EECE 5640 - High Performance Computing

Final Project Report

Topic:

Evaluation of ML workloads on CPU, GPU and Multi-GPU

Submitted by,

Balaji Sundareshan

Syed Shahbaaz Ahmed

Department of Electrical and Computer Engineering

Northeastern University

December 2021

The specs of the system used for this project is mentioned below:

Model name	Intel(R) Core(TM) i9-7920X CPU @ 2.90GHz
CPU(s)	0-23
NUMA node(s)	1
Architecture	x86_64
CPU max MHz	4400
CPU min MHz	1200
GPU(s)	4
GPU name	NVIDIA GeForce RTX 2080 Ti
CUDA version	10.0

Workloads:

The following are the workloads that will be used in the project:

1. Linear Regression
2. Logistic Regression
3. Decision Trees
4. Image classification using Convolutional Neural Networks (CNN)

Datasets:

Two types of datasets will be used. One with tabular data (for ML workload) and other, image (matrix-type) data.

- Datasets for ML:
    1. UCI Higgs Dataset (classification)
    2. UCI YearPredictionMSD (regression)

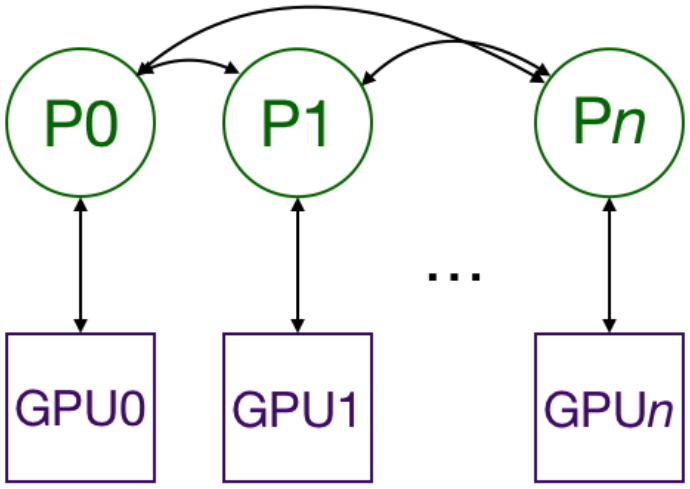
- Datasets for DL:
    1. CIFAR 100 Dataset (classification)

Platform:

- **Framework**:
  - Pytorch which uses C++ and CUDA in the backend, for parallel implementation in CPU and GPU respectively.
  - Xgboost, an implementation of gradient boosted decision trees designed for speed and performance, written in C++.
- **Programming Language**: Python

Multi-GPU Implementation:

- The multi-gpu training was implemented in pytorch using **nn.DistributedDataParallel** module. In this module, the machine has one process per GPU, and each model is controlled by each process. The GPUs can all be on the same node or across multiple nodes. Only gradients are passed between the processes/GPUs.



- During training, each process loads its own mini-batch from the disk and passes it to its GPU.
- Each GPU does its forward pass, then the gradients are all-reduced across the GPUs.
- Gradients for each layer do not depend on previous layers, so the gradient all-reduce is calculated concurrently with the backward pass to further alleviate the networking bottleneck.
- At the end of the backward pass, every node has the averaged gradients, ensuring that the model weights stay synchronized.

Workloads and Experiments:

Linear Regression:

- It's a method to predict a target variable by fitting the best linear relationship between the dependent and independent variable.

$$Least\ Squared\ Error = \frac{1}{2}(h_\theta(x^{(i)}) - y^{(i)})^2$$

$$Cost\ Function = J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2, \quad m = number\ of\ sample\ data$$

Parameters:

- number of training samples used: 463715
- number of test samples used: 51630
- batch size: 1024
- number of epochs: 100
- input dimensions: 90
- output dimensions: 1
- learning rate: 5e-3
- learning rate steps: [50]
- learning rate drop: 0.1

Model Summary:

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                [-1, 180]          16,380
              ReLU-2                [-1, 180]               0
            Linear-3                [-1, 180]          32,580
              ReLU-4                [-1, 180]               0
            Linear-5                 [-1, 90]          16,290
              ReLU-6                 [-1, 90]               0
            Linear-7                  [-1, 1]              91
================================================================
Total params: 65,341
Trainable params: 65,341
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.01
Params size (MB): 0.25
Estimated Total Size (MB): 0.26
----------------------------------------------------------------
```

	Average training time per epoch (sec)	Training Mean Absolute Error (MAE)
CPU	3.48	8.206
GPU	3.396	8.204
Multi-GPU(2)	3.04	7.86
Multi-GPU(4)	2.442	8.322

Observation:

- As seen from the above table, the average training time per epoch reduces as we increase the number of GPUs.
- The training time mentioned above contains forward pass, backward pass and gradient computation.
- The training time is less for a single GPU when compared to CPU. As the complexity of the network and the number of parameters are low, the difference between time taken in CPU and single GPU is less.
- The training mean absolute error is similar across all configurations.

	Total time elapsed for testing (secs)	Total prediction time elapsed for testing (secs)	Testing Mean Absolute Error (MAE)
CPU	12.54	5.371	8.1760
GPU	12.55	5.35	8.1750

- In the above table, the total time taken for testing includes data loading, host - device communication and forward pass for all the test images.
- The prediction time taken for testing includes only the forward pass for all the test images. As the complexity of the network is less, this time taken is similar for both single CPU and single GPU.
- The testing mean absolute error is similar across all configurations.

Logistic Regression:

It's a classification algorithm that is used where the response variable is categorical. The idea of Logistic Regression is to find a relationship between features and probability of particular outcome.

Cost function:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} Cost(h_\theta(x^{(i)}), y^{(i)})$$

$$J(\theta) = \frac{1}{m}[\sum_{i=1}^{m} -y^{(i)} log(h_\theta(x^{(i)})) + (1 - y^{(i)}) log(1 - h_\theta(x^{(i)}))]$$

$$m = number\ of\ samples$$

Model Summary:

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                [-1, 56]           1,624
              ReLU-2                [-1, 56]               0
            Linear-3                [-1, 56]           3,192
              ReLU-4                [-1, 56]               0
            Linear-5                [-1, 28]           1,596
              ReLU-6                [-1, 28]               0
            Linear-7                 [-1, 1]              29
           Sigmoid-8                 [-1, 1]               0
================================================================
Total params: 6,441
Trainable params: 6,441
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.00
Params size (MB): 0.02
Estimated Total Size (MB): 0.03
----------------------------------------------------------------
```

Parameters:

number of training samples used = 1050000
number of testing samples used = 50000
batch size = 512*2
Number of epochs = 100
Input dimensions = 28
Output dimensions = 1
learning rate = 5e-2
learning rate steps = [20, 40, 60, 80]
learning rate drop = 0.3

	Average training time per epoch (sec)	**Training Mean Absolute Error (MAE)**
CPU	7.632	0.1705
GPU	7.26	0.1696
Multi-GPU(2)	4.26	0.166
Multi-GPU(4)	2.418	0.1676

Observation:

- As seen from the above table, the average training time per epoch reduces significantly as we increase the number of GPUs.
- The training time mentioned above contains forward pass, backward pass and gradient computation.
- The training time is less for a single GPU when compared to CPU. As the complexity of the network and the number of parameters are low, the difference between time taken in CPU and single GPU is less.
- The training mean absolute error is similar across all configurations.

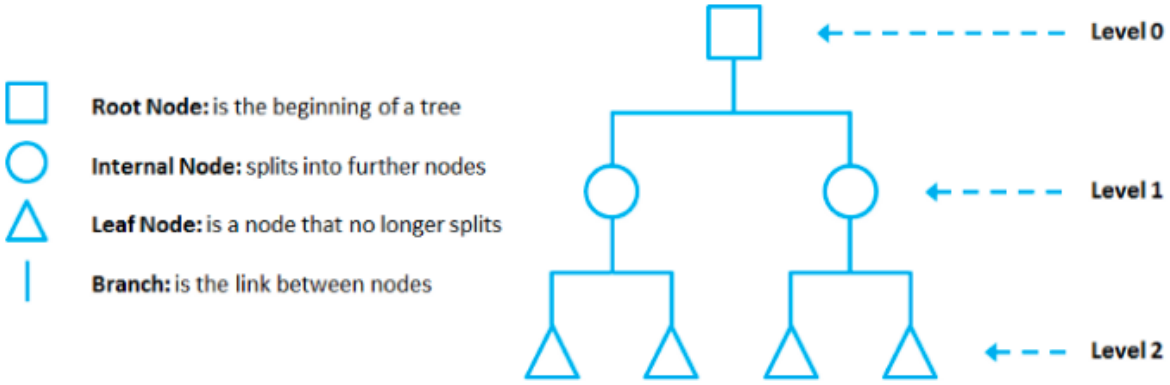
	Total time elapsed for testing (secs)	**Total prediction time elapsed for testing (secs)**	**Mean Accuracy**
CPU	61.98	4.023	74.3
GPU	62.0366	4.005	74.5

- In the above table, the total time taken for testing includes data loading, host - device communication and forward pass for all the test images.

- The prediction time taken for testing includes only the forward pass for all the test images. As the complexity of the network is less, this time taken is similar for both single CPU and single GPU.
- The testing mean absolute error is similar across all configurations.

Decision Tree:

- Decision trees are built by recursively splitting our training samples using the features from the data that work best for the specific task. This is done by evaluating the entropy for categorical decision trees.
- Decision Trees (DT) are composed of nodes, branches, and leaves. Each node represents an attribute (or feature), each branch represents a rule (or decision), and each leaf represents an outcome. The depth of a Tree is defined by the number of levels, not including the root node.



In this example, a DT of 2 levels.

Number of training samples = 1050000
Number of testing samples = 50000
Number of trees = 1000

	Training time (secs)	Testing time (secs)	Testing Accuracy
CPU	259.35	0.245	73.6
GPU	14.86	0.097	73.14
Multi-GPU(2)	12.46	— (single GPU testing)	73.47
Multi-GPU(4)	12.35	— (single GPU testing)	73.45

Observation:

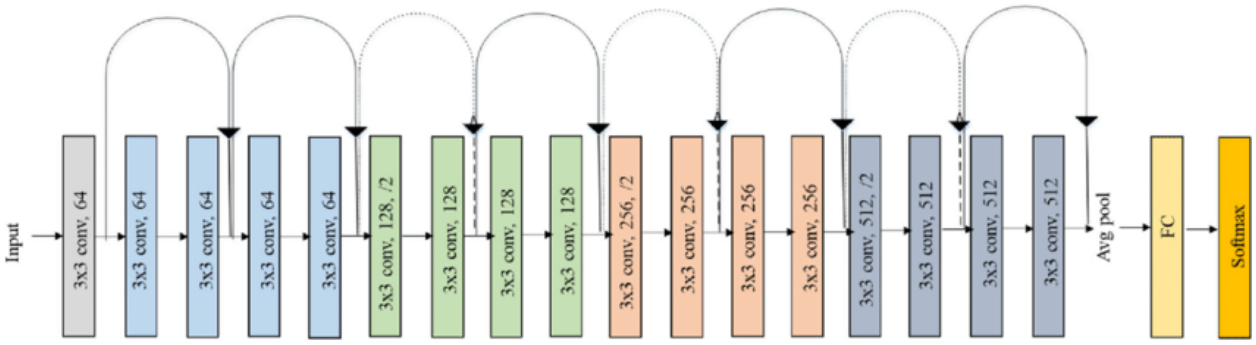
- There is a significant difference between the time taken in training in CPU and GPU. This huge difference is due to the optimization done by the xgboost library on GPUs.
- The difference between training time in single GPU and Multi-GPUs are pretty close. This might be due to the additional time taken in the host-device communication and synchronization.
- The testing time is calculated by summing all the time elapsed in predicting each sample. As prediction happens on one sample at a time, validation on multiple GPUs is not possible.
- The testing accuracy is pretty similar across all the configurations.

Image Classification (CNN):

- Image Classification is the technique to extract the features from the images to categorize them in the defined classes. For this problem, a Resnet network containing 18 layers was trained on the CIFAR100 dataset. This dataset contains images for 100 classes, each containing 500 images.

The architecture of the network is shown below

Resnet architecture:



Model params:

```
Total params: 19,480,612
Trainable params: 19,480,612
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 11.69
Params size (MB): 74.31
Estimated Total Size (MB): 86.02
```

Parameters:

- Batch size = 512
- Number of epochs = 100
- Learning rate = 1e-3
- Learning rate steps = [40, 80]
- Learning rate drop = 0.1

	Training Time per epoch (secs)
GPU - FP32	22.62
Multi-GPU(2) - FP32	12.06
Multi-GPU(4) - FP32	7.152
GPU - FP16	16.2
Multi-GPU(2) - FP16	12.9
Multi-GPU(4) - FP16	5.16

Observation:

- The training time per epoch reduces with the increase in the number of GPUs. As the complexity of the network is heavy, the reduction in time is quite significant with an increase in the number of GPUs.
- The training time reduces when using mixed precision, compared to single precision for each of the training configurations.
- Below table shows the training loss for both single precision and mixed precision. As expected the training loss for single precision is less compared to mixed precision.

	Single Precision (FP32)	Multi Precision (FP16)
Training Loss	0.18	0.23

	Single Precision (FP32)	Multi Precision (FP16)
Testing time (secs)	0.905	0.648

- The above table shows the time taken to predict 5000 images by model trained using single precision and multi precision formats.

- The testing time is calculated by summing all the time elapsed in predicting each sample. As prediction happens on one sample at a time, validation on multiple GPUs is not possible.

Code for this project is available here.

Reference:

1. https://medium.com/data-science-group-iitr/linear-regression-back-to-basics-e4819829d78b
2. https://towardsdatascience.com/how-to-scale-training-on-multiple-gpus-dae1041f49d2
3. https://medium.com/data-science-group-iitr/logistic-regression-simplified-9b4efe801389
4. https://medium.com/@ODSC/the-complete-guide-to-decision-trees-part-1-aa68b34f476d
5. https://developer.nvidia.com/blog/gradient-boosting-decision-trees-xgboost-cuda/
6.