Performance of some image processing algorithms in TensorFlow

Damir Demirović, Emir Skejić, Amira Šerifović-Trbalić
University of Tuzla, Faculty of Electrical Engineering, Franjevačka 2, 75000 Tuzla, Bosnia and Herzegovina
damir.demirovic@untz.ba, emir.skejic@untz.ba

Abstract **Signal, image and Synthetic Aperture Radar imagery algorithms in recent time are used in a daily routine. Due to huge data and complexity, their processing is almost impossible in a real time. Often image processing algorithms are inherently parallel in nature, so they fit nicely into parallel architectures multicore Central Processing Unit (CPU) and Graphics Processing Unit GPUs. In this paper image processing algorithms were evaluated, which are capable to execute in parallel manner on several platforms CPU and GPU. All algorithms were tested in TensorFlow, which is a novel framework for deep learning, but also for image processing. Relative speedups compared to CPU were given for all algorithms. TensorFlow GPU implementation can outperform multi-core CPUs for tested algorithms, obtained speedups range from 3.6 to 15 times.**

***Keywords*–image processing; tensorflow; parallel processing; central processing unit; graphics processing unit;**

I. INTRODUCTION

Google TensorFlow [1] is a platform for building models in machine learning. It's mostly used for developing neural networks.In the last time has a fast-growing community in the field of deep learning networks[3], [6], the focus is on training and inference besides other compelling fields where it can be used, like optimization, image processing, etc. This can be done because of modular extensible design, which doesn't limit developer to specific applications. TensorFlow is an open-source software capable to work in heterogeneous environment consisting multicore Central Processing Unit (CPU), Graphics Processing Unit (GPU) on a desktop, server and mobile devices and Tensor Processing Unit (TPU) in production data centers. TensorFlow is a C++ based deep learning framework along with Python APIs developed and open sourced under an open source Apache 2.0 license by Google recently.

The basic unit in TensorFlow is a computational graph, which consists of nodes, which are operations, and edges which represent tensors. Tensors are arbitrary dimensional arrays, i.e. high order generalization of matrix or a vector. Each node can take multiple inputs and give multiple outputs, and tensors are passed from one node to another. TensorFlow uses two types of tensors: placeholders and variables. The edges control the flow of computation. TensorFlow uses sessions for graph computations. The computational graph at the beginning is empty and there are no variables. The session interprets user commands to initialize variables and build the computational graph. Afterwards, run the computational graph.

TensorFlow determines order of computation of a graph by creating the queue of nodes without dependencies. The program executes the nodes in queue in some order to ensure decreasing the unresolved dependencies until whole graph is computed. Parallelization is done in the manner that each node is assigned to device for computation rather than running the whole graph in parallel on multiple devices. Before the real execution, TensorFlow runs a simulation of the graph to determine the how long will take to compute and determine the computation order, then assigns nodes to devices depending whether is the kernel used or used for the operation on that device. In fact, not all operations have GPU implementation. The support exists on a wider variety of processor and non-processor architectures. TensorFlow uses abstract interface for vendors to implement hardware backends, domain specific compilers and linear algebra, etc. Support on GPU exists for specific NVIDIA cards, using the related version of CUDA toolkit [8]. TensorFlow runs on any platform that supports 64-bit Python development environment.

The core of TensorFlow is implemented in C++ programming language, however the main language is Python, but there are several communities or vendor-developed third-party TensorFlow bindings for programming languages C#, Julia, Rust, Scala, etc. There are several benefits using TensorFlow which are computational graph model, simple-to-use Application Programming Interface (API), flexible architecture, distributed processing and performance.

In [7] authors were evaluated five software frameworks for deep learning architectures, i.e. extensibility, hardware utilization and speed. They tested algorithms on a single machine multi-threaded CPU and GPU. They find that TensorFlow, albeit very flexible performance on a single GPU is not competitive as other studied frameworks.

In [2] authors make comparative study of the state-of-the-art GPU-accelerated deep learning software for training the neural network, using two CPU and two GPU platforms, and distributed versions with multiple GPUs. Their finding in general is that TensorFlow has relatively better scalability compared with other tools. Also GPU platform has a much better efficiency than many-core CPU.

In this paper the performance of TensorFlow implementation of basic algorithms used in image processing

978-1-5386-6979-2/18/$31.00 ©2018 IEEE

were evaluated on a CPU and GPU. We have evaluated several algorithms implemented in TensorFlow on a CPU and GPU which are relevant to image processing: convolution, linear algebra, partial differential equations, and image operations like gradient calculation, edge detection, filtering, image transformation and segmentation.

II. METHODS

Parallel processing in the last time has become most dominant for high-performance computing. The amount of data in signal, image and Synthetic Aperture Radar imagery processing constantly rises. GPUs were originally designed for computer graphics, today they are used for parallel processing [4],[5]. There is viable alternative to CPUs in time-consuming tasks. Today, two dominant parallel computing platforms are NVIDIA CUDA and OpenCL. Both are software frameworks for writing programs that run across heterogenous platforms like CPUs, GPUs digital signal processors (DSP) and field-programmable gate arrays (FPGA).

For the purpose of benchmarking, in this work were used the implementation of two input data sets for all algorithms, smaller and bigger.

A. Convolution

The one of the use of convolution in image processing is the edge detection. This can be done with use of a kernel, or a mask, which is usually square matrix with odd size. 2-D and 3-D convolution were implemented with the following parameters. For 2-D image was 256x256 pixels, and kernel size 5x5 and for bigger input image size is 4096x4096 and kernel size 11x11. For 3-D, image size is 64x64x64 with kernel 5x5x5 and image size 128x128x128 with kernel size 7x7x7.

B. Edge detection

Edge detection is often the first step in image preprocessing, usually a lower stage before image recognition. There are the vast of methods for image detection. In this work were implemented simple methods like image gradient and convolution and complex Canny filter. For all experiments for the edge detection the input images were 512 x 512 and 2048 x 2048 for bigger data set.

1) Image gradient

Image gradient is one kind of simpler edge detection methods. The image gradient represents directional change in intensity or color in an image and can be calculated as a convolution with a kernel size 3 x 3 in both directions.

2) Canny filter

Another robust method for the edge detection is Canny algorithm, which is a multi-stage edge detector used to extract useful structural information.

C. Gaussian filtering

This is often-used method for reducing Gaussian noise in images, which is mathematically convolution with Gaussian kernel. Gaussian smoothing is one of simple preprocessing methods for suppressing noise in images, which produces reduced image details. Gaussian kernel was used with the side length of size 5 pixels and standard deviation of 1. Input images were with size 512 x 512 and 2048 x 2048 for bigger data set.

D. Matrix multiplication

Matrices are crucial in the image representation. Simple image transformations can be treated as matrix operations. Two integer matrices were used 1000x1000 and 11000x11000.

E. Image resize

The image interpolation guesses the intensity values at missing location, using neighbor's intensity values which are known. It is useful, for example, for image scaling. For the purpose of this work, 2D implementation of bicubic and bilinear interpolation were carried. Bicubic interpolation and bilinear interpolation is an extension of linear interpolation for interpolating functions of two variables. The idea is to perform interpolation in one direction, and then again in another direction. For both experiments, the evaluation is made with image sizes 2048x2048 and 4096x4096.

F. Image segmentation

K-means is a well-known method for clustering, but in image processing is used for the segmentation. The implemented versions of K-means segmentan image into 4 clusters, with the image size 100x100 pixels, or 10000 elements, and for bigger data size 3162 x 3162 pixels or approximately 1000000 pixels.

G. Image deblurring

Deblurring is an important task in image restoration. It's a process of deconvolution, usually treated in frequency domain, but the implementation used in this work uses the deconvolution in spatial domain. The algorithm uses original image, blurred image and produce restored image. Deblurring was done using convolution matrix. Deblurring can be seen as a process of optimization while reducing the sum of square errors between blurred version and final image. For optimization, gradient descent optimizer was used, with the same learning parameter for both experiments. Input images for smaller and bigger data set were 512x512 and 2048x2048, respectively.

H. Image rotation

For the image rotation, the angle of 45 degrees was used for both data sets, using the transformation matrix and nearest interpolation method.

I. Partial differential equation

Partial differential equation (PDE) is used in image processing for image smoothing. For evaluation the discretized 2-D heat equation was used, first for image size 512 x 512, and then 2048 x 2048, keeping the same parameters ϵ and k which represent time step and damping, respectively.

III. RESULTS

All experiments are performed on a single machine running 64-bit GNU/Linux with two-core Intel(R) Xeon(R) CPU @

978-1-5386-6979-2/18/$31.00 ©2018 IEEE

2.30GHz with 2 processors, and GPU Tesla K80 (Table I). For the experiments we use algorithms given in Table II. Speedups were expressed relatively to CPU, which speedup is one. In the table, columns 1 and 2 represent smaller and bigger input size, respectively.

TABLE I SPECIFICATION OF CPU AND GPU USED IN ALL EXPERIMENTS

Processor	CPU	GPU
Number of cores	2	2 x 2496
Max clock frequency	3.0 GHz	875 MHz
Global memory size	46080 kB	2 x 12 GB
Power rating	95 W	300W

TABLE II EXECUTION TIMES (IN SECONDS) FOR ALL EXPERIMENTS FOR DATASET 1 AND 2

Algorithm	GPU		CPU	
	1	2	1	2
Canny filter	0.81	**1.03**	0.38	5.65
Image deblurring	**0.77**	**8.20**	2,75	33.68
Gaussian filter	0.46	1.94	0.10	1.53
Gradient filter	0.47	180	0.10	1.33
Kmeans segmentation	0.52	**3.56**	0.29	53.35
Matrix multiplication	**0.46**	188	0.70	193
Bicubic image resize	**0.60**	0.40	0.88	0.12
Bilateral image resize	0.38	1.71	0.07	1.09
Image rotation	0.37	**0.42**	0.03	2.29
2-D image convolution	0.03	20.88	0.02	20.63
3-D image convolution	0.92	19.66	0.67	18.84
PDE - Heat equation	0.77	**0.66**	0.28	4.50

It can be seen that for the smaller data size CPU outperforms GPU in most cases, i.e. data size is not adapted well for the GPU. For bigger input data, GPU gave better performance especially for algorithms: rotation of image, deblurring and Canny edge detection, and PDE. Figure 1 shows the speedups obtained for the smaller data size, and in Figure 2 presents relative speedup for the bigger data set. For the smaller and bigger data set highest speedup is 3.6 times, and 15 times respectively, and for bigger data set lowest speedup is 0.1 and 0.3, respectively. Three algorithms from the smaller dataset gain speedups: bicubic resize matrix multiplication and deblur. From the obtained results we can see that five algorithms from bigger data set can gain speedups: PDE, image rotation, K-means, deblur and Canny. Deblur is only algorithm with speedups for all datasets. Also, K-means can gain highest speed up for bigger data size. Poor performance using gradient and Gaussian smoothing is in fact due to the small kernel size which is not suitable for the execution on a GPU.

From the experiments we cannot expect significant improvements for GPU using CUDA, for example in [4] and [5]. The one possibility to overcome this problem would be using a cluster of TensorFlow servers, so the separate operation can be carried on different machines in a distributed manner.

IV. CONCLUSION

In this work, the image processing algorithms were evaluated on different parallel processing units using TensorFlow platform. Almost all tested algorithms can get speed gain using parallelization on a GPU. TensorFlow implementation on a CPU and GPU was used, and according to this research the running times highly depend on input data size. From obtained results GPU speedups can be expected for most of algorithms from 3.6 times to 15 times. The computation on the smaller data set obtain lower speedups due to the fact how TensorFlow handling computation, i.e. each node is assigned to device for computation rather than running the whole graph in parallel on multiple devices.

Future research can investigate performance gain using distributed TensorFlow or investigate possible optimizations on a GPU.

V. ACKNOWLEDGEMENT

The authors express their gratitude to NATO SPS Programme for the support of the presented work provided by the project grant NATO SPS 985208.

REFERENCES

[1] M. Abadi, P. Barham, J. Chen, at all."TensorFlow: a system for large-scale machine learning". In *Proceedings of the 12th USENIX conference on Operating Systems Design and Implementation* (OSDI'16). USENIX Association, Berkeley, CA, USA, 265-283., 2016.

[2] S. Shi, Q. Wang, P. Xu and X. Chu, "Benchmarking State-of-the-Art Deep Learning Software Tools," *2016 7th International Conference on Cloud Computing and Big Data (CCBD)*, Macau, 2016, pp. 99-104.

[3] Y. Kochura, S. Stirenko, O. Alienin, M. Novotarskiy and Y. Gordienko, "Comparative analysis of open source frameworks for machine learning with use case in single-threaded and multi-threaded modes," *2017 12th International Scientific and Technical Conference on Computer Sciences and Information Technologies (CSIT)*, Lviv, 2017, pp. 373-376.

[4] Z. Yang, Y. Zhu and Y. Pu, "Parallel Image Processing Based on CUDA," 2008 International Conference on Computer Science and Software Engineering, Wuhan, Hubei, 2008, pp. 198-201.

[5] I. K. Park, N. Singhal, M. H. Lee, S. Cho and C. Kim, "Design and Performance Evaluation of Image Processing Algorithms on GPUs," in IEEE Transactions on Parallel and Distributed Systems, vol. 22, no. 1, pp. 91-104, Jan. 2011.

[6] J. Lawrence,J. Malmsten, A. Rybka at all. "Comparing TensorFlow Deep Learning Performance Using CPUs, GPUs, Local PCs and Cloud." 2017.

[7] S. Bahrampour, N. Ramakrishnan, L. Schott. And M. Shah, M. Comparative Study of Caffe, Neon, Theano, and Torch for Deep Learning". CoRR, abs/1511.06435. 2015.

[8] CUDA, http://docs.nvidia.com/cuda/

978-1-5386-6979-2/18/$31.00 ©2018 IEEE

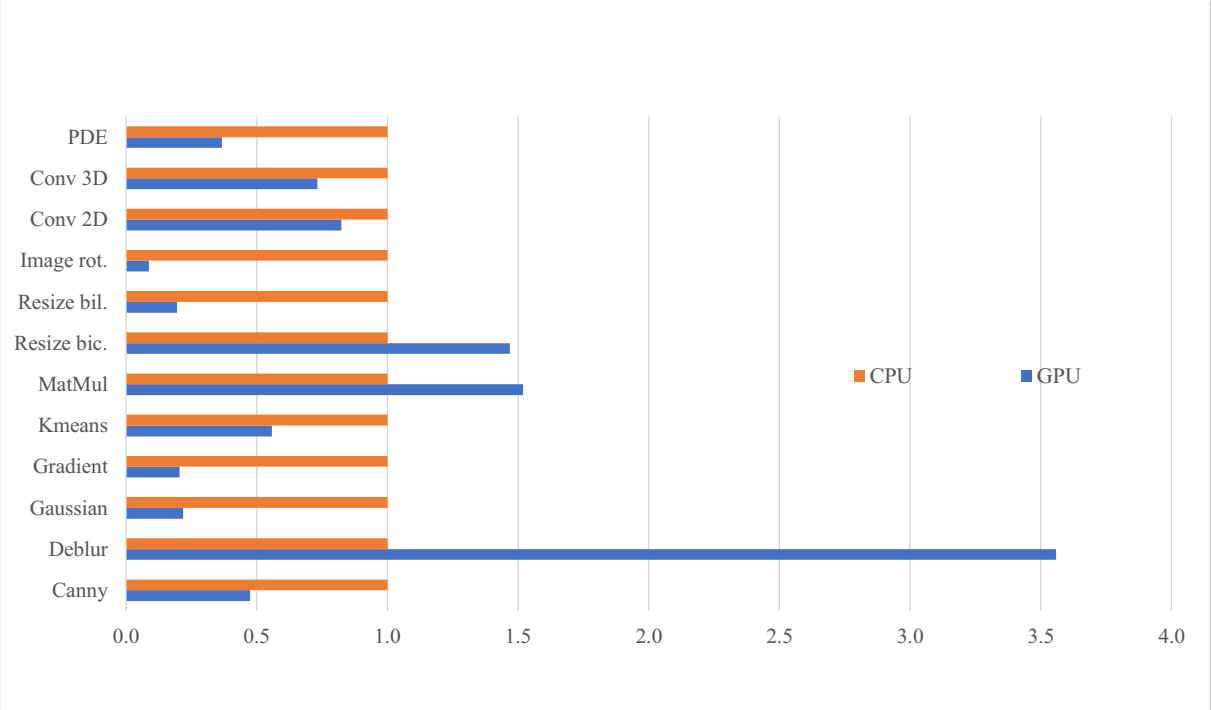

Figure 1 Speedups for smaller data sizes (Table II)

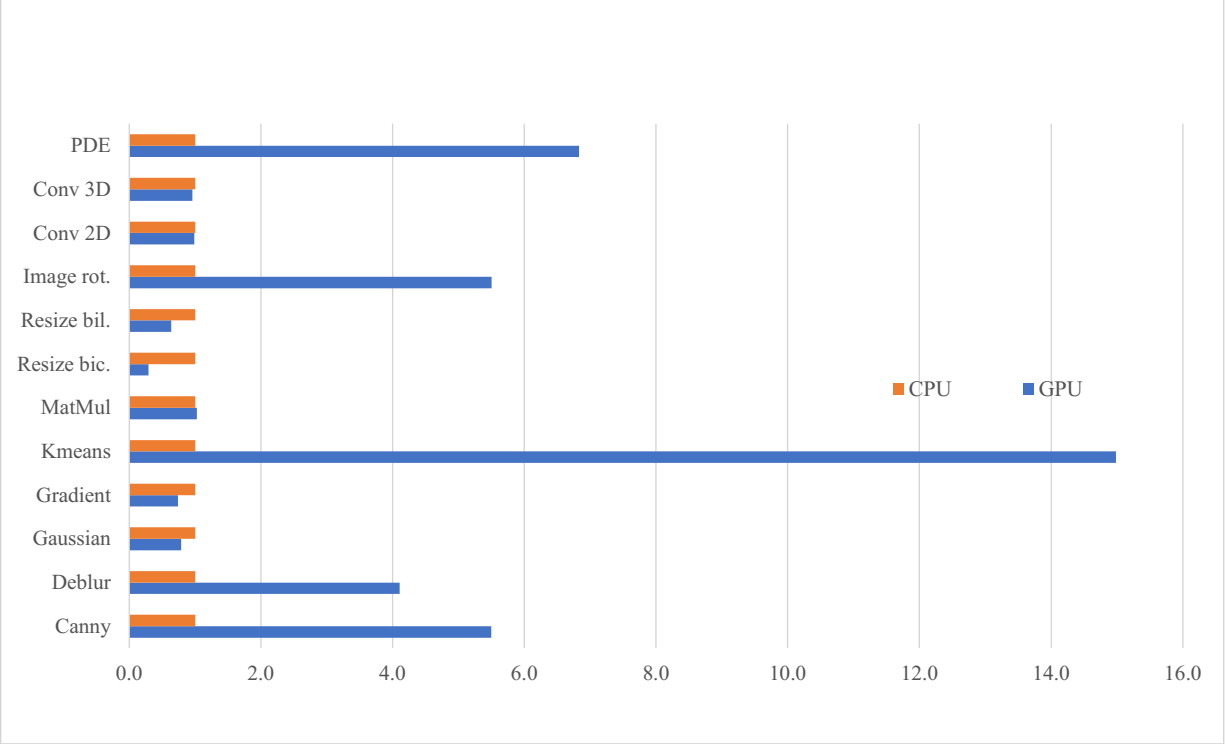

Figure 2 Speedups for bigger data sizes (Table II)

978-1-5386-6979-2/18/$31.00 ©2018 IEEE