the direction of the negative gradient of the loss with respect to the weights. This happens in the **backward propagation** step.

$$w_{updated} = w - \eta\frac{\partial L(\theta)}{\partial w}$$

The rate of update, or the learning rate, η is a hyperparameter, fixed to be 1.0 in this work. The forward and backward propagation steps happen once per epoch, and in our case, the number of epochs have been set to 250.

Parallelization.

The training of the neural network is parallelized using OpenACC. The only scope for parallelism exists in the forward and backward propagation steps, since the epochs are forced to be executed sequentially.

Every epoch contains the following steps:

1) Compute the prediction for each input vector after forward propagation
2) Compute the loss averaged over all data points with the predictions
3) Compute the gradients during back propagation and update the parameters

Figure 4 illustrates the levels of parallelization achievable using OpenACC. In this work, each gang takes up one epoch of computations, which involves all the data points. Within this, the workers parallelly perform computations on different sets of data points. On each set of data points, vectors parallelize the computations of forward and backward propagation (steps 1, 2 and 3). The loss calculation is inherently present in the back-propagation step. The gang calculates the loss averaged over all the data points (step 2).

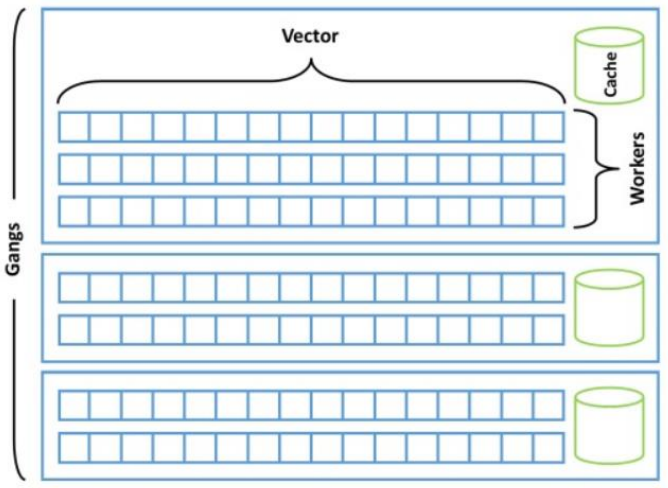


Figure 4. Parallelization levels in a GPU using OpenACC.

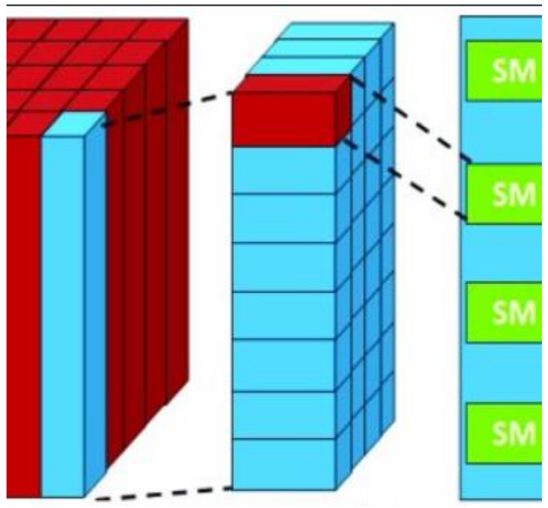


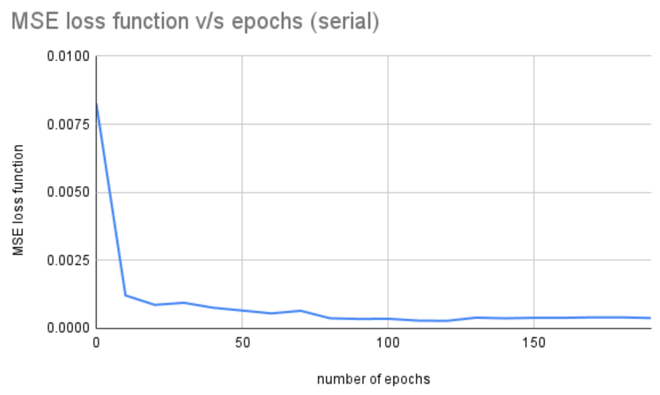
Figure 5. Data parallelization in a GPU.

The parallelization of the problem domain is illustrated in Figure 5. This is a three-dimensional analogy of how parallelization occurs. Each worker can be understood to have been allotted a blue block, and each vector works on the red sub-block that is a part of the blue block. In the case of a large number of vectors, this results in one vector performing operations with a single data point.

RESULTS

The efficiency of the training process, that is, the number of epochs taken for convergence of the neural network, remains nearly the same with OpenACC, as seen in Fig. 6. On training for 250 epochs, the MSE loss achieved by both the models is comparable.

Figure 7 illustrates the results of inference from both the serial and parallel implementations of the deep neural network. These are the temperature distribution functions as predicted by the trained neural network given only the four boundary conditions. This further reinforces the result in Fig. 6, showing that the network succeeded in learning the temperature distribution patterns given a set of boundary conditions.



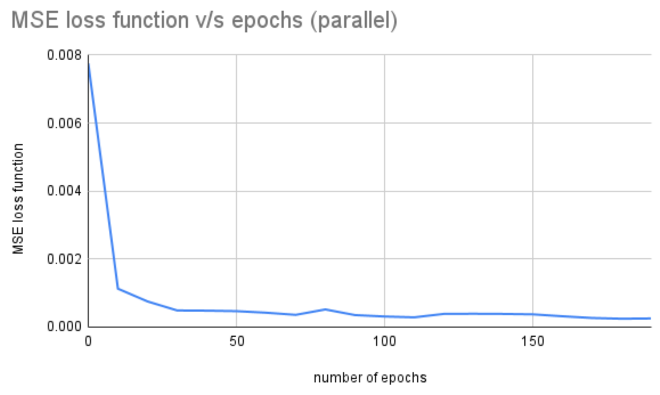


Figure 6. Loss function vs. epochs for serial and parallel implementations of the DNN.

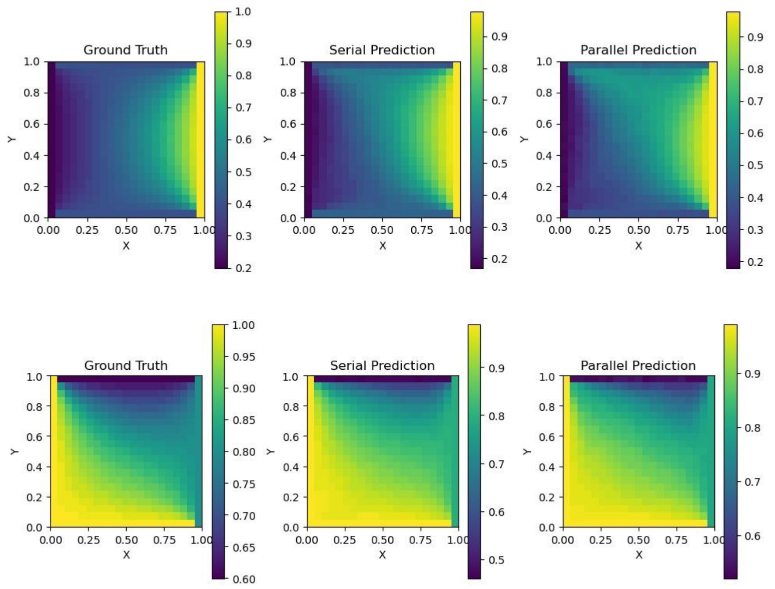


Figure 7. Predictions of the serial and parallel implementations of the DNN alongside the results from the Jacobi iterative method.

Results from Parallelization.

An important metric in parallelization is speedup, calculated as follows.

$$\psi(n,p) = \frac{\textit{Time taken with serial code}}{\textit{Time taken with parallelised code}}$$

The generation of data shows an improvement in latency using MPI in the process, with a speedup of **~2**. The time taken with the parallel code, thus, takes half the time, and is marginally better as the problem size, that is, the number of data points, increases. This is shown in Fig. 8.

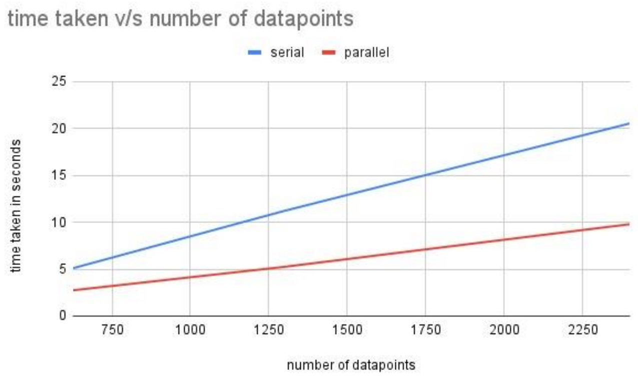


Figure 8. Time taken with different number of data points generated.

The time taken for the data generation process decreases with a higher number of processors, as shown in Fig. 9. This is shown for generating 1296 data points, which were used for the training process. The corresponding speedup values are shown in Table 1.

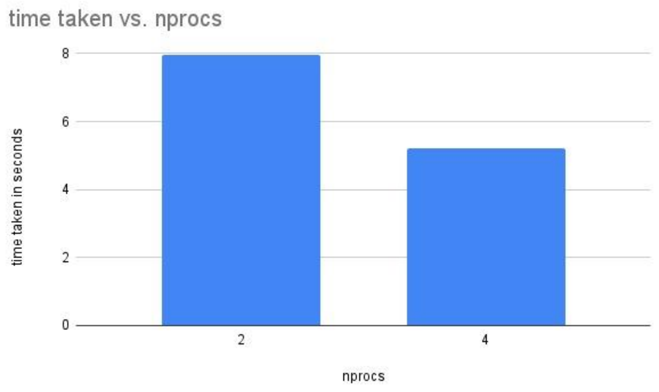


Figure 9. Time taken for data generation with different no. of processors.

Number of Processors	Speedup
2	1.405
4	2.138

Table 1. Speedup with different no. of processors for data generation.

The time taken for the training process shows a clear decrease when employing OpenACC for parallelism, taking around 3-4x lesser than the serial implementation, as shown in Fig. 10.

Comparing speedup for different numbers of epochs reveals the performance of the parallelized version. In this case, it turns out to be ~3.8, as shown in Fig. 11.

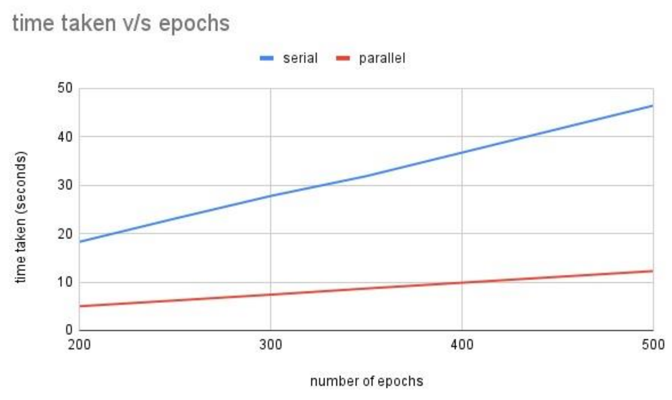


Figure 10. Time taken for training with different no. of epochs.

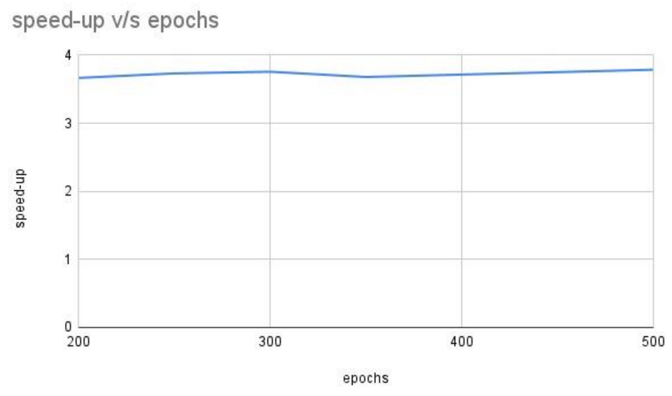


Figure 11. Speed-up in training for different numbers of epochs.

CONCLUSION

The data generation process has a significant speedup on using MPI for parallelization, and this is better with a higher problem size. Parallelizing the training process of the neural network for steady-state heat conduction problems proves to be beneficial, providing a speed-up of ~3.8. OpenACC was shown to be an effective tool in the parallelization of this problem, and thus can be exploited for related and more complicated problems, where speed-up will become a great necessity. Deep neural networks are effective in learning conduction problems under simple constraints.

ACKNOWLEDGMENTS

We sincerely thank Dr. Kameswararao Anupindi for this opportunity and this excellent learning experience.

REFERENCES

Yagawa, G., Aoki, O. (1995). A Neural Network-Based Finite Element Method on Parallel Processors. In: Batra, R.C. (eds) Contemporary Research in Engineering Science. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-80001-6_36

Ehsan Kharazmi, Zhongqiang Zhang, George E.M. Karniadakis, hp-VPINNs: Variational physics-informed neural networks with domain decomposition, Computer Methods in Applied Mechanics and Engineering, Volume 374, 2021, 113547, ISSN 0045-7825, https://doi.org/10.1016/j.cma.2020.113547

Ameya D. Jagtap, Ehsan Kharazmi, George Em Karniadakis, Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to forward and inverse problems, Computer Methods in Applied Mechanics and Engineering, Volume 365, 2020, 113028, ISSN 0045-7825, https://doi.org/10.1016/j.cma.2020.113028

Khemraj Shukla, Ameya D. Jagtap, George Em Karniadakis, Parallel physics-informed neural networks via domain decomposition, Journal of Computational Physics, Volume 447, 2021, 110683, ISSN 0021-9991, https://doi.org/10.1016/j.jcp.2021.110683

S. Deng, Y. Hwang, Applying neural networks to the solution of forward and inverse heat conduction problems, International Journal of Heat and Mass Transfer, Volume 49, Issues 25–26, 2006, Pages 4732-4750, ISSN 0017-9310, https://doi.org/10.1016/j.ijheatmasstransfer.2006.06.009