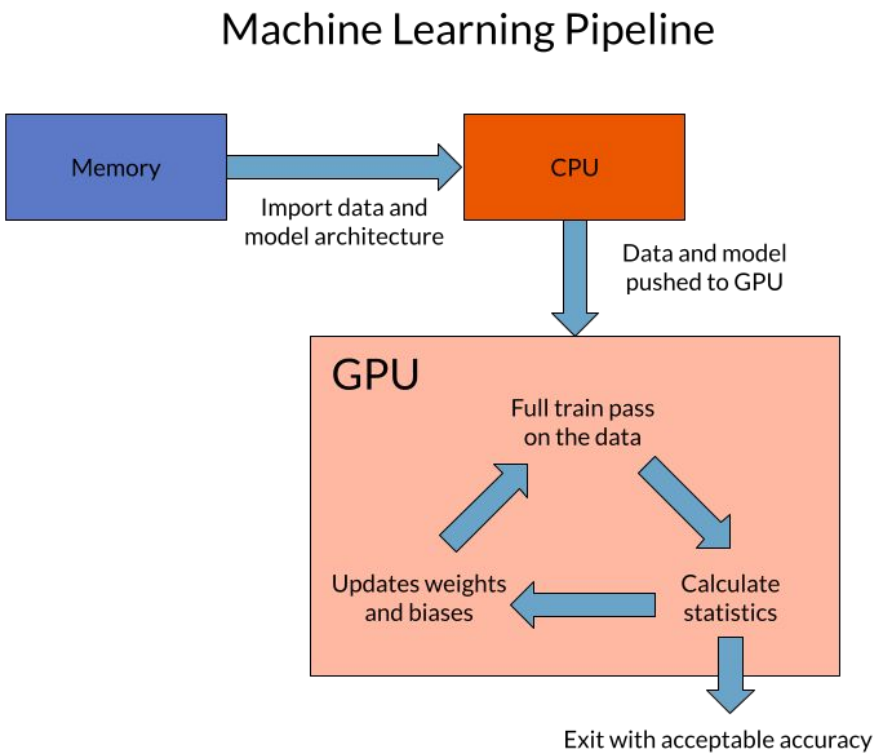
# Synchronization

Omri Steinberg-Tatman and Jackson Vaughn

# The Background Research

# Conventional Model Training




Model training on a large datasets takes considerable time

- The larger the dataset, the longer it takes to train a model.

Only so much can be done with a single GPU

- Data has to fit on GPU
- GPU only has a set amount of processing power

# A Single GPU

How can we use a single GPU most efficiently?

Possible speedup solutions:

- Speedup training pass
  - Reduce dataset size
  - Reduce model depth
- Store more information on GPU
  - Smaller individual data size
    - Lower granularity (ex: lower definition images)
  - Reduce model size

However, we want to use the same data and same model.

- By utilizing multiple GPUs, the model could be trained in parallel to reduce training time. How is this done?

# The Problem

How do you train separate models and transform them into one cohesive model?

# The Solution

Synchronization

# Methods of Parallelization

There are two main types of model parallelization

*Bulk Synchronous Parallel (BSP)*

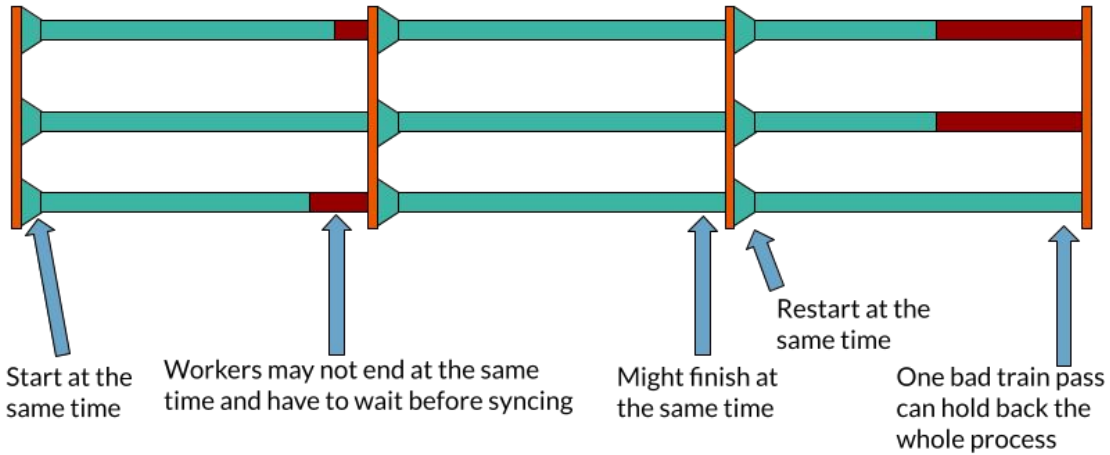
- Model synchronize at the end of each superstep
- GPUs will not proceed until the model parameters have been fully updated by *all* workers

*Asynchronous Parallel (ASP)*

- Model do not synchronize at the end of each superstep
- Model starts the next superstep before receiving the latest parameters
- Usually accomplished by a different synchronization implementation

**Reference:** C. Zhang, H. Tian, W. Wang and F. Yan, "Stay Fresh: Speculative Synchronization for Fast Distributed Machine Learning," *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, Vienna, Austria, 2018, pp. 99-109, doi: 10.1109/ICDCS.2018.00020.

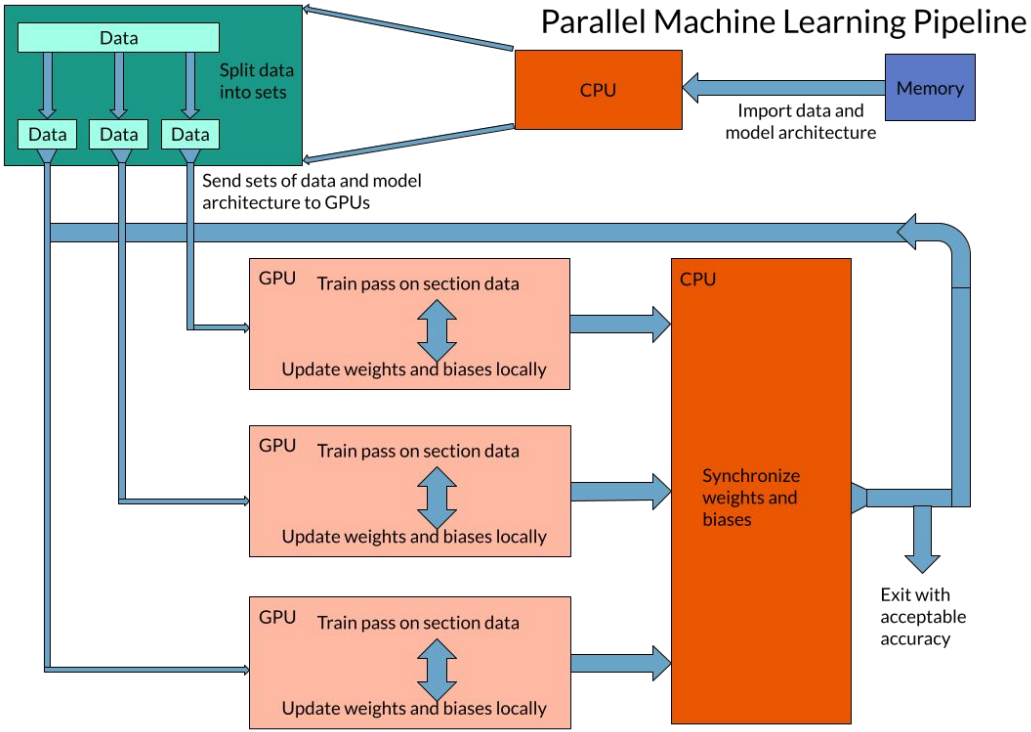
# Bulk Synchronous Parallelism




- Parameters are synchronized only at the end of each superstep
  - Next superstep is blocked until updated model parameters are calculated
- Simpler and more predictable
  - Usually results in better convergence
- Hefty synchronization overhead for larger models
- All workers may have to wait for slowest worker
- A very popular scheme for synchronization of large scale models

**Reference:** Y. Li, H. Wan, B. Jiang and X. Long, "More Effective Synchronization Scheme in ML Using Stale Parameters," *2016 IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, Sydney, NSW, Australia, 2016, pp. 757-764, doi: 10.1109/HPCC-SmartCity-DSS.2016.0110.

# Our Synchronization Pipeline




The process of combining the training multiple models into one cohesive model

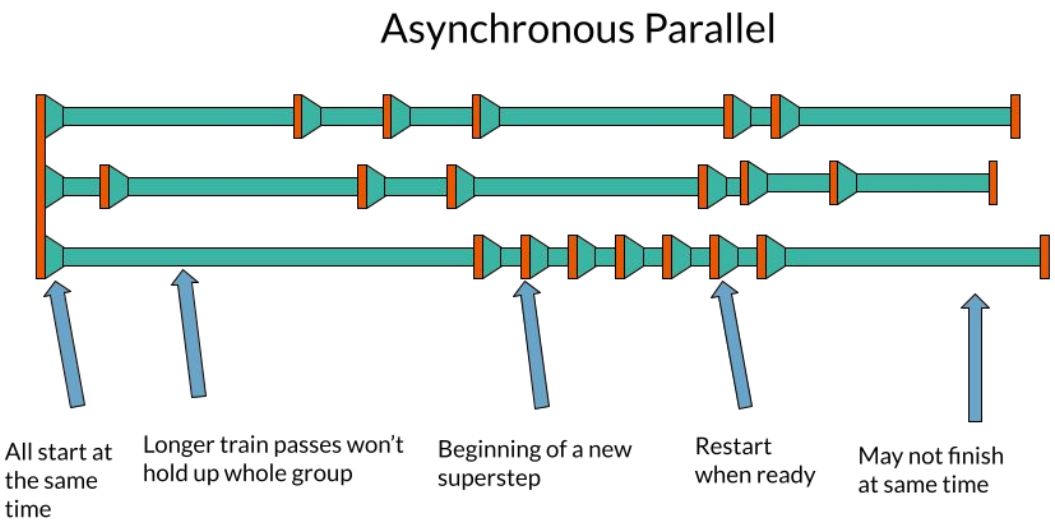
After synchronization, each GPU is updated, and the model is run again.

Training step is still fully done, but the work is spread out over multiple GPUs

Considerations:

- What method of synchronization do you use?
- How often do you synchronize?

# Asynchronous Parallelism




- Due to the shortcomings of BSP, various asynchronous models have been proposed
- Workers can start the next superstep before receiving updated parameters
- The "synchronizing" functionality is achieved through other means (shared buffers, etc.)
- Generally has faster convergence times than BSP
  - Model is not guaranteed to converge

**Reference:** Y. Li, H. Wan, B. Jiang and X. Long, "More Effective Synchronization Scheme in ML Using Stale Parameters," *2016 IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, Sydney, NSW, Australia, 2016, pp. 757-764, doi: 10.1109/HPCC-SmartCity-DSS.2016.0110.

# Asynchronous Parallelism Implementations

Shared Data Structures

- Use shared memory or buffers
- Synchronization through data dependencies

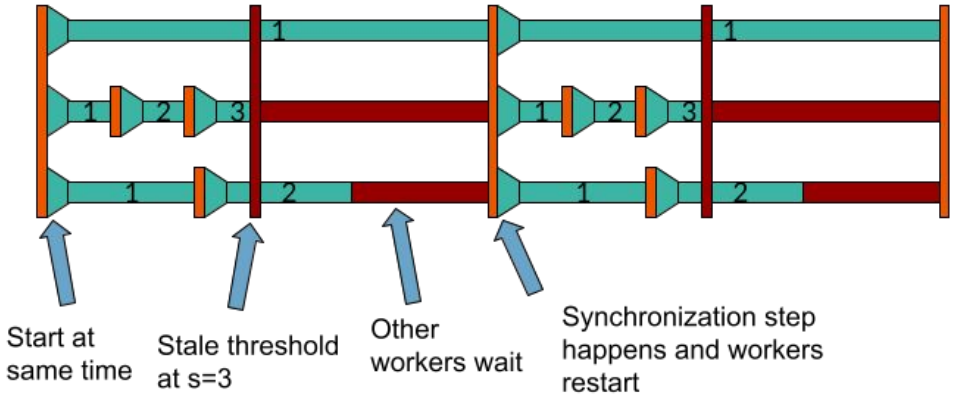
Continuous Communication between Parallel Nodes

- Ongoing exchange among parallel workers
- Updates are shared in real time
- No strict synchronization points

Local Synchronization Points

- Multiple, smaller, bulk synchronization parallelism
- Use other method of asynchronous parallelism to communicate

# Combining BSP and ASP




## Stale-Synchronous Parallel (SSP)

- Uses a Parameter Server (PS) to aggregate local updates of concurrent workers
- Workers train asynchronously and independently to update model parameters to the PS
- To ensure each local model is not “stale”, faster workers are blocked and they can not exceed slower workers by *n* supersteps.
  - This effectively enforces synchronization
- SSP generally suffers from high generalization error when compared to BSP
  - SSP processes less samples than BSP which causes the gradient to be noisier and less accurate
- *At its best, it can run faster than BSP but still has good convergence*

**Reference:** S. Tyagi and M. Swany, "Accelerating Distributed ML Training via Selective Synchronization (Poster Abstract)," 2023 IEEE International Conference on Cluster Computing Workshops (CLUSTER Workshops), Santa Fe, NM, USA, 2023, pp. 56-57, doi: 10.1109/CLUSTERWorkshops61457.2023.00023.
Yang, Rui & Zhang, Jilin & Wan, Jian & Zhou, Li & Shen, Jing & Zhang, Yunchen & Wei, Zhenguo & Zhang, Juncong & Wang, Jue. (2019). Parameter Communication Consistency Model for Large-Scale Security Monitoring Based on Mobile Computing. IEEE Access. PP. 1-1. 10.1109/ACCESS.2019.2956632.

# When to Synchronize?

Synchronization can be done at many points during the process, and the superstep can be defined by many things

- Can be based on train length
  - Epochs
  - Iterations
  - Timing
- Based on values
  - Accuracy
  - Loss
- Based on outside factors
  - Resource utilization
  - Dynamic

Anything that fits your hardware and model

# Methods of Synchronization

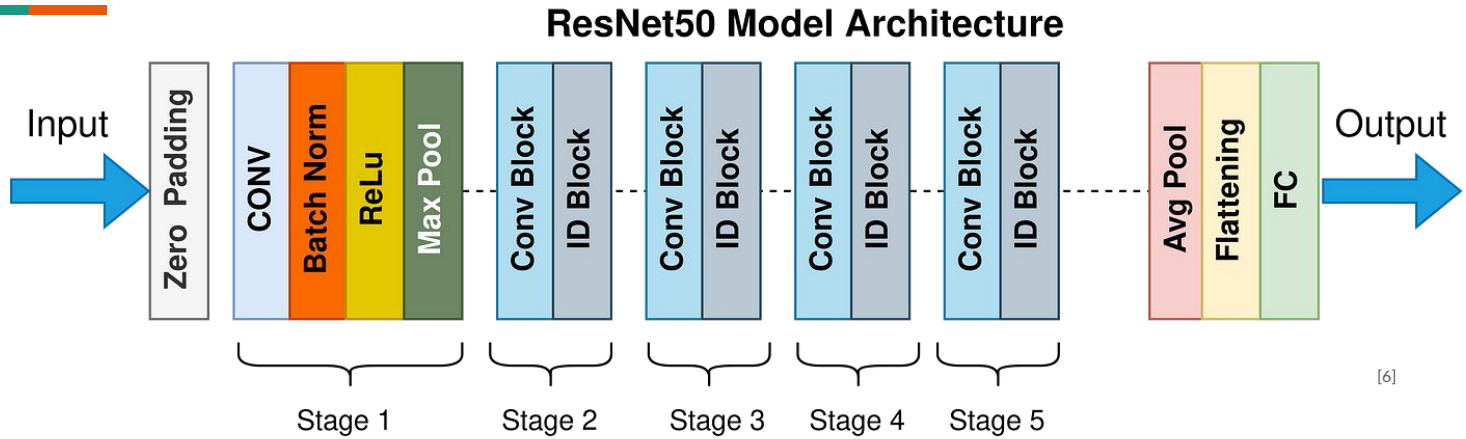
There are many different methods to synchronize model parameters

- Random Forest
  - Integrates the results of many decision trees to arrive at a single outcome
- Weighted Average
  - Aggregates predictions from many models with each model being weighted by some measure
- Outlier Processing
  - Takes the minimum or maximum, or removes outliers above or below some threshold then takes a measure
- Distribution Measures
  - Takes the mean, or the median, or other distribution measure of the parameters

**Reference:** T. T. Aurpa, S. M. Jeba and S. U. Rasel, "Ensemble Methods of Machine Learning Algorithms for Early Diabetic Detection in Comparison," 2023 1st International Conference on Circuits, Power and Intelligent Systems (CCPIS), Bhubaneswar, India, 2023, pp. 1-6, doi: 10.1109/CCPIS59145.2023.10291566.

# The Project

# Model Setup and Architecture




[6]

We used the RESNET50 pre designed model within PyTorch

The model was entirely pretrained by us

- We found 15 epochs to be a good amount of pretraining for the synchronization tests

With this model, we used an edited version of the Stanford Cars pre labeled dataset

**Reference:** Koonce, B. (2021). ResNet 50. In: Convolutional Neural Networks with Swift for Tensorflow. Apress, Berkeley, CA. https://doi.org/10.1007/978-1-4842-6168-2_6

# RESNET50 Applications

As RESNET50 is a widely used and documented model architecture, it has been used and tested on many tasks

RESNET50 is sometimes retrained in its entirety, and used to perform tasks, or the FC output is changed for transfer learning

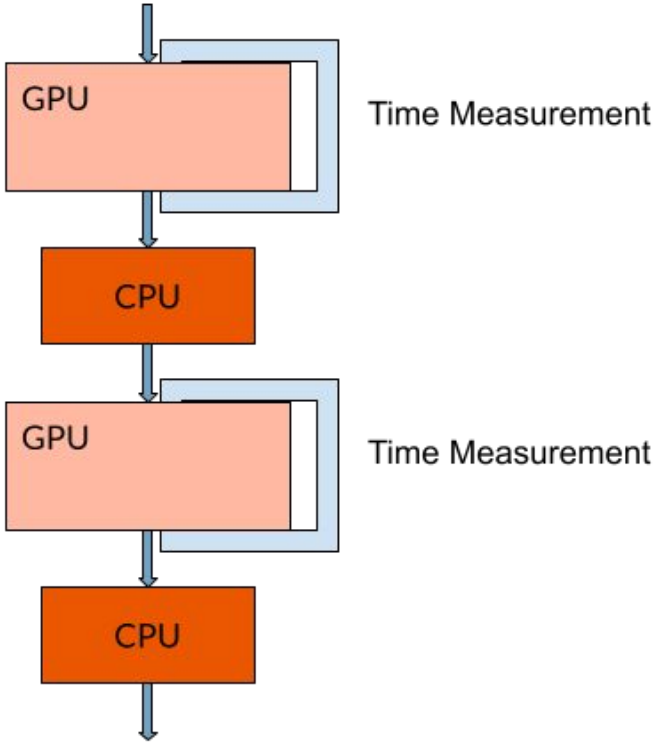
Our model was retrained in the interest of consistency, as we had full control over the save and stopping point for the pretraining

Other tasks can be similar enough to use a pretrained model in transfer learning
From there the output layers of the model are changed, and trained a bit more to fit to the new task

- One paper we explored used RESNET50 as a model to do food image recognition
- Another paper did the same, but instead using RESNET50 for malicious software classification
    - While this was a pretty different task, it was still useful with this model

**Reference:** Z. Zahisham, C. P. Lee and K. M. Lim, "Food Recognition with ResNet-50," 2020 IEEE 2nd International Conference on Artificial Intelligence in Engineering and Technology (IICAIET), Kota Kinabalu, Malaysia, 2020, pp. 1-5, doi: 10.1109/IICAIET49801.2020.9257825
E. Rezende, G. Ruppert, T. Carvalho, F. Ramos and P. de Geus, "Malicious Software Classification Using Transfer Learning of ResNet-50 Deep Neural Network," 2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA), Cancun, Mexico, 2017, pp. 1011-1014, doi: 10.1109/ICMLA.2017.00-19.

# Project Setup




We only had access to one relatively small GPU, so we ran each “synchronous” GPU one at a time

- Synchronization was performed after each superstep using this single GPU
- Timing was calculated in relation to how long it would’ve taken in parallel
    - The max of the timings was taken

Runs were set up in dictionaries with information passed through an automated pipeline

- Run types were predetermined and information was stored in the dictionaries
- Predetermined values were set for items like batch size, learning rate, and epochs

Values were then extracted for the final train and test loss and accuracy, along with time taken

# Synchronization

Synchronization was done at different superstep sizes for different runs

- All supersteps were based on epochs, with the number of epochs varying (30, 10, 5, 3)
- All runs were done with 30 epochs

Synchronized using:

- Max parameter
- Min parameter
- Median parameter
- Mean parameter

Related data was taken for each one

- Output statistics were taken at the end of each run

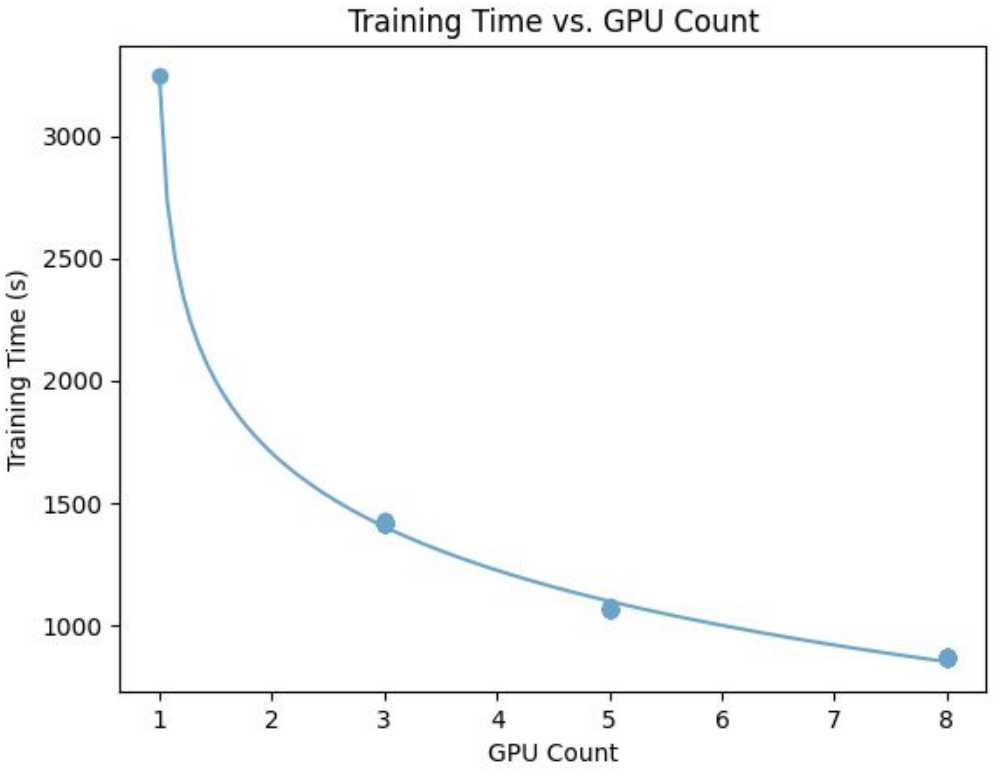
# The Data

# Training Speedup

Training our model on 8 GPUs  runs faster than 1 GPU by ~3.73 faster.

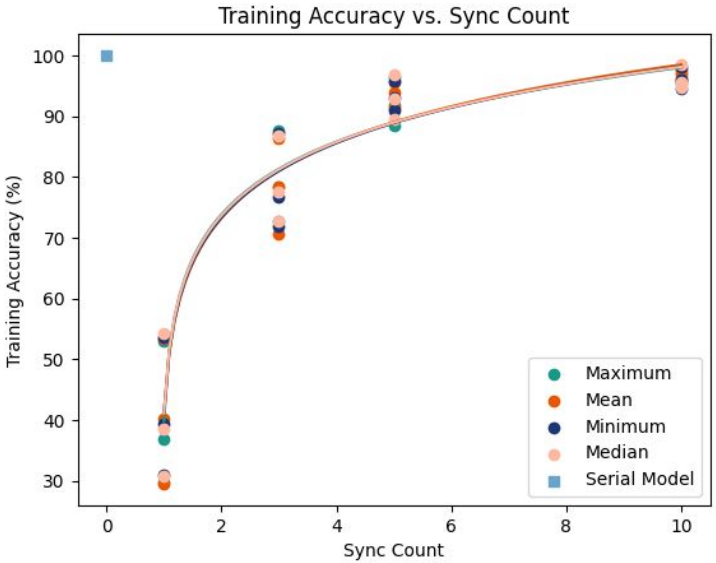
At a point, more GPUs only slightly increases training.

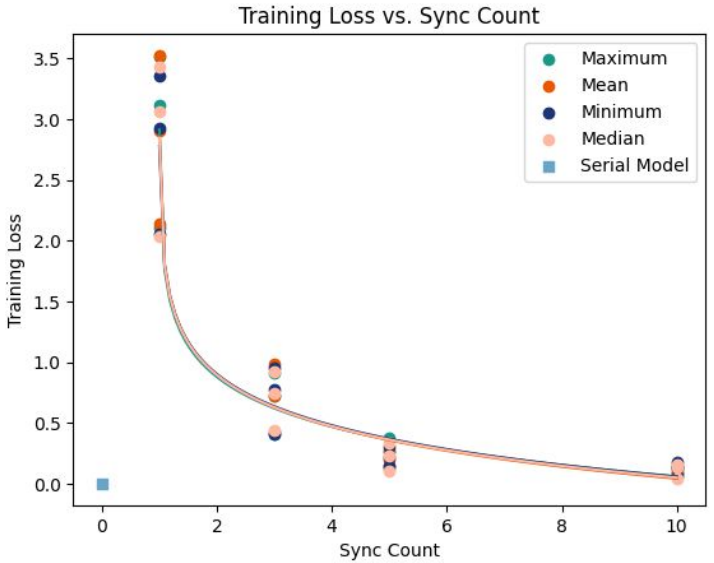
- 1 GPU to 3 GPUs exposes ~2.3 speedup
- 3 GPUs to 5 GPUs only exposes ~1.3 speedup




* Each point actually represents 16 unique data points

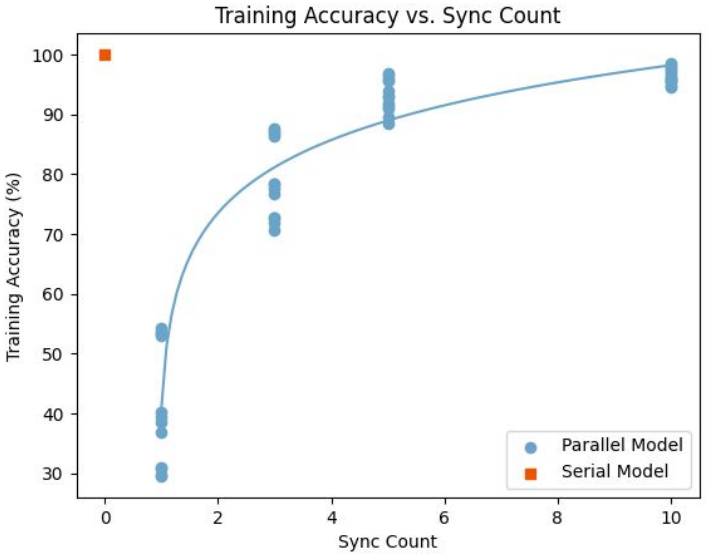
# How to Synchronize

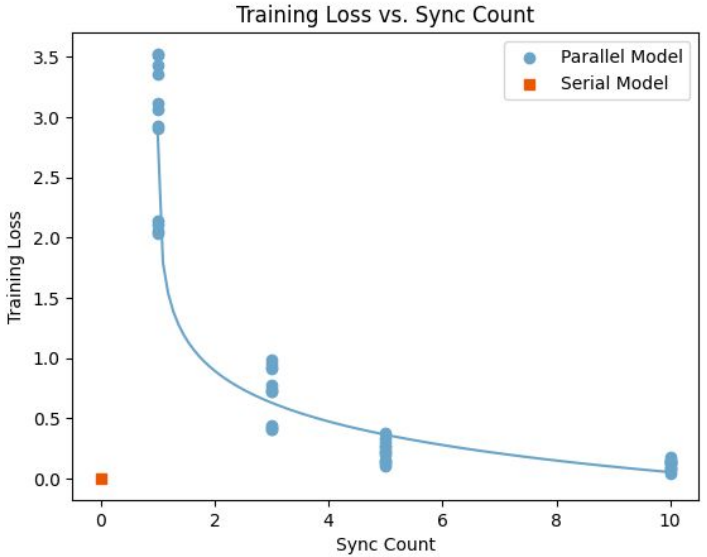



The fit of all our synchronization methods is the same. In  this case, the synchronization method does not affect the loss and accuracy of the model.

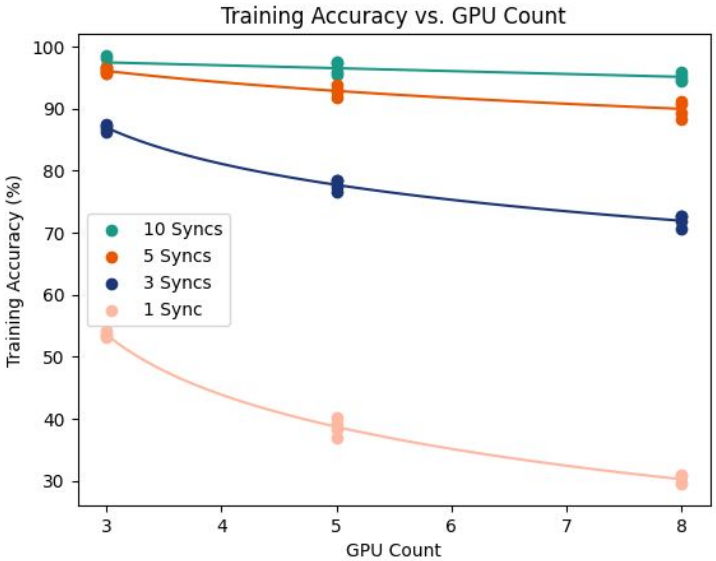
# When to Synchronize

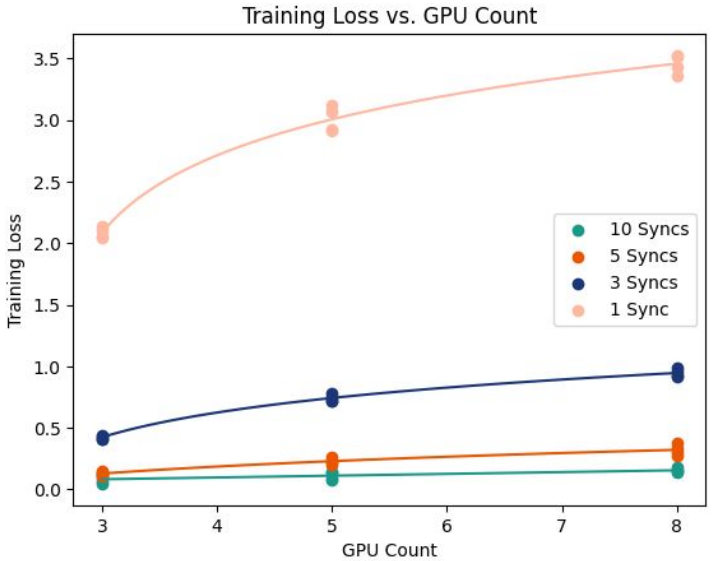



As the model is synchronized more, the parallel model performs better and approaches the loss and accuracy of the serial model.

# How many GPUs




- Training accuracy reduces as the model is trained more parallelly, but training time is significantly reduced!
- As there are more and more syncs the loss in accuracy is reduced.
    - As you add more syncs, the training more closely matches the serial training.

# Conclusion & Next Steps

# Conclusions

- Synchronization can be done for many reasons
  - Speed of the training
  - Size of the model
  - Size of the data
  - Available resources
- Synchronization can lead to significantly faster convergence times
  - Asynchronous options for parallelization exist too, although they may not converge, they can be faster
- There are many types that can be used, and choice will vary based on task and model
  - Can vary based on type of function used, or when synchronization happens
- Parallel model training can significantly reduce training time but can also reduce accuracy. *Tradeoff must be considered in design*.

# Next Steps

- Run more data
  - Larger datasets if possible
- Run different models
  - Deeper and wider models
  - Different tasks
- Run more synchronizations
  - More types of synchronization
    - Weighted average
    - Random
    - Remove outliers
    - Thresholding
    - If you can think it, you can try it!
- Run more epochs
  - Train the model longer to get a better of idea of when  to stop synchronization

# References


1. C. Zhang, H. Tian, W. Wang and F. Yan, "Stay Fresh: Speculative Synchronization for Fast Distributed Machine Learning," *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, Vienna, Austria, 2018, pp. 99-109, doi: 10.1109/ICDCS.2018.00020.
2. Y. Li, H. Wan, B. Jiang and X. Long, "More Effective Synchronization Scheme in ML Using Stale Parameters," *2016 IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, Sydney, NSW, Australia, 2016, pp. 757-764, doi: 10.1109/HPCC-SmartCity-DSS.2016.0110.
3. S. Tyagi and M. Swany, "Accelerating Distributed ML Training via Selective Synchronization (Poster Abstract)," 2023 IEEE International Conference on Cluster Computing Workshops (CLUSTER Workshops), Santa Fe, NM, USA, 2023, pp. 56-57, doi: 10.1109/CLUSTERWorkshops61457.2023.00023.
4. Yang, Rui & Zhang, Jilin & Wan, Jian & Zhou, Li & Shen, Jing & Zhang, Yunchen & Wei, Zhenguo & Zhang, Juncong & Wang, Jue. (2019). Parameter Communication Consistency Model for Large-Scale Security Monitoring Based on Mobile Computing. IEEE Access. PP. 1-1. 10.1109/ACCESS.2019.2956632.
5. T. T. Aurpa, S. M. Jeba and S. U. Rasel, "Ensemble Methods of Machine Learning Algorithms for Early Diabetic Detection in Comparison," 2023 1st International Conference on Circuits, Power and Intelligent Systems (CCPIS), Bhubaneswar, India, 2023, pp. 1-6, doi: 10.1109/CCPIS59145.2023.10291566.
6. Mukherjee, Suvaditya. “The Annotated Resnet-50.” *Medium*, Towards Data Science, 18 Aug. 2022, towardsdatascience.com/the-annotated-resnet-50-a6c536034758.
7. Z. Zahisham, C. P. Lee and K. M. Lim, "Food Recognition with ResNet-50," 2020 IEEE 2nd International Conference on Artificial Intelligence in Engineering and Technology (IICAIET), Kota Kinabalu, Malaysia, 2020, pp. 1-5, doi: 10.1109/IICAIET49801.2020.9257825
8. E. Rezende, G. Ruppert, T. Carvalho, F. Ramos and P. de Geus, "Malicious Software Classification Using Transfer Learning of ResNet-50 Deep Neural Network," 2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA), Cancun, Mexico, 2017, pp. 1011-1014, doi: 10.1109/ICMLA.2017.00-19.
9. Koonce, B. (2021). ResNet 50. In: Convolutional Neural Networks with Swift for Tensorflow. Apress, Berkeley, CA. https://doi.org/10.1007/978-1-4842-6168-2_6