PARALLEL TRAINING OF GRU NETWORKS WITH A MULTI-GRID SOLVER FOR LONG SEQUENCES

Anonymous authors
Paper under double-blind review

ABSTRACT

Parallelizing Gated Recurrent Unit (GRU) is a challenging task, as the training procedure of GRU is inherently sequential. Prior efforts to parallelize GRU have largely focused on conventional parallelization strategies such as data-parallel and model-parallel training algorithms. However, when the given sequences are very long, existing approaches are still inevitably performance limited in terms of both training time and model accuracy. In this paper, we present a novel parallel training scheme (called parallel-in-time) for GRU based on a multigrid reduction in time (MGRIT) solver. MGRIT partitions a sequence into multiple shorter sub-sequences and trains the sub-sequences on different processors in parallel. The key to achieving speedup is a hierarchical correction of the hidden state to accelerate end-to-end communication in both the forward and backward propagation phases of gradient descent. Experimental results on the HMDB51 dataset, where each video is an image sequence, demonstrate that a new parallel training scheme of GRU achieves up to $6.5\times$ speedup over a serial approach. As efficiency of our new parallelization strategy is associated with the sequence length, our parallel GRU algorithm achieves significant performance improvement as the length of sequence increases. Further, the proposed approach can be applied simultaneously with batch and other forms of model parallelism.

1 INTRODUCTION

Recently, the model complexity of Deep Neural Networks (DNN) have grown to keep pace with the increasing scale of data. Hence, parallelization of DNN is becoming important to achieve high quality models in tractable computational run times (Ben-Nun & Hoefler, 2019). To this end, the main focus of this paper is parallelization of the Gated Recurrent Unit (GRU) applied to classification problems. GRU is a recurrent neural network (RNN) that is used for processing sequential data as input and capturing temporal dependencies sequence data.

Parallel algorithms for training neural networks are generally broken into two classes; *data-parallelism* distributes a batch across processors, and *model-parallelism* distributes the architecture across processors (Ben-Nun & Hoefler, 2019). Data-parallelism inherent in mini-batch stochastic gradient descent (SGD) is the primary strategy for parallelizing GRU (You et al., 2019; Huang et al., 2013). However, the accuracy of mini-batch SGD degrades as the number of sequences in each mini-batch increases. Moreover, although multiple mini-batches are processed in parallel, the training procedure of each sequence (within a mini-batch) is invariably sequential. Alternatively, the GRU model can be parallelized across hidden layers (Wu et al., 2016; MXNet, 2018). This model-parallelism approach is beneficial when the network requires a large number of layers. However, in practice, a large-scale recurrent neural networks such as Google Neural Machine Translation (GNMT) developed by Wu et al. (2016) have 8 LSTM layers. Therefore, the scalability of model-parallelism is limited by the number of layers. A common problem in data-parallel and model-parallel algorithms for GRU is that the training execution time increases with the sequence length (i.e., the number of data samples in each sequence). At the heart of this difficulty is a fundamental need to process sequences in a serial fashion, yielding forward and backward propagation algorithms that are inherently sequential, and thus hard to parallelize.

We propose a novel parallel-in-time (PinT) training method for GRU networks with long sequences. To achieve this, we adapt a multigrid reduction in time (MGRIT) solver for forward and backward

propagation of the network. Within the numerical methods community a significant resurgence in PinT has recently occurred as a result of increasing computational resources (Gander, 2015; Ong & Schroder, 2020). These efforts have even been applied to the training of Neural ODEs (Schroder, 2017; Gunther et al., 2020; Kirby et al., 2020), where inexact forward and back propagation is exploited to achieve parallel speedups. To date, these techniques have not been applied to RNNs.

Following Jordan et al. (2021), GRU networks can be written as an ordinary differential equation (ODE) to facilitate application of MGRIT. Different from existing parallel training algorithms for RNNs, our MGRIT parallel training scheme partitions a sequence into multiple shorter sub-sequences and distributes each sub-sequence to different processors. By itself this provides local improvement of the hidden states, yet global errors remain. To correct these errors, propagation is computed on a sub-sequence requiring less computational work but providing an improvement to the original hidden states. This process is iterated to the accuracy required for training. Applying this algorithm to the classic GRU network will achieve parallelism. However, the algorithm will not yield well trained networks. To fix this we develop a new GRU architecture, or discretization of the ODE, we refer to as *Implicit GRU* that handles stiffness in the architecture. This is required for application of the MGRIT algorithm where sub-sequences providing the correction, correspond to discretizations with more severe numerical stability restrictions.

Two datasets are chosen for evaluation of the PinT algorithm. The UCI-HAR dataset is used to demonstrate the algorithm can obtain similar accuracy to classical training approaches, while also demonstrating speedup. Evaluation with the HMDB51 video dataset demonstrates that the parallel training scheme achieves up to $6.5\times$ speedup over the serial approach while maintaining accuracy.

The paper is organized as follows. In Section 2, we present background of GRU network and how the classic GRU can be reformulated as an ODE description. Section 3 provides details of MGRIT method and our PinT training scheme for GRU. In Section 4 we compare the performance of the proposed approach on the sequence classification tasks. Finally, we discuss further extensions of our PinT training algorithm in Section 5.

2 Gated Recurrent Unit

The GRU network, proposed by Cho et al. (2014), is a type of RNN and a state-of-the-art sequential model widely used in applications such as natural language processing and image/video classification. Each GRU cell at time t takes input data x_t and the previous hidden state h_{t-1}, and computes a new hidden state h_t. The cell resets or updates information carefully regulated by gating units in response to prior hidden state and input sequence data stimulation as follows:

$$\begin{aligned} r_t &= \sigma(W_{ir}x_t + b_{ir} + W_{hr}h_{t-1} + b_{hr}) \\ z_t &= \sigma(W_{iz}x_t + b_{iz} + W_{hz}h_{t-1} + b_{hz}) \\ n_t &= \varphi(W_{in}x_t + b_{in} + r_t \odot (W_{hn}h_{t-1} + b_{hn})) \\ h_t &= z_t \odot h_{t-1} + (1 - z_t) \odot n_t. \end{aligned} \tag{1}$$

We refer to this method as *Classic GRU* in the text.

GRUs to ODEs Recent work has shown how RNN can be modified to obtain an ODE description (Chang et al., 2019; Habiba & Pearlmutter, 2020). Of particular interest here is those works for GRU networks (De Brouwer et al., 2019; Jordan et al., 2021) where the ODE is

$$\partial_t h_t = \underbrace{-(1 - z_t) \odot h_t}_{\text{stiff mode}} + (1 - z_t) \odot n_t. \tag{2}$$

A detailed description of the behavior of this network can be found in Jordan et al. (2021). For our purposes, note that the first term represents a stiff mode in the dynamics that biases the hidden state h towards a manifold specified by the data. The second term injects changes from the data. Finally, the term z_t controls the decay rate towards the manifold for each component of the hidden state. If z_t is near 1, then the rate is slow, while if z_t is closer to zero the decay rate is rapid and the first term can be stiff constraining the choice of time step (see Wanner & Hairer (1996) for discussion on the numerical treatment of stiff ODE's).

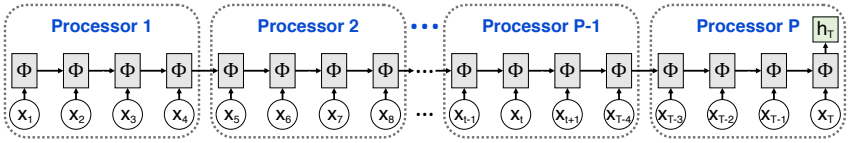

Figure 1: Parallel distribution of work in parallel-in-time training.

The classic GRU method in Eq. 1 is rederived using an explicit Euler scheme with a unit time step to integrate Eq. 2. Adding and subtracting h_{n-1} on the right hand side of the hidden state update, and for notational purposes multiplying by $\gamma = 1$, yields

$$h_t = \Phi_\gamma(x_t, h_{t-1}) := h_{t-1} + \gamma\left((1 - z_t) \odot h_{t-1} + (1 - z_t) \odot n_t\right). \tag{3}$$

The notation Φ_γ, introduced here, denotes the classic GRU hidden state update if $\gamma = 1$.

3 Parallel-in-Time Training

Training GRU networks is typically done with the BPTT algorithm (Sutskever, 2013). This algorithm computes a forward evaluation of the neural network, followed by backpropagation to compute the gradient. Both forward and backward propagation are computationally expensive algorithms that don't naturally lend themselves to parallelization in the time domain. Despite this we develop a methodology achieving the parallel distribution shown in Figure 1. This distribution is enabled by computing the forward and backward propagation inexactly, trading inexactness for parallelism and ultimately run time speedups. MGRIT, a multigrid method, is used to compute the hidden states for both forward and backward propagation.

Multigrid methods are used in the iterative solution of partial differential equations (Brandt, 1977; Douglas, 1996; Briggs et al., 2000; Trottenberg et al., 2000). Recent advances multigrid, have yielded the MGRIT method achieving PinT evolution of ODEs like Eq. 2 (see for instance Falgout et al. (2014); Dobrev et al. (2017); Gunther et al. (2020)). Thus multigrid and MGRIT methods are reasonably well developed; however, they remain quite complex. Here, we briefly outline the algorithm for forward propagation and refer the reader interested in further exploration to other references. In particular, the publications Gunther et al. (2020); Cyr et al. (2019); Kirby et al. (2020) are focused on neural networks, while the papers Falgout et al. (2014); Dobrev et al. (2017) provide an overview of the theory and practice of the MGRIT algorithm employed here. In this work, we focus on the key advance that enables MGRIT to be used in training GRU networks. In that respect, Section 3.3 proposes an implicit formulation for GRU that handles stiff modes and enables MGRIT even when time steps on the order of unity are chosen for propagation.

3.1 Notation and Preliminaries

Denote a sequence by $\vec{u} = \{u_t\}_{t=0}^T$ where the length T will be clear from context, and $u_0 = 0$ by assumption. This notation is used for the input data sequence $\vec{x}$ and the sequence of hidden states $\vec{h}$. The GRU forward propagator in Eq. 3 can be written as the system of nonlinear algebraic equations

$$f(\vec{h}) = \{h_t - \Phi_1(x_t, h_{t-1})\}_{t=1}^T. \tag{4}$$

When $f(\vec{h}) = 0$, then $\vec{h}$ is the hidden state for the GRU network with input sequence $\vec{x}$.

A two-level multigrid algorithm uses a *fine* sequence on level $l = 0$, corresponding to a data input of length T. The *coarse* level, defined on level $l = 1$, has data of length T/c_f where c_f is an integer coarsening factor. Sequences are transferred from the fine to coarse level using a *restriction* operator

$$\vec{U} = \mathcal{R}(\vec{u}) := \{u_{c_f t}\}_{t=1}^{T/c_f}. \tag{5}$$

Applying $\mathcal{R}$ to $\vec{x}$ gives a coarse sub-sequence of the original. Time points on the fine level that are included in the coarse level are referred to as *coarse points* (see Figure 2). Transfers from coarse to fine grids are denoted by the *prolongation* operator

$$\vec{u} = \mathcal{P}(\vec{U}) := \{U_{\lfloor t/c_f \rfloor}\}_{t=1}^T. \tag{6}$$

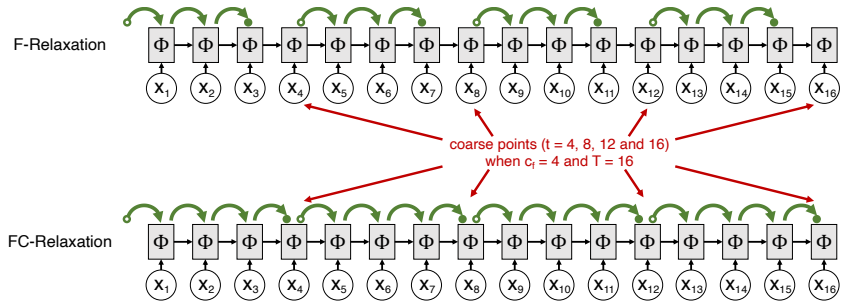

Figure 2: For $c_f = 4$ with $T = 16$, this image shows F- and FC-relaxation. In both cases forward propagation is performed serially starting at the open green circle and ending at the closed one. Each of these propagation steps is performed in parallel, yielding four-way parallelism.

Prolongation fills intermediate values using replication of the prior coarse point. The nonlinear system from Eq. 4 is modified using $\Delta t = c_f$

$$f_c(\vec{H}) = \left\{ H_t - \Phi_{c_f}(X_t, H_{t-1}) \right\}_{t=1}^{T/c_f} . \tag{7}$$

Notice that in addition to a larger time step, the sequence $\vec{H}$ length is shorter relative to the fine grid.

Finally, we introduce the notion of *relaxation* of the hidden state. Here we define this formally, while in Figure 2 a schematic view of these algorithms is taken. Relaxation applies propagation to an initial condition for the hidden state in a parallel fashion. F*-relaxation* is defined in Algorithm 1 as $\Psi_F(c_f, \vec{x}, \vec{h})$ and demonstrated in the top of Figure 2. F-relaxation uses the the hidden state at the coarse points, and fills in the fine points up to the next coarse point (exclusive). FC-relaxation, Ψ_{FC}, is identical to F-relaxation except the next coarse point is included (the bottom row in Figure 2, where closed circles fall on the coarse grid point). Combining, F- and FC-relaxation, yields FCF-*relaxation* used in MGRIT to ensure sequence length independent convergence

$$\vec{h}' = \Psi_{FCF}(c_f, \vec{x}, \vec{h}) := \Psi_F(c_f, \vec{x}, \Psi_{FC}(c_f, \vec{x}, \vec{h})). \tag{8}$$

The available parallelism for all relaxation methods is equal to the number of coarse points.

Algorithm 1: $\Psi_F(\gamma, \vec{x}, \vec{h})$ - F-relaxation

1. **parallel_for** $t_c \leftarrow 1$ **to** T/γ
2. | $t = \gamma t_c$
3. | $h'_t = h_t$
4. | **for** $\tau \leftarrow t$ **to** $t + \gamma - 1$
5. | | $h'_{\tau+1} = \Phi_\gamma(x_\tau, h'_\tau)$
6. **return** $\vec{h}'$

3.2 Multi-grid in Time

Algorithm 2 presents pseudocode for the two-level MGRIT method for forward propagation. A related schematic is presented in Figure 3 where line numbers are included in parentheses. In the figure, the fine grid on the top corresponds to an input sequence of length 16. It is depicted as being distributed across 4 processors. The coarse grid representation is a sub-sequence of length 4, its propagation is limited to a single processor. Red arrows in Figure 3 indicate grid transfers, while green arrows indicate relaxation and propagation through the network.

The goal of Line 1 in Algorithm 2 is perform a relaxations on the initial guess and improve the fine grid hidden state. FCF-relaxation is typically used to obtain sequence length independent convergence, though often on the fine level F-relaxation is used. Regardless, relaxation is applied in parallel yielding a processor local improved hidden state. The fine level residual given the relaxed hidden state is

$$\vec{r} = f(\vec{h}'). \tag{9}$$

Algorithm 2: MGProp$(\vec{x}, \vec{h})$ - Forward propagation using multigrid

// Relax the initial guess

1. $\vec{h}' = \Psi_{FCF}(\vec{x}, \vec{h})$

// Transfer approximate hidden state to the coarse grid

2. $\vec{r}_c = \mathcal{R}(f(\vec{h}'))$
3. $\vec{h}'_c = \mathcal{R}(\vec{h}')$

// Compute coarse grid correction, return to fine grid

4. Solve for $\vec{h}^*_c$: $f_c(\vec{h}^*_c) = f_c(\vec{h}'_c) - \vec{r}_c$
5. $h'' = h' + \mathcal{P}(h^*_c - h'_c)$

// Update the fine grid solution, and relax on fine level

6. **return** $\Psi_F(\vec{x}, \vec{h}'')$

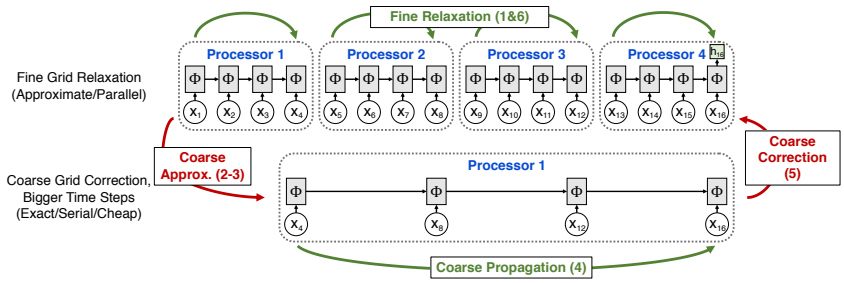

Figure 3: The two level MGRIT method. Green arrows represent relaxation, and forward propagation. Grid transfers are represented by red arrows. Numbers in parentheses are lines in Algorithm 2.

Let $\vec{h}^*$ be the forward propagated hidden state that satisfies $f(\vec{h}^*) = 0$. This is easily computed by serial forward propagated. Rewriting $\vec{r}$ in terms of $\vec{h}'$ and $\vec{h}^*$ gives

$$\vec{r} = f(\vec{h}') - f(\vec{h}^*). \tag{10}$$

Coarsening this expression using the restriction operator from Eq. 5, we arrive at an equation for the coarse grid approximation of $\vec{h}^*$

$$f_c(\vec{h}^*_c) = f_c(\mathcal{R}(\vec{h}')) - \mathcal{R}(r). \tag{11}$$

The coarse grid approximation is computed using Eq. 11 in Line 4 of the algorithm. This computation has reduced run time relative to the fine grid because the input sequence is shorter. Line 5 transfers a coarse approximation to the fine grid. The final step in the algorithm is F-relaxation on Line 6 that propagates the corrected solution from the coarse points to the fine points.

Remark 1 (Multiple Iterations)**.** *MGRIT is an iterative algorithm. One pass through Algorithm 2 may not provide sufficient accuracy in the hidden state for use in neural network training. However, in Gunther et al. (2020), relatively few iterations were needed for successful training.*

Remark 2 (Beyond Two-Level)**.** *As presented this is a two level algorithm. The coarse grid correction in line 4 provides an end-to-end correction at a reduced run-time of T/c_f. However, for large T this may be prohibitively expensive. The solution is to recognize that the coarse grid problem itself can be solved using MGRIT in a recursive fashion. Thus new levels are added to reduce the run time of the coarse problem. The run time of level l with T time steps scales as $c(l) = T/P + c(l+1)$, where P is the number of processors. Thus the total run time for L levels scales as*

$$c(0) = \sum_{l=0}^{L-1} \frac{T}{Pc_f^l} \leq \frac{T}{P} \sum_{l=0}^{\infty} \frac{1}{c_f^l} \leq \frac{T}{P} \frac{c_f}{c_f - 1}. \tag{12}$$

3.3 Coarse Grid Propagators

The discussion above assumes that all levels use the classic GRU propagator in Eq. 3 (forward Euler for Eq. 3 with time step of $\Delta t = 1$). This presents a problem for coarsening. Consider only the stiff

term in the dynamics from Eq. 2 and assume that $z \in [0, 1]$ is constant for all time. This results in the model ODE $\partial_t h = -(1 - z) \odot h$. Discretizing with forward Euler, stable time steps satisfy

$$|1 - \Delta t(1 - z)| \leq 1 \Rightarrow 0 \leq \Delta t \leq 2(1 - z)^{-1}. \tag{13}$$

The model ODE shares the stiff character of the GRU formulation, and the time step is restricted to be $\Delta t < 2$ assuming the worse case $z = 0$ ($z = 1$ implies there is no time step restriction). On the fine grid with $\Delta t = 1$ this is not unstable. However, on coarse grids, where the time step is equal to c_f^l, stability of forward and back propagation is a concern. In practice, we found instabilities on coarse grids materialize as poor training, and generalization.

To alleviate problems with stiffness, the traditional approach is to apply an implicit method. For instance, applying an backward Euler discretization to the model ODE yields a stability region of

$$|1 + \Delta t(1 - z)|^{-1} \leq 1. \tag{14}$$

Thus the time step restriction is unlimited for the model problem (for $1-z < 0$ the ODE is unstable).

A backward Euler discretization of Eq. 2 requiring a nonlinear solver is possible; however, it is likely needlessly expensive. We hypothesize, considering the model problem from above, that the stiffness arises from the first term in Eq. 2. The second term incorporates the input sequence data, changing the hidden state relatively slowly to aggregate many terms over time. Finally the nonlinearity in z may be problematic; however, this term primarily controls the decay rate of the hidden state.

As a result of this discussion, we develop an implicit scheme to handle the term labeled "stiff mode" in Eq. 2. The discretization we use computes the updated hidden state to satisfy

$$(1 + \Delta t(1 - z_t)) \odot h_t = h_{t-1} + \Delta t(1 - z_t) \odot n_t. \tag{15}$$

We refer to this method as *Implicit GRU*. Here the term $(1 - z_t)$ on the left hand side and the right hand side are identical and evaluated at the current data point x_t and the old hidden state h_{t-1}. Compared to classic GRU, this method inverts the left-hand side operator to compute the new hidden state h_t. This is easily achieved because $(1 + \Delta t(1 - z_t))$ is a diagonal matrix. Thus the evaluation cost of Eq. 15 is not substantially different then Eq. 3. Yet the (linear) stiffness implied by first term in the ODE is handled by the implicit time integration scheme.

To simplify notation, Eq. 15 is rewritten by defining an update operator Φ_γ (similar to Eq. 3)

$$h_t = \Phi_\gamma(x_t, h_{t-1};) := (1 + \gamma(1 - z_t))^{-1} \odot (h_{t-1} + \gamma(1 - z_t) \odot n_t) \,. \tag{16}$$

the subscript γ is the time step size, and the learned parameters are identical to classic GRU. This propagator replaces the classic GRU propagator in all levels of the MGRIT algorithm.

3.4 Training with MGRIT

Training using MGRIT follows the traditional approaches outlined in Goodfellow et al. (2016). In fact, the MGRIT algorithm has been implemented in PyTorch (Paszke et al., 2019), and the ADAM implementation is used without modification. The difference comes in the forward propagation, and backward propagation steps. These are replaced with the MGRIT forward and backward propagation. However, the iterations used for these steps are much fewer then would be required to achieve zero error in the propagation. Following Gunther et al. (2020), we found that using 2 MGRIT iterations for the forward propagation, and 1 iteration for back propagation was all that is required to reduce the training loss. Further generalization errors are on par with that seen with serial training.

Remark 3 (Serial vs. Parallel Inference)**.** *As training with MGRIT is inexact, a question arises: does inference with the network evaluated in parallel differ from when the network is evaluated in serial? Our results below indicate consistency between serial and parallel inference even when training is done with MGRIT. This suggests that error introduced by MGRIT is not a severe limitation.*

4 Experimental Evaluation

In this section, we evaluate the performance our new parallel-in-time training approach on the sequence classification tasks such as human activity recognition and video classification. All the experiments were run on a 2.3 GHz Intel Xeon, 2 Sockets, 18 Cores each (36 cores per node). Further, for the video classification data 9 OpenMP threads were used per MPI rank (4 ranks per node).

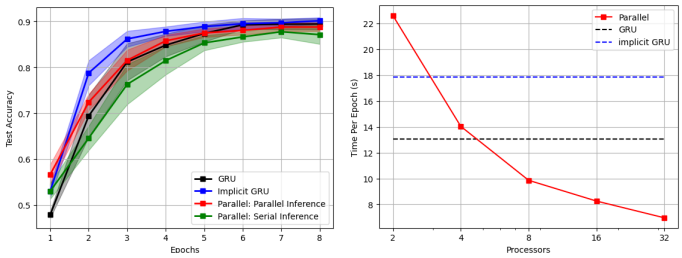

Figure 4: For the UCI-HAR dataset: (left) Accuracy for classic, implicit and parallel training of a GRU network on the UCI-HAR dataset. (right) Run time per epoch for training.

We used two publicly available datasets – UCI-HAR (Anguita et al., 2013) and HMDB51 (Kuehne et al., 2011). The UCI-HAR dataset is a collection of 10299 sequences of human activities labeled with 6 activities such as walking, walking upstairs, walking downstairs, sitting, standing, and laying. Each sequence consists of 128 consecutive input data samples of size 9 measuring three different types of motion recorded by smartphone accelerometers. The HMDB51 dataset contains approximately 6700 video clips distributed in 51 human activities. In several of our experiments, to limit computational costs, a subset of classes are used (chew, eat, jump, run, sit and walk). Further for consistency, each video clip is converted (truncated/padded) into 128 frames. After generating image sequences, each image of size 240×240 pixels is passed through the ResNet18, 34 or 50 pre-trained on ImageNet to extract features for the GRU network (Deng et al., 2009; He et al., 2016).

The software implementation of the MGRIT GRU propagation algorithm is a combination of the C++ library [redacted] and PyTorch. The combined capability is referred to as [redacted][1]. The implementation leverages performance enhancements developed for the classic GRU within PyTorch. While the implicit GRU has been carefully implemented, further optimizations are certainly possible to achieve greater computational performance. A GPU implementation of the MGRIT algorithms is under development. Finally, its important to note that the time parallel implementation does not interfere substantially with the more traditional data and model forms of parallelism.

4.1 UCI-HAR Dataset

In Figure 4, the left image shows the test accuracy variability over 16 training runs for the UCI-HAR data set. Solid lines are mean accuracy, while the colored shadows indicate one standard deviation. The classic GRU (black) achieves 90% accuracy. Implicit GRU (blue line) does as well though with a more rapid increase as a function of epochs. When training with MGRIT three levels are with a coarsening rate of 4 are used for the length 128 sequences. This yields a hierarchy of grids of size 128, 32, and 8. Here we see that the the test accuracy using parallel inference (red line) is modestly lower then purely serial case. Inference with the serial algorithm suffers a little more. Finally, the plot on the right demonstrates a roughly $2\times$ speedup over the classic GRU method in serial (black dashed line) when using the MGRIT parallel algorithm. We note that this data set is very small, and likely suffers from substantial effects of Amdahls' law (see Sun & Ni (1993) for discussions of Amdahl's law). Thus even achieving $2\times$ speedup in this case is notable. Beyond 32 MPI ranks there is no further speedup due to communication swamping any potential gains from parallelism.

4.2 HMDB51 Dataset

The first HMDB51 result is an examination of the number of iterations required to converge the MGRIT forward and backward prorogation. The norm of the fine level residual (see Eq. 4) as a function of iterations for a representative batch of size 100 is plotted in Figure 5. Here rapid convergence is demonstrated achieving roughly a digit of accuracy per iteration. Its notable that the backward computation even after one iteration is substantially better then the forward. Based on these convergence plots and experimentation, only 2 iterations for forward MGRIT propagation and 1 iteration of backward MGRIT propagation were required to obtained sufficient accuracy for

[1] Available at https://github.com/[redacted]

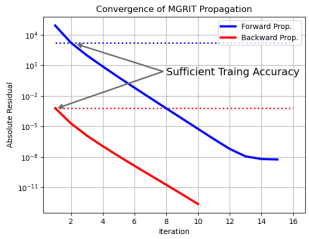

Figure 5: For HMDB51: The rapid convergence of of MGRIT for forward and backward propagation. Sufficient accuracy for training is achieved with 2 forward, and 1 backward iteration.

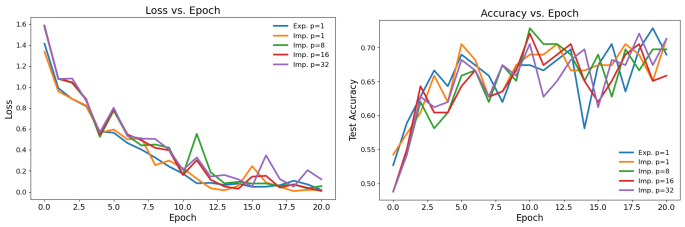

Figure 6: For a Subset of HMDB51: The loss and accuracy of the training with fixed learning rate for the classic GRU method and implicit GRU using the parallel MGRIT propagation algorithm.

training (as indicated in the figure). Further iterations did not noticeably improve the accuracy of training; a similar was observed in Gunther et al. (2020) and replicated in Kirby et al. (2020).

Figure 6 shows the loss and accuracy for a subset of the dataset. In all cases the ADAM optimizer is used, and the learning rate is a fixed 10^{-3}. In this result, the loss and accuracy for the parallel algorithms are nearby the result obtained for classic GRU, with variations coming from both different seeds and the use of the MGRIT algorithm. Note that due to the choice of relaxation schemes, the use of different processor counts doesn't yield a different algorithm.

To further study the parallel performance of MGRIT training of GRU, we consider now the full HMDB51 data set. Figure 7 shows the run time per batch (with a batch size of 100) when different pretrained ResNet architectures. In all cases, implicit GRU networks are trained with 32 MPI ranks and 9 OpenMP threads, achieving roughly $6.5\times$ speedup over the classic GRU algorithm trained with 9 OpenMP threads. The largest amount of speedup occurs after only 4 MPI ranks. Figure 8 shows a breakdown of those times for the ResNet18 case. This *log* plot shows consistent gains with parallelism for the total (red), forward (green), backward (blue) and disk compute times (orange). The expense of the forward simulation, even for classic GRU on the far left is dominated by the evaluation of the ResNet at each time step. The backward propagation cost is reduced because sensitivities with respect to the ResNet are not required. Finally, the disk time can be reduced by simply storing these in memory or overlapping computation and disk read. Note that both of these options are available to the MGRIT training as well. The robustness of MGRIT training applied to

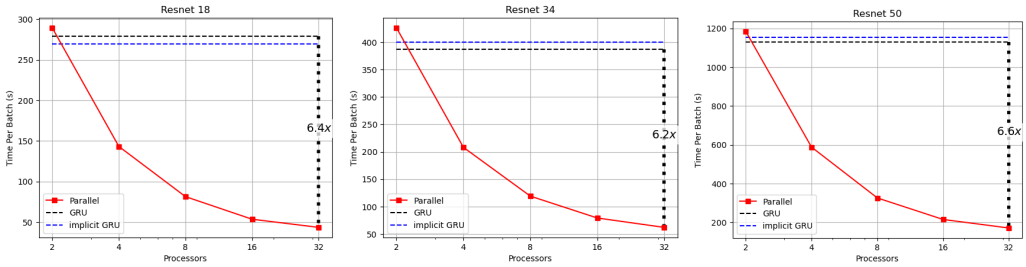

Figure 7: For HMDB51: Per batch run times per batch for training of a GRU network with ResNet18, 34 and 50 (left to right).

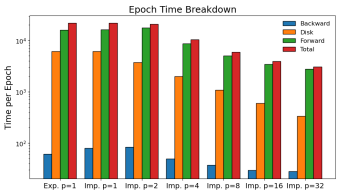

Figure 8: For HMDB51: Per epoch breakdown of run times with ResNet18 preprocessing images.

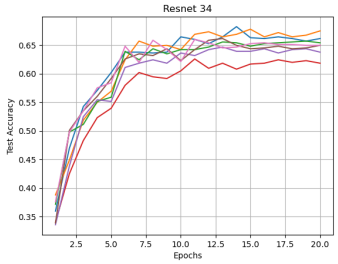

Figure 9: For HMDB51: Test accuracy of as a function of epochs for a GRU network with ResNet34. Trained on 32 MPI ranks with 9 OpenMP threads per rank. Each color is a different initialization.

implicit GRU using Resnet34 on 32 MPI ranks is investigated with respect to initialization with 7 different random seeds in Figure 9. A learning rate schedule starting from 10^{-3} and being reduced by $1/2$ after 5 epochs is used to achieve roughly 65% accuracy. The variation is relatively small, and the overall performance is robust.

5 Conclusion

This paper presented a PinT approach for forward and backward propagation of a GRU network using MGRIT. Parallelism is possible because the algorithm admits controlled errors in the hidden states computed. These errors arise out of a multigrid hierarchy used to reduce the run time through parallelism of the forward and backward propagation phases. A key development to make this possible was a new *implicit GRU* network based on the ODE formulation of the classical GRU methodology. Extending MGRIT propagation approach to other types of RNN, like LSTM's or those proposed in Chang et al. (2019), is possible with an implicit solver.

The implementation of this approach leverages existing automatic differentiation frameworks for neural networks. PyTorch was used in the present work using the standard ADAM optimizer. The parallel algorithm thus compared to the PyTorch GRU implementation on CPUs using OpenMP, demonstrating a $6.5x$ speedup for the HMDB51 dataset. A GPU version of the MGRIT technology is currently underdevelopment, though related work has demonstrated potential on Neural ODEs (Kirby et al., 2020). Other forms of data and model parallelism are complementary to the proposed technique, though undoubtedly there are as of yet unexplored optimizations with communication. The key thing this algorithm enables is accelerated parallel training of GRUs on longer sequences. This is a unique capability that permits growth in this dimension. The intended goal then is to allow further exploration on very long sequences via time-parallel training.

References

Davide Anguita, Alessandro Ghio, Luca Oneto, Xavier Parra, Jorge Luis Reyes-Ortiz, et al. A public domain dataset for human activity recognition using smartphones. In *Esann*, volume 3, pp. 3, 2013. URL `https://archive.ics.uci.edu/ml/datasets/human+activity+recognition+using+smartphones`.

Tal Ben-Nun and Torsten Hoefler. Demystifying parallel and distributed deep learning: An in-depth concurrency analysis. *ACM Computing Surveys (CSUR)*, 52(4):1–43, 2019.

Achi Brandt. Multi-level adaptive solutions to boundary-value problems. *Mathematics of computation*, 31(138):333–390, 1977.

William L Briggs, Van Emden Henson, and Steve F McCormick. *A multigrid tutorial*. SIAM, 2000.

Bo Chang, Minmin Chen, Eldad Haber, and Ed H Chi. Antisymmetricrnn: A dynamical system view on recurrent neural networks. *arXiv preprint arXiv:1902.09689*, 2019.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

Eric C Cyr, Stefanie Günther, and Jacob B Schroder. Multilevel initialization for layer-parallel deep neural network training. *arXiv preprint arXiv:1912.08974*, 2019.

Edward De Brouwer, J Simm, Ádám Arany, and Y Moreau. Gru-ode-bayes: Continuous modeling of sporadically-observed time series. *Proc. of Advances in Neural Information Processing Systems (NeurIPS)*, 32:7377–7388, 2019.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

V. A. Dobrev, Tz. Kolev, N. A. Petersson, and J. B. Schroder. Two-level convergence theory for multigrid reduction in time (mgrit). *SIAM Journal on Scientific Computing*, 39(5):S501–S527, jan 2017. doi: 10.1137/16m1074096. URL https://doi.org/10.1137%2F16m1074096.

Craig C Douglas. Multigrid methods in science and engineering. *IEEE Computational Science and Engineering*, 3(4):55–68, 1996.

Robert D Falgout, Stephanie Friedhoff, Tz V Kolev, Scott P MacLachlan, and Jacob B Schroder. Parallel time integration with multigrid. *SIAM Journal on Scientific Computing*, 36(6):C635–C661, 2014.

Martin J Gander. 50 years of time parallel time integration. In *Multiple shooting and time domain decomposition methods*, pp. 69–113. Springer, 2015.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

Stefanie Gunther, Lars Ruthotto, Jacob B Schroder, Eric C Cyr, and Nicolas R Gauger. Layer-parallel training of deep residual neural networks. *SIAM Journal on Mathematics of Data Science*, 2(1):1–23, 2020.

Mansura Habiba and Barak A Pearlmutter. Neural ordinary differential equation based recurrent neural network model. In *2020 31st Irish Signals and Systems Conference (ISSC)*, pp. 1–6. IEEE, 2020.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Zhiheng Huang, Geoffrey Zweig, Michael Levit, Benoit Dumoulin, Barlas Oguz, and Shawn Chang. Accelerating recurrent neural network training via two stage classes and parallelization. In *2013 IEEE Workshop on Automatic Speech Recognition and Understanding*, pp. 326–331. IEEE, 2013.

Ian D Jordan, Piotr Aleksander Sokół, and Il Memming Park. Gated recurrent units viewed through the lens of continuous time dynamical systems. *Frontiers in computational neuroscience*, pp. 67, 2021.

Andrew Kirby, Siddharth Samsi, Michael Jones, Albert Reuther, Jeremy Kepner, and Vijay Gadepally. Layer-parallel training with gpu concurrency of deep residual neural networks via nonlinear multigrid. In *2020 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, sep 2020. doi: 10.1109/hpec43674.2020.9286180. URL https://doi.org/10.1109%2Fhpec43674.2020.9286180.

Hildegard Kuehne, Hueihan Jhuang, Estíbaliz Garrote, Tomaso Poggio, and Thomas Serre. Hmdb: a large video database for human motion recognition. In *2011 International conference on computer vision*, pp. 2556–2563. IEEE, 2011. URL https://serre-lab.clps.brown.edu/resource/hmdb-a-large-human-motion-database/.

Apache MXNet. Training with multiple gpus using model parallelism. Technical report, 2018. URL https://mxnet.incubator.apache.org/versions/1.2.1/faq/model_parallel_lstm.html.

Benjamin W Ong and Jacob B Schroder. Applications of time parallelization. *Computing and Visualization in Science*, 23(1):1–15, 2020.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

Jacob B Schroder. Parallelizing over artificial neural network training runs with multigrid. *arXiv preprint arXiv:1708.02276*, 2017.

Xian-He Sun and Lionel M Ni. Scalable problems and memory-bounded speedup. *Journal of Parallel and Distributed Computing*, 19(1):27–37, 1993.

Ilya Sutskever. *Training recurrent neural networks*. University of Toronto Toronto, Canada, 2013.

Ulrich Trottenberg, Cornelius W Oosterlee, and Anton Schuller. *Multigrid*. Elsevier, 2000.

Gerhard Wanner and Ernst Hairer. *Solving ordinary differential equations II*, volume 375. Springer Berlin Heidelberg, 1996.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

Yang You, Jonathan Hseu, Chris Ying, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. Large-batch training for lstm and beyond. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–16, 2019.