Image Classification using Parallel CPU and GPU Computing

Prathamesh Borhade, Rajvardhan Deshmukh, Samridhi Murarka, Rishav Agarwal

Abstract*: Image classification algorithms such as Convolutional Neural Network used for classifying huge image datasets takes a lot of time to perform convolution operations, thus increasing the computational demand of image processing. Compared to CPU, Graphics Processing Unit (GPU) is a good way to accelerate the processing of the images. Parallelizing multiple CPU cores is also another way to process the images faster. Increasing the system memory (RAM) can also decrease the computational time of image processing. Comparing the architecture of CPU and GPU, the former consists of a few cores optimized for sequential processing whereas the later has thousands of relatively simple cores clocked at approx. 1Ghz. The aim of this project is to compare the performance of parallelized CPUs and a GPU. Python's Ray library is being used to parallelize multicore CPUs. The benchmark image classification algorithm used in this project is Convolutional Neural Network. The dataset used in this project is Plant Disease Image Dataset. Our results show that the GPU implementation achieves 80% speedup compared to the CPU implementation.

***Keywords*: Convolutional Neural Network, parallel computing, speedup.**

I. INTRODUCTION

In recent years, parallel computing and soft computing has become a rapidly evolving field of study. The demand for parallel processing in increasing day by day. There are various software tools and libraries by which we can parallelize our programs. For example, we have OPENMP in c++ for parallel computing. OPENMP supports FORTRAN, C and C++. It is basically an Application Programming Interface for shared Memory Model programming. Python has its separate parallel processing module named Multiprocessing. Multiprocessing module enables to spawn multiple processes, allowing programmer to fully leverage the computing power of multiple processors. The main drawback of Python's Multiprocessing module is that it cannot be used for handling large numeric data. It cannot be used in Deep Learning Frameworks such as Keras as it decreases the accuracy of the models. Shared variables cannot be used in the Multiprocessing Module. Python also has a Parallel and Distributed computing framework called Ray. Ray can be used for developing emerging AI applications such as image classification, face recognition etc. Parallelizing multiple cores of CPU using Ray can also increase the speedup of the model significantly. The benchmark image classification algorithm used in this project is Convolutional Neural Network. The Dataset used in this project is Plant Disease Image dataset containing around 30000 images. The system is configured with 16 GB RAM with 4 CPU Cores and Tesla P100 GPU. This project compares the performance of 2-core, 3-core and 4-core parallelized CPUs with GPU.

II. COMPARISON OF CPU AND GPU ARCHITECTURE

A Graphics Processing Unit (GPU) is mostly used in hardware devices to support applications which require heavy graphics processing such as 3-D modeling, designing, etc. GPUs are mostly used in gaming PCs to accelerate gaming graphics processing. Nowadays, GPGPU (General Purpose Graphics Processing Unit) are used to accelerate computational graphics processing workloads. The main purpose of GPU is to project textured polygons onto the screen in a fiercely competitive consumer-facing industry. The design goals of GPU were to increase the throughput and not focus on single threads. GPUs basically hide memory latency through parallelism. They let the programmer deal with raw storage hierarchy. The most important design goal for GPU is to avoid high frequency clock speed. The GPU consists of multiple independent CPU cores. The original design goals of CPU were to make single threads faster, reduce latency through large caches and use prediction and speculation for the instruction stream to guard against the branches in the instruction stream. Modern CPU-Style core design emphasizes individual thread performance. Each core of the CPU executes scalar or vector operations whereas each GPU core only executes vector instructions. CPU uses Single Instruction Multiple Data (SIMD) parallelism through ILP and vector execution units whereas GPU uses Single Instruction Multiple Data (SIMD) parallel execution of all operations. The GPU cores are designed and engineered to switch quickly between threads to recover stalls. A GPU has multiple cores and each core has one or more wide SIMD vector units. These wide SIMD vector units execute one instruction stream and also have a pool of shared memory. Each core of the GPU shares a registry file shared privately among all the ALUs.

Revised Manuscript Received on April 25, 2020.
* Correspondence Author

Prathamesh Borhade*, Computer science and engineering, Vellore Institute of Technology, Vellore, India. Email: prathamesh.borhade29@gmail.com

Rajvardhan Deshmukh, Computer science and engineering, Vellore Institute of Technology, Vellore, India. Email: rajvardhan1999@gmail.com

Rishav Agarwal, Computer science and engineering, Vellore Institute of Technology, Vellore, India. Email: rishavagarwal2717@gmail.com

Samridhi Murarka, Computer science and engineering, Vellore Institute of Technology, Vellore, India. Email: samridhi.m98@gmail.com

Each core quickly switches thread blocks to hide memory latency. Modern GPUs combine multiple wide vector processing cores with local and global-shared memory

III. PERFORMANCE COMPARISON OF CPU AND GPU

We all need to understand that the latency optimization takes place in CPUs while GPUs are bandwidth optimized. For Deep Learning, GPUs hide latency via thread parallelism thus offering high bandwidth. This is the reason why GPUs are widely used in training deep learning models. Compared to CPUs, GPUs cannot have more memory capacity. 24GB of RAM is known to be the highest amount of memory that a GPU can hold whereas CPUs can have RAM up to 1TB. The main advantage of using GPUs in Deep Learning is that they can perform matrix operations faster than CPUs. Deep Learning largely comprises of large matrix operations. Researchers say that the GPUs can gain speedup up to 60% more than that of CPUs.

IV. PARALLELIZING MULTIPLE CPU CORES

To parallelize multiple cores of CPU, the OS has to be a multiprocessing OS. Multiprocessing OS uses two or more CPU cores to run a single program. Each core works on different parts of the same task, or on two or more different tasks. They are used for high speed computations and to increase the power of the computer as the execution takes place in parallel. Python’s Multiprocessing module can be used to parallelize all the CPU cores present in the machine. Multiprocessing module uses Pool property to parallelize CPUs. The pool distributes the workload of the program to the available processors using First In First Out (FIFO) scheduling. But this module cannot be used with Deep Learning frameworks such as Keras as these frameworks are not multiprocessing safe.

Python has a parallel and distributed framework called Ray which exclusively used for building AI applications by parallelizing the CPU cores. Ray automatically detects the number of CPU and GPU cores present in the system

V. RELATED WORKS

[1] The paper talks about the several types of Leaf Diseases and the image processing techniques to categorize them. The paper contains a study of all the different papers that talk about several techniques for leaf detection techniques. **[2]** The paper gives an in depth knowledge of Convolutional Neural Networks. It tells us about the growth of precision and results in image classification and detection based on numerous CNN techniques. This paper tells us about the simple but effective architecture and several layers of the Convolutional Neural Networks like convolution layer, ReLu, pooling, etc and the ways to set them to create a model that gives excellent results for detecting complex patterns. **[3]** This paper gives us a complete overview of the architecture of the CPU as well as the GPU. The paper has shown experiments for detecting the edges of an image i.e. creating a boundary of an image. The authors of this paper have used CUDA for the parallelizing of the algorithm used for the above-mentioned purpose and used MATLAB as their programming language. The paper states that the CUDA supported GPU gives much faster and better results as compared to the CPU. **[4]** The paper compares the GPU and CPU computations of multiple algorithms by parallelizing them into independent threads. The paper points out some of the exceptional cases where the CPU exceeds the expectations and give better and faster results than the GPU. Thus the paper gives us a set of guidelines to help us determine when and in which way should the GPU be used for parallelization. **[5]** This paper talks about requirements and necessities that led to the development of this distributed framework, Ray. The paper states the applications of Ray in numerous places like parallelization, complex user and problem specific computations, etc. with limited resources and a single execution engine. The paper describes and highlights the power of Ray by experimenting with 1.8 million tasks per second. **[6]** This paper also gives an overview of GPU and Image processing in general. The paper explains the steps involved in building a model for a particular image processing application. It explains steps like RGB to Gray conversion, Morphology Applications, etc. The paper also compares normal sequential and CPU image processing with the image processing using the model that was just built on GPU. **[7]** This paper not only talks about image processing of 2D images but also the 3D images. The paper implements several techniques like Single Image Transmission and Multi Pixel Processing, Multi Image Transmission and Single Pixel Processing, etc. on GPU and CPU. In conclusion the paper gives the results and comparison of each technique on single CPU, GPU, multicore CPU and multicore GPGPU.

VI. FRAMEWORK MODEL

A. Approach

The following approach is being followed to compare the performance of parallelized CPUs and GPUs.

- Downloading the required dataset for our model. In this case, it is Plant disease image dataset. The dataset is taken from Kaggle’s dataset library.
- Designing the architecture for Convolutional Neural Network.
- Preprocessing all the images of the dataset.
- Dividing the given dataset into training set and testing set.
- Parallelizing CPU cores with Ray and training the model. Varying the number of CPU cores
- Using GPU to train the model
- Observing the execution time for all the epochs for GPU and CPU computing
- Observing the accuracy for CPU and GPU computing
- Analysis of results by plotting the appropriate graphs.

B. Architecture of Convolutional Neural Network

Deep learning is a sub domain of machine learning which is based on neural networks. The architecture of neural network is based on the anatomy of the human neuron.

That's the reason these algorithms are called neural networks. Convolutional neural network is a type of neural network which is used mostly in visual imagery.

As the name suggests, this neural network uses convolution operation to process images. Convolution operation of two functions *f* and *g* is defined as the integral product of these two functions after one is reversed and shifted. The CNN consists of multiple hidden layers which are bounded by an input layer and an output layer. These hidden layers are basically a series of convolution operations. The common operations present in the hidden layer are Convolution, ReLu, Max pooling, Flattening and Full connection

Step 1: Convolution layer

In this layer, the feature/filter is moved to each potential position on the image. Firstly, the features and image are lined up. Then each image pixel is multiplied by the corresponding feature pixel. Then these values are added and the sum is divided by the total number of pixels in the feature/filter.

Step 2: ReLU Layer

In this layer all the negative values from the filtered images are replaced with zeros. We do this to avoid summing up of the values up to zero. ReLU (Rectified Linear Unit) function activates the node when the input is greater than a certain threshold value, otherwise it returns 0. The input becomes linear to dependent variable when it increases after certain threshold.

Step 3: Pooling Layer

Shrinking of image stack takes place in this layer. First we choose a window size (usually 2 or 3). Then we pick up a stride (usually 2) to traverse the whole image. Then the window is walked across the filtered images. From each window, the maximum value is found and stored.

Step 4: Flattening Layer

In this step the 2-D matrix obtained from the pooling layer is converted into a 1-D matrix (vector) so that it can be fed into the fully connected layer

Step 5: Stacking the layers

In order to increase the accuracy of the model, we can repeat the stack of above layers before feeding the vectors into the fully connected layer.

Step 6: Fully connected layer

The actual classification happens in this layer. This is the final layer of CNN. In this layer the filtered and shrinked images are put into a single list. This layer connects every node of one layer to every node of the another layer. The vector obtained from flattening layer is passed through layer to predict the class

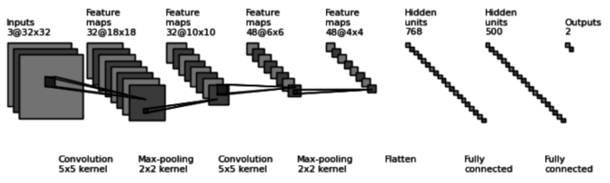

Figure 1. Layers of CNN

C. Parallelizing CPUs using RAY

Python has a parallel and distributed module called Ray which is used to build scalable AI applications. Ray helps in easily parallelizing python applications. Ray is an open source project built by Machine Learning department of University of California, Berkeley. OpenMP, Python Multiprocessing and ZeroMQ are low-level primitive parallel programming libraries whereas Ray is a high-level parallel and distributed framework. Ray automatically detects the number of CPU and GPU cores present in the hardware system. The API *ray.init()* initializes the ray context and the number of workers of the cluster. We can also specify the number of resources such as number of CPUs and GPUs. By default Ray initializes number of CPUs as the number of CPU cores present in the system. We can overwrite the number of CPUs and GPUs using the following *ray.init(num_cpus=2, num_gpus=1).* The decorator *@ray.remote(num_cpus=3)* is called before a function. This decorator assigns the resources to the function for the computation. The API *ray.get()* returns the output of the function and the Ids of the objects of the function.

D. About the dataset

The dataset used in this project is Plant Disease Detection image dataset. The same is available on Kaggle Data Science Platform. This dataset contains the images of leaf of the plant having a particular disease. The dataset is divided into following 15 categories: Pepper bell Bacterial spot, Pepper bell healthy, Potato Early blight, Potato healthy, Tomato Spider mites Two spotted spider, Potato Late blight, Tomato Bacterial spot, Tomato Early blight, Tomato healthy, Tomato Late blight, Tomato Leaf mold, Tomato Septoria leaf spot, Tomato target spot, Tomato Tomato mosaic, virus and Tomato Tomato Yellow Leaf Curl Virus.

VII. RESULTS

To compare the performance of GPU with CPU, we have configured the following setup.
For CPU: Intel Core i5 7th generation with 16GB RAM and 2GB Nvidia 940MX 2GB graphics card.
For GPU: Intel Core i5 7th generation with 16GB RAM and Tesla P100 graphics card

The dataset was split into training set and testing set with 80% of the images in the training set.

To record the first observation, we observed the execution time and the accuracy of the model after training the same for serial python. We observed the execution time for each epoch. The accuracy was observed to be 88.75%.

Table 1. Execution time for serial python

Epoch sequence number	Execution time (in seconds)
1	538
2	540
3	537
4	535
5	538

After parallelizing 3 CPU cores using *ray.init(num_cpus=3)*, we got an accuracy of 92.32%. Following were the execution time of all the epochs

Table 2. Execution time for 3 CPU cores

Epoch sequence number	Execution time (in seconds)
1	620
2	610
3	615
4	612
5	612

After parallelizing 4 CPU cores using *ray.init(num_cpus=4)*, we got an accuracy of 93.23%. Following were the execution time of all the epochs.

Table 3. Execution time for 4 CPU cores

Epoch sequence number	Execution time (in seconds)
1	577
2	569
3	562
4	572
5	565

Lastly, the model was trained on a GPU (Tesla P100). The Ray was initialized as *ray.init(num_gpus=1)*
and following observations were noted down. The accuracy was observed to be 94.67%

Table 4. Execution time for GPU

Epoch sequence number	Execution time (in seconds)
1	50
2	46
3	47
4	46
5	46

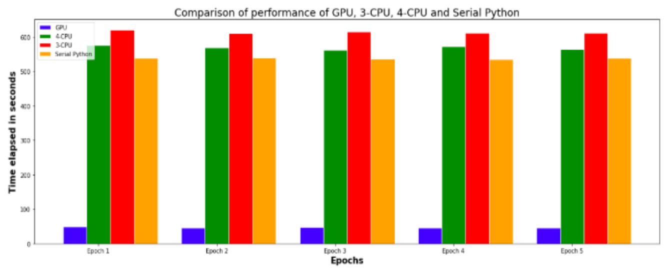

Fig 2. Performance comparison

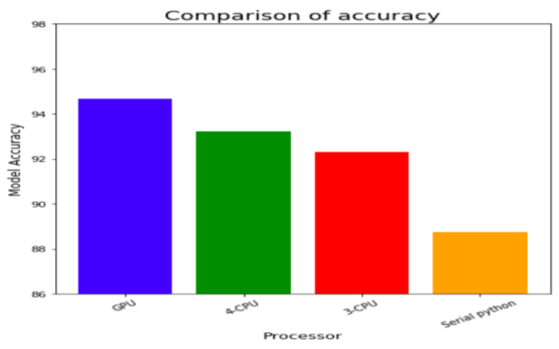

Fig 3. Comparison of accuracy

VIII. CONCLUSION

Training an image classification Convolutional Neural Network on GPU not only decreased the computation time but also increased the accuracy of the model compared to parallelized CPUs. Our results show that the GPU implementation achieves 80% speedup compared to the CPU implementation. Most of the existing studies show that the GPU implementation of deep learning model such as the CNN can yield significant speedup compared to CPUs. Due to highly parallel architecture, it helps GPU compute vector operations, matrix operations and image processing faster than CPU. The latency is overridden by the number of small cores present in the GPU. In summary, the results of the experiment showed that the computation time taken by the GPU about 80% less than the time taken by CPU to train the model. CPU is very efficient in handling small data compared to GPU. A traditional CPU might outperform GPU in some aspects, so we should not blindly choose for any computational applications.

REFERENCES

1. "Image Processing Techniques for Detection of Leaf Disease" by Arti N. Rathod, Bhavesh Tanawal, Vatsal Shah.
2. "An Introduction to Convolutional Neural Networks" by Keiron Teilo O'Shea.
3. "Performance Analysis of GPU V/S CPU for Image Processing Applications" by B. N. Manjunatha Reddy, Dr. Shanthala S. , Dr. B. R. VijayaKumar.
4. "A Comparative evaluation of the GPU vs. the CPU for Parallelization of Evaluation Algorithms through multiple independent runs" by Anna Syberfeldt and Tom Ekblom.
5. "Ray: A Distributed Framework for Emerging AI Applications" by Philipp Moritz , Robert Nishihara.
6. "Image Processing Application on Graphics processors" by Chouchene Marwa, Bahri Haythem, Sayadi Fatma Ezahra, Atri Mohamed.
7. "Real-Time Image Processing Applications on Multicore CPUs and GPGPU" by R. Samet, O.F. Bay, S. Aydın, S. Tural, A. Bayram.

AUTHORS PROFILE

Prathamesh Borhade is a third-year undergraduate pursuing computer science and engineering from Vellore institute of technology. He is a machine learning enthusiast and an aspiring data scientist currently working at Samsung Prism Project. He has also worked at Rapid Circle as a Machine Learning Intern. Email: prathamesh.borhade29@gmail.com

Rajvardhan Deshmukh is currently pursuing his B.Tech in Computer Science and Engineering from Vellore Institute of Technology, Vellore, India. He is a full stack web-developer with an industrial experience in Pune Municipal Corporation. He is also a Data Science and Machine Learning enthusiast and has done several projects based on it during his three years in college. He is the Projects Head of IEEE Computer Society, Vellore and has been actively participating in several Hackathon. He has also conducted a few Hackathons at the university level. Email: rajvardhan1999@gmail.com

Rishav Agarwal is in third-year. He has developed several projects in the platform of Web Developement. Recently, he's been involved in the field of Data Sciences. He has been an active part of several chapters like IEEE-Computer Society and also been performing as part of the VIT Dance Club. Email: rishavagarwal2717@gmail.com

Samridhi Murarka, is currently a third year undergraduate student pursuing Computer Science from Vellore Institute of Technology, Vellore. She is a UI/UX designer. She is an active IEEE Computer Society member and has led the media and design for the community in several events. She has a passion for integration of technology and business. She is currently researching more on Natural Language Processing and has developed an extractive summarizer as a part of a team of 4 members. Email: samridhi.m98@gmail.com
.