1 Introduction

Due to the large number of layers in deep neural networks (DNNs) [11, 12], DNN training is time-consuming and there are demands to reduce training time these days. Recently, multi-GPU parallel computing has become an important topic for accelerating DNN training [2, 6]. In particular, Günther et al. [6] considered the layer structure of ResNet [8] as the forward Euler discretization of a specific ODE and applied a nonlinear in-time multigrid method [3] by regarding the learning process of the network as an optimal control problem.

In this work, we propose a novel paradigm of multi-GPU parallel computing for DNNs, called parareal neural network. In general, DNN has a feed-forward architecture. That is, the output of DNN is obtained from the input by sequential compositions of functions representing layers. We observe that sequential computations can be interpreted as time steps of a time-dependent problem. In the field of numerical analysis, after a pioneering work of Lions et al. [14], there have been numerous researches on parallel-in-time algorithms to solve time-dependent problems in parallel; see, e.g., [5, 15, 16]. Motivated by these works, we present a methodology to transform a given feed-forward neural network to another neural network called parareal neural network which naturally adopts parallel computing. The parareal neural network consists of fine structures which can be processed in parallel and...
a coarse structure which approximates the fine structures by emulating one of the parallel-in-time algorithms called parareal [14].

Note that both the proposed parareal neural network and the work of Günther et al. [6] seem to be very closely related in that they parallelize and accelerate the training of neural networks using a parallel time integration approach. However, unlike the work of Günther et al., the proposed network has the advantage of being more general by focusing on layer propagation in an arbitrary feed-forward network.

The parareal neural network can significantly reduce the time for inter-GPU communication because the fine structures do not communicate with each other but communicate only with the coarse structure. Therefore, the proposed methodology is effective in reducing the elapsed time for dealing with very deep neural networks. Numerical results confirm that the parareal neural network gives similar or better performance to the original network even with less training time.

2 The parareal algorithm

The parareal algorithm proposed by Lions et al. [14] is a parallel-in-time algorithm to solve time-dependent differential equations. For the purpose of description, the following system of ordinary differential equations is considered:

\[
\dot{\mathbf{u}}(t) = A\mathbf{u}(t) \quad \text{in} \quad [0, T], \quad \mathbf{u}(0) = \mathbf{u}_0,
\]

where \( A: \mathbb{R}^m \to \mathbb{R}^m \) is an operator, \( T > 0 \), and \( \mathbf{u}_0 \in \mathbb{R}^m \). The time interval \([0, T]\) is decomposed into \( N \) subintervals \( 0 = T_0 < T_1 < \cdots < T_N = T \). First, an approximated solution \( \{\mathbf{u}^k_j\}_{j=0}^N \) on the coarse grid \( \{T_j\}_{j=0}^N \) is obtained by the backward Euler method. The key step of the parareal algorithm is to correct residuals \( \{\mathbf{r}^k_j\}_{j=1}^{N-1} \) occurring in each interface. It is well-known that the algorithm converges to the exact solution uniformly [1, 4]. We briefly summarize the parareal in Algorithm 1.

3 Parareal neural networks

In this section, we propose a methodology to design a parareal neural network by emulating the parareal algorithm introduced in Section 2 from a given feed-forward neural network. The resulting parareal neural network has an intrinsic parallel structure and is suitable for parallel computation using multiple GPUs with distributed memory simultaneously.

Let \( f_\theta: X \to Y \) be a feed-forward neural network, where \( X \) and \( Y \) are the spaces of inputs and outputs, respectively, and \( \theta \) is a vector consisting of parameters. Since many modern neural networks such as [9, 10, 17] have block-repetitive substructures, we may assume that \( f_\theta \) can be written as the composition of three functions \( C_\theta: X \to W_0, \ g_\varphi: W_0 \to W_1, \) and \( h_\varepsilon: W_1 \to Y, \) i.e.,
Algorithm 1: The parareal algorithm for (1)

\[
\Delta t_j = T_{j+1} - T_j \quad \text{and} \quad 0 = T_0 < T_1 < \cdots < T_N = T.
\]

for \( j \leftarrow 0 \) to \( N - 1 \) do
\[
\text{Solve } \frac{U^j_{j+1} - U^j_{j}}{\Delta T_j} = A U^j_{j+1}, \quad U^j_0 = u_0
\]
end

for \( k \leftarrow 1, 2, \ldots \) do

for \( j \leftarrow 0 \) to \( N - 1 \) in parallel do
\[
\text{Solve } u^j_k(t) = A u^j_k(t) \text{ in } [T_j, T_{j+1}], \quad u^j_k(T_j) = U^j_k.
\]
end

for \( j \leftarrow 0 \) to \( N - 1 \) do
\[
S^j_{j+1} = u^j_{j+1}(T_{j+1}) - U^j_{j+1}, \quad S^j_0 = 0.
\]
\[
\delta^j_{j+1} - \delta^j_k \frac{\Delta T_j}{\Delta T_j} = A \delta^j_{j+1} + S^j_k, \quad \delta^j_0 = 0.
\]
\[
U^j_{j+1} = U^j_{j+1} + \delta^j_{j+1}.
\]
end

end

\[
f_\theta = h_\varepsilon \circ g_\varphi \circ C_\delta, \quad \theta = \delta \oplus \varphi \oplus \varepsilon.
\]

where \( W_0 \) and \( W_1 \) are vector spaces, \( g_\varphi \) is a block-repetitive substructure of \( f_\theta \) with parameters \( \varphi \), \( C_\delta \) is a preprocessing operator with parameters \( \delta \), and \( h_\varepsilon \) is a postprocessing operator with parameters \( \varepsilon \). Note that \( \oplus \) represents a concatenation.

For appropriate vector spaces \( X_0, X_1, \ldots, X_N \), we further assume that \( g_\varphi \) can be partitioned into \( N \) subnetworks \( \{g^j_\varphi; X_{j-1} \rightarrow X_j\}_{j=1}^{N} \) which satisfy the followings:

- \( X_0 = W_0 \) and \( X_N = W_1 \),
- \( \varphi = \bigoplus_{j=1}^{N} \psi_j \),
- \( g_\varphi = g^N_\varphi \circ g^{N-1}_\varphi \circ \cdots \circ g^1_\varphi \).

In forward and backward propagations through \( g_\varphi \), propagations are done sequentially through the subnetworks \( \{g^j_\varphi\}_{j=1}^{N} \). Regarding the subnetworks as subintervals of a time-dependent problem and adopting the idea of the parareal algorithm introduced in Section 2, we construct a new neural network \( f_\bar{\theta} \): \( X \rightarrow Y \) which contains \( \{g^j_\varphi\}_{j=1}^{N} \) as parallel subnetworks; the precise definition for parameters \( \bar{\theta} \) will be given in (4).

Since the dimensions of the spaces \( \{X_j\}_{j=0}^{N-1} \) are different for each \( j \) in general, we introduce preprocessing operators \( C^j_\delta; X \rightarrow X_{j-1} \) such that \( C^1_\delta = C_\delta \) and \( C^j_\delta \) for \( j = 2, \ldots, N \) play similar roles to \( C_\delta \); particular examples will be given in Section 4. We write \( x_j \in X_{j-1} \) and \( y_j \in X_j \) as follows:

\[
x_j = C^j_\delta(x) \quad \text{for } x \in X, \quad y_j = g^j_\varphi(x_j).
\]
Then, we consider neural networks $F^j_{\eta_j} : X_j \rightarrow X_{j+1}$ with parameters $\eta_j$ for $j \geq 1$ such that it approximates $g^{j+1}_{\varphi_{j+1}}$ well while it has a cheaper computational cost than $g^{j+1}_{\varphi_{j+1}}$, i.e., $F^j_{\eta_j} \approx g^{j+1}_{\varphi_{j+1}}$ and $\dim(\eta_j) \ll \dim(\varphi_{j+1})$. Emulating the coarse grid correction of the parareal algorithm, we assemble a network called coarse network with building blocks $F^j_{\eta_j}$. With inputs $x_{j+1}$, $y_j$, and an output $y \in Y$, the coarse network is described as follows:

$$
\begin{align*}
\mathbf{r}_N &= \mathbf{0}, \quad \mathbf{r}_j = y_j - x_{j+1} \quad \text{for} \quad j = 1, \ldots, N - 1, \\
\mathbf{r}_1 &= \mathbf{r}_j, \quad \mathbf{r}_{j+1} = F^j_{\eta_j}(\mathbf{r}_j) \quad \text{for} \quad j = 1, \ldots, N - 1, \\
\hat{y} &= y_N + \mathbf{r}_N.
\end{align*}
$$

That is, in the coarse network, the residual $\mathbf{r}_j$ at the interface between layers $g^j_{\varphi_j}$ and $g^{j+1}_{\varphi_{j+1}}$ propagates through shallow neural networks $F^1_{\eta_1}, \ldots, F^{N-1}_{\eta_{N-1}}$. Then the propagated residual is added to the output.

Finally, the parareal neural network $\tilde{f}_\tilde{\theta}$ corresponding to the original network $f_\theta$ is defined as

$$
\tilde{f}_\tilde{\theta}(x) = h_\varepsilon(\hat{y}), \quad \hat{y} = \left( \bigoplus_{j=1}^{N} (\delta_j \oplus \varphi_j) \right) \oplus \left( \bigoplus_{j=1}^{N-1} \eta_j \right) \oplus \varepsilon.
$$

That is, $\tilde{f}_\tilde{\theta}$ is composed of the preprocessing operators $\{C^j_{\delta_j}\}$, parallel subnetworks $\{g^j_{\varphi_j}\}$, the coarse network $\{F^j_{\eta_j}\}$, and the postprocessing operator $h_\varepsilon$. Figure 1(b) illustrates $\tilde{f}_\tilde{\theta}$.

Since each $g^j_{\varphi_j} \circ C^j_{\delta_j}$ lies in parallel, all computations related to $g^j_{\varphi_j} \circ C^j_{\delta_j}$ can be done independently. Therefore, multiple GPUs can be utilized to process $\{g^j_{\varphi_j} \circ C^j_{\delta_j}\}$ simultaneously for each $j$. In this case, one may expect significant decrease of the elapsed time for training $\tilde{f}_\tilde{\theta}$ compared to the original network $f_\theta$. On the other hand, the coarse network cannot be parallelized since $\{F^j_{\eta_j}\}$ is computed in the sequential manner. One should choose $F^j_{\eta_j}$ whose computational cost is as cheap as possible in order to reduce the bottleneck effect of the coarse network.

Now, we want show that the proposed parareal neural network $\tilde{f}_\tilde{\theta}$ is consistently constructed in the sense that it recovers the original neural network $f_\theta$ in the setting where nonlinearity is removed. By collecting all the residuals in each interface and dealing with them sequentially, the following proposition is obtained.

**Proposition 1 (Consistency)**

Assume that the original network $f_\theta$ is linear and $F^j_{\eta_j} = g^{j+1}_{\varphi_{j+1}}$ for $j = 1, \ldots, N-1$. Then we have $\tilde{f}_\tilde{\theta}(x) = f_\theta(x)$ for all $x \in X$.

Proposition 1 presents a guideline on how to design the coarse network of $\tilde{f}_\tilde{\theta}$. Under the assumption that $f_\theta$ is linear, a sufficient condition to ensure that $\tilde{f}_\tilde{\theta} = f_\theta$
is $F_{\eta_j} = g^{j+1}_{\phi_{j+1}}$ for all $j$. Therefore, we can say that it is essential to design the coarse network with $F_{\eta_j} = g^{j+1}_{\phi_{j+1}}$ to ensure that the performance of $f_{\hat{\theta}}$ is as good as that of $f_{\theta}$. Detailed examples will be given in Section 4.

4 Application to ResNet-1001

The proposed parareal neural network can be applied to a general feed-forward neural network. However, since most of the current very deep neural networks have residual structures, we applied it to ResNet-1001 [9], which is one of the typical very deep convolutional neural network for classification problems. First, we describe the structure of ResNet-1001 with the terminology introduced in Section 3. Inputs for ResNet-1001 are 3-channel images with $32 \times 32$ pixels, i.e., $X = \mathbb{R}^{3 \times 32 \times 32}$. The output space $Y$ is given by $Y = \mathbb{R}^m$, where $m$ is the number of classes of images. ResNet-1001 has a block-repetitive substructure consisting of 333 residual units (RUs), so that we may set $g_{\phi_j}: W_0 \to W_1$ as the composition of those RUs with $W_0 = \mathbb{R}^{16 \times 32 \times 32}$ and $W_1 = \mathbb{R}^{256 \times 8 \times 8}$. Then the preprocessing operator $C_{\phi}: X \to W_0$ is a single $3 \times 3$ convolution layer and the postprocessing operator $h_{\varepsilon}: W_1 \to Y$ consists of global average pooling and fully connected layers.

The design of a parareal neural network with $N$ parallel subnetworks for ResNet-1001, denoted as Parareal ResNet-$N$, can be completed by specifying the structures $g^{l}_{\phi_j}$, $C^{l}_{\phi_j}$, and $F^{l}_{\eta_j}$. For convenience, the original neural network ResNet-1001 is called Parareal ResNet-1. We assume that $N = 3N_0$ for some positive integer $N_0$. We note that $g_{\phi}$ can be decomposed as

![Fig. 1: A feed-forward neural network and its corresponding parareal neural network. (a) Feed-forward neural network $f_{\theta}$, (b) Parareal neural network $f_{\hat{\theta}}$ with $N$ parallel subnetworks ($N = 3$).](image-url)
where each of $g_{\varphi_j}^j : X_{j-1} \rightarrow X_j$ consists of $\lceil 333/N \rceil$ RUs with

$$X_j = \begin{cases} \mathbb{R}^{64 \times 32 \times 32} & \text{for } j = 1, \ldots, N_0, \\ \mathbb{R}^{128 \times 16 \times 16} & \text{for } j = N_0 + 1, \ldots, 2N_0, \\ \mathbb{R}^{256 \times 8 \times 8} & \text{for } j = 2N_0 + 1, \ldots, N, \\ \end{cases} \varphi = \bigoplus_{j=1}^{N} \varphi_j.$$

The main role of the preprocessing operator $C^j_{\delta_j} : X \rightarrow X_{j-1}$ is to transform an input $x \in X$ to fit in the space $X_{j-1}$. In this perspective, we simply set $C^1_{\delta_1} = C_\delta$ and $C^j_{\delta_j}$ for $j > 1$ consists of a $1 \times 1$ convolution to match the number of channels after appropriate number of $3 \times 3$ max pooling layers with stride 2 to match the image size. For the coarse network, we first define a coarse RU consisting of two $3 \times 3$ convolutions and skip-connection. If the downsampling is needed, then the stride of first convolution in coarse RU is set to 2. We want to define $F^j_{\eta_j} : X_j \rightarrow X_{j+1}$ having smaller number of (coarse) RUs than $g_{\varphi_{j+1}}^{j+1}$ but a similar coverage to $g_{\varphi_{j+1}}^{j+1}$. Let $N_c$ be the number of coarse RUs in $F^j_{\eta_j}$ of the coarse network. Note that the receptive field of $g_{\varphi_j}^j$ covers the input size $32 \times 32$. In the case of $N = 3$, even if we construct $F^j_{\eta_j}$ with $N_c = 4$ coarse RUs, it can cover $31 \times 31$ pixels which are similar coverage to the parallel subnetwork $g_{\varphi_j}^j$. Generally, if we use $N$ parallel subnetworks ($N \geq 3$), each $333/N$ RUs in $g_{\varphi_j}^j$ can be approximated by the $N_c$ RUs in $F^j_{\eta_j}$ whenever we select $N_c = \lceil 12/N \rceil$.

## 5 Numerical results

In this section, we present numerical results of the Parareal ResNet-$N$ with various $N$. First, we present details on the datasets we used. The CIFAR-$m$ ($m = 10, 100$) dataset consists of $32 \times 32$ colored natural images and includes 50,000 training and 10,000 test samples with $m$ classes. The SVHN dataset is composed of $32 \times 32$ colored digit images; there are 73,257 and 26,032 samples for training and test, respectively, with additional 531,131 training samples. However, we did not use the additional ones for training. MNIST is a classic dataset which contains handwritten digits encoded in $28 \times 28$ grayscale images. It includes 55,000 training, 5,000 validation, and 10,000 test samples. In our experiments, the training and validation samples are used as training data and the test samples as test data. We adopted a data augmentation technique in [13] for CIFAR datasets; four pixels are padded on each side of images, and $32 \times 32$ crops are randomly sampled from the padded images and their horizontal flips.

All neural networks in this section were trained using the stochastic gradient descent with the batch size 128, weight decay 0.0005, momentum 0.9, and weights initialized as in [7]. The initial learning rate is set to 0.1, and is reduced by a factor
Parareal Neural Networks

Table 1: Error rates (%) on the CIFAR-10, CIFAR-100, MNIST, and SVHN datasets of Parareal ResNet-$N$ ($N = 1, 3, 6, 12, 18$) with $N_c = \lceil 12/N \rceil$.

<table>
<thead>
<tr>
<th>$N$</th>
<th>Parameters per subnetwork</th>
<th>Parameters of coarse network</th>
<th>Total Parameters</th>
<th>CIFAR-10</th>
<th>CIFAR-100</th>
<th>MNIST</th>
<th>SVHN</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-</td>
<td>-</td>
<td>10.3M</td>
<td>4.96</td>
<td>21.13</td>
<td>0.34</td>
<td>3.17</td>
</tr>
<tr>
<td>3</td>
<td>3.4M</td>
<td>5.6M</td>
<td>15.9M</td>
<td>4.61</td>
<td>21.14</td>
<td>0.31</td>
<td>3.11</td>
</tr>
<tr>
<td>6</td>
<td>1.7M</td>
<td>5.7M</td>
<td>16.1M</td>
<td>4.20</td>
<td>20.87</td>
<td>0.31</td>
<td>3.21</td>
</tr>
<tr>
<td>12</td>
<td>0.9M</td>
<td>5.8M</td>
<td>16.2M</td>
<td>4.37</td>
<td>20.42</td>
<td>0.28</td>
<td>3.25</td>
</tr>
<tr>
<td>18</td>
<td>0.6M</td>
<td>8.9M</td>
<td>19.4M</td>
<td>4.02</td>
<td>20.40</td>
<td>0.33</td>
<td>3.29</td>
</tr>
</tbody>
</table>

Table 2: Forward/backward computation time for Parareal ResNet-$N$ ($N = 1, 3, 6, 12, 18$). The time is measured in one iteration for CIFAR-100 dataset input $x \in \mathbb{R}^{3 \times 32 \times 32}$ with batch size 128.

<table>
<thead>
<tr>
<th>$N$</th>
<th>Preprocessing</th>
<th>Parallel subnetworks</th>
<th>Coarse network</th>
<th>Postprocessing</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.25/6.46</td>
<td>443.81/1387.62</td>
<td>-</td>
<td>0.06/3.18</td>
<td>444.12/1397.26</td>
</tr>
<tr>
<td>3</td>
<td>0.25/6.45</td>
<td>131.92/458.87</td>
<td>10.01/97.60</td>
<td>0.06/3.71</td>
<td>142.24/566.63</td>
</tr>
<tr>
<td>6</td>
<td>0.27/6.42</td>
<td>67.59/219.72</td>
<td>14.68/137.08</td>
<td>0.06/3.33</td>
<td>82.60/366.55</td>
</tr>
<tr>
<td>12</td>
<td>0.28/6.59</td>
<td>48.47/113.33</td>
<td>17.97/149.52</td>
<td>0.06/3.63</td>
<td>66.78/273.07</td>
</tr>
<tr>
<td>18</td>
<td>0.29/6.17</td>
<td>30.40/77.84</td>
<td>27.87/163.25</td>
<td>0.06/3.64</td>
<td>58.62/250.90</td>
</tr>
<tr>
<td>24</td>
<td>0.29/6.58</td>
<td>22.71/58.04</td>
<td>41.03/242.87</td>
<td>0.06/3.54</td>
<td>64.09/311.03</td>
</tr>
</tbody>
</table>

of 10 in the 80th and 120th epochs. All networks were implemented in Python with PyTorch and all computations were performed on a cluster equipped with Intel Xeon Gold 5515 (2.4GHz, 20C), NVIDIA Titan RTX and the operating system Ubuntu 18.04 64bit.

With fixed $N_c = \lceil 12/N \rceil$, we report the classification results of Parareal ResNet with respect to various $N$ on datasets CIFAR-10, CIFAR-100, SVHN, and MNIST. Table 1 shows that the error rates of Parareal ResNet-$N$ are usually smaller than ResNet-1001.

Next, we investigate the elapsed time for forward and backward propagations of parareal neural networks. Table 2 shows the virtual wall-clock time for forward and backward computation of Parareal ResNet-$N$ with various $N$ for the input $x \in \mathbb{R}^{3 \times 32 \times 32}$. As shown in Table 2, the larger $N$, the shorter the computing time of the parallel subnetworks $g^l_{\Phi_{ij}}$, while the longer the computing time of the coarse network.

This is because as $N$ increases, the depth of each parallel subnetwork $g^l_{\Phi_{ij}}$ becomes shallower while the number of $F_{t_{ij}}$ in the coarse network increases. On the other hand, each preprocessing operator $C^l_{\Phi_{ij}}$ is designed to be the same as or similar to the preprocessing operator $C_{\Phi}$ of the original neural network and the postprocessing operator $h_{\Phi}$ is the same as the original one. Therefore, the computation time for the pre- and postprocessing operators does not increase even as $N$ increases.
Table 3: The wall-clock time and relative speedup on the CIFAR-100 dataset. The wall-clock time is the total time taken to train a given network by 200 epochs.

<table>
<thead>
<tr>
<th>Network</th>
<th>Parameters</th>
<th>Wall-clock time (h:m:s)</th>
<th>Relative speedup (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet-1001</td>
<td>10.3M</td>
<td>22:44:53</td>
<td>0.0</td>
</tr>
<tr>
<td>Parareal ResNet-3</td>
<td>15.9M</td>
<td>16:28:38</td>
<td>27.6</td>
</tr>
<tr>
<td>Parareal ResNet-6</td>
<td>16.1M</td>
<td>11:48:13</td>
<td>48.1</td>
</tr>
</tbody>
</table>

Finally, we measure the wall-clock time of the Parareal ResNet with the CIFAR-100 dataset. Table 3 shows that Parareal ResNet’s wall clock time is reduced by about half as \( N \) increases to 6.

In conclusion, despite the large number of layers, the parareal neural network can accelerate the training of the very deep CNN using multiple-GPU. To the best of our knowledge, the proposed methodology is a new kind of multi-GPU parallelism in the field of deep learning.

References
