JOIV : Int. J. Inform. Visualization, 6(2) - June 2022 411-417

INTERNATIONAL JOURNAL ON INFORMATICS VISUALIZATION

journal homepage : www.joiv.org/index.php/joiv

High-Performance Computing on Agriculture: Analysis of Corn Leaf Disease

Evianita Dewi Fajrianti [a], Afis Asryullah Pratama [a], Jamal Abdul Nasyir [b], Alfandino Rasyid [a], Idris Winarno [b,*], Sritrusta Sukaridhoto [b]

[a] *Electrical Engineering Department, Politeknik Elektronika Negeri Surabaya, Jl. Raya ITS, Surabaya, 60111, Indonesia*
[b] *Information and Computer Engineering Department, Politeknik Elektronika Negeri Surabaya, Jl. Raya ITS, Surabaya, 60111, Indonesia*
*Corresponding author: *idris@pens.ac.id*

***Abstract*— In some cases, image processing relies on a lot of training data to produce good and accurate models. It can be done to get an accurate model by augmenting the data, adjusting the darkness level of the image, and providing interference to the image. However, the more data that is trained, of course, requires high computational costs. One way that can be done is to add acceleration and parallel communication. This study discusses several scenarios of applying CUDA and MPI to train the 14.04 GB corn leaf disease dataset. The use of CUDA and MPI in the image pre-processing process. The results of the pre-processing image accuracy are 83.37%, while the precision value is 86.18%. In pre-processing using MPI, the load distribution process occurs on each slave, from loading the image to cutting the image to get the features carried out in parallel. The resulting features are combined with the master for linear regression. In the use of CPU and Hybrid without the addition of MPI there is a difference of 2 minutes. Meanwhile, in the usage between CPU MPI and GPU MPI there is a difference of 1 minute. This demonstrates that implementing accelerated and parallel communications can streamline the processing of data sets and save computational costs. In this case, the use of MPI and GPU positively influences the proposed system.**

***Keywords*— Corn leaf disease; image analysis; GPU; MPI.**

Manuscript received 11 Feb. 2022; revised 28 Mar. 2022; accepted 17 Apr. 2022. Date of publication 30 Jun. 2022.

I. INTRODUCTION

Indonesia is a country that has a large food source [1]. One of the food products of this country is corn. Corn is a food crop that is widely used for staple foods, snacks, and other food preparations [2]. To maintain a balanced supply, it is necessary to empower plants to stay healthy. One of the efforts that can be made to maintain the supply of corn is by paying attention to the health of corn. Some cases of health can affect crop yields. Poor yields can harm farmers financially, and the corn processing industry is experiencing delays for consumers [3]. Diseases of corn occur in many leaves, ears, and roots. On corn leaves, specific diseases such as leaf spots, spots, and fungus often occur on corn. To eliminate diseases on the leaves that need to be detected early for immediate treatment of plants [4]. One way to do this is by analyzing images of each type of disease on the leaves. To achieve the goal of disease in leaves, of course, requires a collection of data with complete information that can be used using computer vision [5].

Computer technology has been widely applied to read images, pattern recognition for early detection, and automate some image analysis. However, new problems arise when studying large amounts of data, usually gigabytes. This can increase the computer load, which can cause high losses. For this reason, additional acceleration is needed to read large amounts of data collection [6].

Several studies on image analysis have been implemented in certain conditions. For example, diagnosis in the health field using computer vision [12]. Research about pothole recognition to find the pothole spot in the city [13]. The experiment was conducted using a camera node device attached to a vehicle to recognize a pothole by scanning the road. The system consists of 2 devices. The first device is Raspberry zero, with a Raspberry Pi camera attached to the vehicle's license plate to take a video and stream the frame to the processing device. The second device is a laptop that receives the video frame and then compares it with the classification model. The classification algorithm uses a 1000 image dataset separated into 700 for training data and 300 for testing data.

The other experiment is about surface segmentation [14] and the segmentation for quality control in the tile industry [15]. This experiment focused on detecting the defect of each tile that has been produced. The dataset that has been used is about 410 data which are divided into 246 training data and 164 testing data. On the processing device, a GPU used is NVIDIA GeForce 930mx GPU with CUDNN acceleration. The experiment was conducted with several methods that have been recorded, which is SegNet-VGG15 takes time 115 min 36 sec, Deeplabv3plus-Resnetl8 takes time takes time 115 min 36 sec, Deeplabv3plus-Mobilenetv2 takes time 65 min 26 sec. Also, Deeplabv3plus-Xception (takes 100 min 52 sec), and the most accurate model is the model from Deeplabv3plus-Xception 99.10%.

Object recognition based on computer vision has also been implemented in a self-driving car to detect a traffic light in real-time [16]. The self-driving car is one of the solutions for urban people to minimize the use of fatigue in a constant concentration. Then this vehicle could raise the safety aspect of the urban street. Object recognition is one of the issues that must be done in this scope. The system uses a high-end cloud-based GPU (NVIDIA Tesla T4 GPU) for the computation, Keras and Tensorflow backend in the Google Collaboratory cloud platform, and RetinaNet deep learning architecture for the model. The dataset that has been used is about 5000 data for training and 8334 data for testing.

Another experiment used a parallel computing system in image processing to convert an image using raspberry pi's CPU [17]. The cluster computer method is chosen since the image segmentation process is running with limited parameters while using a single raspberry pi. Here was compared the interval data while using a single raspberry pi and while using clustered two raspberry pi. The experiment uses Open MPI to make the computation parallel. From the result, the author shows that the cluster or parallel method is faster than the single one. Although for various data sizes, the cluster computation's performance is still great.

Another parallel computing experiment is used along with a GPU framework to detect multitemporal hyperspectral images [18]. For the overview, this is a part of the change detection experiment. It is used to detect changes on a pixel of a hyperspectral image. The experiment consists of 4 main processes. The first is the Segmentation process, which is intended to generate the segmented hyperspectral image. Second is the Fusion process to identify the difference in the image. The third is the thresholding process using *Otsu's* method to create a binary image according to the threshold. The last is the spatial regularization process to make a spatially regularized change map. This experiment is evaluated both in CPU Intel Core i5-3470 and GPU TITAN X, and the result is projection using GPU reach up to 46.5 times speeds up compared to OpenMP CPU version. For the accuracy, it reaches 96.96%.

This study proposes HPC (High-Performance Computing) as a big data training solution. However, it is not easy to implement HPC for imaging applications. HPC has good potential to be applied to high computational image recognition operations [7]. To improve the parallel programming environment, it is necessary to implement an MPI (Message Passing Interface) to suppress communication between GPU and HPC [8]. MPI is a standard to make implementing the message passing concept easier [9]. Specifically for the parallel MPI programming implementation, the Open MPI framework is chosen. Open MPI iso is not just about "open-source" but also because it has an impressive performance, although for the heterogeneous platform [10], [11].

The parallelism is implemented through CUDA as the main execution engine at every level to get the value that attracts performance and scalability. Integrating GPU with MPI can be used to solve complex scalability problems where MPI can act as a communication link between GPUs. In this research, implementation is carried out in various ranges for use between CPU, GPU, CPU with MPI, and GPU with MPI. The experiment was carried out using 3 instances with details of 1 instance serving as master, 2 instances as a slave. The master instance is responsible for distributing the load on the slave instances.

II. Materials and Method

A. CPU vs. GPU

CPU (Central Processing Unit) is the most important hardware component or device for processing data on a computer. CPU controls all activities and runs all programs, including applications and software in it, so the CPU can be called the brain of a computer [19]. At least one "processor" or chip is built into the CPU, which is involved in the computing process of the computer. The main function of the CPU is to perform arithmetic and logical operations on data retrieved from memory or information provided by hardware.

CPU and GPU are both critical machines whose purpose is to handle data. But CPUs and GPUs have different architectures and were created for different purposes [20]. CPUs are designed to handle a wide variety of workloads, especially if latency or per-core performance is important. GPUs have evolved into general-purpose parallel processors, handling growing applications [21]. The use of large data with the composition of graphic calculations requires the GPU to accomplish this. The deep learning algorithm is implemented using a GPU accelerated approach. This can significantly improve performance and bring real-world problems within easy reach of work.

B. GPU vs. MPI

GPUs consist of thousands of cores that can execute commands in parallel [22]. GPU-accelerated graphics cards allow these programmers to apply programming using parallel computing technology to the GPU core along with the CPU. GPUs can consist of thousands of cores working in parallel. One implementation of GPU acceleration is CUDA. CUDA is an advanced GPU architecture that is used to assist the CPU in graphics computing purposes. CUDA is designed to use more than one block and requires synchronization from the CPU. A GPU can contain a series of cores that work with the same instructions, therefore CUDA is widely used to work in parallel [23].

MPI (Message Passing Interface) is an API that allows communication between computers to complete a task over a network [24]. MPI implementation provides a unique approach to building software with a specific function. MPI can be run through languages such as Fortran, C, and C++.

MPI was also portable in supporting various platforms. MPI relies on network bandwidth and throughput to be able to explore parallel computing. MPI supports point-to-point communication and is collective.

In this scenario, the mpi4py library was used to provide python binding for the standard use of passing interface messages (MPI). The use of mpi4py can exploit many processors on workstations, clusters, and supercomputers. The application of this library also supports point-to-point communication. Also, use the OpenCV library to do image processing. The use of OpenCV supports cross platforms that use that can be developed a real-time computer vision application. This study compared the performance of corn leaf disease detection in four scenarios. The four scenarios cover CPU, CUDA, and MPI usage. It can be seen in Table 1.

TABLE I
CORN LEAF DISEASE DETECTION PERFORMANCE COMPARISON SCENARIO

Scenario	CPU	Hybrid	MPI	Nodes
1	√			1
2		√		1
3	√		√	2
4		√	√	2

The first experiment implemented 1 node with CPU to perform corn leaf disease detection, starting with reading the input image. Then pre-processing is done using the CPU. Next, feature extraction was performed as input for linear regression, Figure 1a. The second experiment implementation used 1 node with CUDA for acceleration on corn leaf disease detection with the same steps as the CPU, but in pre-processing it is run using the GPU, Figure 1b.

The third scenario is the implementation using 3 nodes with MPI to perform corn leaf disease detection, starting with the master reading the input image then scattering to be divided into slave 1 and slave 2 using the CPU, then feature extraction is carried out. After obtaining features, gather from both slaves to obtain linear regression input, Figure 2a. The fourth scenario is the implementation using 3 nodes with CPU+MPI with almost the same steps as the third scenario. However, there is a slight difference in the pre-processing done using CUDA, Figure 2b.

Fig. 1 (a) CPU Image Analysis on Corn Leaf Disease, (b) Hybrid Image Analysis on Corn Leaf Disease.

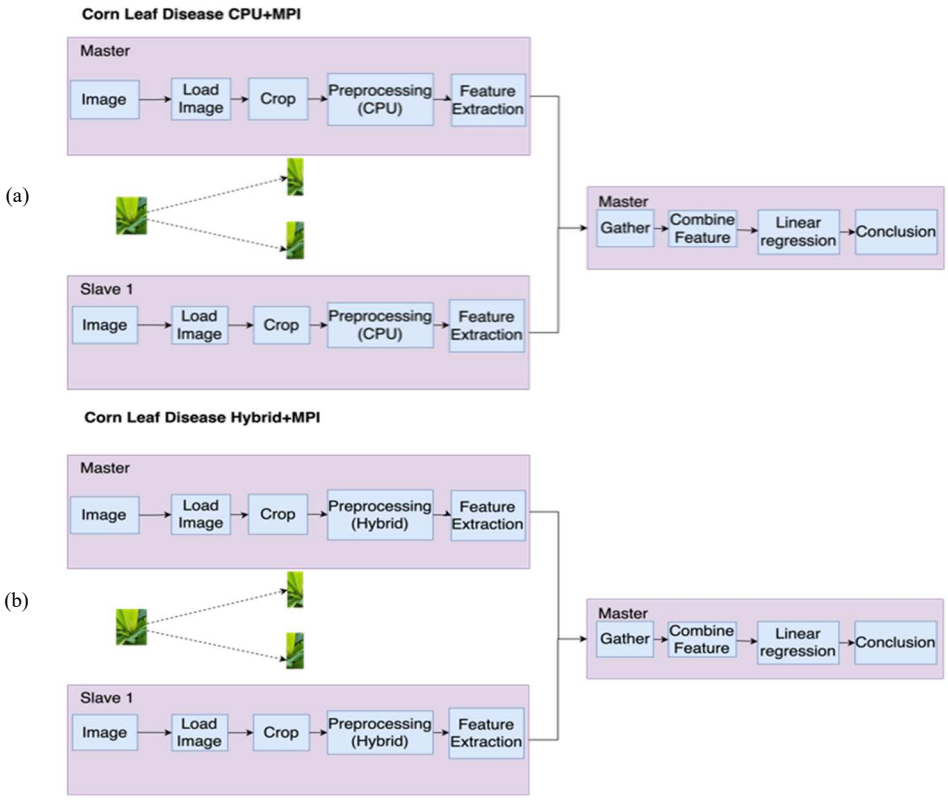

Fig. 2 (a) CPU+MPI Image Analysis on Corn Leaf Disease, (b) Hybrid+MPI Image Analysis on Corn Leaf Disease

The amount of data learned by algorithm design using CPU requires power optimization. Introducing GPUs to integrate with CPUs has been widely implemented to predict execution time. This study discusses the integration of four different scenarios for the implementation of image analysis on corn leaf disease. The dataset used is open source from Kaggle[https://www.kaggle.com/qramkrishna/corn-leaf-infection-dataset]. The dataset is taken based on the need to test the efficiency of using CUDA and MPI; high-resolution images are applied with many datasets. The dataset obtained is 14.04 GB in size which consists of two classes of leaf conditions, i.e., healthy and infected. A total of 4225 images obtained with 2000 healthy and 2226 infected compositions have also been annotated. Splitting of data 60% training and 40% testing is done to avoid over-learning and poor performance. The proportion of the dataset resulting from splitting obtained 2535 trains and 1690 tests. In the image pre-processing process, the following processes occur, Figure 3. The first stage is reading the image for conversion to HSV and grayscale. In HSV conversion, masking is performed, while gray-level correlation is performed in grayscale. Then the feature extraction is carried out, namely the mean, standard deviation, variance, entropy, root mean square, and smoothness of the masked HSV as well as the contrast, energy, and homogeneity features of the gray level correlation matrix.

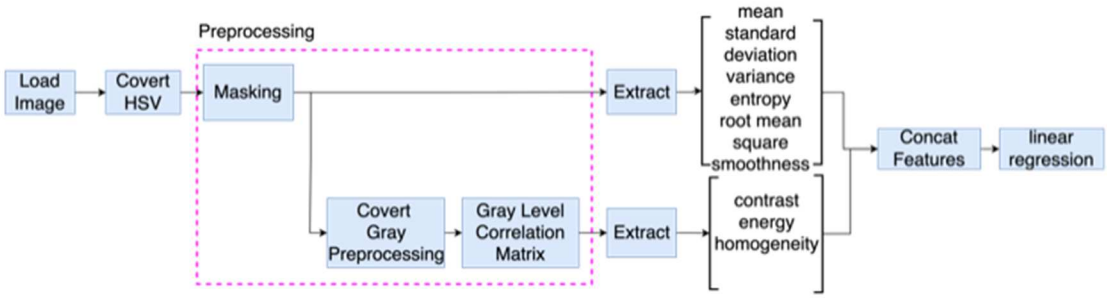

Fig. 3 Preprocessing Image Leaf Corn Disease

Parallel computing is implemented through a virtual personal computer with NVIDIA Tesla T4 professional graphics card. The device specifications used in the implementation of this study are shown in Table 2. Implementation of the use of Google Cloud Platform to be easily accessible on mobile with the Google Compute Engine Virtual Machine model with 13 GB of RAM. The virtual personal computer is set up for CUDA and no CUDA usage scenarios.

TABLE III
TECHNICAL SPECIFICATION

Specification	
Manufacture	Google Cloud Platform
Model	Google Compute Engine Visual Machine
Processor	Intel(R) Xeon(R) CPU @ 2.20GHz 2.20GHz
Installed Memory (RAM)	13.0 GB
System Type	64-bit Operating System, x64-based processor

Algorithm 1. explains that pre-processing images without using a GPU accelerator, namely CUDA, is carried out by converting the image to HSV, then creating an image mask based on the lower threshold of 14, 32.64, 22.185, and the upper threshold of 34, 255, 232, 815. If the image mask is between the threshold, it reads yellow. Outside the threshold, it is changed to 0 or black. The masked image due to the threshold value is converted to gray for the Gray Level Correlation Matrix (GLCM) process. The results from GLCM are then returned to feature extraction.

Algorithm 1. Pseudocode on a pre-processing image without CUDA

Pre-processing without CUDA
Input: (1) image, (2) lower threshold for mask, (3) upper threshold for mask
Output: (1) masked HSV image, (2) GLCM (gray level correlation matrix)
1: Convert the input image to HSV
2: Create a image mask, based on lower & upper threshold
3: Create a masked image with the image mask
4: Create a gray image from masked image
5: Make a GLCM (gray level correlation matrix) from gray
6: image
Return masked image and GLCM

Algorithm 2. explains that pre-processing images using hybrid (CPU and CUDA) which is done by converting the image to HSV, then creating an image mask based on the lover threshold and upper threshold. The difference in pre-processing using CUDA lies in masking the image done through CUDA.

Algorithm 2. Pseudocode on a pre-processing image with CUDA

Pre-processing with CUDA
Input: (1) image, (2) lower threshold for mask, (3) upper threshold for mask
Output: (1) masked HSV image, (2) GLCM (gray level correlation matrix)
1: Create a GPU CUDA image object
2: Upload input image into the GPU CUDA image
3: Convert the GPU CUDA image to HSV
4: Download the HSV image from the GPU CUDA image
5: Create an image mask, based on lower & upped threshold
6: Create a masked HSV image with the image mask
7: Create GPU CUDA HSV masked image object
8: Upload HSV masked image into the GPU CUDA HSV masked image
9: Convert the GPU CUDA HSV masked image to gray
10: Download the grey image from GPU CUDA HSV masked image

11: Return a GLCK (gray level correlation matrix) from grey image
12: Return masked HSV image and GLCM

III. Results and Discussion

Tests were carried out using four tests with the same datasets. The first test was performed using one CPU. In this process, images of various sizes and sizes are sent for pre-processing through the CPU. The processing time for each image depends on the image size, and figure 4 shows the reading time of each image. The results of image pre-processing using a single CPU get a total value of 44 minutes. This needs to be compared with the use of the GPU to determine the effect of adding acceleration to image reading.

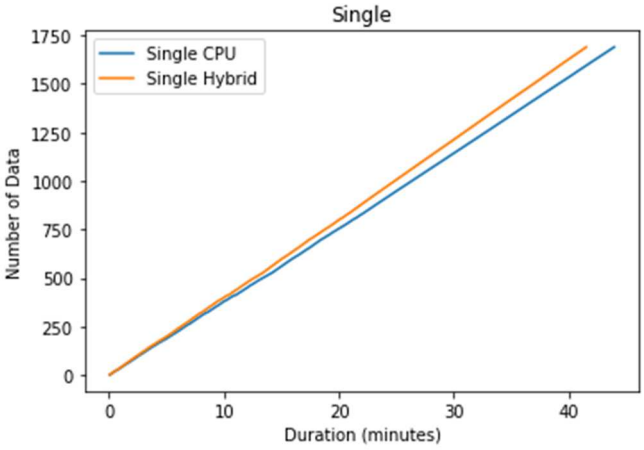

Fig. 4 Image analysis time using Single CPU and Single hybrid

In image processing based on sequence, several images take about 3 s per frame, but in using a single CPU, image processing is dominant in the range of 1 – 1.5 ms. Tests by adding a GPU are also applied to determine the effect of GPU usage on image analysis. However, the use of this GPU cannot be perfectly applied. The CPU participates in completing the image task. Figure 4 shows the result of reading a sequence of images against the length of time it takes for each frame. From the Hybrid test for image pre-processing, the total value is 42 minutes. If the two are compared, namely between single GPU and Hybrid, it shows a significant increase in the efficiency of time requirements in the pre-processing process, namely, the time difference between the two is about 2 minutes.

The next experiment was to add MPI to the CPU for parallel communication, Figure 5. The results of this test obtained a total value of 23 minutes for image pre-processing. In the MPI addition test, there was a significant increase in time when compared to without using MPI. The use of MPI, in this case, can communicate in parallel between networks. Communication between networks in MPI uses its modular nature to accelerate portable parallel development, so the time required is relatively less. This means that the time used to perform image pre-processing is getting smaller. Efficient use of time can reduce computing costs.

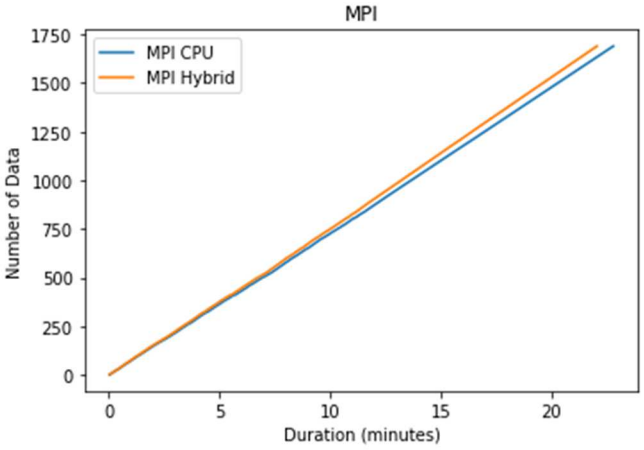

Fig. 5 Image analysis time using CPU MPI and using MPI Hybrid

The use of MPI has a big influence on time requirements. The next experiment implemented MPI with the GPU, Figure 5. The results of this test obtained a total value of pre-processing images of 22 minutes. When compared between the use of MPI on the CPU and GPU, there is a significant increase, namely the difference of 1 minute. This shows that the use of CUDA acceleration with MPI results in significant time cuts. This efficiency can cut computational costs and speed up pre-processing with bloated amounts of data.

A confusion matrix can be applied to measure the classification performance of outputs with two or more classes. The Confusion Matrix is a table with four combinations of predicted and actual values. Four terms represent the results of the classification process in the confusion matrix, namely True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN). Equation 1. is a formula for calculating accuracy values in two classes.

$$\text{accuracy} = (\text{TP} + \text{TN})/(\text{TP} + \text{TN} + \text{FP} + \text{FN}) \quad (1)$$

$$accuracy = 83.37\%$$

$$precision = TP/(TP + FP) \quad (2)$$

$$precision = 86.18\%$$

$$recall = TP/(TP + FN) \quad (3)$$

$$recall = 82.71\%$$

$$F1\ score = 2((precision \cdot recall)\ /\ (precision + recall)) \quad (4)$$

$$F1\ score = 0.8441$$

Figure 6. shows the predicted result with the true value. Where in the implementation of image analysis is classified into two classes, namely the infected class and the health class. The combination resulting from the confusion matrix is True positive 761, False Positive 122, False Negative 159, and True Negative 648. So, the accuracy value obtained from pre-processing corn leaf disease is 83.37%. Also obtained a precision value of 86.18%, recall value of 82.71%, and F1 score of 0.8441, which describes the accuracy between the actual data and the prediction results given by the model.

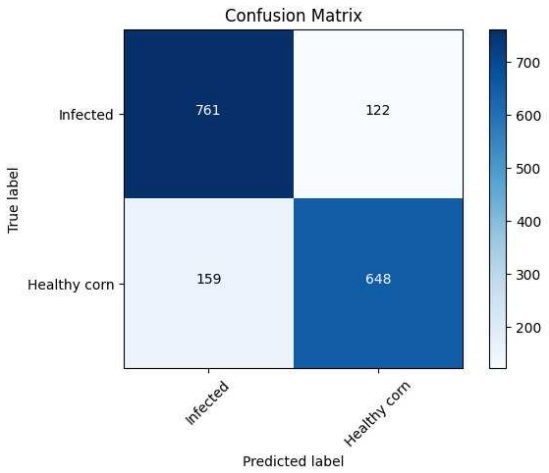

Fig. 6 Confusion Matrix Value on CPU, Hybrid, CPU MPI, and Hybrid MPI Preprocessing

To measure processing performance can be measured through the speed-up factor. The speed-up factor refers to the increase in performance from the addition of processing elements in Equation 3.

$$S(p) = ts/tp \quad (3)$$

Equation 3 is the execution time on a single processor and the execution time on a multiprocessor. Speed up increases its speed by using a multiprocessor. Figure 7 is a speed-up graph that refers to an increase in processing performance.

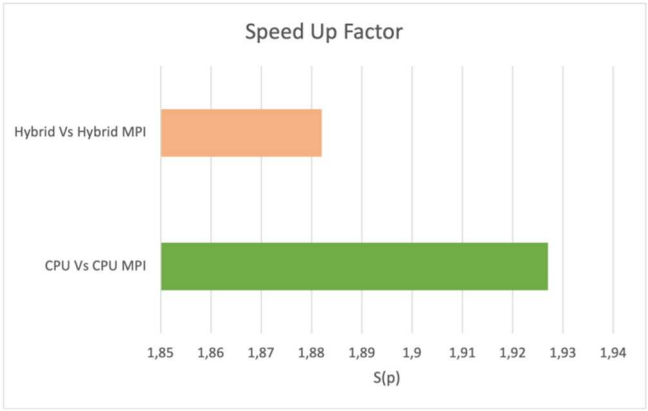

Fig. 7 Speed-up Factor

From the speed-up factor measurement, we get a value of 1.927 for a single CPU against CPU MPI, which indicates that the value is greater than 1.0 and that the program has accelerated more than the CPU increase. As for the speed-up factor for a single Hybrid against Hybrid MPI, getting a value of 1.882 means the program has benefited from parallel execution.

IV. Conclusion

In the case of image processing, producing accurate and precise readings requires large training data. The larger the training data provided, the greater the computational requirements. One way to reduce computational costs is to take advantage of CUDA acceleration and parallel communication on MPI. In this study, the utilization of CUDA runs as expected, which can reduce the time required by 2 minutes compared to a single CPU. The application of MPI with CUDA also has a good effect on time efficiency compared to the use of MPI with CPU. The difference between the two is 1 minute. Thus, MPI and CUDA can be a new alternative for processing large image data. This study discusses the comparison of parallel computing that applies CUDA and MPI. The use of MPI has a good effect on the case of image pre-processing of corn leaf disease. MPI can reduce the time required for large data. However, the implementation of MPI with GPU still uses a hybrid scenario due to a bug in the OpenCV library so that pure CUDA cannot be implemented, but in future developments, it can be implemented using CUDA without dependence on the CPU.

Acknowledgment

The authors thank to Politeknik Elektronika Negeri Surabaya for supporting research. This research was under the project of the high-performance computing course in 2021.

References

[1] Z. Rozaki, "Chapter Five - Food security challenges and opportunities in indonesia post COVID-19," in Advances in Food Security and Sustainability, vol. 6, M. J. Cohen, Ed. Elsevier, 2021, pp. 119–168. doi: 10.1016/bs.af2s.2021.07.002.

[2] N. Palacios-Rojas et al., "Mining maize diversity and improving its nutritional aspects within agro-food systems," Compr. Rev. Food Sci. Food Saf., vol. 19, no. 4, pp. 1809–1834, 2020, doi: 10.1111/1541-4337.12552.

[3] D. S. Mueller et al., "Corn Yield Loss Estimates Due to Diseases in the United States and Ontario, Canada, from 2016 to 2019," Plant Health Prog., vol. 21, no. 4, pp. 238–247, Jan. 2020, doi: 10.1094/PHP-05-20-0038-RS.

[4] R. Meng et al., "Development of Spectral Disease Indices for Southern Corn Rust Detection and Severity Classification," Remote Sens., vol. 12, no. 19, Art. no. 19, Jan. 2020, doi: 10.3390/rs12193233.

[5] K. P. Panigrahi, A. K. Sahoo, and H. Das, "A CNN Approach for Corn Leaves Disease Detection to support Digital Agricultural System," in 2020 4th International Conference on Trends in Electronics and Informatics (ICOEI)(48184), Jun. 2020, pp. 678–683. doi: 10.1109/ICOEI48184.2020.9142871.

[6] L. I. U. Zhentao, Y. Yi, W. Dongchao, and X. I. E. Xiaoyao, "A CUDA-based parallel accelerating geographically weighted regression algorithm for big data," Bull. Surv. Mapp., vol. 0, no. 12, p. 1, Jan. 2021, doi: 10.13474/j.cnki.11-2246.2020.0379.

[7] M. A. Elaziz, K. M. Hosny, A. Salah, M. M. Darwish, S. Lu, and A. T. Sahlol, "New machine learning method for image-based diagnosis of COVID-19," PLOS ONE, vol. 15, no. 6, p. e0235187, Jun. 2020, doi: 10.1371/journal.pone.0235187.

[8] A. Gupta, D. Singh, and M. Kaur, "An efficient image encryption using non-dominated sorting genetic algorithm-III based 4-D chaotic maps," J. Ambient Intell. Humaniz. Comput., vol. 11, no. 3, pp. 1309–1324, Mar. 2020, doi: 10.1007/s12652-019-01493-x.

[9] W. Gropp, E. Lusk, N. Doss, and A. Skjellum, "A high-performance, portable implementation of the MPI message passing interface standard," Parallel Comput., vol. 22, no. 6, pp. 789–828, Sep. 1996, doi: 10.1016/0167-8191(96)00024-5.

[10] E. Gabriel et al., "Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation," in Recent Advances in Parallel Virtual Machine and Message Passing Interface, Berlin, Heidelberg, 2004, pp. 97–104. doi: 10.1007/978-3-540-30218-6_19.

[11] R. L. Graham, G. M. Shipman, B. W. Barrett, R. H. Castain, G. Bosilca, and A. Lumsdaine, "Open MPI: A High-Performance, Heterogeneous MPI," in 2006 IEEE International Conference on Cluster Computing, Sep. 2006, pp. 1–9. doi: 10.1109/CLUSTR.2006.311904.

[12] R. P, M. A, M. B, and G. K. S, "Lung Cancer Diagnosis and Treatment Using AI and Mobile Applications," Int. J. Interact. Mob. Technol. IJIM, vol. 14, no. 17, Art. no. 17, Oct. 2020, doi: 10.3991/ijim.v14i17.16607.

[13] A. Rasyid et al., "Pothole Visual Detection using Machine Learning Method integrated with Internet of Thing Video Streaming Platform,"

in 2019 International Electronics Symposium (IES), Sep. 2019, pp. 672–675. doi: 10.1109/ELECSYM.2019.8901626.

[14] K. Salhi, E. M. Jaara, and M. T. Alaoui, "Texture Image Segmentation Approach Based on Neural Networks," Int. J. Recent Contrib. Eng. Sci. IT IJES, vol. 6, no. 1, Art. no. 1, Mar. 2018, doi: 10.3991/ijes.v6i1.8166.

[15] E. D. Fajrianti, E. Suryawati Ningrum, A. Risnumawan, and K. V. Madalena, "Tile Surface Segmentation Using Deep Convolutional Encoder-Decoder Architecture," in 2020 International Electronics Symposium (IES), Sep. 2020, pp. 364–370. doi: 10.1109/IES50839.2020.9231575.

[16] A. N. Aneesh, L. Shine, R. Pradeep, and V. Sajith, "Real-time Traffic Light Detection and Recognition based on Deep RetinaNet for Self Driving Cars," in 2019 2nd International Conference on Intelligent Computing, Instrumentation and Control Technologies (ICICICT), Jul. 2019, vol. 1, pp. 1554–1557. doi: 10.1109/ICICICT46008.2019.8993293.

[17] R. F. Rahmat, T. Saputra, A. Hizriadi, T. Z. Lini, and M. K. M. Nasution, "Performance Test of Parallel Image Processing Using Open MPI on Raspberry PI Cluster Board," in 2019 3rd International Conference on Electrical, Telecommunication and Computer Engineering (ELTICOM), Sep. 2019, pp. 32–35. doi: 10.1109/ELTICOM47379.2019.8943848.

[18] J. López-Fandiño, D. B. Heras, F. Argüello, and M. Dalla Mura, "GPU Framework for Change Detection in Multitemporal Hyperspectral Images," Int. J. Parallel Program., vol. 47, no. 2, pp. 272–292, Apr. 2019, doi: 10.1007/s10766-017-0547-5.

[19] C. Shen, C. Chen, and J. Zhang, "Micro-architectural Cache Side-Channel Attacks and Countermeasures," in 2021 26th Asia and South Pacific Design Automation Conference (ASP-DAC), Jan. 2021, pp. 441–448.

[20] J. C. Phillips et al., "Scalable molecular dynamics on CPU and GPU architectures with NAMD," J. Chem. Phys., vol. 153, no. 4, p. 044130, Jul. 2020, doi: 10.1063/5.0014475.

[21] M. Liu, H. Li, M. Zhang, and T. Wang, "Graphics Processing Unit - Based Match and Locate (GPU - M&L): An Improved Match and Locate Method and Its Application," Seismol. Res. Lett., vol. 91, no. 2A, pp. 1019 – 1029, Jan. 2020, doi: 10.1785/0220190241.

[22] G. M. J. Barca, J. L. Galvez-Vallejo, D. L. Poole, A. P. Rendell, and M. S. Gordon, "High-Performance, Graphics Processing Unit-Accelerated Fock Build Algorithm," J. Chem. Theory Comput., vol. 16, no. 12, pp. 7232–7238, Dec. 2020, doi: 10.1021/acs.jctc.0c00768.

[23] D. Rosenberg, P. D. Mininni, R. Reddy, and A. Pouquet, "GPU Parallelization of a Hybrid Pseudospectral Geophysical Turbulence Framework Using CUDA," Atmosphere, vol. 11, no. 2, Art. no. 2, Feb. 2020, doi: 10.3390/atmos11020178.

[24] L. Clarke, I. Glendinning, and R. Hempel, "The MPI Message Passing Interface Standard," in Programming Environments for Massively Parallel Distributed Systems, Basel, 1994, pp. 213–218. doi: 10.1007/978-3-0348-8534-8_21.