Hindawi
Scientific Programming
Volume 2023, Article ID 1228614, 10 pages
https://doi.org/10.1155/2023/1228614

Research Article

Heterogeneous Hadoop Cluster-Based Image Processing Workload Distribution Framework between CPU and GPU

Najia Naz,[1] Islam Zada,[2] Abdul Haseeb Malik,[3] Muhammad Nadeem,[2] and Sikandar Ali[4]

[1]*Department of Computer Science, Bacha Khan University Charsadda, Khyber Pakhtunkhwa, Peshawar 24420, Pakistan*
[2]*Department of Computer Science, and Software Engineering, Faculty of Computing and Information Technology, International Islamic University, Islamabad 44000, Pakistan*
[3]*Department of Computer Science, University of Peshawar, Peshawar 25120, Khyber Pakhtunkhwa, Pakistan*
[4]*Department of Information Technology, The University of Haripur, Haripur 22620, Khyber Pakhtunkhwa, Pakistan*

Correspondence should be addressed to Islam Zada; islam.zada@iiu.edu.pk and Sikandar Ali; sikandar@cup.edu.cn

Received 1 July 2022; Revised 15 August 2022; Accepted 30 August 2022; Published 5 May 2023

Academic Editor: Mian Ahmad Jan

Due to the rapid development of image data and the necessity to analyze it to extract meaningful information, heterogeneous systems have gained prominence. One of the most critical aspects of distributed systems is load balancing. When it comes to the distribution of workload in a balanced manner in a cluster, some heterogeneous systems are used for image processing. When workloads are allocated in these systems, the computational power of the processors is not considered. As a result, in these heterogeneous systems for image processing applications, an uneven workload distribution issue is found. A workload distribution programming framework is presented and discussed in this paper. The proposed strategy consists of two parts. As a first step, image data is split into optimal split sizes and distributed across nodes, then the image data is distributed across CPU and GPU in a second step for processing. A heterogeneous environment is created by combining the CPU and GPU. The OpenCL Java bindings are used to set up both the CPU and GPU to run the program. To assess the performance of the suggested technique, certain tests are carried out and compared to current platforms. For image processing applications in heterogeneous clusters, the proposed workload distribution approach distributed image data efficiently. The results of the proposed solution (Hadoop + GPU) show that using an effective workload allocation mechanism in heterogeneous systems reduces average execution time while improving overall application performance.

1. Introduction

A vast quantity of digital data is created everywhere in today's technology world from many sources such as the Internet, networked cameras, mobile phones, sensors, and so on. Digital data used to be measured in Megabytes and Gigabytes, but today it is measured in terabytes and petabytes. Because 70% of digital data is unstructured, such a large volume of data needs additional storage and processing power [1]. Unstructured data includes images, which are a two-dimensional representation of pixels with variable intensity values, among other forms. In addition, images include intrinsic data-level parallelism that must be handled to extract relevant information, a process termed image processing. It is useful in a variety of applications, including medical imaging, satellite imaging, document analysis, and so on.

The constraints of limited memory capacity and data access speed arise when storing such a vast amount of data on a single processing device, impacting the performance of the application. Due to these problems and the programs' high computing demands, it was impossible to improve the performance of a single CPU or processing system. Due to the need to maximize the efficiency of running data independently of one another across several computing units in parallel, distributed and parallel architectures had to be developed.

Hadoop is a well-known and simple-to-use technology with a loosely linked design and distributed environment. The Hadoop Distributed File System (HDFS) and the MapReduce programming methodology make up Hadoop. Google's file system (GFS) is based on HDFS, which is a free and open-source version of it. In essence, it uses a map-reduce method to distribute enormous amounts of data across commodity computers, which is used by a variety of companies including Yahoo, Amazon, Facebook, and Google. The maps reduction technique provides dependable data synchronization, load balancing, as well as dynamic allocation of jobs among multiple, compute units.

The GPU architecture is configured so that the hardware has several multiprocessors. Each multiprocessor is composed of a collection of SIMD (Single Instruction Multiple Data) architecture-based 32-bit processors. Every clock cycle, a multiprocessor executes the same instruction on a group of threads known as a warp. The quantity of threads in the warp determines the size of the warp. Each streaming multiprocessor (SM) has 8 scalar thread processors (SP), and the block's threads share 16 kb of on-chip memory for communication. Programmers write two different types of code for GPU execution: kernel and host code. The kernel code is executed concurrently on the GPU. The CPU's host code manages data transmission between the GPU and main memory, as well as starting kernels on the GPU.

Massive parallel processing cores combined with GPU are provided by heterogeneous clusters, which give high speed and scalability data dissemination to faraway consumers. Because of the commodity PCs' network and GPU capability, it is adaptable. Heterogeneous computing is the utilization of heterogeneous architectures by applications. Figure 1 demonstrates how heterogeneous architectures are made up of various processor types, each of which has a distinct set of advantages and disadvantages, such as GPUs and CPUs with multiple cores. A variety of hardware can be used on these platforms, varying in power consumption and performance [2]. As a result, heterogeneous systems improve performance while lowering energy usage [3].

1.1. Problem Statement. By efficiently processing a vast amount of image data, state-of-the-art heterogeneous frameworks for image processing applications provide high performance. It is important to note that for these techniques to achieve good application performance, they need a minimum amount of support to distribute data between nodes and between CPUs and GPUs based on processing power. Nevertheless, this can be achieved by utilizing a load balancing technique that divides and distributes data between the nodes as well as between the CPU and GPU on each node, depending on their computational capabilities. As a result, an effective workload allocation policy must be implemented to improve application performance.

1.2. Aims and Objectives. The following aims and objectives are deemed to support the problem description and will take us to the desired goal:

(i) To give programmers an easy-to-use image processing framework that automatically distributes workload over a heterogeneous cluster, resulting in improved performance

(ii) Within the node, automatically partition image data between CPU and GPU

1.3. Contributions

(i) A new method for partitioning data into optimal split sizes which ensures locality for computations by ensuring that images within the given split cannot exceed the boundaries of the split

(ii) To maximize resource efficiency and minimize data transfer, splits are dispersed across nodes and within nodes according to their computational capabilities

(iii) Instead of acquiring expensive supercomputers or specialist vector machines, a commodity computer systems cluster can readily manage huge amounts of image data

The rest of the paper is organized as follows:

Section 2 describes the literature review, Section 3 proposed the framework, Section 4 is results and analysis while Section 5 contains the conclusion and future work.

2. Literature Review

It aims to provide image processing in a heterogeneous environment by combining multicore CPU and GPU methods. In a heterogeneous cluster, diverse computing capability processors are paired together to offer a programming framework that allows for an optimal split size, even/balanced job assignment, and maximum resource efficiency.

2.1. Image Processing in a Distributed Environment. In image processing applications, distributed systems have quickly become the preferred platform because of their fast-processing speed, scalability, and efficiency. It is feasible to handle large quantities of uploaded image data using distributed systems. Due to the growth of distributed systems, applications with large storage and processing needs are becoming more and more popular. Data sharing, device sharing, device connectivity, and task distribution flexibility are all advantages of distributed systems versus single processor systems. Many open-source programming paradigms, such as Spark, have been created to assist in efficient data processing in distributed environments [7],

The UC Berkeley MapReduce and Storm6 Spark [4] tools allow appealing computations such as data mining and machine learning. The storm is a distributed real-time computing system that successfully handles unbounded data streams. The MapReduce architecture is a popular choice for large data analysis because of its capacity to handle semi-structured and unstructured data in parallel [2].

In addition to image processing applications [5], face and gesture recognition [6], face tracking [7], video

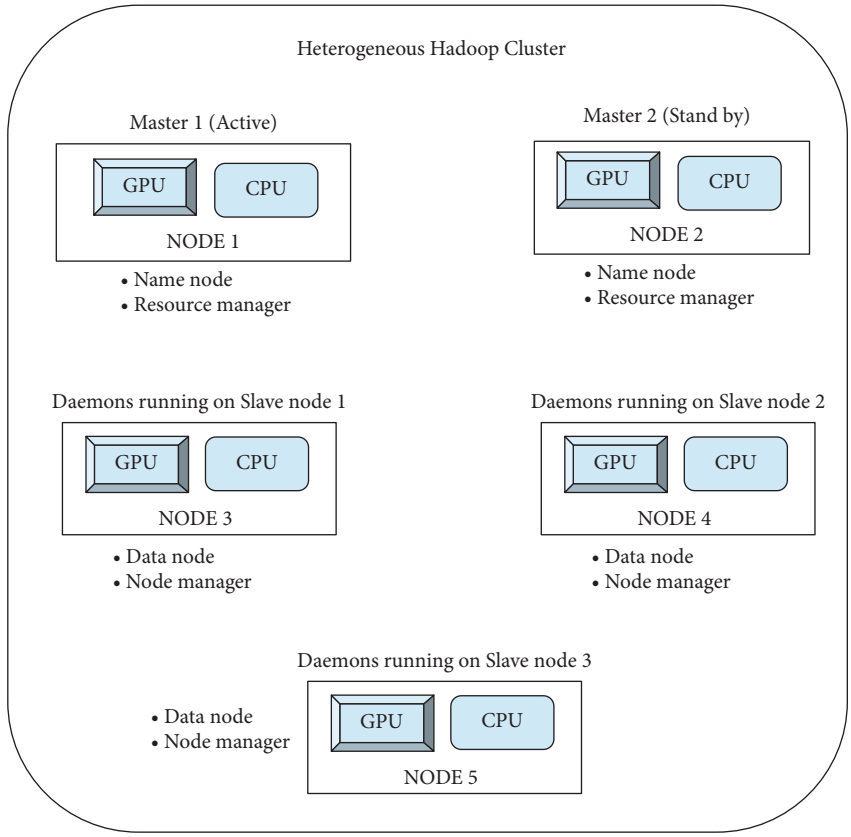

Figure 1: A generic diagram of the heterogeneous Hadoop cluster.

detection of textual words from online lecture videos [9], as well as video surveillance [10], the Apache Hadoop framework has been used in many other applications. The Hadoop MapReduce architecture was used to create a satellite image application for a Qatari environmental study center [11]. Hadoop has also been used to manage enormous amounts of image data in content-based image retrieval (CBIR) [12].

2.2. GPU-Based Image Processing. Using the OpenGL graphics library, the GPU was employed for feature extraction and tracking [13]. GPUs have been employed in applications such as Canny edge detection [14], satellite image processing [15], and medical image processing [16]. The GPU is faster than the CPU at processing the integral of photos, as shown in a study of face detection using the viola-jones method [17]. When image data is generated in real-time from satellites and must be processed fast, GPUs have been employed for image smoothing [18] and cloud removal [19] operations. It has been demonstrated that GPUs can be used in the medical field to detect brain tumor cells, complete several stages of operations, and provide fantastic performance when processing large amounts of image data rapidly [20].

2.3. Image Processing Using Heterogeneous Hadoop Clusters. Using parallel processing cores and GPUs, heterogeneous clusters assist in delivering data at high speeds and scalability to distant consumers [21]. To effectively analyze large amounts of data, the CUDA on the Hadoop framework was employed [22], which improves application performance by combining Hadoop's distributed computing capabilities with the GPU's high parallel processing structure [23, 24]. Mars is a framework that combines GPU capabilities with the Hadoop framework, and it is designed to process web documents (searches and logs) [25]. There are also three frameworks that combine GPU and Hadoop for high performance, though they were designed for specialized scientific tasks rather than image processing. These frameworks include MAPCG [26], StreamMR [27], and GPMR [28]. Hadoop Image Processing Interface (HIPI) was created to handle large amounts of tiny image data effectively; however, it does not support GPU [29].

3. Proposed Framework

In the section before, some of the challenges that result in an uneven work distribution in a diversified context were covered. The creation of a programming framework that can efficiently distribute data across nodes, and then within each node between CPUs and GPUs based on their processing capabilities, in a heterogeneous environment, is required to address these issues.

According to Figure 2, the proposed programming framework demonstrates how heterogeneous data may be distributed efficiently in a large amount of image data. The process involves two phases

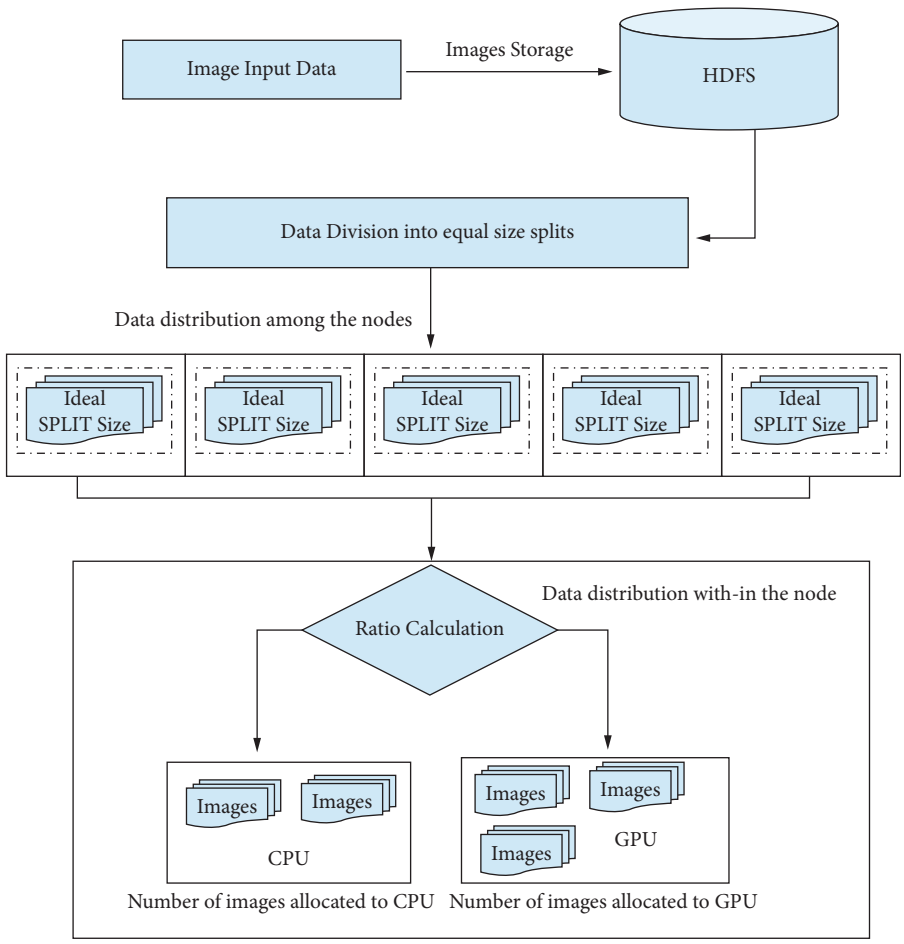

Figure 2: Proposed framework for workload distribution.

(1) Data Distribution among Nodes: the nodes in the cluster will distribute data during this phase

(2) Distribution of Workload between CPU and GPU: data is dispersed to individual nodes and then distributed within each node between the CPU and GPU in this step

This work focuses on efficiently workload distribution in a heterogeneous cluster.

3.1. Proposed Distribution of Workload among Nodes. Workloads can be divided into Hadoop workloads and cluster node workloads. The nodes process the data when it has been received. Images must be evenly distributed across cluster nodes in this study to make the best use of the cluster's processing and memory capabilities. A new distribution policy is advised for the distribution of photos. The images were chosen for the suggested technique in the same size, even if the photographs come in a variety of sizes, and each split will include one or more images. However, doing so would degrade performance because one image cannot be split into numerous parts.

Images in the proposed distribution scheme are grouped together so that every split contains many images that fit within the split size. To prevent the image from exceeding

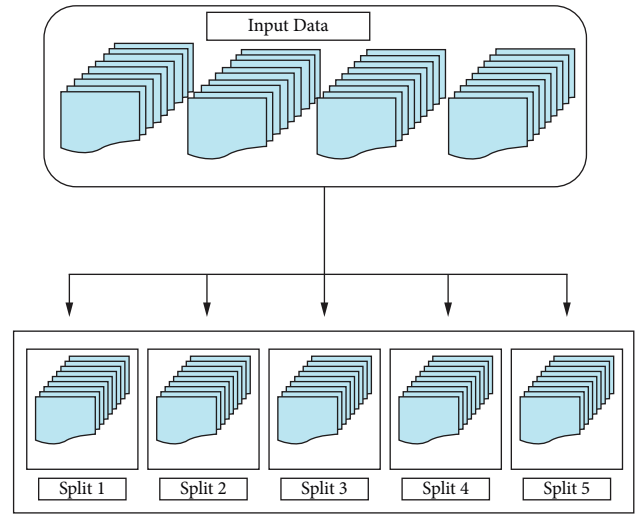

Figure 3: Partitioning data into splits.

the split boundaries, the split size is determined by the image size. Figure 3 depicts several photos that have been divided and are ready to be distributed among nodes, while Figure 4 depicts the ideal split size based on the block default size.

To address the issue of uneven splits, when photos are dispersed unevenly across splits, the split size must be

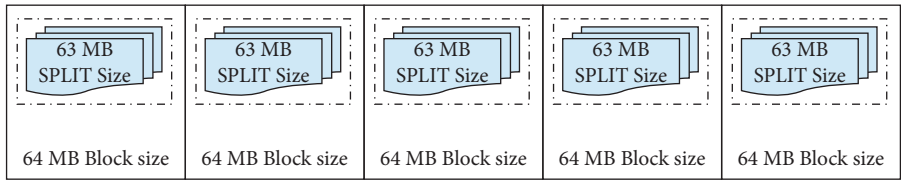

Figure 4: Partitioning image data into ideal split sizes.

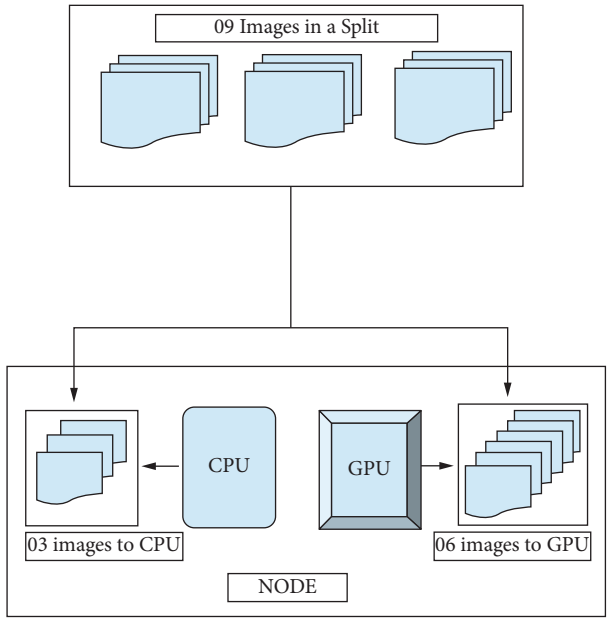

Figure 5: Data distribution within a node between CPU and GPU based on their computing capabilities.

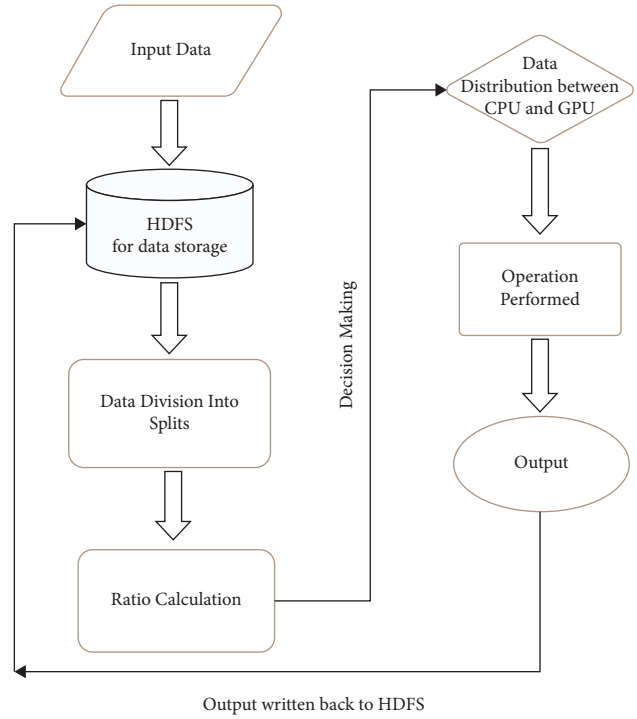

Figure 6: Basic flow chart of the proposed work.

calculated depending on the sizes of the images. When choosing an input split size in HDFS, the Input split size must be set according to the image size, then several images of the same size can be accommodated.

3.1.1. Selection of Input Split Size. The ideal input split size represents I here, the default split size represents D, the image size represents S, and no represents the maximum number of images that can be accommodated by the ideal input split. The total number of images in the dataset represents Ti, and the number of splits with an equal distribution represents Sn.

Ideal input split size = I.

Size of Image = s.

Size of Input split of default Hadoop = d.

Size of ideal split that can accommodate the number of images = no.

Divide d by s and use the floor function to disregard the fractional component to compute no. To get I, multiply s by no.

$$\begin{aligned} \text{no} &= \lfloor d/s \rfloor, \\ I &= \text{no} * s. \end{aligned} \tag{1}$$

This method computes output splits based on the size of the input image, and no image can cross two input splits.

3.2. Workload Distribution between CPUs and GPUs within a Node. A heterogeneous cluster has coprocessor GPUs, which are faster than CPUs. As a result, job assignments must be based on the processors' computational capabilities for best performance. A new effective workload allocation technique in a heterogeneous Hadoop cluster is suggested to achieve excellent performance for image processing applications.

This load balancing method is visually explained in Figure 5. This phase involves splitting up each map task into fixed-size images. The map function is called every time a split occurs, which takes an input pair of a key and value. By using the map function, each image in the split is read. By applying the proposed approach, the map function checks ratios for all images within the split and assigns images based on their computational capabilities to the CPU and GPU. A sample image is executed simultaneously on CPU and GPU to determine the execution time of both CPU and GPU on an algorithm. An image ratio is created by comparing the execution times of each processor and comparing how many

Table 1: The CPU-GPU class.

Class Public class CPU-GPU Methods	
Public static	setSplitSize() Each division size is determined using this method.
Public static	GPUDetect (float pixels, int width, int height). Processing of GPU is determined using this method.
Public void	CalculateRatio (BufferedImage bi, int width, int height). The ratio between GPU and CPU can be calculated using this method.
Public static void	Concat (BufferedImage bi, int *c*). Using this method, the images are concatenated into a single bundle.
Public static buffered image[]	Split (int *c*, int width, int height). This method divides data into equal splits.

Table 2: The ImageMapper class.

Class Public static class ImageMapper extends mapper < bundle header, image bundle, text intwritable> Methods	
Protected void	Map (bundle header key, image bundle value, context), the mapper that consumes the split and processes it is determined by this method.

Table 3: Variations in execution time (milliseconds) for CPU and GPU for edge detection.

Image resolution	CPU	GPU
1024 × 768	81	42
1600 × 900	110	50
1920 × 1080	236	61
2560 × 1440	382	69

Table 4: Average execution time (milliseconds) for CPU and GPU for edge detection.

Image resolution	CPU	GPU
1024 × 768	81	42
1600 × 900	110	50
1920 × 1080	236	61
2560 × 1440	382	69

images the GPU and CPU can process simultaneously. The Efficient Workload Distribution Implementation details process and basic flow chart of the proposed work are depicted in Figure 6 as below.

3.3. Efficient Workload Distribution Implementation. This section describes the implementation of the proposed approach.

Splits are formed by defining three variables in the setSplitSize() method of the class CPU-GPU given in Table 1. Using HDFS block size as a reference for split size calculation, one can specify the following formats for file information: (i) file size, (ii) default split size, and (iii) optimal split size. The default split size is substituted by the optimal determined split size thanks to the conf setting in run ().

The sample image, image width, and image height are the three inputs for the CalculateRatio() function, which calculates the ratio. The Concate() method is used to combine the images based on the ratio that will be sent to the GPU, the image class detail is given in Table 2. A method on the GPU called GPUdetect() is used to generate an edge detection operation based on the width, height, and the total number of pixels in an image. Once the photos have been processed, they are separated once more to separate out each individual image, which is then saved in an output file. The (key, value) pair is created by using two parameters in the

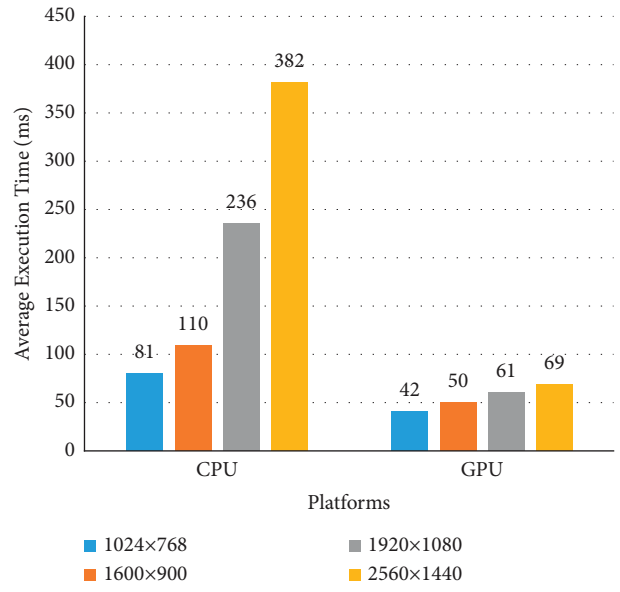

Figure 7: Average execution time (milliseconds) of CPU and GPU using a single image.

map () function of the ImageMapper class to specify details about the packaged photos and their actual data, the CPU-GPU, and the ImageMapper Class are shown in Table 1 and Table 2, respectively, while the variations in execution time

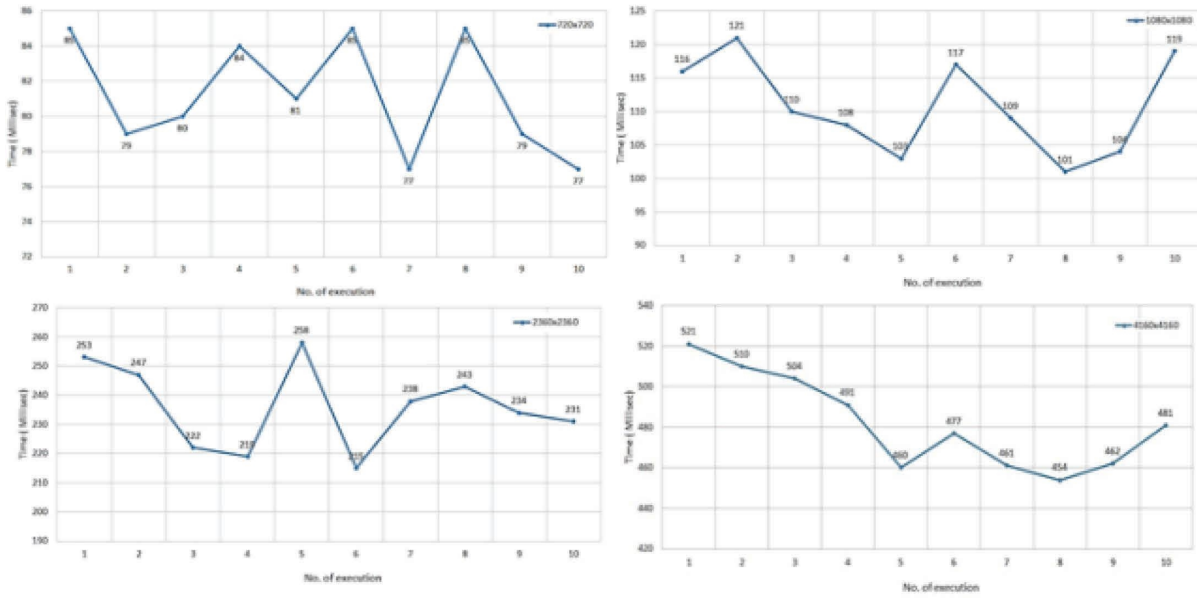

Figure 8: Variations in execution time of a single image on CPU.

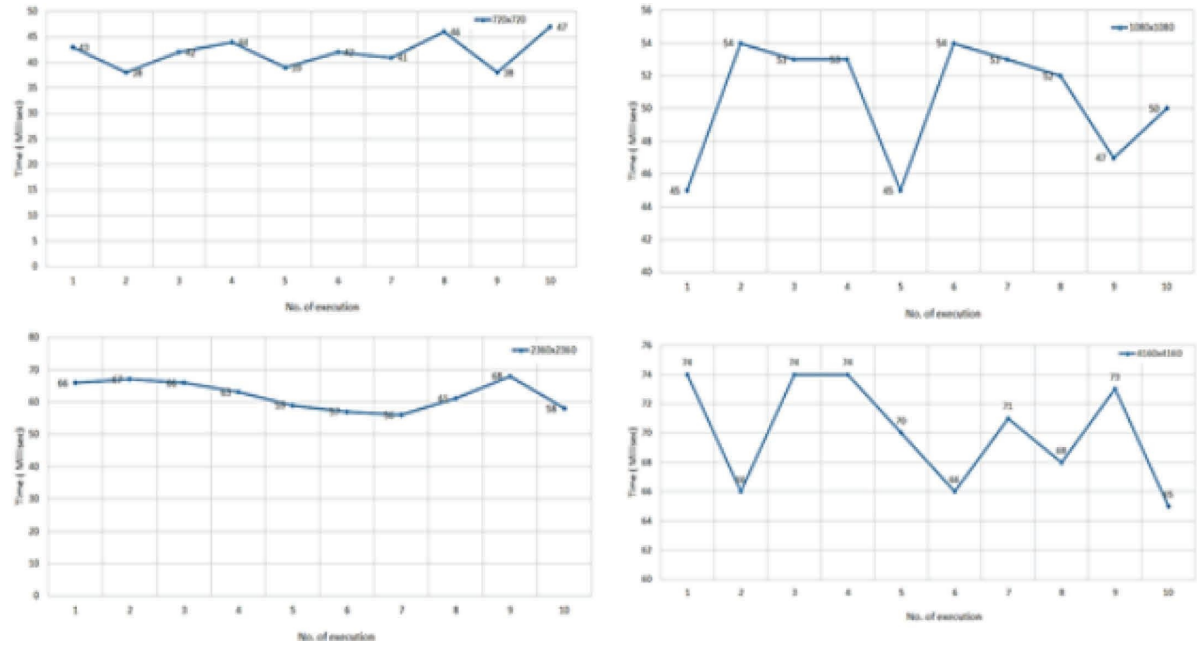

Figure 9: Variations in execution time of a single image on GPU.

(milliseconds) for CPU and GPU for Edge Detection details are given in Table 3.

4. Results and Analysis

In this section, all the results of the conducted experiments are shown and analyzed in detail.

4.1. Comparative Analysis of CPU and GPU. The results in Table 4 and a bar chart diagram in Figure 7 show the execution times for four different resolution images on a CPU and GPU, respectively, in milliseconds. In the experiment, it was discovered that when the image size rose, both the CPU and GPU execution times also increased; however, the GPU execution times increased more slowly than the exponentially rising CPU execution times. The variations in execution times for the CPU and GPU are seen in Figures 8 and 9, respectively. Table 5 demonstrates that as image size increases, the variance in GPU execution time is less than it is for CPU execution [27]. By demonstrating the effect of increasing image size on calculation time, this

Table 5: Average execution time (milliseconds) for proposed approach (Hadoop + GPU) vs existing platforms.

Resolution of image	HIPI	HIPI + GPU	Hadoop + GPU	Proposed approach (Hadoop + GPU)
1024 × 768	221	184	149	97
1600 × 900	260	225	192	133
1920 × 1080	442	371	269	172
2560 × 1440	686	489	342	191

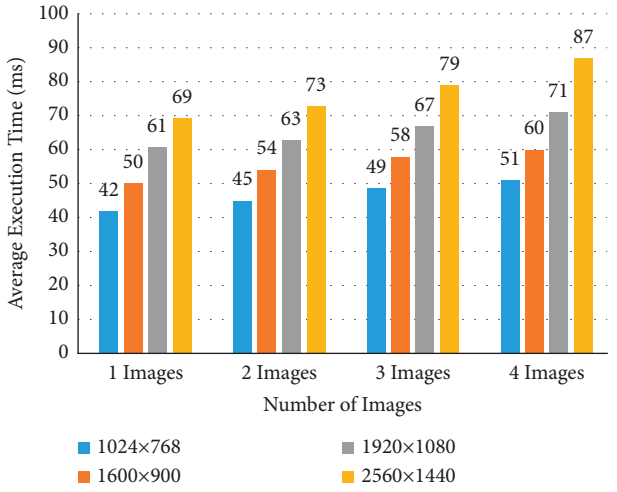

Figure 10: Average execution time (milliseconds) of an increasing number of images on a GPU.

Table 6: Average execution time (milliseconds) while scaling the number of images on a GPU.

Resolution of image	No. of images			
	I	II	III	IV
1024 × 768	42	45	49	51
1600 × 900	50	54	58	60
1920 × 1080	61	63	67	71
2560 × 1440	69	74	80	88

Table 7: Increase in execution time of an increasing number of images on a GPU.

Resolution of image	No. of images			
	I	II	III	IV
1024 × 768	3	3	3	3
1600 × 900	4	4	3	3
1920 × 1080	4	4	3	3
2560 × 1440	4	4	3	3

experiment illustrates how the GPU can be used to improve application performance by considering the overhead of loading data from CPU memory to GPU.

4.2. Varying/Increasing Number of Images on GPU. The experiment in Figure 10 demonstrates how the integration of images affects the performance of the program on a GPU. The *x*-axis represents the number of images integrated with each other, while the *y*-axis shows the execution duration in milliseconds. In Tables 6 and 7, the execution time and standard deviation for this experiment are presented by integrating four images of different resolutions (1, 2, 3, and 4 images). Based on experiments, a single image with a resolution of 1024 × 768 is processed independently in 42 milliseconds, but four images with the same resolution are combined in 51 milliseconds. According to the experimental results, the variation in execution time is directly attributed to the number of exchanges between the CPU and GPU for the transfer of each image and the subsequent writing of the result to the CPU. So, these data shifting and loading processes are executed for each individual image, but when it comes to image integration, all the integrated photos are handled in one cycle.

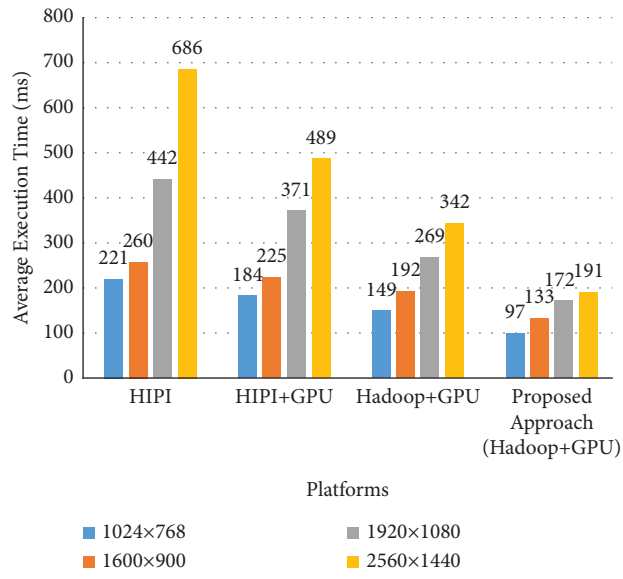

Figure 11: Average execution time (milliseconds) for proposed approach (Hadoop + GPU) vs existing platforms.

4.3. Comparative Analysis of Performance on Different Platforms. Figure 11 shows the average execution time for the suggested solution (Hadoop + GPU) versus current approaches. For all resolutions of photos displayed in Table 5, the suggested framework takes much less time to execute than other current frameworks (HIPI, HIPI + GPU, Hadoop + GPU). The results of the proposed solution (Hadoop + GPU) show that using an effective workload allocation mechanism in heterogeneous systems reduces average execution time while improving overall application performance. Table 8 demonstrates the significant execution time variance between the suggested method (Hadoop + GPU) and the existing platforms (HIPI, HIPI + GPU, and Hadoop + GPU). The findings of the

Table 8: Time comparison between proposed approach (Hadoop + GPU) and existing platforms (milliseconds).

Image resolution	HIPI	HIPI + GPU	Hadoop + GPU	Proposed approach (Hadoop + GPU)
1024 × 768	7	10	7	3
1600 × 900	11	8	7	5
1920 × 1080	8	15	11	5
2560 × 1440	16	12	8	5

experiment show that the main factor that significantly improves application performance and completely utilizes the resources available is the suggested efficient workload allocation policy.

5. Conclusion and Future Work

The goal of this paper is to introduce a novel programming framework utilizing the Hadoop MapReduce programming model and graphics processing units (GPUs). The suggested technique offers these advantages over existing approaches for image processing applications on heterogeneous clusters. A new method for partitioning data into optimal split sizes ensures locality for computations by ensuring that images within the given split cannot exceed the boundaries of the split, to maximize resource efficiency and minimize data transfer, splits are dispersed across nodes and within nodes according to their computational capabilities and instead of acquiring expensive supercomputers or specialist vector machines, a commodity computer systems cluster can readily manage huge amounts of image data.

As a result, future work will focus on developing a split size that can easily support varied image sizes and divide them among nodes as well as inside each node between CPU and GPU. Real-time image processing refers to the completion of certain activities in a set period. In certain image processing applications, a stream of images is created that must be processed within a certain amount of time to ensure that an image does not miss its deadline. The suggested technique in heterogeneous systems will be used to process this stream of images within the stated timeframe in the future study.

Data Availability

The data used to support the findings of this study are included within the article.

Conflicts of Interest

All the authors declare that they have no conflicts of interest.

Authors' Contributions

All authors contributed equally to this study.

Acknowledgments

This study is carried out by the collaboration of University of Peshawar, International Islamic University Islamabad, and University of Haripur.

References

[1] P. Vijay and B. Keshwani, "Emergence of big data with Hadoop: a review," *IOSR Journal of Engineering*, vol. 6, no. 3, pp. 50–54, 2016.

[2] S. A. Mahmoudi, E. Ozkan, P. Manneback, S. Tosun, E. Jeannot, and J. Zilinskas, "Taking advantage of heterogeneous platforms in image and video processing," in *High-Performance Computing on Complex Environments*John Wiley & Sons, Hoboken, NJ, USA, 2014.

[3] R. Kumar, D. M. Tullsen, N. P. Jouppi, and P. Ranganathan, "Heterogeneous chip multiprocessors," *Computer*, vol. 38, no. 11, pp. 32–38, Nov. 2005.

[4] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: cluster computing with working set," in *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing*, California, Berkeley, 2010.

[5] A. Sozykin and T. Epanchintsev, "MIPr-a framework for distributed imge processing using Hadoop," in *Proceedings of the 9th International Conference on Application of Information and Communication Technologies (AICT)*, pp. 35–39, Rostov on Don, Russia, October 2015.

[6] H. Tan and L. Chen, "An approach for fast and parallel video processing on Apache Hadoop clusters," in *Proceedings of the IEEE International Conference on Multimedia and Expo (ICME)*, pp. 1–6, Chengdu, China, July 2014.

[7] C. Ryu, D. Lee, M. Jang, C. Kim, and E. Soe, "Extensible video processing framework in Apache Hadoop," in *Proceedings of the IEEE 5th International Conference on Cloud Computing Technology and Science(CloudCom)*, pp. 305–310, Bristol, UK, December 2013.

[8] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[9] M. Husain, S. M. Meena, A. K. Sabarad, H. Hebballi, S. M. Nagaralli, and S. Shetty, "Counting occurrences of textual words in lecture video frames using Apache Hadoop framework," in *Proceedings of the IEEE International Advance Computing Conference (IACC)*, pp. 1144–1147, IEEE, Banglore, India, June 2015.

[10] A. A. Aradhye, H. S. Inamdar, G. S. Patil, and Y. B. Jagdale, "Object detection avenue for video surveillance using Hadoop," *International Research Journal of Engineering and Technology(IRJET)*, vol. 3, no. 5, pp. 497–499, 2016.

[11] M. H. Almeer, "Cloud Hadoop map reduce for remote sensing image analysis," *Journal of Emerging Trends in Computing and Information Sciences*, vol. 3, no. 4, pp. 637–644, 2012.

[12] S. P. Dravyakar, S. B. Mane, and P. K. Sinha, "Private content based multimedia information retrieval using map-reduce," *International Journal of Computer Science Engineering and Technology*, vol. 4, no. 4, pp. 125–128, 2014.

[13] S. N. Sinha, J.-M. Frahm, M. Pollefeys, and Y. Genc, "GPU-Based video feature tracking and matching," University of North Carolina, Chapel Hill, NC, USA, TR 06-012, 2006.

[14] Z. Yang, Y. Zhu, and Y. Pu, "Parallel image processing based on CUDA," *International Conference on Computer Science and Software Engineering*, vol. 3, pp. 198–201, 2008.

[15] H. M. Patel, K. Panchal, P. Chauhan, and M. B. Potdar, "Satellite image processing using CUDA and Hadoop architecture," *International Journal of Scientific Engineering and Research*, vol. 7, no. 5, 2016.

[16] L. Shi, W. Liu, H. Zhang, Y. Xie, and D. Wang, "A Survey of GPU-based medical image computing techniques," *Quantitative Imaging in Medicine and Surgery*, vol. 2, no. 3, pp. 188–206, 2012.

[17] V. Jain and D. Patel, "A GPU based implementation of robust face detection system," in *Proceedings of the 2016 International Conference on Computational Science*, pp. 156–163, Istanbul, Turkey, June 2016.

[18] J. Ke, A. Sowmya, Y. Guo, T. Bednarz, and M. Buckley, "Efficient GPU computing framework of cloud filtering in remotely sensed image processing," in *Proceedings of the 2016 International Conference on Digital Image Computing: Techniques and Applications (DICTA)*, pp. 1–8, Gold Coast, QLD, Australia, November 2016.

[19] Y. Guo, F. Li, P. Caccetta, D. Devereux, and M. Berman, "Cloud filtering for Landsat TM satellite images using multiple temporal mosaicing," in *Proceedings of the IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, pp. 7240–7243, Beijing, China, July 2016.

[20] W. Phusomsai, C. So-In, C. Phaudphut, C. Thammasakorn, and W. Punjaruk, "Brain tumor cell recognition schemes using image processing with parallel ELM classifications on GPU," in *Proceedings of the 13th International Joint Conference on Computer Science and Software Engineering (JCSSE)*, pp. 1–6, Khon Kaen, Thailand, July 2016.

[21] Z. Wang, P. Lv, and C. Zheng, "CUDA on Hadoop: a mixed computing framework for massive data processing," in *Foundations and Practical Applications of Cognitive Systems and Information Processing*Springer, Berlin Germany, 2014.

[22] W. Fang, Q. Lou, N. K. Govendaraju, and T. Wang, "Mars: a MapReduce framework on graphics processors," in *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, pp. 260–269, ACM, Toronto, Ontario, Canada, October 2008.

[23] H. Chuntao, C. Dehao, and C. Wenguang, "MAPCG: writing parallel program portable between CPU and GPU," in *Proceedings of the 19th International Conference on Parallel Architectures and Compilation Techniques*, pp. 217–226, Vienna, Austria, September 2010.

[24] M. Elteir, H. Lin, and W. C. Feng, "StreamMR: an optimized MapReduce framework for AMD GPU," in *Proceedings of the 17th International IEEE Conference on Parallel and Distributed Systems*, pp. 364–371, December 2011.

[25] J. A. Stuart and J. D. Owens, "Multi- GPU MapReduce on GPU cluster," in *Proceedings of the 2011 IEEE International Parallel & Distributed Processing Symposium (IPDPS)*, pp. 1068–1079, Anchorage, AK, USA, May 2011.

[26] C. Sweeney, L. Liu, S. Arietta, and J. Lawrence, "HIPI: A Hadoop Image Processing Interface for Image-Based MapReduce Tasks," Master Thesis, University of Virginia, Charlottesville, Va, USA, 2011.

[27] N. Naz, A. Haseeb Malik, A. B. Khurshid et al., "Efficient processing of image processing applications on CPU/GPU," *Mathematical Problems in Engineering*, vol. 2020, Article ID 4839876, 14 pages, 2020.

[28] N. Naz, A. Haseeb Malik, A. B. Khurshid et al., "Efficient processing of image processing applications on CPU/GPU," *Mathematical Problems in Engineering*, vol. 2020, Article ID 4839876, 14 pages, 2020.

[29] X. Zhang, Z. Tang, X. Zhang, and K. Li, "Co-concurrency mechanism for multi-GPUs in distributed heterogeneous environments," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 12, pp. 4935–4947, Article ID 4839876, 2022.