SC20: International Conference for High Performance Computing, Networking, Storage and Analysis | 978-1-7281-9998-6/20/$31.00 ©2020 IEEE | DOI: 10.1109/SC41405.2020.00049

GEMS: GPU-Enabled Memory-Aware Model-Parallelism System for Distributed DNN Training

Arpan Jain
Department of Computer Science and Engineering
The Ohio State University
jain.575@osu.edu

Ammar Ahmad Awan
Department of Computer Science and Engineering
The Ohio State University
awan.10@osu.edu

Asmaa M. Aljuhani
Department of Computer Science and Engineering
The Ohio State University
aljuhani.2@osu.edu

Jahanzeb Maqbool Hashmi
Department of Computer Science and Engineering
The Ohio State University
hashmi.29@osu.edu

Quentin G. Anthony
Department of Computer Science and Engineering
The Ohio State University
anthony.301@osu.edu

Hari Subramoni
Department of Computer Science and Engineering
The Ohio State University
subramoni.1@osu.edu

Dhableswar K. Panda
Department of Computer Science and Engineering
The Ohio State University
panda.2@osu.edu

Raghu Machiraju
Department of Computer Science and Engineering
The Ohio State University
machiraju.1@osu.edu

Anil Parwani
Department of Pathology
The Ohio State University
anil.parwani@osumc.edu

***Abstract*—Data-parallelism has become an established paradigm to train DNNs that fit inside GPU memory on large-scale HPC systems. However, model-parallelism is required to train out-of-core DNNs. In this paper, we deal with emerging requirements brought forward by very large DNNs being trained using high-resolution images common in digital pathology. To address these, we propose, design, and implement GEMS; a GPU-Enabled Memory-Aware Model-Parallelism System. We present several design schemes like GEMS-MAST, GEMS-MASTER, and GEMS-Hybrid that offer excellent speedups over state-of-the-art systems like Mesh-TensorFlow and FlexFlow. Furthermore, we combine model-parallelism and data-parallelism to train a 1000-layer ResNet-1k model using 1,024 Volta V100 GPUs with 97.32% scaling-efficiency. For the real-world histopathology whole-slide-image (WSI) of 100,000 x 100,000 pixels, we train custom ResNet-110-v2 on image tiles of size 1024 x 1024 and reduce the training time from seven hours to 28 minutes.**

***Index Terms*—DNN, Model Parallelism, Keras, TensorFlow, Eager Execution, MPI**

I. Introduction

The popularity and effectiveness of Deep Learning (DL) have led to advances in several areas, including image recognition, speech processing, autonomous vehicles, and precision medicine. DL is a subset of Machine Learning (ML) that allows us to discover relationships between the input and output by reducing the input into distinguishable features and applying the model to accurately predict the output. Large-scale Deep Neural Networks (DNNs) are a key component to DL methods. Domain-specific modifications like convolution and recurrent layers are integrated into DNNs to increase the accuracy of models made for computer vision and natural language processing, respectively. DNNs offer higher prediction accuracy for these tasks, but training complex DNNs is a significantly compute-intensive and memory-hungry [1], [2] operation. Due to its relative simplicity, researchers employ *Data Parallelism* [3] to train DNNs by replicating the model on multiple processing elements (PEs) such as CPUs and/or GPUs. Each replica of the DNN operates on a different subset of the input data called a batch. This results in a set of gradients for each replica that are accumulated with an Allreduce operation [4].

While data parallelism works well for distributed DNN training and offers near-linear scaling [5], it is bounded by the size of the model itself, that is; even at the finest granularity of a unit batch size (i.e., a single training example), the entire model does not fit in the PE's memory. To address this fundamental limitation of data parallelism, a new parallelism strategy called "Model Parallelism" is gaining attention in the CS and AI communities. As shown in Figure 1, several real-world image sizes are much bigger than the current 224 $\times$ 224 resolution of the widely used ImageNet [6] dataset. These emerging application areas, including digital pathology, are the primary motivators of model and hybrid parallelism (a combination of model and data parallelism).

Given the advent of digital pathology over the last decade, digital whole slide image (WSI) is now fast replacing the glass slide for diagnostic purposes. A typical WSI is often extremely

SC20, November 9-19, 2020, Is Everywhere We Are
978-1-7281-9998-6/20/$31.00 ©2020 IEEE

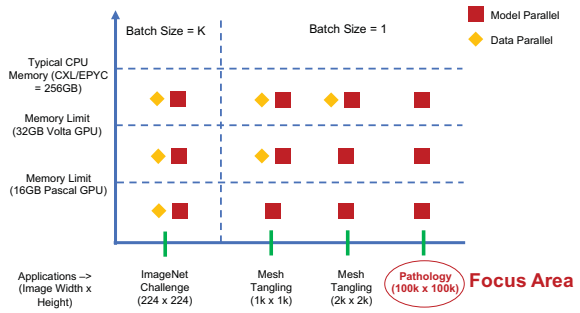


Fig. 1. The emerging need for Model Parallelism

large, and it is typical for pathologists to view images that are 100,000×100,000 pixels in size and were captured at very high magnification levels (10-60×). A pathologist will typically follow a multi-scale resolution approach to examine tissue regions on a whole slide image (WSI) as shown in Figure 2. The goal of such examination is to objectively measure the degree of abnormality (or *grade*) of a tumor for a disease such as cancer.

To address these new requirements, we investigate various parallelism approaches for DNNs in this paper. The basic approach to implement model parallelism involves dividing the model itself into smaller partitions and assigning these partitions to different PEs. During training, each PE computes on a given partition and communicates the result with neighboring PEs during the forward pass and the backward pass. Because of the data dependency in the forward and backward pass, at any given time, only one PE does the computation while the rest of the PEs remain idle. For example, an arbitrary layer L_i requires data from $L_{(i-1)}$ in the forward pass, forming a dependency edge from Layer $L_{(i-1)}$ to L_i. The inverse holds true for the backward pass. The inherent dependencies of forward and backward pass serialize the workers involved in model parallel training leading to the under-utilization of resources as shown in Figure 3.

A. Motivation

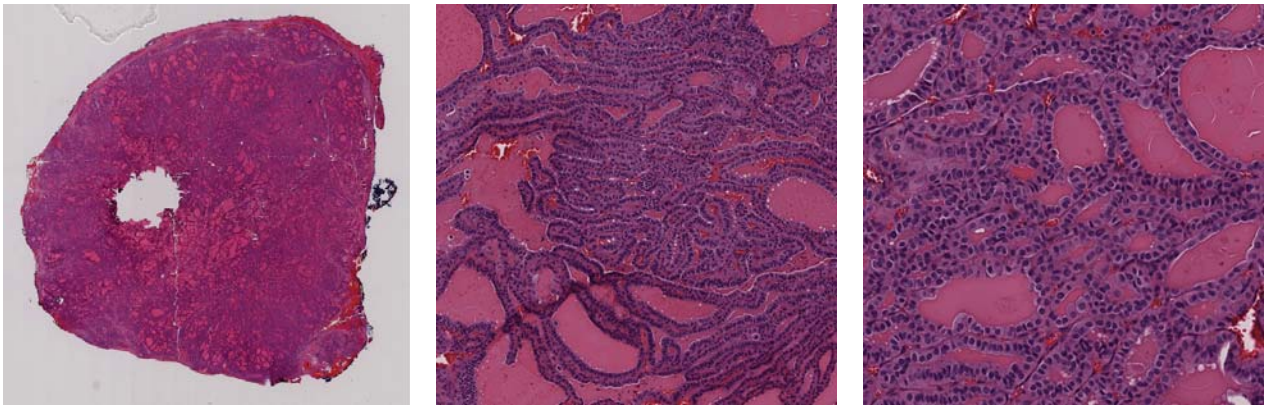
Recent research studies [7], [8], [9] address some of the requirements of model parallelism. However, under-utilization (Figure 3) of resources largely remains unaddressed. Existing systems employ techniques like pipelining to overcome the limitation of inherently serialized forward and backward passes. However, pipeline parallelism, which is a special case of model parallelism, is only applicable if several training examples (i.e., batch size = K, where K >> 1) can fit inside the PE's memory. This is not always possible as certain state-of-the-art models only fit in the PE's memory for a unit batch size. Any increase in batch size overflows the PE's memory, and hence the model becomes out-of-core and not trainable. These problems exist for all out-of-core models. However, this is exacerbated by pressing application-level needs for training on even larger images.

In this context, we discuss a concrete digital pathology scenario. To diagnose a disease or its grading, a pathologist must identify certain histologic structures such as elongated follicles, necrosis, glands, and cancer nuclei. The morphology appearance of these structures is an important indicator for the presence and the severity of a disease. Some of these histologic structures are analyzed on a low magnification level to capture their architecture arrangements (i.e tram tracks in tall cell variant (TCV) of papillary thyroid cancer (PTC)) while others need to be analyzed at a high magnification level to differentiate diagnosis (i.e nuclear stratification). Therefore, due to their relatively large size, the images are tiled into patches at different resolution levels depending on the histologic structure being analyzed. Larger tile sizes at a high magnification level preserve the cellular features as well as their architectural arrangement. These application-level requirements cannot be satisfied by existing DNN training systems and thus there is a need for a truly memory-efficient system. The broad challenge that we solve in this paper is as follows: *How can we design a model parallelism solution that is 1) memory-efficient, 2) offers better training speed compared to state-of-the-art systems, and 3) supports emerging real-world use cases like digital pathology?* Several concrete design and implementation challenges lie in designing such a system:

- How can we reduce the under-utilization of memory in a model parallelism system?
- How can we design a GPU-enabled system that can avoid unnecessary data movement between CPU and GPU caused by limitations in TensorFlow and other high-level frameworks?
- How can we design a memory-efficient system that not only fully utilizes the memory but also offers better throughput by exploiting additional compute?
- Can model parallelism be made as scalable as data parallelism by a novel integration of data and model parallelism?
- Can we design sufficiently deep neural networks that can capture both finer-level cellular characteristics and coarse-level tissue organization in high-resolution histopathology images?

B. Contributions

To the best of our knowledge, no state-of-the-art model parallelism system is truly memory-efficient. Related studies on data, model, and hybrid-parallelism and their major features are compared with the proposed system in Table I. We extensively study and implement existing model parallelism approaches to understand various limitations and address them systematically in the proposed system called GEMS. Furthermore, to highlight the real-world impact of GEMS, we collaborate with pathologists and deploy GEMS to analyze WSIs pertaining to the analysis for tall cell variant (TCV) of the papillary thyroid carcinoma (PTC) manifest as histopathological images. The key contributions of this paper are as follows:



(a) A whole slide image (WSI) (b) A tile at 10× magnification level (c) A tile at 20× magnification level

Fig. 2. 2(a) A Hematoxylin and Eosin stained whole slide image labeled as Tall Cell Variant (TCV) of the papillary thyroid cancer (PTC). 2(b) A 1024×1024 image tile at 10× magnification level shows histologic feature of elongated follicles arranged in parallel cords or tram tracks. 2(c) A 1024×1024 image tile at 20× magnification level shows cellular features of tall cells (cf. Section IX-A)

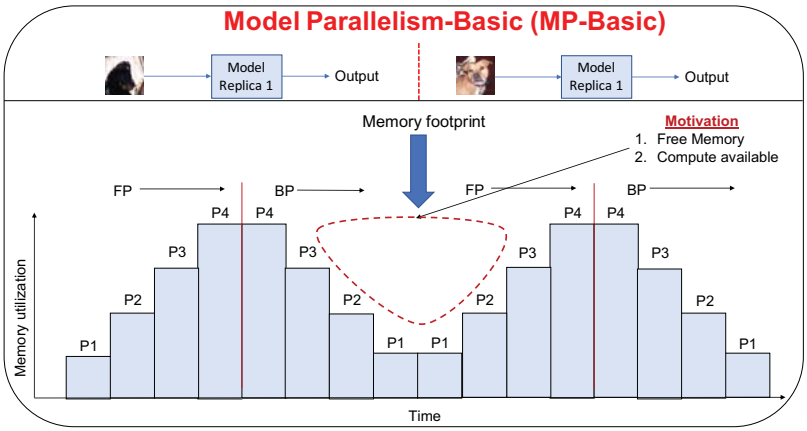


Fig. 3. Under-utilization of Memory in Existing Model/Pipeline Parallelism Systems. The available (unused) memory is represented by the dotted bubble.

Existing and Proposed Studies on Model Parallelism (MP)	Features							
	Synchronized Training	Out of Core DNN training Max BS=1	Out of Core DNN training Max BS=N	Efficient Memory Utilization	Efficient Compute Utilization	Pipelining using Batch Size	Training with Out-of-core Batch size	Speedup for Out-of-core Batch Size
Basic Model Parallelism	✔	✔	✔	×	×	×	×	×
Model Parallelism - Delayed Synchronization	✔	✔	✔	×	×	×	✔	×
Pipeline (GPipe [7] and HyPar-Flow [10])	✔	×	✔	✔	✔	✔	×	×
PipeDream [8]	×	✔	✔	✔	✔	✔	×	×
FlexFlow [11]	✔	✔	✔	✔	✔	×	×	×
LBANN [12]	✔	✔	✔	✔	✔	×	×	×
Mesh-TensorFlow [9]	✔	✔	✔	✔	✔	×	×	×
Proposed Designs (GEMS)	✔	✔	✔	✔	✔	✔	✔	✔

TABLE I
FEATURES OFFERED BY GEMS COMPARED TO EXISTING FRAMEWORKS

- We propose, design, and evaluate GEMS; an integrated system that provides memory-efficient model parallel training and scalable hybrid parallel training.
- To overcome the limitations in existing systems, we propose several design schemes: 1) Basic Model Parallelism (GEMS-Basic), 2) Memory Aware Synchronised Training (GEMS-MAST), 3) Memory Aware Synchronised Training with Enhanced Replications (GEMS-MASTER), and 4) GEMS-Hybrid that combines model and data parallelism.
- GEMS offers up to 1.36× speedup with MAST and 1.83× speedup with MASTER over existing model parallelism techniques including GEMS-Basic.
- We develop a comprehensive analytical model for MAST and MASTER that guides our performance expectations and verify its effectiveness through experimental evaluation.
- We report near-linear scaling (97.32%) of hybrid parallel

training on 1,024 Volta V100 GPUs for GEMS-Hybrid.

- We develop a high-level TCV classifier (a custom ResNet-110-v2) as an initial step to provide a robust diagnosis on much-larger 1024×1024 tiles that is only made possible through the proposed GEMS system.
- GEMS provides reduced training time (from 7.25 hours to 28 minutes) for out-of-core training of the newly developed ResNet-110-v2 based TCV classifier on 128 Volta V100 GPUs.

II. Background

A. DNN Training

A DNN is a directed, weighted graph of *neurons* with non-linear mappings between a set of inputs *x* and a set of learned outputs *y*. Each neuron is composed of a non-linear activation function *g* applied to the neuron's inputs. Each subsequent set of neurons are grouped into *layers*. Training the DNN entails adjusting the weights so that a prediction loss function L is minimized; weights are adjusted iteratively in *training steps*. At each training step, an input sample is fed through each layer of the network, and each neuron's weights/activations are applied sequentially. The loss gradient L is then propagated back through the network, and *neuron* weights are updated to reduce the loss. This concludes a training step, and the process is repeated until either the desired loss is achieved or the loss function reaches a global/local minima. Deep Learning frameworks like TensorFlow and PyTorch have dynamic memory requirements for each DNN training step: 1) As each layer of the DNN is computed in the forward pass, its values (and all values from previous layers) must be held in memory. 2) As each weight update is computed in the backward pass, its values may be removed from memory. Therefore, the memory requirements at each training step increase for each layer of the forward pass, then decrease for each layer of the backward pass. This insight is the key to our work. Distributed training can be realized using different strategies like data, model, and hybrid parallelism.

III. Challenges in Designing Memory-Aware Distributed DNN Training

We now highlight challenges in designing a memory-aware DNN training framework like GEMS that can support emerging real-world requirements including the high-resolution images common in digital pathology.

Challenge-1: GPU-based Communication in TensorFlow
Communication is a necessary evil in distributed/parallel training of DNNs. To improve its performance, we have to either optimize the blocking communication primitives for GPU communication or overlap it with the computation. In basic Model Parallelism, DNN is divided into smaller partitions and each partition is assigned to a different GPU. Partition P_i cannot start computation before receiving output from P_{i-1}. Therefore, model parallelism suffers from the serializability problem (i.e. there are data dependencies in the computation) and there exists little opportunity to overlap computation and communication. MPI runtimes offer optimized communication primitives and rely on CUDA-Awareness to optimize GPU-to-GPU communication. mpi4py provides high-level MPI bindings for Python. However, mpi4py cannot be directly used with TensorFlow because of two main reasons: 1) mpi4py requires NumPy or Numba arrays as data. At the same time, TensorFlow has its own data management for storing Tensors, thereby making usage of Numba arrays impossible, and 2) TensorFlow's Tensor objects can be converted to NumPy objects, but it requires an additional copy from GPU to CPU which increases the overhead for GPU-based communication.

Challenge-2: Memory management in TensorFlow
To develop a memory-aware model parallelism solution, it is essential that deep learning framework provides memory management functionalities. However, TensorFlow allocates the memory at the beginning of the program and manages this memory itself. Even if the memory is free for some time, TensorFlow does not provide any functionality to deallocate memory so that it can be used by other processes. The absence of explicit memory management increases the difficulty of implementing memory-aware designs for model parallelism.

Challenge-3: Scaling Memory-Aware solutions
A memory-aware solution that cannot scale to hundreds of GPUs is not very useful. Generally, model parallelism is used to fit the DNN into a small number of GPUs, and data parallelism is used to scale out the DNN to a higher number of GPUs. Therefore, a scalable solution for out-of-core models is required. Applying data parallelism to the memory-aware model parallelism is a non-trivial problem since a model's parameters are spread across multiple GPUs, which is in contrast to basic data parallelism, where parameters are stored in a single GPU. Therefore, we have to design an advanced version of data parallelism that supports parameters split across GPUs, and can also give similar or better speedup than basic data parallelism.

IV. Limitations in Existing Approaches for Model Parallelism

We provide an overview of various existing model parallelism approaches and discuss their limitations.

A. Basic Model Parallelism

A basic approach to train out-of-core DNNs is to split the DNN across multiple GPUs before applying a distributed forward and backward pass. The out-of-core DNN is partitioned, and each partition is placed on a single GPU. Send and Recv operations are used to implement distributed forward and backward propagation. Figure 4 shows how a DNN is split into two partitions using two GPUs with send and recv communication.

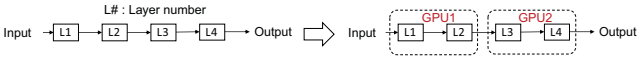


Fig. 4. Conversion of a sequential DNN to basic model parallel approach.

In forward propagation, the data is input to the first model partition at GPU1, and the result of the L2 layer is sent to

GPU2 using send and recv primitives. At GPU2, the received activations are used to compute the final prediction and its error (the loss value), which is used to calculate the gradient in the backpropagation algorithm. Partial errors are needed at GPU1 to implement the backward pass; therefore, GPU2 sends partial errors to GPU1 in the backward pass. In basic model parallelism, the DNN can be partitioned into more than two partitions, but the maximum number of partitions is limited by the maximum number of layers. There are two main issues with basic model parallelism: 1) Under-utilization of resources (cf. Figure 3) and 2) A non-trivial and complex implementation compared to data parallelism. Under-utilization happens as follows. If a model is partitioned across eight GPUs, then at any given time, only one GPU is doing computation while the other seven GPUs are idle; therefore, the basic implementation of model parallelism suffers from under-utilization of available computation resources. On the implementation complexity front, model parallelism is challenging to design since existing DL frameworks do not provide distributed back-propagation implementations. Partitioning a DNN itself is a complicated task as there can be skip connections or residual connections [13] in the DNN topology.

B. Model Parallelism with Pipelining

To mitigate resource under-utilization in basic model parallelism, pipelining divides the input batch into smaller batches called micro-batches. The number of such micro-batches can be called *parts*. When the number of *parts=1*, the pipelining design becomes similar to basic model parallelism. Figure 5 shows the pipelining approach on 4 GPUs with 4 parts to fill the pipeline. Typically, the number of parts should be equal to the number of DNN splits to utilize the pipeline fully. There are two main issues with the pipelining approach; 1) the maximum number of parts is limited by the batch size, and 2) the performance is poor compared to Data Parallelism or Hybrid Parallelism. If the batch size is smaller than the length of pipelining, the micro-batches will not fill the pipeline. Further, when the largest batch size is 1 for an out-of-core DNN on multiple GPUs, it is not possible to use pipelining as a batch cannot be further divided. The above two cases will result in under-utilization of GPU resources. The second issue becomes more relevant when training on a large number of GPUs. Pipelining can improve the utilization of resources, but it's still limited in performance compared to DP after a certain number of GPUs, and the length of the pipeline is limited by the batch size. Because of the above two issues, there is a need to further optimize both basic and pipeline model parallelism to get better performance.

C. Model Parallelism with Delayed Synchronization

When the DNN size and image size become too large, it is not possible to train the DNN with a Batch Size (BS) > 1 on multiple GPUs. Training hyperparameters like the learning rate and momentum depend on the BS, and training a model with BS equal to 1 can lead to accuracy degradation as gradients are not stable. Past research [14] has shown that the optimal

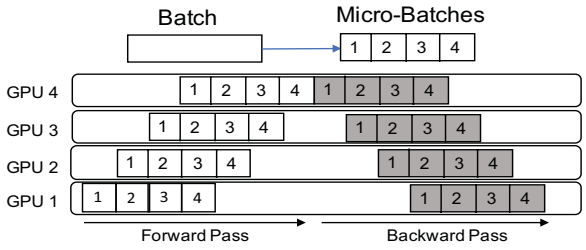


Fig. 5. Computation pattern in Model Parallelism with pipelining approach. Input batch is divided into smaller micro-batches to optimise the performance of basic model parallelism.

BS is between 2 and 32, while [15] shows that increasing BS can improve DNN training. A simple way to emulate BS equal to N behavior is via the Delayed Synchronization approach. In this approach, the global parameter update is not performed immediately after the backward pass. In each backward pass, gradients are added to the previous gradients. After N backward passes, the accumulated gradients are used to update the DNN parameters. Figure 6 shows an example computation pattern of Model Parallelism using a DS approach. Parameters are updated after the 2nd backward pass in Figure 6. The main issue with this approach is that it does not increase the performance as we increase the BS and throughput (images/second) remains constant in this approach. Therefore, we need an approach that can train a DNN model with any batch size and achieve better performance.

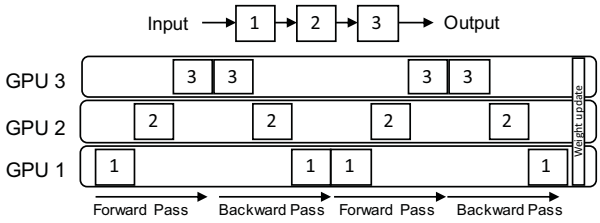


Fig. 6. Model Parallelism with Delayed Synchronization approach.

V. Proposed GPU Enabled Memory Aware Model Parallelism System (GEMS)

We investigate several design schemes to address the limitations in existing approaches. Due to lack of publicly available implementations of basic model parallelism, pipeline parallelism, and model parallelism with delayed synchronization, we implement all existing approaches as well. We call them GEMS-Basic and GEMS-Pipeline. These are in addition to the three proposed approaches discussed below.

A. Memory Aware Synchronised Training (GEMS-MAST)

To address the limitations discussed in Section IV, we propose a new design called Memory Aware Synchronised Training (GEMS-MAST).

Memory View (Motivation): One of the major problems with basic and pipelining model parallelism is the under-utilization of resources. Figure 3 shows the memory vacuum

during forward and backward propagation. After completing the forward and backward passes for a given model partition, the GPU has both free memory and free compute. This free memory can be utilized to perform the forward and backward passes of a new model. GEMS-MAST uses this free memory and compute by training a replica of the same DNN in an inverted manner. Figure 7 shows a memory view of the GEMS-MAST design for forward and backward passes of two model replicas and the improvement made possible by it.

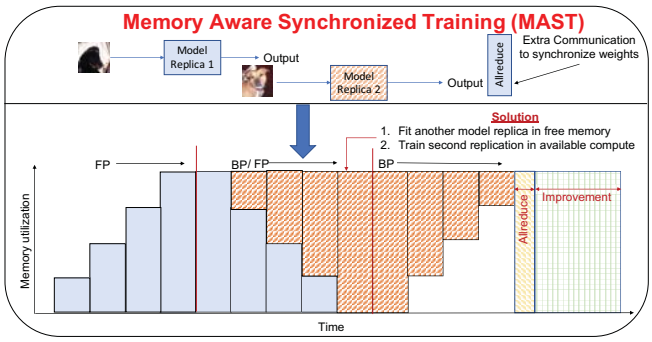


Fig. 7. Memory-Aware Synchronous Training (GEMS-MAST): Compared to GEMS-Basic, the proposed GEMS-MAST design utilizes free memory available and fully utilizes the GPU computation to train an additional model replica.

Computational View (Detailed Design): The proposed design can be viewed as training two model replicas on the same number of resources before applying data parallelism in the end to synchronize the parameters. Therefore, each process/GPU will have two model partitions that are trained independently, and their parameters are synchronized in the end. Figure 8(b) shows the order of computation and communication in the proposed GEMS-MAST approach. Training model replica 1 is similar to training in GEMS-Basic. After completing the forward and backward passes on model replica 1, GPU4 has free memory and compute while the other GPUs are applying backpropagation to their respective partitions of model replica 1. GPU4 will start the forward pass for model replica 2, essentially training model replica 2 in an inverted way. After applying forward and backward pass for both model replicas, the parameters are synchronized with an allreduce operation. It is important to note here that two allreduce operations are required to synchronize parameters as each rank has two model partitions; however, these allreduce operations are not global. For example, to synchronize parameters for model partition 1, processes on GPU4, and GPU1 will perform an allreduce, while at the same time processes on GPU3 and GPU2 will perform an allreduce for model partition 2. Similarly, an allreduce is performed for model partitions 3 and 4. Hence, the allreduce operation can be overlapped.

B. Memory Aware Synchronized Training with Enhanced Replications (GEMS-MASTER)

Instead of synchronizing the parameters after applying the forward and backward passes for model replicas 1 and 2, more compute can be stacked together. Specifically, the allreduce overhead can be minimized by stacking compute. It is important to note that we are still using two model replicas even if we are applying more than two forward and backward passes, as the same model replicas can be used for subsequent forward and backward passes. Each backward pass on the same model replica has different gradients because the input batch is different. Gradients for the same model replica can be reduced locally with a summation operation. In the end, we have to synchronize the parameters of two model replicas; therefore, we need only two allreduce operations.

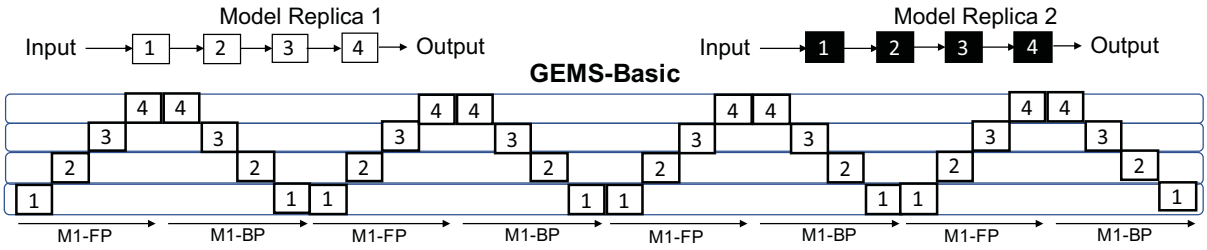
This approach enables researchers to train a model with any batch size on the same number of resources and gives improvement as you increase the batch size, which is not possible in GEMS-Basic with delayed synchronization (Section IV-C). Figure 8(c) shows the performance improvement possible by using GEMS-MASTER compared to GEMS-Basic and GEMS-MAST. It also shows the computation and communication (allreduce) pattern for the GEMS-MASTER design and contrasts them with other approaches. GEMS-MASTER is a generalized version of the GEMS-MAST design. The number of compute replications (denoted by η in Section VI) is 2 for GEMS-MAST and 4 for GEMS-MASTER.

C. Integrating Memory Aware Model Parallelism and Data Parallelism (GEMS-Hybrid)

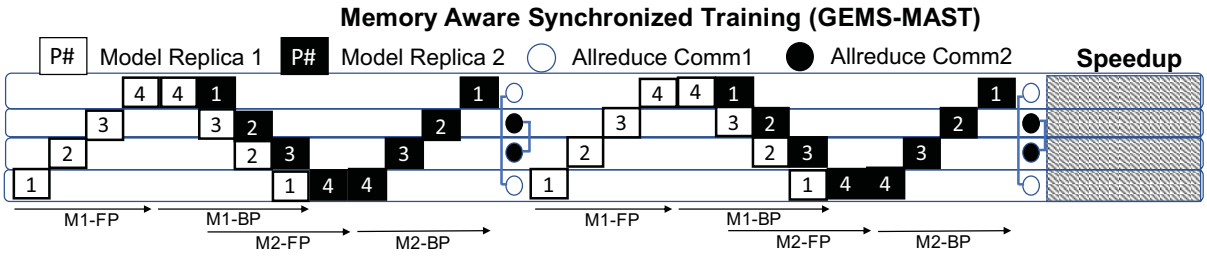
Model parallelism is suitable for out-of-core DNNs, but it is generally not as scalable as Data Parallelism (DP). Traditionally, DP has given near-linear speedup for models trainable on a single GPU. With some minor modifications, model parallelism methods can be combined with DP such that out-of-core DNNs can be trained while maintaining the performance speedup of DP.

Hybrid Model Parallelism (GEMS-HY Basic): Both basic and pipeline model parallelism can be combined with data parallelism. At the end of the backward pass, each GPU/process initiates an allreduce operation for the model partition it is responsible for. Figure 9(a) shows a basic implementation of the hybrid design. An Allreduce operation starts once all the processes have finished the backward pass for their model partition. There is no overlap of computation and communication. But in hybrid model parallelism, the allreduce communicator does not have all the processes, but only the processes which have the same model partition. There will be several simultaneous allreduce operations with a different set of processes. The number of simultaneous allreduce operations is equal to the number of model partitions. Figure 9(b) shows the computation and communication patterns with respect to time for two model parallelism replicas.

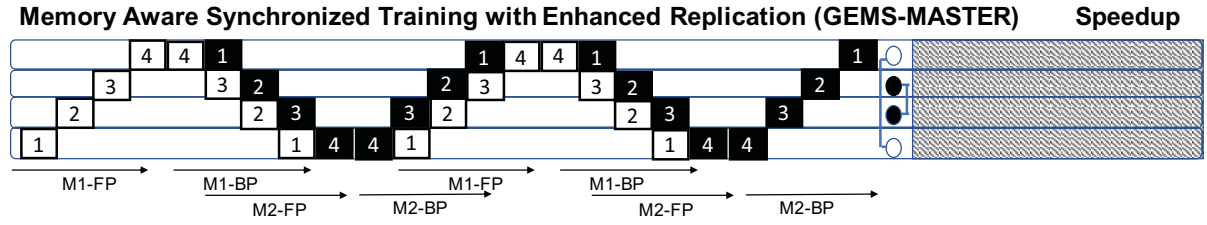
Hybrid Memory Aware Synchronized Training: Similar to the GEMS-HY Basic, DP can be combined with the proposed GEMS-MAST and GEMS-MASTER; labeled **GEMS-HY MAST** and **GEMS-HY MASTER** in Section VII, respectively. Similar to GEMS-MAST, there are two allreduce operations at the end of each model partition in the hybrid design. Each allreduce starts after all processes have finished



(a) Delayed synchronization can be used in GEMS-Basic to train a model with any batch size but there is no computation overlap possible.

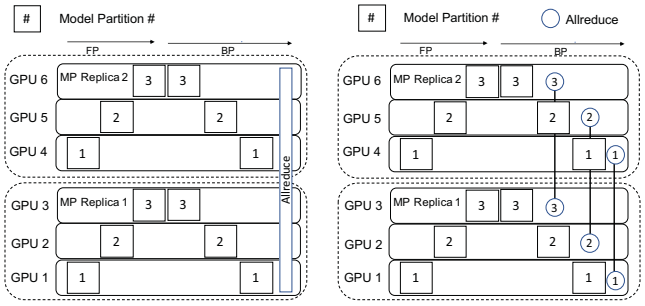


(b) The proposed design overlaps the computation by efficiently utilizing GPU memory (Figure 7), and enables researchers to train larger batch sizes with better performance. Allreduce is used to synchronize the parameters of two model replicas.



(c) The proposed GEMS-MASTER design can further enhance the performance for any batch size by overlapping computation with more replications.

Fig. 8. Comparison of proposed GEMS-MAST and GEMS-MASTER designs with GEMS-Basic



(a) Basic Allreduce (b) Optimized Allreduce

Fig. 9. Hybrid Design (GEMS-HY Basic). In Figure 9(a), an allreduce operation is called once all the processes have finished a backward pass. This implementation can lead to congestion and reduced allreduce performance. While in Figure 9(b), communication is overlapped with computation. Since an allreduce here is not global, the allreduce operations can be applied as soon as a partition finishes the backward pass.

the backward pass due to the data dependency. Unlike GEMS-HY Basic, the number of processes in each allreduce is twice the number of DP clusters since each DP cluster has two model parallelism replicas. In the GEMS-HY MASTER approach, the number of allreduce operations and the number of processes in an allreduce operation remains the same; therefore, the allreduce cost can be minimized by increasing the number of replications (η). Figure 10 shows the overall architecture of the GEMS-HY MAST approach.

VI. Analytical Model for Model Parallelism

Basic Model Parallelism: We present an anlytical model to calculate the time spent in computation and communication in the basic model parallelism. In the end, we calculate the idle time per GPU for basic model parallelism. The number of model partitions as well as the number of GPUs are represented by n. The computation and communication time are denoted by $T_{compute}$ and $T_{communication}$, respectively. $T_{compute}$ is equal to the time spent in forward and backward pass in each partition. $T_{communication}$ is given by Eq 2. $UnderutilizedCompute_i$ denotes the time when the i^{th} GPU is free (idle) during one complete forward and backward pass in DNN training. FP_i and BP_i denote the forward and

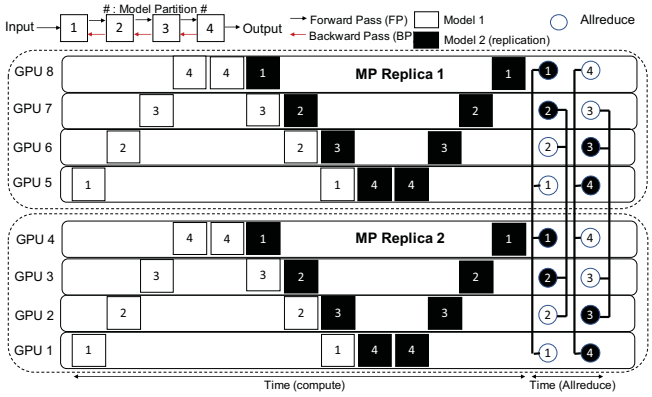


Fig. 10. Proposed GEMS-Hybrid approach (GEMS-HY MAST). The GEMS-MAST design is combined with Data parallelism to improve performance with a scale-out approach. GEMS-MAST is replicated, and an allreduce operation is used to synchronize the performance. Instead of two processes involved in allreduce as in Hybrid Model Parallelism (Figure 9(b)), there are four processes in each allreduce in GEMS-HY MAST since each partition is on 4 GPUs in two GEMS-MAST replicas.

backward pass times respectively for i^{th} model partition. n_i is the number of send-recv operations between model partition i and $i+1$. The latency of communicating a given message is denoted by lat, e.g., lat_{ij} denotes the latency of sending a message from i to j.

$$T_{compute} = \sum_{i=1}^{n} FP_i + \sum_{i=1}^{n} BP_i \tag{1}$$

$$T_{communication} = 2 \times \sum_{i=1}^{n-1} (\sum_{j=1}^{n_i} lat_{ij}) \tag{2}$$

$$T_{total} = T_{compute} + T_{communication} \tag{3}$$

$$UnderutilizedCompute_i(\%) = 100 \times \frac{(T_{compute} - (FP_i + BP_i))}{T_{compute}} \tag{4}$$

Model Parallelism with Delayed Synchronization: The time required for a batch size of 'N' can be seen in Eq 5. Similarly, $T_{communication}$ is equal to $T_{communication}$ (Eq 2) multiplied by 'N'. The total time is given by Eq 3.

$$T_{compute} = N \times (\sum_{i=1}^{n} FP_i + \sum_{i=1}^{n} BP_i) \tag{5}$$

GEMS-MAST: Eqs. 6 and 7 show the time spent in computation and communication for the proposed GEMS-MAST design. Eq. 8 calculates the underutilized compute per GPU. It may be directly inferred that the under-utilized compute for MAST is less than MP-Basic by comparing Eqs. 8 and 4.

$$T_{compute\_MAST} = \sum_{i=1}^{n} FP_i + 2 \sum_{i=1}^{n} BP_i + FP_n \tag{6}$$

$$T_{communication} = 2 \times T_{allreduce} + 4 \times \sum_{i=1}^{n-1} (\sum_{j=1}^{n_i} lat_{ij}) \tag{7}$$

$$UnderutilizedCompute_i(\%) = (1 - \frac{(FP_i + BP_i + FP_{n-i+1} + BP_{n-i+1})}{T_{compute\_MAST}}) \times 100 \tag{8}$$

GEMS-MASTER: Eq. 9 shows the computation time for MASTER with respect to the number of replications (η) and forward/backward pass time. The total time for MASTER is T_{c_MASTER} + $T_{allreduce}$ (time required for allreduce operation) + $T_{communication}$ (time required for send/recv operation in forward and backward pass, similar to Eq 2). The time improvement can be calculated by subtracting Eq 9 and $T_{allreduce}$ from Eq 5. It is hard to calculate time for forward and backward pass for a particular partition as it needs profiling variables in implementation. Thus, Eq 10 can be rewritten as Eq 11 where η is the number of replications (Analysis in Section VII-E).

$$T_{c\_MASTER(\eta)} = \begin{cases} \frac{\eta+1}{2} \sum_{i=1}^{n} (FP_i + BP_i) \\ +(\frac{\eta-1}{2})(FP_n + BP_n) & \text{when } \eta \text{ is odd} \\ \\ \frac{\eta}{2} \sum_{i=1}^{n} (FP_i + BP_i) \\ +(\frac{\eta}{2} - 1)(FP_n + BP_n) + FP_n \\ \sum_{i=1}^{n} BP_i & \text{else} \end{cases} \tag{9}$$

$$T_{improved}(\eta) = \frac{\eta}{2} \sum_{i=1}^{n-1} FP_i + (\frac{\eta}{2} - 1) \sum_{i=1}^{n-1} BP_i - T_{allreduce} \tag{10}$$

$$T_{improved}(\eta) = (\frac{\eta}{2} - 1)(T_{improved}(4) - T_{improved}(2)) + T_{improved}(2) \tag{11}$$

VII. Performance Evaluation

We first provide details on the evaluation platform and important performance metrics. Next, we divide the performance evaluation section into three broad categories:

1) In-depth Performance Evaluation of various GEMS approaches (Sections VII-D, VII-E, and VII-F).
2) GEMS vs. existing frameworks that offer model/hybrid parallelism (Section VIII).
3) Impact of GEMS on a real-world digital pathology problem (Section IX-B).

A. Evaluation Platform

All the experiments were conducted on LLNL/Lassen, which is an OpenPOWER system equipped with POWER9 processors and 4 NVIDIA Volta V100 GPUs. Each node of the cluster is a dual-socket machine, and each socket is equipped with 22-core IBM POWER9 processors and 2 NVIDIA Volta V100 GPUs with 16 GB HBM2. NVLink is used to connect GPU-GPU and GPU-Processor. X-Bus is used to connect two NUMA nodes.

MPI Library: MVAPICH2-GDR [16] 2.3.3 is used.

Deep Learning Framework: TensorFlow v1.14 [17] is used for performance evaluation and Keras model definition module included in TensorFlow is used to define DNNs.

Deep Neural Networks: We used ResNet variants defined in Keras examples/applications [18].

B. Evaluation Setup and Performance metrics

Two performance metrics are used to evaluate the performance of proposed approaches; 1) Time per batch and 2) Images per sec. Performance metrics and terms used in the characterization are explained below.

- **Time per batch:** Time required to complete one weight update step.
- **Images per sec:** Number of images processed per sec in DNN training. Images per sec = EBS / Time per batch.
- η: Number of model replications in GEMS-MASTER or GEMS-Basic.
- **Partition:** A slice of DNN, which is trained on single GPU in model parallelism.
- **Parts:** Number of micro-batches in model parallelism with pipelining.
- **Replica:** A copy of DNN with same parameters.
- **Effective Batch Size (EBS):** This is equal to the effective number of samples used in each weight update during backward pass. It is calculated as follows:
  - GEMS-Basic: EBS = BS (Batch Size) $\times$ η
  - GEMS-MAST: EBS = 2 $\times$ BS
  - GEMS-MASTER: EBS = η $\times$ BS
  - GEMS-HY MP: EBS = number of DP replicas (DPR) $\times$ BS
  - GEMS-HY MAST: EBS = DPR $\times$ 2 $\times$ BS
  - GEMS-HY MASTER: EBS = DPR $\times$ η $\times$ BS

C. Evaluation Methodology

In this section, we will describe our experimental methodology used to conduct the experiments. Performance can be evaluated in two ways; 1) the number of images processed per sec (higher the images/sec less will be the time required to complete one epoch) and 2) Time needed to reach a particular accuracy for the given DNN. We evaluate our proposed designs on both fronts. Broadly, our experiments can be divided into three categories; 1) Performance analysis of proposed designs on different DNNs, 2) Comparison of GEMS with existing strategies, and 3) Showing the benefit of GEMS on a real dataset.

In Sections VII-D, VII-E, and VII-F, we evaluate the performance of proposed designs in terms of time required to process one batch of inputs. One of our main motivations is to accelerate the training of DNNs, where the pipelining approach is not possible. In other words, the maximum trainable batch size on the given number of resources is 1. Therefore in Section VII-D, we increase the size of the DNN so that maximum trainable batch size remains one and we can show the effect of the number of model partitions on the speedup of GEMS-MAST design. GEMS-MASTER is a generalized version of GEMS-MAST; therefore, we pick one DNN evaluated in Section VII-D to demonstrate the speedup for the different numbers of replications η (a parameter used in GEMS-MASTER). Our motive in this paper is to enable the training of large DNNs; hence we scale ResNet-1k on 512$\times$512 image size to 1024 GPUs and compare proposed GEMS-HY MAST and GEMS-HY MASTER with GEMS-HY Basic.

In Section VIII, we compare our proposed designs existing solutions like spatial parallelism and pipelining. We compare two types of DNNs in comparison with spatial parallelism: 1) DNNs trainable with BS$>$1 and 2) DNNs trainable with maximum BS=1 on a given number of resources. We select ResNet-56 model to compare both pipelining and spatial parallelism approaches. We have considered ResNet-110 in Section IX-B; therefore, we provide a comparison for this model also. Further, we compare two different types of DNNs to show speedup over GPipe. We consider AmoebaNet model (used in [7] to show the speedup of pipeline parallelism) and ResNet-110 (used in Section IX-B). In the end, we train ResNet-110 in histopathology images and show speedup possible with proposed designs in terms of training a DNN model to the required accuracy.

D. Memory Aware Synchronized Training (GEMS-MAST)

We evaluated many ResNet variants to compare GEMS-Basic and the proposed GEMS-MAST design. The main issue with the use of pipelining for model parallelism is that it requires an EBS greater than 1, which is not possible in large DNNs that barely fit in the memory of multiple GPUs. For instance, the ResNet-164 v2 model requires at least 4 GPUs to train on an image input of 1024. Even on 4 GPUs, it requires an EBS of 1, which makes pipelining impossible to use. To train this model with an EBS greater than 1, we have to increase the number of GPUs. If we double the number of GPUs, we cannot train the model with EBS $>$ 2. Therefore, the pipeline cannot be filled completely. Even if we use EBS = 2 on 8 GPUs, the pipelining performance remains a challenge compared to hybrid MP + DP (Section V-C) with two DP replicas. Our proposed GEMS-MAST design enables the researchers to train a model with EBS = 2 and get speedup over GEMS-Basic on the same number of GPUs, which is not possible with any state-of-the-art approach.

Figure 11 shows the performance improvement when using GEMS-MAST over GEMS-Basic. We evaluated different models to show performance improvement for DNNs with different layer counts. Speedup or percent improvement in time depends on the number of model partitions (can be inferred from Eq. 8). In Figure 11, the speedup increases as the number of model partitions increase, which corroborates with Eq. 8. Theoretically, the maximum speedup possible with GEMS-MAST is always less than 1.5x as we can only overlap one full FP and BP in 2 FP's and 2 BP's. Allreduce is also required in the proposed approach to synchronize the parameters. Thus, the speedup is always less than 1.5$\times$. For the 16 GPU case, we are able to get 1.36$\times$ speedup over GEMS-Basic for the ResNet-326 model evaluated on an input image size of 1024 $\times$ 1024.

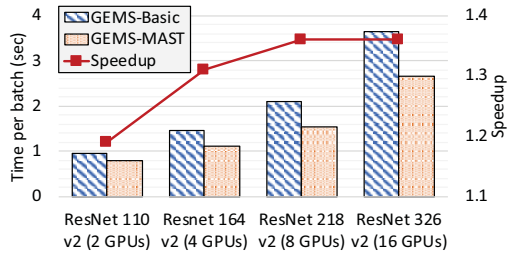


Fig. 11. Performance comparison of GEMS-Basic and GEMS-MAST for ResNet variants on 1024 × 1024 image size

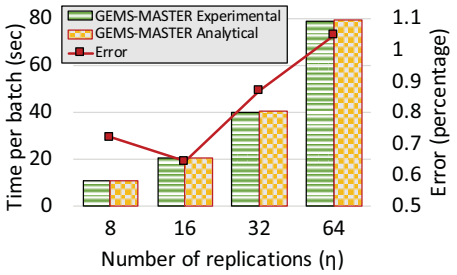


Fig. 13. Experimental Validation of the Analytical Model for GEMS-MASTER(ResNet-1K with 512×512 image size on 8 GPUs)

E. Memory Aware Synchronized Training with Enhanced Replications (GEMS-MASTER)

GEMS-MASTER enables researchers to train DNN with any batch size, which is not possible with GEMS-MAST. Training hyperparameters such as the learning and momentum rates depend on a particular batch size. If the batch size is changed then these hyperparameters must be tuned again to maintain accuracy. Another reason for using this design is to train the model faster, which is not possible in GEMS-Basic with delayed synchronization (Section IV-C). GEMS-MASTER is equivalent to the proposed GEMS-MAST design when $\eta = 2$.

We evaluated GEMS-MASTER on different models trainable on 4 and 8 GPUs to showcase the effectiveness of the proposed design. Figure 12 compare GEMS-MASTER to GEMS-Basic (with delayed synchronization) for ResNet-164. We show up to 1.83× speedup for ResNet-164 on 1024×1024 image size using GEMS-MASTER.

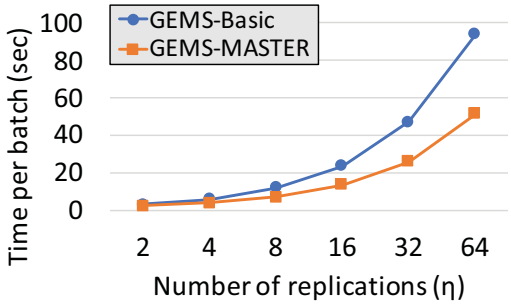


Fig. 12. GEMS-Basic and GEMS-MASTER performance comparison (ResNet-164 for 1024×1024 image size on 4 GPUs)

We present the validation of the analytical model in Figure 13 for ResNet-1k models. Equation 10 or Equation 11 can be used to calculate the amount of improvement possible with GEMS-MASTER for a given η. Equation 10 needs profiling variables to be introduced to calculate the improvement, which is not the primary objective of this study. Therefore we use Equation 11 to calculate the improvement for $\eta > 4$ since we have values for $T_{improved}(\eta = 2)$ and $T_{improved}(\eta = 4)$. Figure 13 shows the time per batch predicted by the analytical model and the actual time along with the error in percentage.

F. GEMS-Hybrid Designs

To showcase the scalability of proposed designs, we trained ResNet-1K on up to 1024 GPUs using hybrid designs (Section V-C). We are able to get near-linear speedup for proposed designs. GEMS-MASTER ($\eta = 64$) gives a better speedup than other approaches because the computation to communication ratio is higher. The number of DP replicas on 1024 GPUs is 128 since the model is partitioned across 8 GPUs; therefore, the ideal speedup is 128×. GEMS-HY MAST and GEMS-HY MASTER give 89× and 124.58× speedup on 128 model replicas (1024 GPUs), respectively.

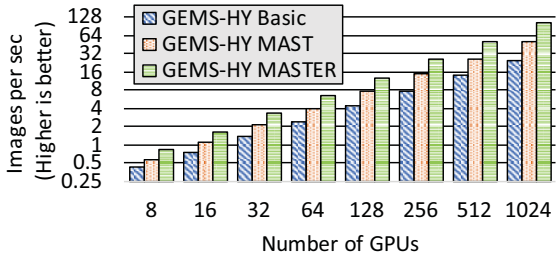


Fig. 14. GEMS-Hybrid: Scaling ResNet-1K on 1,024 GPUs.

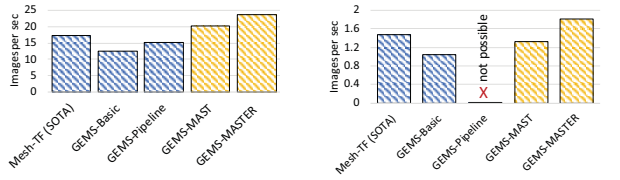
VIII. GEMS vs. Existing Frameworks

In this section, we compare GEMS with state-of-the-art existing frameworks like Mesh-TensorFlow (MTF) and GPipe. Model Parallelism frameworks can be broadly into two categories; 1) Frameworks that use approaches like spatial and channel parallelism and 2) Pipelining frameworks like Gpipe, HyPar-Flow. Therefore we present comparison results for both types of frameworks.

A. GEMS vs. Frameworks with Spatial Parlallelism

There are a few frameworks that implement spatial parallelism to distribute DNN training across multiple nodes. The performance of any parallelization strategy highly depends on the DL framework [19], [20], [21] that is being used in the backend. Therefore, we select MTF for the comparison as it has support for spatial parallelism and uses TensorFlow as backend. In this way, we ensure that the performance gain is the result of the proposed design, not because of the

underlying framework. We compare two types of DL models to demonstrate the effectiveness of our proposed designs: 1) DNN trainable with BS $>$ 1 and 2) DNN trainable only with BS=1. Figure 15(a) shows the comparison for ResNet-56 using 512×512 images trainable with maximum BS=8. We show that GEMS-MAST is 1.31× and 1.16× better than pipelining and MTF, respectively, while the GEMS-MASTER is up to 1.54× and 1.36× better than pipelining and MTF, respectively. Figure 15(b) shows the comparison for ResNet-110 using 1024×1024 images (considered in Section IX-B) trainable with maximum BS=1. Pipelining approach is not possible for this DNN as the maximum BS trainable is 1. We show that GEMS-MASTER is up to 1.74× and 1.24× better than GEMS-Basic and MTF, respectively.



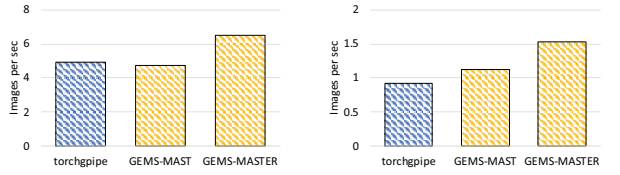
(a) ResNet-56 on 4 GPUs using 512×512 images

(b) ResNet-110 on 4 GPUs using 1024×1024 images

Fig. 15. Comparison of GEMS with Mesh-TensorFlow.

B. GEMS vs. Frameworks with Pipeline Parallelism

We do not compare with GPipe's public version for TensorFlow as it is based on the deprecated *tf.Session* API. However, there exists an implementation of GPipe for PyTorch called torchgpipe [22]. torchgpipe uses python's multiprocessing module to implement the pipelining approach. Therefore, torchgpipe works only on a single node. Directly comparing our TensorFlow implementation with torchgpipe will be unfair as it will also include the performance difference of PyTorch and TensorFlow DL frameworks. Therefore, we implement GEMS-MAST and GEMS-MASTER designs in PyTorch to give a fair comparison with torchgpipe. Figure 16(a) shows the comparison of GEMS-MAST and torchgpipe for AmoebaNet network with 1024×1024 image size. We have used the AmoebaNet network with num_layers=18 and num_filters=208 provided in models directory in torchgpipe's GitHub repo and show that GEMS-MASTER is 1.32× better than torchgpipe. Figure 15(b) shows the comparison for ResNet-110 using 1024×1024 images (considered in Section IX-B) trainable with maximum BS=1. We show that GEMS-MAST is 1.21× and GEMS-MASTER is 1.65× better than torchgpipe.



(a) AmoebaNet on 4 GPUs using 1024×1024 images

(b) ResNet-110 on 4 GPUs using 1024×1024 images

Fig. 16. Comparison of GEMS with torchgpipe.

IX. Digital Pathology

A. Need for Deep Learning

Tall cell variant (TCV) is the most aggressive subtype of Papillary thyroid carcinoma. TCV was first defined as a papillary thyroid cancer when at least 30% tall cells by Hawk and Hazard in 1976 [23]. A cell is considered tall when its height is at least twice or thrice its width. However, there is still controversy regarding the proportion of tall cells in thyroid tissue and the appropriate aspect ratios (height vs. width) for diagnosis. The inherent subjectivity of TCV diagnoses and inter-observer variability provides an opportunity for DL methods to objectively classify TCVs. There is a need to understand both the histopathology and the molecular basis in order to be able to diagnose and treat afflicted patients. A subgroup among authors is working to understand the morphology and histological basis of TCVs and PTCs.

B. Benefits of GEMS for Digital Pathology

To highlight the benefits and impact of GEMS on a real-world use-case, we obtained 16 Hematoxylin and Eosin (H&E) stained WSI, 8 classified as classical papillary thyroid carcinoma (PTC), and 8 classified as the tall cell variant or TCV. Each WSI was divided into 1024×1024 tiles at ×10 magnification level (Figure 2). It should be noted that some tiles will contain little or no tissue. For this study, the tiles with the occupancy of 50% of the area were chosen for further analyses. Our goal is to achieve higher levels of classification accuracy in a reasonable time frame. In this section, we provide results that include both the accuracy and training speed for different models on high-resolution histopathology images shown in Figure 2.

We experimented with several models pre-trained on the ImageNet dataset and fine-tuned them on the digital pathology dataset described above. We eventually settled on a custom ResNet110-v2 model that offered the best accuracy for our use-cases. We trained this model on image tiles of 1024×1024 resolution. Because of memory limitation, ResNet110-v2 with image size 1024×1024 can only be trained with BS 1 using GEMS-BASIC. BS is 2 and 16 for GEMS-MAST and GEMS-MASTER designs, respectively. We have trained the network using the Adam optimizer [24] Table II shows the time required to train the ResNet110-v2 model to 90% accuracy for various GEMS variants including GEMS-Basic, GEMS-MAST, GEMS-MASTER, and GEMS-Hybrid.

It is important to note that the ResNet110-v2 model is an out-of-core DNN, and it is not possible to train this model with a batch size of two on four GPUs with GEMS-Basic. Training accuracy is similar for GEMS-Basic, GEMS-MAST, and GEMS-MASTER. Proposed designs are able to train the DNN to 90% accuracy. We observed a 1.16× speedup for GEMS-MAST and a 1.51× speedup for GEMS-MASTER in terms of the time required to complete one epoch. We scaled the DNN training to a large number of GPUs to demonstrate the scaling efficiency of GEMS-Hybrid. We were able to decrease the training time from 7.2 hours to 28 minutes

Approach	# Nodes (#GPUs)	Epochs	Learning Rate	Total Time in hours (Total Time in mins)	Training speed (img/sec)	Speedup Over GEMS -MASTER
GEMS-Basic	1 (4)	8	0.001	7.25 (435)	1.04	-
GEMS-MAST	1 (4)	8	0.001	6.28 (377)	1.19	-
GEMS-MASTER	1 (4)	7	0.001	4.21 (252)	1.56	1
GEMS-Hybrid	2 (8)	7	0.001	2.51 (151)	2.99	1.9
GEMS-Hybrid	4 (16)	7	0.001	1.34 (80)	5.65	3.6
GEMS-Hybrid	8 (32)	10	0.0025	0.98 (59)	10.69	6.8
GEMS-Hybrid	16 (64)	12	0.0025	0.65 (39.5)	19.18	12.3
GEMS-Hybrid	32 (128)	15	0.004	0.46 (27.8)	34.51	22.1

TABLE II
SCALING RESNET-110 V2 ON 1024 × 1024 IMAGE TILES EXTRACTED FROM HISTOPATHOLOGY DATA (CF. FIGURE 2)

using GEMS-Hybrid. We conducted these experiments to show the impact of the GEMS framework and how GEMS can benefit researchers in finding the right DL model for their applications.

X. RELATED WORK

With the growth of scientific applications requiring massive data sample sizes (e.g. high-resolution images [25]), some models have correspondingly become intractably large to run on a single GPU [26]. Early experiments in model parallelism techniques on GPUs were performed by Alex Krizhevsky in [27]. Krizhevsky's work in [28] introduced a single-tower design that used data parallelism in convolutional layers but model parallelism in fully-connected layers. LBANN introduced many model parallel solutions including support for spatial convolutions split across nodes in [12]. GPipe [7] employs pipeline parallelism to enable the training of a variety of extremely large models like AmoebaNet [29] on Google TPUs and accelerators. GPipe's public version is based on the deprecated tf.Session API, so we implemented their pipelining approach for GEMS (GEMS-Basic and GEMS-Pipeline). Furthermore, we implement proposed designs on the top of pipelining and exploit it wherever possible. FlexFlow [11] searches through parallelization strategies with simulation algorithms and highlights different DNN parallelism dimensions. FlexFlow is built with Legion [30] for communication within a node and GASNet across nodes. We attempted to compare our work with FlexFlow but ran into issues with large models on our system. Mesh-TensorFlow (MTF) [9] is a popular language for distributed DNN training which distributes tensors across a processor mesh. MTF, like GPipe, only works with deprecated TF APIs (sessions, graphs, etc.). Further, MTF users need to re-write their model. Unlike MTF, GEMS does not require any changes to the code/model. Experimental comparisons with MTF and FlexFlow have been discussed in Section VIII. Out-of-core methods like [1], [2] take a different approach to deal with large models, which is not directly comparable to model/hybrid-parallelism. We summarize these related studies and their features in Table I.

XI. CONCLUSION

In this paper, we presented GEMS — an integrated system that provides compute- and memory-efficient model parallel training and scalable hybrid (data + model) parallel training. GEMS employed two novel design strategies (MAST and MASTER) proposed in this paper for efficient out-of-core training. The proposed designs are evaluated against state-of-the-art model parallel approaches from the literature using large DL models such as ResNet-1K. We reported up to 1.36× speedup for MAST and up to 1.83× speedup when using MASTER design. Furthermore, GEMS-Hybrid, a hybrid scheme that combines MAST and MASTER with data-parallelism to scale-out on up to 1,024 GPUs was presented. We achieved a near-linear scaling efficiency of 97.32% for 1,024 Volta GPUs. Finally, we collaborated with pathologists to develop a custom ResNet-110-v2 mode for TCV classification using high-resolution histopathology images. The proposed designs reduced the training time from 7.25 hours to just 28 minutes by exploiting 128 Volta V100 GPUs. We believe that GEMS will pave a way forward for solving overarching problems in digital pathology, computer science, and artificial intelligence. GEMS is an effort that highlights the importance of collaboration across different research themes to solve real-world problems by exploiting large-scale HPC systems.

ACKNOWLEDGEMENT

This research is supported in part by NSF grants #1450440, #1565414, #1664137, #1818253, #1854828, #1931537, #2007991, and XRAC grant #NCR-130002.

REFERENCES

[1] A. A. Awan, C. Chu, H. Subramoni, X. Lu, and D. K. Panda, "OC-DNN: Exploiting Advanced Unified Memory Capabilities in CUDA 9 and Volta GPUs for Out-of-Core DNN Training," in *2018 IEEE 25th International Conference on High Performance Computing (HiPC)*, Dec 2018, pp. 143–152.

[2] P. Markthub, M. E. Belviranli, S. Lee, J. S. Vetter, and S. Matsuoka, "DRAGON: Breaking GPU Memory Capacity Limits with Direct NVM Access," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*, ser. SC '18. Piscataway, NJ, USA: IEEE Press, 2018, pp. 32:1–32:13. [Online]. Available: http://dl.acm.org/citation.cfm?id=3291656.3291699

[3] A. Sergeev and M. Del Balso, "Horovod: Fast and Easy Distributed Deep Learning in TensorFlow," *CoRR*, vol. abs/1802.05799, 2018. [Online]. Available: http://arxiv.org/abs/1802.05799

[4] A. Jain, A. A. Awan, Q. Anthony, H. Subramoni, and D. K. Panda, "Performance Characterization of DNN Training using TensorFlow and PyTorch on Modern Clusters," in *2019 IEEE International Conference on Cluster Computing (CLUSTER)*, 2019, pp. 1–11.

[5] P. Goyal, P. Dollár, R. B. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, "Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour," *CoRR*, vol. abs/1706.02677, 2017. [Online]. Available: http://arxiv.org/abs/1706.02677

[6] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A Large-Scale Hierarchical Image Database," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE, 2009, pp. 248–255.

[7] Y. Huang, Y. Cheng, D. Chen, H. Lee, J. Ngiam, Q. V. Le, and Z. Chen, "Gpipe: Efficient training of giant neural networks using pipeline parallelism," in *NeurIPS*, 2019.

[8] D. Narayanan, A. Harlap, A. Phanishayee, V. Seshadri, N. R. Devanur, G. R. Ganger, P. B. Gibbons, and M. Zaharia, "Pipedream: Generalized pipeline parallelism for dnn training," in *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, ser. SOSP '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 1–15. [Online]. Available: https://doi.org/10.1145/3341301.3359646

[9] N. Shazeer, Y. Cheng, N. Parmar, D. Tran, A. Vaswani, P. Koanantakool, P. Hawkins, H. Lee, M. Hong, C. Young, R. Sepassi, and B. Hechtman, "Mesh-tensorflow: Deep learning for supercomputers," in *Advances in Neural Information Processing Systems 31*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds. Curran Associates, Inc., 2018, pp. 10 414–10 423. [Online]. Available: http://papers.nips.cc/paper/8242-mesh-tensorflow-deep-learning-for-supercomputers.pdf

[10] A. A. Awan, A. Jain, Q. Anthony, H. Subramoni *et al.*, "HyPar-Flow: Exploiting MPI and Keras for Scalable Hybrid-Parallel DNN Training using TensorFlow," *arXiv preprint arXiv:1911.05146*, 2019. [Online]. Available: http://arxiv.org/abs/1911.05146

[11] Z. Jia, M. Zaharia, and A. Aiken, "Beyond data and model parallelism for deep neural networks," *SysML 2019*, 2019.

[12] N. Dryden, N. Maruyama, T. Benson, T. Moon, M. Snir, and B. Van Essen, "Improving strong-scaling of cnn training by exploiting finer-grained parallelism," in *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2019, pp. 210–220.

[13] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[14] D. Masters and C. Luschi, "Revisiting small batch training for deep neural networks," *CoRR*, vol. abs/1804.07612, 2018. [Online]. Available: http://arxiv.org/abs/1804.07612

[15] S. L. Smith, P.-J. Kindermans, and Q. V. Le, "Don't decay the learning rate, increase the batch size," in *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=B1Yy1BxCZ

[16] MVAPICH2: MPI over InfiniBand, 10GigE/iWARP and RoCE, https://mvapich.cse.ohio-state.edu/, 2001, [Online; accessed September 9, 2020].

[17] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin *et al.*, "TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, 2015," *Software available from tensorflow. org*, 2016.

[18] "Keras (CIFAR-10 ResNet)," 2019, [Online; accessed September 9, 2020]. [Online]. Available: https://keras.io/examples/cifar10_resnet/

[19] A. Jain, A. Awan, Q. Anthony, H. Subramoni, and D. K. Panda, "Performance characterization of dnn training using tensorflow and pytorch on modern clusters," September 2019.

[20] R. D. Fonnegra, B. Blair, and G. M. Díaz, "Performance comparison of deep learning frameworks in image classification problems using convolutional and recurrent networks," in *2017 IEEE Colombian Conference on Communications and Computing (COLCOM)*, 2017, pp. 1–6.

[21] A. A. Awan, H. Subramoni, and D. K. Panda, "An in-depth performance characterization of cpu-and gpu-based dnn training on modern architectures," in *Proceedings of the Machine Learning on HPC Environments*. ACM, 2017, p. 8.

[22] C. Kim, H. Lee, M. Jeong, W. Baek, B. Yoon, I. Kim, S. Lim, and S. Kim, "torchgpipe: On-the-fly pipeline parallelism for training giant models," 2020.

[23] W. A. Hawk, J. B. Hazard *et al.*, "The many appearances of papillary carcinoma of the thyroid," *Cleveland Clinic Quarterly*, vol. 43, no. 4, pp. 207–216, 1976.

[24] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015. [Online]. Available: http://arxiv.org/abs/1412.6980

[25] L. Hou, N. J. Parmar, N. Shazeer, X. Song, Y. Li, and Y. Cheng, "High resolution medical image analysis with spatial partitioning," in *High Resolution Medical Image Analysis with Spatial Partitioning*, 2019.

[26] M. Shoeybi, M. A. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, "Megatron-lm: Training multi-billion parameter language models using model parallelism," *ArXiv*, vol. abs/1909.08053, 2019.

[27] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, p. 84–90, May 2017. [Online]. Available: https://doi.org/10.1145/3065386

[28] A. Krizhevsky, "One Weird Trick for Parallelizing Convolutional Neural Networks," *CoRR*, vol. abs/1404.5997, 2014. [Online]. Available: http://arxiv.org/abs/1404.5997

[29] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized Evolution for Image Classifier Architecture Search," *CoRR*, vol. abs/1802.01548, 2018. [Online]. Available: http://arxiv.org/abs/1802.01548

[30] M. Bauer, S. Treichler, E. Slaughter, and A. Aiken, "Legion: Expressing Locality and Independence with Logical Regions," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '12. Los Alamitos, CA, USA: IEEE Computer Society Press, 2012, pp. 66:1–66:11. [Online]. Available: http://dl.acm.org/citation.cfm?id=2388996.2389086

Appendix: Artifact Description/Artifact Evaluation

SUMMARY OF THE EXPERIMENTS REPORTED

The artifact contains the GEMS framework that we have designed for this work. GEMS is a distributed training framework for TensorFlow, which enables users to perform model parallelism on GPUs in a user transparent manner. GEMS has basic model parallelism, pipelining along with the proposed designs MAST, MASTER, hybrid model parallelism, hybrid MAST, and hybrid MASTER. GEMS also implements CUDA aware MPI operations like point to communications and Allreduce communication operations for TensorFlow using TensorFlow operators written in C++.

1 EXPERIMENT WORKFLOW

Datasets can be imported by using TensorFlow's data import functions. In this study, we have used *ImageDataGenerator* class from *tensorflow.keras* module and imported the dataset using *flow_from_directory* function, which gives a generator for training and validation dataset. GEMS uses this generator to generate input and output for the distributed DNN training.

One can use following command to run the GEMS framework on 4 GPUs by using the proposed optimization schemes as follows. For **GEMS-MASTER** ,

```
$ mpirun_rsh --export-all -n 4 --hostfile hosts
MV2_USE_CUDA=1 MV2_SUPPORT_DL=1 python gems.py
--strategy master
--mp-size 4
--save-data /PATH/TO/FOLDER
--learning-rate $LR
--batch-size $BS
--replications $eta
--dataset /PATH/TO/DATASET
--num-epochs 100
```

When the number of processes is more than the number of partitions (–mp-size) for DNN training, GEMS will automatically use the hybrid approach for the given strategy (Basic, MAST, MASTER). For **GEMS-Hybrid (MASTER)**,

```
$ mpirun_rsh --export-all -n 16 --hostfile hosts
MV2_USE_CUDA=1 MV2_SUPPORT_DL=1 python gems.py
--strategy master
--mp-size 4
--save-data /PATH/TO/FOLDER
--learning-rate $LR
--batch-size $BS
--replications $eta
--dataset /PATH/TO/DATASET
--num-epochs 100
```

2 EVALUATION AND EXPECTED RESULT

The detail log and results will be generated to the standard I/O, and a CSV file containing the train and test log will also be generated. CSV file contains information like training loss, training accuracy, test loss, test accuracy, time spent in training, time spent in testing for every epoch. The output of the CSV file is used to generate Table II in the paper. By default, model weights are saved in HDF5 data format after every epoch that can be used to continue the training later or make predictions.

3 EXPERIMENT CUSTOMIZATION

GEMS provides customization for model hyperparameters and model parallelism hyperparameters in the form of command-line arguments. GEMS also provides the process to GPU mapping support in TensorFlow for HPC systems, which have multiple GPUs per node. Currently, we have the following command-line arguments to customize the training

- –learning-rate
- –batch-size
- –num-epochs
- –dataset (/PATH/TO/DATASET/FOLDER)
- –save-data (/PATH/TO/FOLDER where model weights and CSV file should be saved)
- –strategy (Basic, MAST, and MASTER)
- –mp-size (number of model splits)
- –lps (number of layers per split)
- –parts (number of microbatches when pipelining is possible)
- –num-gpus-per-node (number of GPUs per node)

How software can be obtained (if available)
In the future, we plan to release the software through the project website and it will be downloadable without any restrictions.

ARTIFACT AVAILABILITY

Software Artifact Availability: Some author-created software artifacts are NOT maintained in a public repository or are NOT available under an OSI-approved license.

Hardware Artifact Availability: There are no author-created hardware artifacts.

Data Artifact Availability: There are no author-created data artifacts.

Proprietary Artifacts: There are associated proprietary artifacts that are not created by the authors. Some author-created artifacts are proprietary.

BASELINE EXPERIMENTAL SETUP, AND MODIFICATIONS MADE FOR THE PAPER

Relevant hardware details: System LLNL/Lassen, IBM Power9, NVIDIA Volta V100-NVLink GPUs

Operating systems and versions: RHEL7.6

Compilers and versions: xl/2020.03.18

Applications and versions: TensorFlow 1.14

Libraries and versions: MVAPICH2-GDR 2.3.3

Key algorithms: Distributed Forward and Back Propagation for Deep Neural Networks

Input datasets and versions: Histopathology Images: 16 Hematoxylin and Eosin (H&E) stained WSI, 8 classified as classical papillary thyroid carcinoma (CPTC), and 8 classified as TCV