Term Project Evaluation Form CSCE614

Name: Aaryan Kothapalli, Michell Brito
Project Title: CSCE 614 - Term Project Report
Date & Time this report submitted: 12/3/20

Evaluation	MAX. Score	Your Score
Overall Organization Title, abstract (problem attempted, outline of your results, improvements) table of contents, introduction (general description of the problem, motivation, related work, and goals), description of the problem and related definitions and background, description of your work, experimental results, conclusions, appendices (your code may be included here), tables, and figures.	10	
In depth description of the problem and its significance	10	
Techniques used and definitions (should be fairly self-sufficient)	20	
Description of your work and results	40	
Technical evaluation (soundness of approach and depth) of results	60	
Conclusions, summary of work, and directions for further work	15	
Appendices (if applicable)	5	
References	10	
Overall quality of report	30	
Total	200	

CSCE 614 - Term Project Report

Aaryan Kothapalli, Michell Brito

12/4/20

1 Abstract

Machine learning has grown extremely popular in the recent years and provided many opportunities for scientists, but also brought many challenges in the computer architecture field. When it comes to training a system through experience, it often requires very large datasets to be processed quickly, and high energy consumption to enable this processing. To account for these obstacles, specialized architectures like a Graphical Processing Unit can be used that are especially potent in large data processing and repeated instruction processing. Based on the paper *Hardware for Machine Learning: Challenges and Opportunities*, we explore different hardware architectures and their limitations in processing a neural network. We compare MNIST dataset neural network performance with and without optimized hyperparameters using different hardware architectures like a mainstream CPU, mainstream GPU, cloud GPU, cloud CPU, and cloud TPU. The MNIST dataset based Convolutional Neural Network performed the best with the fastest execution time on cloud with TPU hardware acceleration.

2 Introduction

Machine learning grew out of Artificial Intelligence as an attempt to have machines solve the large data problem by autonomously learning from them. This gave rise to big data, where "more data has been created in the past two years than the entire history of the human race". But to make big

data processable, a "significant amount of computation is required". These processing requirements primarily set the limitations to what hardware is viable for Machine Learning. To account for this problem, there are specially designed hardware like TPU, GPUs with large numbers of CUDA cores, and TensorCores to process big data. One example of this is the Nvidia Jetson TX2 which is specifically scaled for Machine Learning applications for developers. There are also cloud applications like Google Colab that provide hardware resources for ML applications. And finally, there are also consumer desktops and laptops with consumer-grade GPUs and CPUs that are also able to run ML applications. The limitations in hardware bandwidth and energy consumption are what primarily differentiate these machines from each other in the context of big data processing.

The paper, Hardware for Machine Learning: Challenges and Opportunities introduces how GPUs and CPUs can perform MAC operations in parallel to reduce the number of multiplications that have to be done in machine learning when generating features. The hardware can do loop unrolling and memory partitioning. Loop unrolling can do multiple calculations per iteration and take larger steps that takes advantage of parallelism. Memory partitioning also takes advantage of parallelism by allowing more instructions to execute concurrently which reduces clock cycles.

Accelerators are introduced as a way to deal with the flow of the dataâs movement which can be optimized by minimizing the access to expensive levels of the memory hierarchy; there are various ones such as output stationary

which the outputs are placed in the register file to minimize movement. The paper also suggests that by modifying the machine learning algorithms or hyper-parameters can minimize the computation, data movement and storage requirements such as removing unnecessary weights that are calculated in machine learning algorithms and can help reduce power consumption.

3 Problem

Machine learning is always dealing with large data and performing operations that result in a lot of energy consumption. Whenever a machine learning algorithm is running, it has to produce feature extraction which is used to transform the raw data into meaningful inputs for the given task [1]. The process of feature extraction is very important in regards to machine learning because the quality of the features it extracted can increase the performance of a learning algorithm both in terms of accuracy and computation time. This leads to the problem of having to find a way to minimize the amount of energy that is consumed when doing feature extractions.

Deep Neural Networks (DNN) in machine learning can learn what features are best directly from the data [2]. To learn the features directly from the data, DNN uses multiple layers representing a feature map. These layers are then merged all together to produce the final layer that has the best feature. This method requires a lot of computation due to all the layers.

Once we have our optimal feature, it is given to us as a vector in which

machine learning algorithms use. There are various vectors such as convolutional neural network (CNN) that can classify the data. To classify the data, CNN and other vectors have to compute the dot product of the features and the weights. This results in a lot of calculations having to be down which increases the energy consumption and the computation time.

4 Solution

To solve the issues regarding energy consumption, computation time, and neural network accuracy, various changes can be made to the hardware and software level such as parallelism, accelerators, and using HyperOpt.

4.1 Hardware

MAC operations are done during the process of feature extraction and DNN, a way to reduce the energy consumption, and computation time are to do the MAC operations in parallel. The GPU and CPU can take advantage of the temporal locality, which allows the matrix multiplication to be tiled to the storage hierarchy. Loop unrolling helps to optimize the execution time of a program, reducing or eliminating instructions that control the loop. To reduce and eliminate instructions, the compiler modifies the loop so multiple iterations of the loop are executed at once.

Accelerators can also be used to reduce energy consumption and computation time. Accelerators optimize the dataflow of the data to minimize

accesses from the memory hierarchy [3]. There are two different dataflows such as Weight stationary (WS), Output stationary (OS). The WS dataflow stores the weights in the register file at the PE, which minimizes the movement cost of the weights, and the OS stores the output in the register file at the PE, which minimizes the cost of the partial sums.

Machine learning models can be trained faster by merely running all operations simultaneously instead of one after the other. Graphics Processing Unit (GPU) can help run these operations parallel because it has more ALUs, control units, and memory cache than CPUs. GPU has a SIMD architecture, which allows it to be able to perform multiple instructions in parallel. Google Colab is a perfect example of using GPU to train machine learning models faster. Google Colab uses a GPU called Nvidia K80 / T4, which contains six Graphics Processing Clusters (GPCs), 36 Texture Processing Clusters (TPCs), and 72 Streaming Multiprocessors (SMs).

Compute Unified Device Architecture (CUDA) Cores can be thought of as cores just like CPU cores but just not as powerful. However, the significant benefit of CUDA cores is that you can have hundreds of CUDA cores compared to 2-16 CPU cores. NVIDIA implements CUDA cores into their GPUs, taking advantage of those hundreds of CUDA cores to perform mathematical operations. Two NVIDIA GPUs that use CUDA cores are the Tesla K80 and 1080 Ti. Google colab uses the Tesla K80 GPU which has 4992 CUDA cores compared to mainstream's 1080 Ti 3584 CUDA cores.

Tensor Processing Unit (TPU) is an accelerator circuit chip created by

Google to improve time-to-accuracy when training large, complex neural network models. The TPU can achieve a high volume of low precision computation, but in return, doesn't have hardware for rasterisation/texture mapping. Google implemented 8 TPUs in their Google Colab which allows it to take large modeles that perform complex matrix computations and finish in significant less time than using other methods.

4.2 Software

For CNN classification, we know that it needs to produce vectors that require a lot of energy and computational power. A way to reduce the energy and computational power that it requires is to make the weights sparse, which leads to a reduction in the multiplication performed [4]. For feature extraction, the data can be sparse by prepossessing, while for DNNs, the amount of MAC operations and weights can be reduced through pruning. Pruning reduces the size of decision trees by removing parts of the tree that do not provide power to classify instances.

Machine learning algorithms have some parameters that can be adjusted, which are called hyperparameters. Hyperparameters are essential in building accurate models; they're able to find a balance between bias and variance, which prevents the model from overfitting or underfitting. Software such as hyperopt can be used to tune the hyperparameters automatically to optimize the model. The random search parameter tuning algorithm from hyperopt implements a randomized search over parameters. Each setting is sampled

from a distribution over possible parameter values; this leads to the best hyperparameters.

4.3 Implementation

- Keras, tensorflow, and sklearn libraries are used for majority of the project. The MNIST dataset (http://yann.lecun.com/exdb/mnist/) is used to train a from-scratch Convolutional Neural Network. The dataset contains 60,000 training images and 10,000 testing images. The images are preprocessed and normalized for better compatibility with the neural network.

- CNN model is built using a sequential model with two Conv2D layers, two hidden dense layers, and one output dense layer. Random hyperparameters are chosen for the model and trained for 15 epochs.

- Hyperopt function is made to take in the unoptimized CNN model and provide an updated CNN model with the best hyperparameters automatically (using random search) to provide the best training and testing accuracies.

- Various hardware are individually tested to measure their performance while processing the neural network. The hardware used are:

  - Mainstream Windows Computer (with GPU acceleration): CPU: Intel 7700k @5.0 Ghz, GPU: NVIDIA GTX 1080Ti @1.7Ghz with

11264 VRAM, RAM: DDR4 16GB @3200Mhz

  - Mainstream Apple computer (using CPU only): CPU: Intel i5 @2.3Ghz with integrated graphics, RAM: LPDDR3 8GB @2133Mhz
  - Google Colab with no acceleration (using CPU only): CPU: Intel Xeon @2.20Ghz
  - Google Colab with GPU acceleration: GPU: NVIDIA Tesla K80
  - Google Colab with TPU acceleration: custom TPU chip

- Same MNIST dataset with the same neural network model and hyperparameters are preserved and tested on each hardware architecture to maintain consistency.

- The system performance metrics are derived using the wandb library. It is a comprehensive library that provides useful realtime metrics from CPU utilization, GPU utilization, to power usage and much more.

- After MNIST data preprocessing, for each run on a hardware architecture, neural network model without optimized hyperparameters is built and trained for 15 epochs. Right after, neural network model with optimized hyperparameters is built and trained for 15 epochs. This is noticeable consistently in all graphs with a dip in the middle to notify the transition from one model to the next.

- The code for these models can be found in our project github: **Term_Project_Github**

5 Analysis

The MNIST dataset based Convolutional Neural Network with and without optimized hyperparameters is tested on five different hardware architectures. Specifically, it is tested on a mainstream Windows computer (using GPU for acceleration), mainstream Apple computer (CPU only), Google Colab with no acceleration (CPU only), Google Colab with GPU acceleration, and finally, Google Colab with TPU acceleration. All hardware architectures eventually produced 100% training accuracy with 0% loss for the optimized neural network.

Observing the epoch graph in Figure 1, there is a dip at the 15th step which signifies the change in the system moving from training the neural network without the optimized hyperparameters to training the neural network with the optimized hyperparameters. This is noticable in every other line graph as well. This is done to observe the effects on system performance when using hyperparameters that provide the best accuracies.

For the accuracy and loss of the neural network, as observed in Figure 1, all hardware architectures performed similarly except the Cloud with TPU acceleration. With TPU, the starting accuracy and loss are significantly better than the other hardware architectures with over 96% accuracy and 12.6% loss, though it later converged with the other architectures for them all to be close to 100% accuracy and 0% loss. This suggests that TPUs can perform significantly better for training multiple low-epoch neural networks.

To get a better understanding of the best performing hardware architecture, relative performance can be calculated from the execution times of the hardware architectures (Figure 1).

$$Relative\ Performance = \frac{Execution\ time\ A}{Execution\ time\ B} \tag{1}$$

Relative performance of Google Colab with TPU acceleration over Google Colab without acceleration (CPU only):

$$\frac{cloud\ no\ acceleration}{cloud\ (TPU)} = \frac{9499}{188} = 50.5 \tag{2}$$

Cloud with TPU acceleration is 50.5 times faster than cloud with no acceleration (CPU).

Relative performance of Google Colab with TPU acceleration over Google Colab with GPU acceleration:

$$\frac{cloud\ GPU}{cloud\ (TPU)} = \frac{390}{188} = 2.1 \tag{3}$$

Cloud with TPU acceleration is 2.1 times faster than cloud with GPU acceleration.

Relative performance of Google Colab with TPU acceleration over mainstream Windows computer with GPU acceleration:

$$\frac{local\ GPU}{cloud\ (TPU)} = \frac{307}{188} = 1.6 \tag{4}$$

Cloud with TPU acceleration is 1.6 times faster than mainstream Windows computer with local GPU acceleration.

Relative performance of Google Colab with TPU acceleration over mainstream Apple computer without acceleration (CPU only):

$$\frac{local\,CPU}{cloud\,(TPU)} = \frac{3103}{188} = 16.5 \tag{5}$$

Cloud with TPU acceleration is 16.5 times faster than mainstream Apple computer without Acceleration (using CPU only).

Based on the information, TPU has a significant lead in relative performance, ranging from being 50.5 times better than using a cloud CPU to being 1.6 times better than a mainstream high-end local GPU (NVIDIA GTX 1080Ti).

Comparing GPU performance, the mainstream Windows Computer with GPU acceleration by an NVIDIA GTX 1080 Ti performed better than Google Colab's GPU acceleration by Tesla K80. This is an interesting find because though Tesla K80 with 4992 CUDA cores has 1408 more than than the GTX 1080Ti with 3584 CUDA cores, it suffers in every other spec such as base clock with 562MHz vs. 1481MHz. Additionally, it is running on cloud with users sharing resources and also, there are predefined limits enforced by Google. Looking at GPU utilization in Figure 2, the local GPU was faster in accessing and processing the neural network than the cloud GPU which suggests that this delay caused the local GPU to perform faster. This delay could have been

either contributed by high latency or server side start-up delay. Something else to consider is the power usage % and power usage (W). Though cloud GPU is using 100% of power usage, it uses 3x-4x times less power than the local GPU. This is another example of the predefined limitation set by Google to prevent resource hoarding.

Comparing CPU performance between the Apple computer (Intel i5) and Google Colab with no acceleration (Intel Xeon CPU), the Apple computer performs significantly better as observed in Figure 3, albeit using either CPUs are unrealistic for ML. While the i5 stayed at an average of 80% CPU utilization compared to 100% utilization from the Intel Xeon CPU, the i5 performed 3x times better in relative performance. This is because while the i5 is quad-core, Google only limits each user to one core from the Intel Xeon CPU. The results are as expected as the i5 performs 3x better with 3 more cores.

Finally, comparing the neural network without optimized hyperparameters to neural network with optimized hyperparameters led to interesting performance results. Even though the training accuracy between the two neural networks was only 5%, there was significant change in GPU and CPU utilization, and local GPU power usage %. After using optimizations, the GPU utilization went up by approx. 2%, CPU utilization went up by approx. 10%, and local GPU usage % went up by 20%. This suggests that by optimizing hyperparameters, the neural network is able to better use more available system resources to produce higher accuracies.

6 Conclusion

As machine learning continues growing in popularity, there are new improvements to the hardware level that will be introduced. We saw that the best ones could increase the ability to perform mathematical operations, either increasing resources that can do more operations or doing these operations in parallel when it comes to changes in the hardware level. At the software level, it came down to optimizing the algorithms being used, so fewer mathematical operations are required when generating the vectors for the model.

Although TPU performed better than GPU and CPU, using a GPU with hundreds of CUDA cores and multiple ALUs is recommended when doing machine learning tasks. It is more accessible and improves time and efficiency, which are crucial for the user. However, the downside is the justification of the cost that is going to take for those time and efficiency improvements. If the data is not extremely large, one can justify not having to spend a lot to improve time and efficiency because it won't be significant.

References

[1] Sze, Vivienne, et al. "Hardware for Machine Learning: Challenges and Opportunities." IEEE.org, ieeexplore.ieee.org/document/7993626.

[2] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," Nature, vol. 521, no. 7553, pp. 436â444, May 2015

[3] M. Sankaradas, V. Jakkula, S. Cadambi, S. Chakradhar, I. Durdanovic, E. Cosatto, and H. P. Graf, "A Massively Parallel Coprocessor for Convolutional Neural Networks," in ASAP, 2009.

[4] A. Suleiman, Z. Zhang, and V. Sze, "A 58.6 mW real-time programmable object detector with multi-scale multi-object support using deformable parts model on 1920Ã 1080 video at 30fps," in Sym. on VLSI, 2016.

Appendex

Figure 1

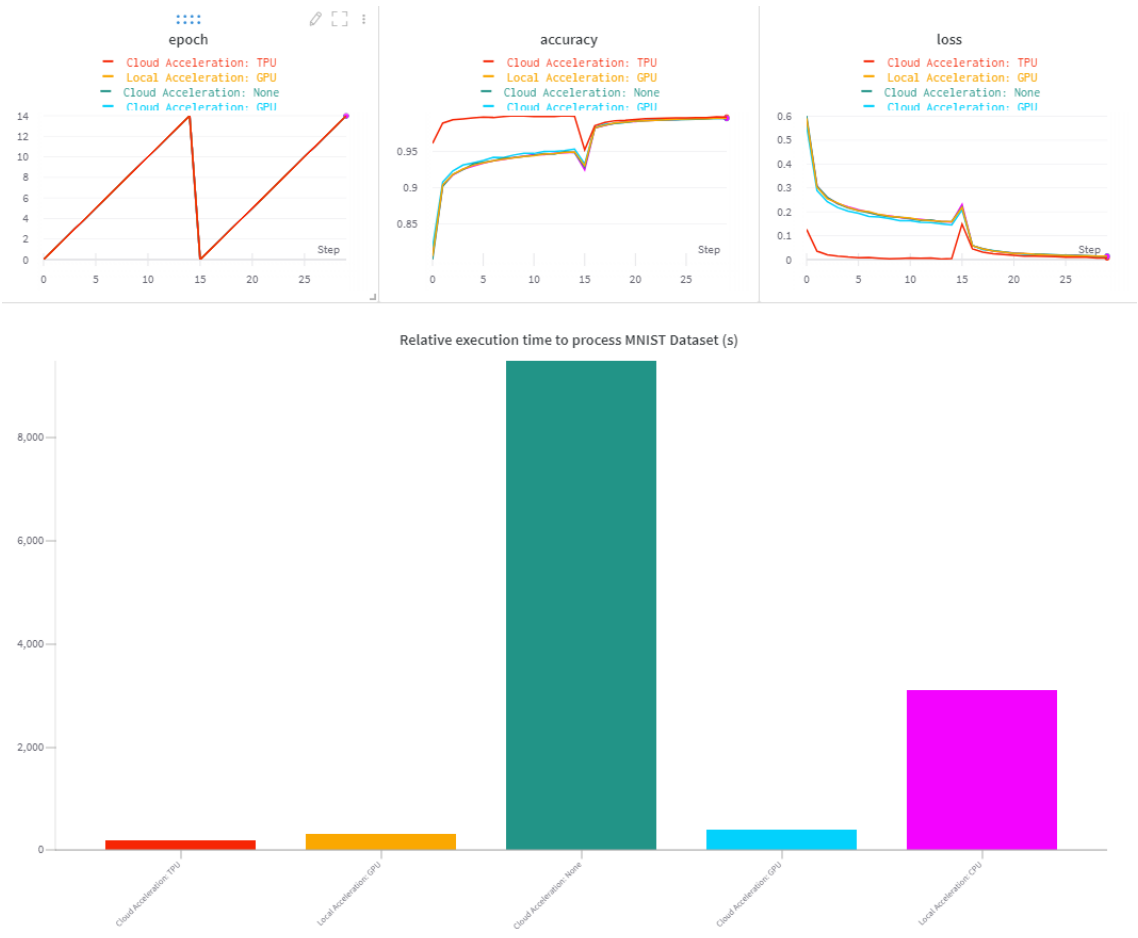


Figure 2

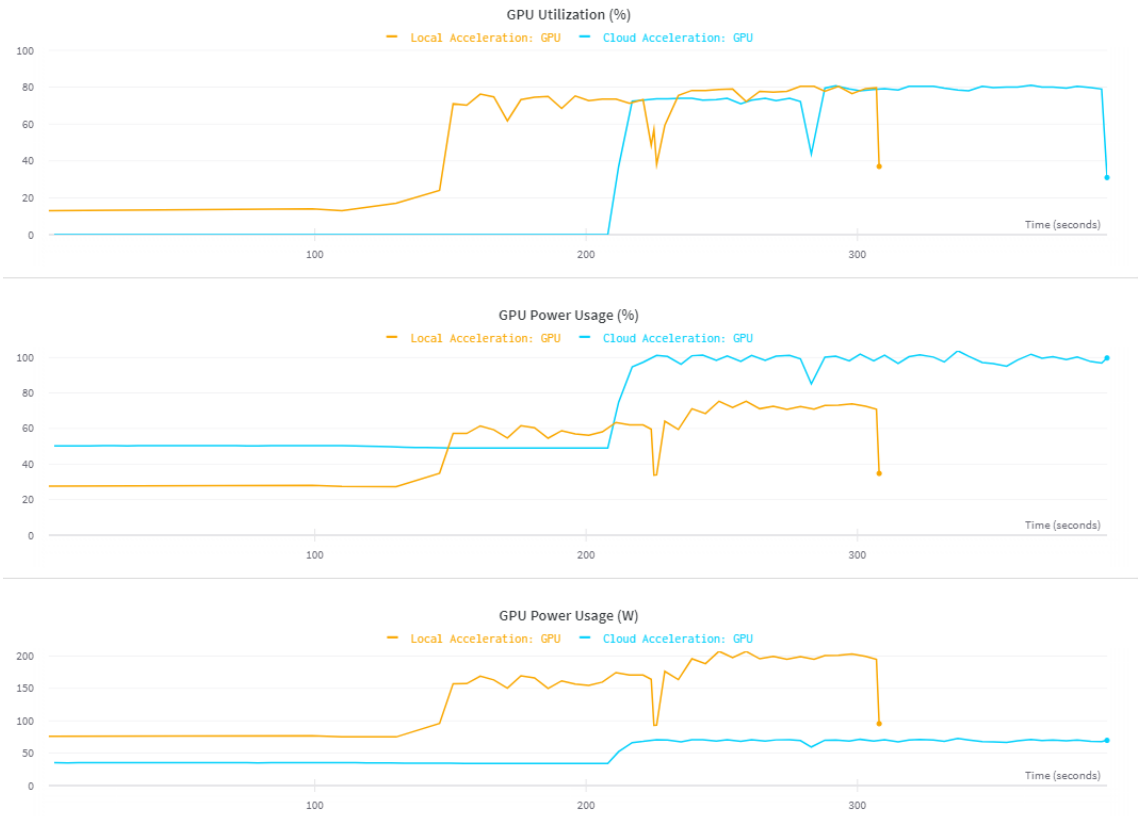
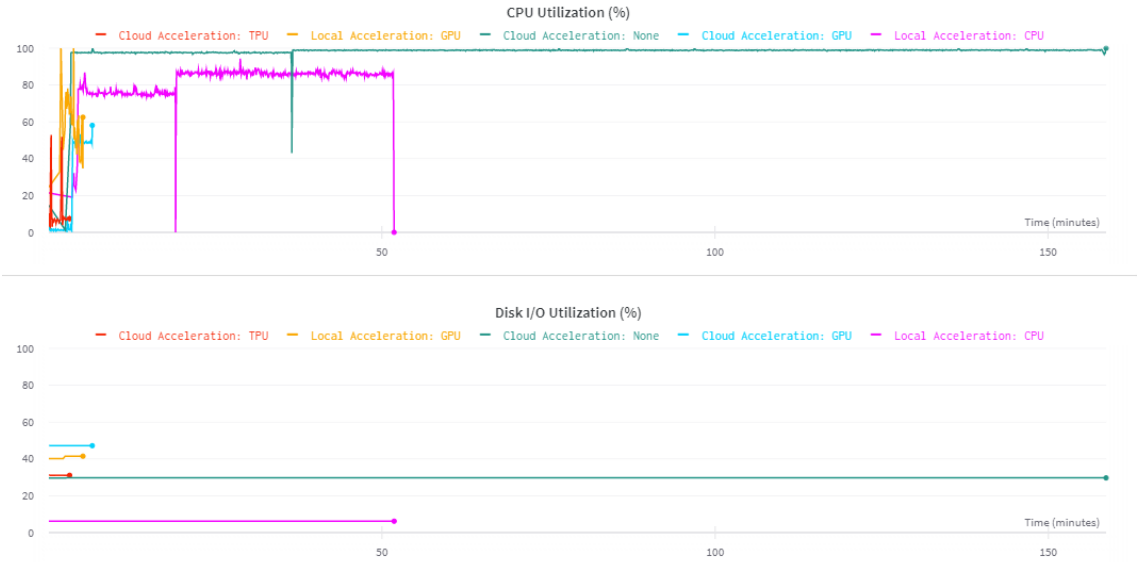

GPU Utilization (%)
Local Acceleration: GPU
Cloud Acceleration: GPU
100
80
60
40
20
0
100
200
300
Time (seconds)
GPU Power Usage (%)
Local Acceleration: GPU
Cloud Acceleration: GPU
100
80
60
40
20
0
100
200
300
Time (seconds)
GPU Power Usage (W)
Local Acceleration: GPU
Cloud Acceleration: GPU
200
150
100
50
0
100
200
300
Time (seconds)

Figure 3


CPU Utilization (%)
Cloud Acceleration: TPU
Local Acceleration: GPU
Cloud Acceleration: None
Cloud Acceleration: GPU
Local Acceleration: CPU
100
80
60
40
20
0
Time (minutes)
50
100
150
Disk I/O Utilization (%)
Cloud Acceleration: TPU
Local Acceleration: GPU
Cloud Acceleration: None
Cloud Acceleration: GPU
Local Acceleration: CPU
100
80
60
40
20
0
Time (minutes)
50
100
150