A Novel Automatic Mixed Precision Approach For Physics Informed Training

Jinze Xue
NVIDIA Corporation
jinzex@nvidia.com

Akshay Subramaniam*
NVIDIA Corporation
asubramaniam@nvidia.com

Mark Hoemmen
NVIDIA Corporation
mhoemmen@nvidia.com

Abstract

Physics Informed Neural Networks (PINNs) allow for a clean way of training models directly using physical governing equations. Training PINNs requires higher-order derivatives that typical data driven training does not require and increases training costs. In this work, we address the performance challenges of training PINNs by developing a new automatic mixed precision approach for physics informed training. This approach uses a derivative scaling strategy that enables the Automatic Mixed Precision (AMP) training for PINNs without running into training instabilities that the regular AMP approach encounters.

1 Introduction

Physics Informed Neural Networks (PINNs) [1] are used in many applications of deep learning to physical systems by training the network directly on the physical governing equations. This training paradigm requires computing higher-order derivatives of outputs of the model with respect to the inputs. This greatly increases the training cost of PINNs compared to typical data driven neural networks. One way to speed up training is using Automatic Mixed precision (AMP) [2] that combines single-precision (FP32) with half-precision (FP16) format to speed up the training process while maintaining comparable accuracy achieved with full single precision training. However, higher-order derivatives required by PINNs make it much more likely to encounter issues with values overflowing FP16's dynamic range. Further, derivatives are required in PINNs even before constructing the loss function. As a result, the gradient "scaler" used in the traditional mixed-precision training algorithm to prevent gradients from underflowing during back-propagation does not solve this derivative overflow problem during the forward pass since it only affects the gradients through loss scaling. In this work, we present a method using a separate derivative scaler for each derivative order or term that successfully solves several example problems and enables mixed-precision training for PINNs.

Mixed-precision training enables use of FP16 for matrix multiplications on tensor cores. It also reduces memory and bandwidth requirements, thus reducing time spent in bandwidth-bound layers and enabling larger batch sizes or models. Thus, successfully integrating AMP could speed up PINNs' training process significantly. In addition, this work provides a general method and insights for enabling mixed-precision training for other deep learning models that may also require computing higher order derivatives in the forward pass.

2 Related Works

Standard AMP training [2] uses a gradient scaler which can preserve small gradient values that cannot be represented in the FP16 range. However, it only tracks the neural network parameter gradients after backpropagation and does not track any derivatives during the forward pass.

*Corresponding author

Machine Learning and the Physical Sciences workshop, NeurIPS 2022.

3 Methods

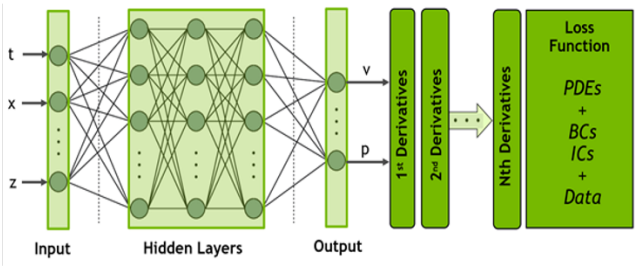


Figure 1: Physics informed neural network and derivatives necessary to formulate the loss function.

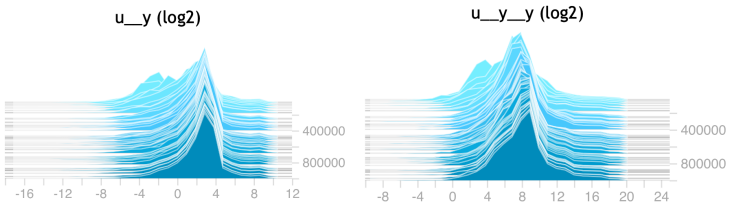


Figure 2: Dynamic range of u__y (du/dy) and u__y__y (d^2u/dy^2) in log2 for the lid driven cavity problem with zero equation model. Here, d^2u/dy^2 overflows the dynamic range of FP16 (2^{-24} to 2^{15}).

Figure 1 shows a schematic of a typical physics informed neural network (PINN) [1]. Since the losses are defined by partial differential equations (PDEs), derivatives of the output of the model with respect to the inputs need to be computed in the forward pass to formulate the loss function. The standard AMP algorithm [2] only addresses dynamic range issues in the backward pass through loss scaling and hence does not work well for PINNs. This section describes the proposed algorithm to enable mixed precision training for PINNs.

Derivative scaler Because higher-order derivatives in PINNs may overflow the dynamic range of FP16 (see Figure 2), we use scaling factors that scale the gradient input (tangent) such that the derivatives calculated using autograd are within the FP16 representable range. If INFs/NaNs are detected in the result, the scaling factor is decreased, and this iteration is skipped. Otherwise, we unscale the result to get the correct derivative values in FP32 and use them to carry out subsequent operations. If no INFs/NaNs are detected for a predefined number of training steps called the growth interval, then the scaling factor is increased to prevent it from going down to 0.

3.1 Coupled autograd graph

Using a single global derivative scaler following the above straightforward scheme could enable the training of some simple PINN networks with AMP. However, because the scaling and unscaling operations for the lower order derivatives happen during the forward pass, they are recorded in the autograd graph. And we need to make sure they do not corrupt the higher-order derivatives and the final backpropagation paths. Figure 5 in the Appendix illustrates the coupled autograd graph for a 2-layer network.

Per derivative order scalers Because of the coupled autograd graph, if there are multi-order derivatives in the model, using a single global derivative scaler can cause the scaling factor to keep decreasing every iteration until it reaches zero. This issue happens because the scaling factor of the second order derivatives get cancelled by the unscaling for the first order derivatives allowing for an INF in the backward path that is independent of the global scaling factor. We can solve this issue by using a different scaler for each derivative order. Appendix A gives more concrete proof of why this is necessary.

Per derivative term scalers The derivative terms of the same order may have different dynamic ranges; some tend to overflow while others can underflow. In this case, users have the option to use a different scaler for each derivative term.

For example, in the lid driven cavity problem with zero equation turbulence model, ν is a term composed of first order derivative terms (Eq 1). The term $d\nu/dx$ is conceptually a second order derivative term, but it has a much lower dynamic range (2^{-20} to 2^{-3}) compared to other second order derivative terms (2^0 to 2^{20}). This results in unstable training when using the same scaler for all second order derivative terms. This could be fixed by registering a special scaler for ν related derivatives.

$$\nu = \sqrt{\left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right)^2 + 2\left(\frac{\partial u}{\partial x}\right)^2 + 2\left(\frac{\partial v}{\partial y}\right)^2} \times \min(0.018, 0.419 \times \mathrm{sdf}(x, y))^2 + 0.0001 \quad (1)$$

Another solution for this example is to flatten out all the terms so that the loss function only has derivatives of output (u, v, p) over input (x, y). In this way, no intermediate terms (e.g., ν) are left in the loss function, and the per-derivative order scalers strategy would just work perfectly.

Scaling factor's range The allowed range of lower-order derivative scaling factors needs to be appropriately selected because the scaling and unscaling operations recorded in the lower-order derivatives could influence the intermediate result's dynamic range in the later autograd graph. This is best illustrated in Appendix Figure 5. When the first order derivative scaling factor, s_1, is too large (e.g., 2^{15}, as long as the first order derivatives do not overflow), the accumulated scaling factor s_2/s_1 (where s_2 is the second order derivative scaling factor) for a region of the second order derivative path would be too small and will make training unstable. We solve this by setting an default upper bound for the scaling factor as 2^0.

3.2 Controlling the scaling factor

In addition, to avoiding a very low scaling factor (e.g., 2^{-10}) that can make the training unstable, we have identified some operations that are likely to overflow the FP16 range and always perform these operations in single precision.

Fused activation function The higher order derivative of the unfused activation function can contain operations where intermediate results overflow the FP16 range. Using a fused activation function where all operations are performed within a single compute kernel can solve this issue.

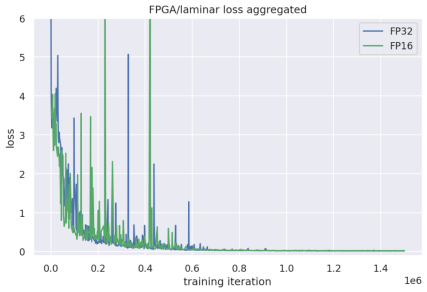
FP32 for a specific layer If a specific layer is likely to produce INFs when calculating derivatives, this layer could always use single precision instead. In our case, because the first layer always produces the final derivatives, it will always overflow if the high-order derivative overflows. Always using FP32 for the first layer makes the scaling factor much more stable.

Recovery mode Running AMP with Fourier feature networks [3] causes a problem that the gradient scaling factor occasionally decreases rapidly, but the growth interval (see [2]) is too slow to keep up. These networks are also susceptible to more dynamic range issues when using very high frequencies due to the nature of the encodings. To counter this instability, the scalers enter a "recovery mode" when the scaling factor is less than a predefined threshold. In this mode scaling factors grow more frequently thereby avoiding instabilities arising from very small scaling factors.

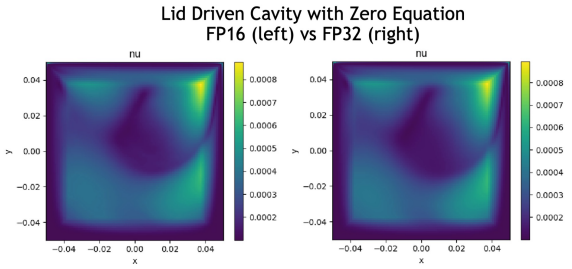
4 Benchmark and Results

We have trained various examples in Modulus [4] successfully with this proposed approach: lid-driven cavity problem, lid-driven cavity problem with zero equation turbulence model, FPGA laminar, FPGA turbulent and three fin 2d heat sink.

- **Baseline FP32** Training on A100, and using TensorFloat32 (TF32) tensor cores for matrix multiplications, all other operations use FP32.



(a) FPGA/laminar training loss curve.



(b) Predicted viscosity ν for lid driven cavity problem.

Figure 3: Comparison of mixed-precision and single precision training.

- **Mixed Precision FP16** All use the same hyper-parameters as the baseline. Using FP16 for matrix multiplications except the first layer, all other operations use FP32. Fused SiLU activation function using NVFuser with TorchScript frontend. Using per-derivative order scaling strategy with all terms flattened in the loss function (e.g., without ν).

They all give similar accuracy compared to the single-precision results. Figure 3a shows the training loss of the FPGA laminar example, and mixed-precision training accuracy matches that of FP32 training. Figure 3b shows the predicted visicosity ν for lid driven cavity problem with zero equation turbulence model, the predicted ν from the FP16 training also matches the baseline run.

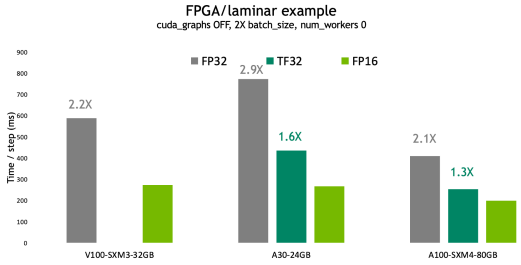


Figure 4: FPGA/laminar performance benchmark.

Figure 4 shows the benchmark result of the FPGA laminar example in Modulus. On V100, FP16 is 2.2X faster than FP32. On A100, FP16 gives 2.1X speedup compared to FP32 and 1.3X speedup compared to TF32. (TF32 is not available in V100 GPUs. It has the same numeric range as FP32 but uses the same 10-bit mantissa as the FP16 to speed up matrix multiplications.)

5 Conclusions and Future Work

Mixed precision training has become essential to speed up various deep learning models. However, it has not yet been studied for models that require high-order derivatives in the forward pass, for example, Physics Informed Neural Networks. This work presents a novel mixed precision training approach for PINNs. We demonstrated that our mixed precision approach matches the baseline models with no loss in accuracy for various examples. This work also provides insights for enabling mixed-precision training for models that require computing higher-order derivatives.

We would like to extend this work with the deferred unscaling strategy shown in Appendix Figure 6 by doing unscaling operations until all-order derivatives are computed. Future directions also include studying forward mode automatic differentiation and statistics of the per layer dynamic range for the intermediate results.

Acknowledgments and Disclosure of Funding

This work was performed at NVIDIA as a part of a summer internship project.

References

[1] Maziar Raissi, Paris Perdikaris, and George E Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational physics*, 378:686–707, 2019.

[2] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, et al. Mixed precision training. In *International Conference on Learning Representations*, 2018.

[3] Matthew Tancik, Pratul Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains. *Advances in Neural Information Processing Systems*, 33:7537–7547, 2020.

[4] Oliver Hennigh, Susheela Narasimhan, Mohammad Amin Nabian, Akshay Subramaniam, Kaustubh Tangsali, Zhiwei Fang, Max Rietmann, Wonmin Byeon, and Sanjay Choudhry. Nvidia simnet™: An ai-accelerated multi-physics simulation framework. In *International Conference on Computational Science*, pages 447–461. Springer, 2021.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See section 3.2
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] See section 4 and we will add more detailed in the final version.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See section 4 and we will add more detailed in the final version.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See section 4
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [Yes]
   (c) Did you include any new assets either in the supplemental material or as a URL? [No]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Per Derivative Order Scaler

Figure 5 shows the autograd graph for the second order derivative u__x__x. This diagram illustrates why INFs/NaNs could not be solved if we only use one single global derivative scaler. In this case, s1 is equal to s2, and there is a block in the path with accumulated scaling factor as 1 (s2/s1). As a result, if there is an INF/NaN on this block, INF/NaN will be persistent and will be independent of the global scaling factor.

B Immediate Unscaling vs Deferred Unscaling

Figure 5 and 6 show the difference between immediate unscaling and deferred unscaling strategies. The deferred unscaling strategy is more convenient for forward mode automatic differentiation.

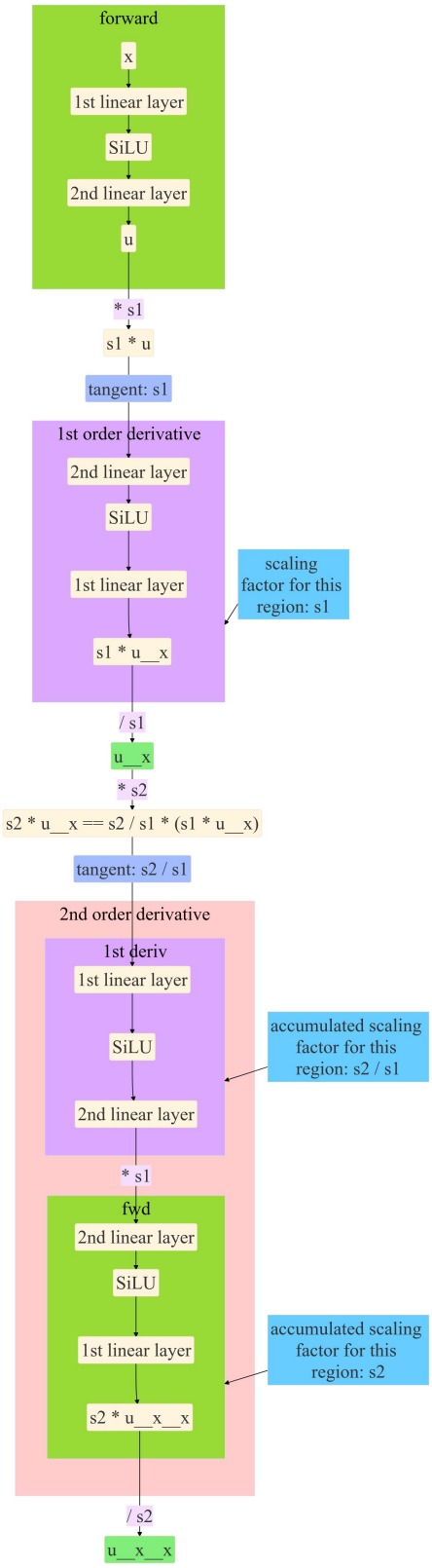


Figure 5: The autograd graph for the second order derivative u__x__x using **immediate unscaling** strategy. The 1st-order derivative scaling factor is denoted as s1, 2nd-order derivative scaling factor is denoted as s2.

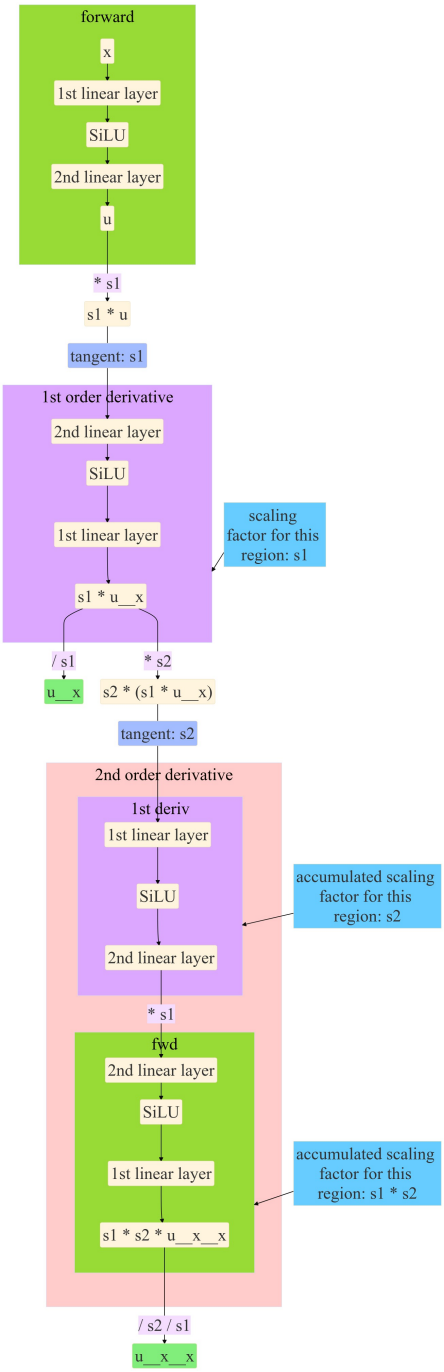


Figure 6: The autograd graph for the second order derivative u__x__x using **deferred unscaling** strategy.