Parallel Efficiency in Image Processing: A Detailed Examination of CUDA and MPI Configurations for Optimizing Image-to-ASCII Character Conversion

Terry Lin[*] , Oscar Li[*]
[*]Rensselaer Polytechnic Institute

ABSTRACT
Image processing is the use of a digital computer to process digital images through an algorithm. It allows a much wider range of algorithms to be applied to the input data and can avoid problems such as the build-up of noise and distortion during processing. Applications range from medical imaging that assists in accurate diagnoses, encrypting messages essential to security, and preservation/reconstruction of lost or damaged cultural artifacts. As the need for faster computations grows, parallel computing methods such as CUDA and MPI have become increasingly popular. These technologies allow the efficient handling of large-scale image datasets by distributing the workload across multiple processors and GPUs. Specifically, our approach uses MPI to distribute images across different processes, while CUDA is utilized to rapidly process each image into ASCII format. Our research also leverages OpenCV for image processing and performing computer vision tasks; we can optimize the conversion of images to ASCII characters, merging the realms of visual data representation and artistic expression. This approach not only maximizes computational efficiency but also significantly reduces the time required for processing large batches of images, which can be used as a scalable solution for many real life applications.

I. INTRODUCTION

In the digital age, image processing has emerged as a cornerstone technology with profound impacts across various industries. This technology not only enhances visual data for better human interpretation but also transforms it for advanced computational tasks. Examples include medical diagnostics [1], security systems [2], and cultural preservation [3]. Medicaling imaging systems include x-rays and y-rays which detect different physical signals from patients and produce images. This is crucial to avoid causing life threatening damages to the body and cells [1]. In security systems, image stitching and image steganography security are used to send encrypted data over networks. The images are encrypted using the AES algorithm, then the cipher text is embedded into the encrypted image, and finally the steganography is performed on the output image where the images are camouflaged by another image using least significant bit replacement [2]. High-resolution images can be analyzed to detect and measure deterioration, guide restoration processes, and create digital archives for long-term preservation. Techniques such as 3D scanning and photogrammetry enable accurate reproductions of physical objects, ensuring access to fragile cultural treasures without risking damage to the original items. Additionally, digital imagery can help in reconstructing lost or damaged heritage sites through virtual reconstruction [3].

This paper explores the integration of CUDA, MPI, and OpenCV to enhance the efficiency of converting images to ASCII art. The use of CUDA enables the leveraging of GPUs to accelerate the processing of large image files by parallelizing the computations. Meanwhile, MPI is used to facilitate communication between processes operating on different nodes in a distributed computing environment. MPI/IO further extends these capabilities by optimizing input/output operations, enabling efficient data management.

The primary aims of utilizing these advanced computational techniques are to reduce processing time, increase scalability, and improve the output quality of ASCII art transformations. By combining the power of CPU and GPU, this approach will have significant performance enhancements over traditional image processing methods.

II. BACKGROUND AND RELATED WORKS

An image mosaic is an assembly of a large number of small tiles. When viewed as a whole, the small miniscule tiles work together to form a single larger image. Each tile approximates a small block of pixels. Besides having aesthetic value, mosaics have been investigated in the context of copyrighted material protection and hiding secret data [8]. This technique would later evolve and be incorporated in cybersecurity techniques where data is encrypted through the network using images. ASCII art is a similar artistic technique utilizing printable characters. It originates from times when printers had limited printing capabilities and transferring or recreating images over computer networks was computationally taxing due to bandwidth constraints [8]. Representing detailed images with limited shapes and placement of characters makes ASCII art a significant challenge. However, by scaling our ASCII art with more pixels, i.e. more printable characters, we can overcome this obstacle which would in turn require more computational power [9]. Image processing has greatly benefited from the development of open-source libraries developed by the community. One example includes IMAGIC-5, an image processing software widely used in biological imaging. IMAGIC-5 is widely used for image alignment, classification, and 3D reconstruction [4]. Another example, scikit-image, is a library written for the Python programming language that offers utilities for image processing and computer vision tasks such as filtering, morphology, segmentation, and transformation [5]. Although these libraries are popular and efficient, they have their respective shortcomings. IMAGIC-5 is mainly used for biological imaging, which may not generalize well to other types of image data or applications outside of its core research focus. Scikit-image is written primarily in Python which conflicts with parallelizing with CUDA and MPI which are C based programming languages. On the other hand, OpenCV is immensely popular for its broad support of computer vision applications and its compatibility across various programming languages and platforms [6]. What makes OpenCV perfect for our study is that although it faces performance challenges, particularly when it comes to utilizing GPU resources efficiently through OpenCL, studies indicate that the performance of OpenCV's object tracking algorithms, such as object detection and optical flow, can be significantly improved by optimizing their implementation to better utilize GPU capabilities. These optimizations can lead to performance improvements of up to 86% for object detection and 10% for optical flow [7]. We believe by leveraging CUDA and its emphasis on parallelized GPU computations, we are able to further optimize OpenCV's applications.

III. METHODOLOGY

The process of converting an image to ASCII art requires mapping the brightness of each pixel in an image to a character from a predefined set. In our research, we have a predefined set as follows:

" .'`^\",:;Il!i><~+_-?][}{1)(|\\//tfjrxnuvczXYUJCLQ0OZmwqpbdkhao*#MWM&8%B@$"

We will begin by assigning an RGB value to each pixel in the image. These RGB values will be transformed into their grayscale equivalents, using a weighting system that prioritizes green, followed by red and blue. Each grayscale value will then correspond to a specific character from a predefined set, arranged to represent various brightness levels from dark to light. This conversion process is depicted in Figure 1.

```
Function ConvertImageToASCII (image):
  Initialize ascii_art as an empty string
  Initialize ascii_image as a matrix

  For each pixel i in the image rows:
    For each pixel j in the image columns:
      Get the RGB values from the pixel (i, j)
      Compute the grayscale value using the formula:
        gray = 0.299 * Red + 0.587 * Green + 0.114 * Blue

      Map the grayscale value to an ASCII character index:
        index = (gray * (number of characters in CHARSET - 1)) / 255
        asciiChar = CHARSET[index]

        If colored flag is true:
          Use original pixel RGB values
        Else:
          Use white color

      Draw the ASCII character on ascii_image at position corresponding to (i, j)
      Append asciiChar to ascii_art string

    Append a newline to ascii_art after each row

  Return ascii_art and ascii_image
```

Fig 1. pseudocode for converting image to ASCII

To enhance the efficiency and scalability of our program, we will utilize MPI to divide the image into 20 segments, based on the MPI rank as illustrated in Figure 2.

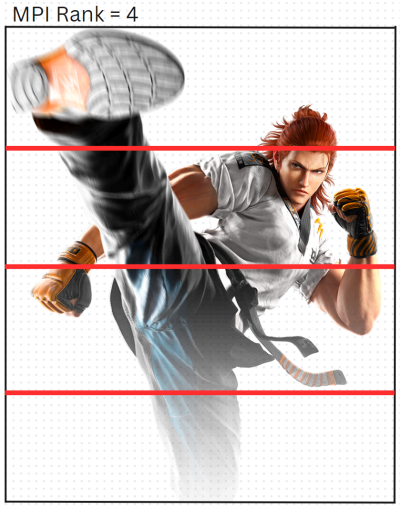

Fig 2. image of Hwoarang, split up into 4 fragments where each fragment is its respective MPI process. All 4 fragments will be executed in parallel, simultaneously converting their respective segments into ASCII.

Subsequently, within each segment, we will employ CUDA and the system's GPU hardware to initiate threads corresponding to the number of pixels in each segment, as demonstrated in Figure 3. Each thread will handle a designated group of pixels, with all threads operating in parallel to convert the pixels into ASCII characters.

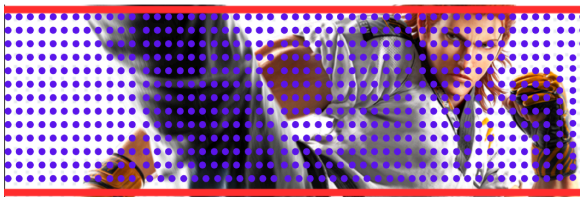

Fig 3. zoomed up image of Hwoarang. Each purple dot represents one cuda thread. Each thread will be executed in parallel, mapping each pixel to an ASCII value.

Finally, once CUDA has completed its execution, the MPI segments will be unified, reconstructing the processed segments back into a cohesive single image.

In our upcoming testing phase, we plan to execute a series of performance evaluations focusing on various aspects of our program. First, we will examine the effects of altering the number of MPI ranks, changing the number of pictures processed in parallel. Next, we aim to modify the number of CUDA thread computations by increasing the scale or width of each picture. This will also result in a clearer, more detailed representation of the image with ASCII. We will also conduct tests for strong scaling, where we keep the problem size constant while incrementally increasing the count of CPUs and GPUs. Additionally, we will explore weak scaling scenarios, where both the problem size and the CPU/GPU count are scaled up. These tests are critical for assessing the efficiency and scalability of our image processing algorithm.

IV. RESULTS

Our program generates txt, PNG, and colorized PNG representations of ASCII art. Fig 4 and Fig 5 exhibit a very pixelated png, characterized by large, blocky regions of color and minimal detail spanning only 50 characters per line. This represents the baseline or starting point of our processing. On the other hand, Fig 6 and Fig 7 showcase a much more refined and detailed version of the same png, processed through our algorithm to highlight the enhancements in definition and clarity spanning 500 characters per line. This contrast showcases the transformative effects of our algorithm.

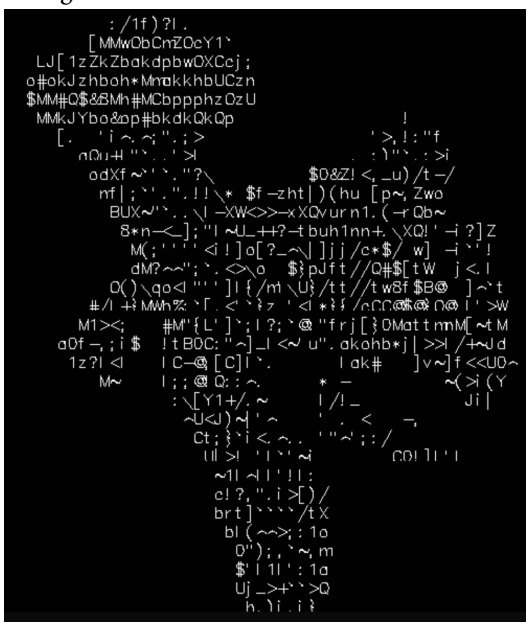

Fig 4. Black and white representation of Hwoarang 50 characters per line

Fig 5. Colored representation of Hwoarang 50 characters per line

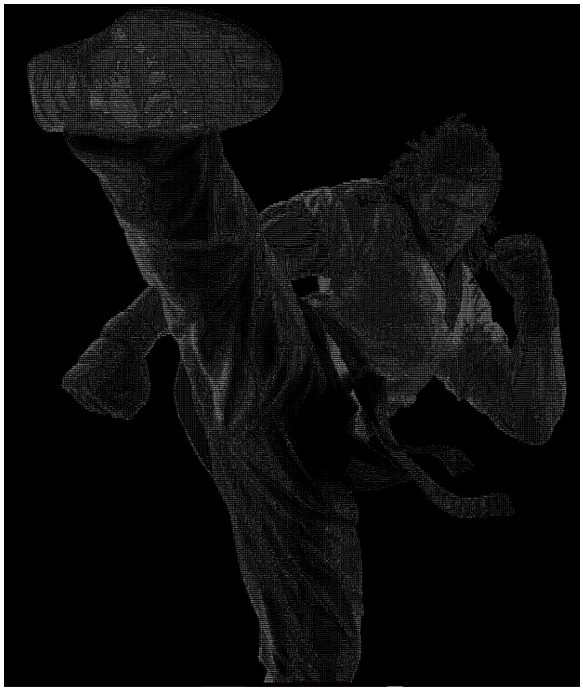

Fig 6. Black and white representation ofHwoarang 500 characters per line

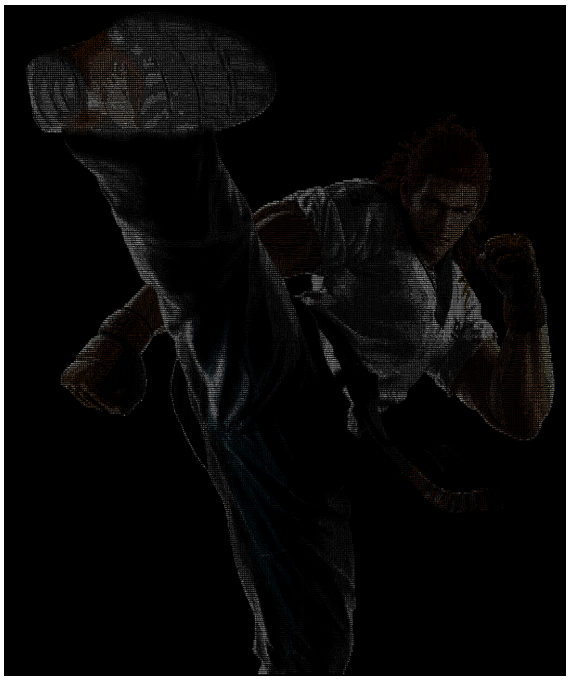

Fig 7. Colored representation ofHwoarang 500 characters per line

We first conducted strong scaling testing where we kept the CUDA Threads and Image width constant, while incrementing the MPI Ranks and their respective GPUs and nodes as shown in Fig 8, 9, and 10. Note that the width represents the number of characters per line.

STRONG SCALING						
MPI Ranks	GPUs	node	CUDA Threads	Image Width	Execution Time (seconds)	MPI I/O Time (Cycles)
1	1	1	256	10000	29.9279	3182222
2	2	1	256	10000	19.0353	5144897
3	3	1	256	10000	13.0422	866780798
4	4	1	256	10000	10.3893	877143805
5	5	1	256	10000	9.31061	867299481
6	6	1	256	10000	8.10278	745049498
12	12	2	256	10000	5.03791	54134210

Fig 8. Strong scaling data

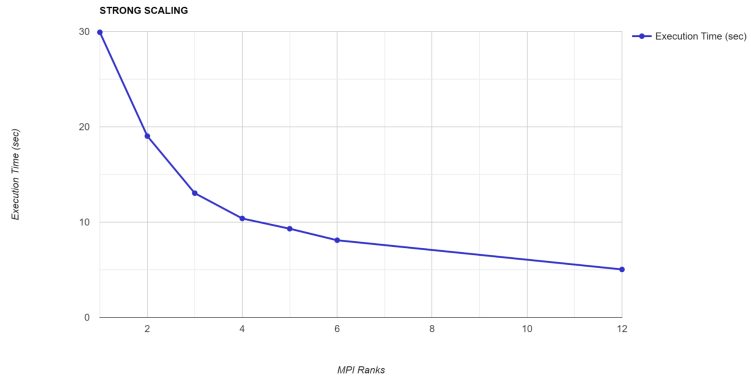

Fig 9. Graph for the execution time

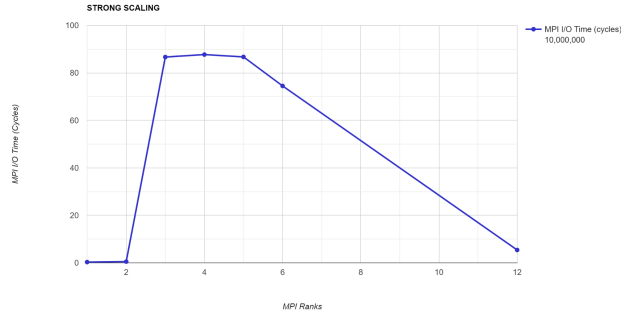

Fig 10. Graph for MPI I/O Times. Note the cycles are devided by 10,000,000 for simplified visual representation.

Firstly, there is a clear trend indicating that as the number of MPI ranks and GPUs increases, the execution time for processing decreases. Since the problem size remains constant, adding more resources leads to shorter execution times. From the data, the execution time decreases from about 29.93 seconds with a single MPI rank and GPU to approximately 8.10 seconds with six MPI ranks and GPUs. This yields a reduction in execution time by over 70%. The ideal expectation in a perfectly scalable system would be a linear decrease in execution time with the addition of resources. However, we observe that the reduction in execution time is less pronounced with each additional MPI rank and GPU. This is typically due to the overhead associated with parallel computing, such as the time taken for inter-process communication and synchronization between the ranks [10]. Additionally, there's a general trend where MPI I/O time increases as more MPI ranks are utilized. This rise in I/O time could be a result of increased communication overhead. The data also shows an abnormal spike in I/O time at 3 and 4 MPI ranks, which may be indicative of inefficiencies or bottlenecks in I/O processing.

We then conducted weak scaling testing where we kept the CUDA Threads, while incrementing the MPI Ranks and their respective GPUs, nodes and Image widths as shown in Fig 11, 12, and 13

WEAK SCALING						
MPI Ranks	GPUs	node	CUDA Threads	Image Width	Execution Time (seconds)	MPI I/O Time (Cycles)
1	1	1	256	1000	1.05725	2774935
2	2	1	256	2000	1.99135	4234972
3	3	1	256	3000	1.74741	69446846
4	4	1	256	4000	2.44693	164309326
5	5	1	256	5000	3.24222	249184502
6	6	1	256	6000	3.86362	328768807
12	12	2	256	12000	6.34882	146995872

Fig 11. Weak scaling data

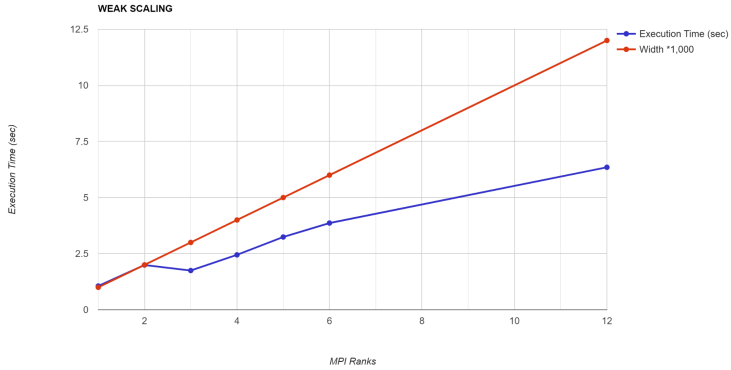

Fig 12. Graph for the execution time. Note the red line represents the width times 1,000 and the blue line represents the execution time in seconds

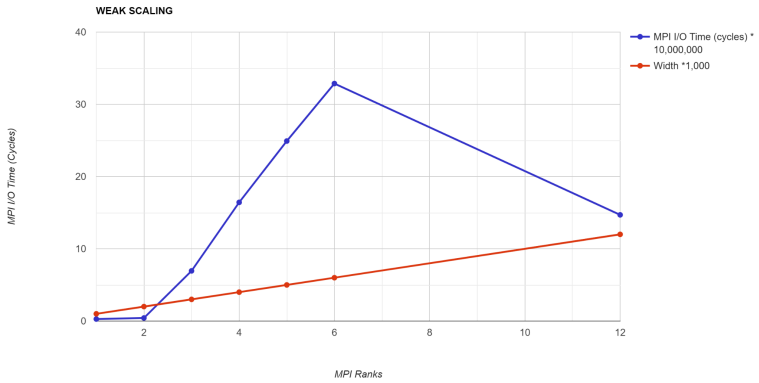

Fig 13. Graph for the MPI I/O cycles. Note the red line represents the width * 1,000 and the blue line represents the execution time in seconds divided by 10,000,000 for simplified visual representation.

We noticed an increase in execution time as the number of MPI ranks and GPUs grew, from 1.05725 seconds with a 1000-pixel width image on a single GPU to 6.34882 seconds with a 12000-pixel width on 12 GPUs. Moreover, the execution time's pattern does not follow a consistent upward trend. There is an instance where the execution time decreases when moving from two to three MPI ranks, despite the workload increasing. The MPI I/O time exhibits a substantial increase with larger image sizes and more ranks. Significant jumps in I/O time occur as the number of MPI ranks rises, which may mean that I/O is a major scaling bottleneck. It's evident that the system's weak scaling performance is reduced by overheads as image width increases. Although computational power is scaled up to match the increased workload, the execution time remains

relatively flat as would be expected in an ideal weak scaling scenario. Finally, the scalability of the system, while beneficial up to a point, shows signs of overheads that limit performance gains. These overheads may stem from latency in communication between nodes, process synchronization challenges, and contention for shared resources like memory and I/O bandwidth. The optimal scalability point appears to be below 12 MPI ranks.

V. CONCLUSION

This study has demonstrated the effective application of CUDA, MPI, and OpenCV in significantly enhancing the process of converting images to ASCII art. Our methodology not only reduces processing times but also improves scalability and output quality. Through our experimental analysis, we established that while our system is capable of strong and weak scaling, there are inherent challenges associated with parallel computing such as increased I/O times and communication overhead. These findings suggest that while the integration of these technologies offers substantial benefits, there is a critical need for ongoing optimization, particularly in managing I/O operations and minimizing latency in data communication. Future research should focus on refining these computational processes and further harnessing the capabilities of GPUs to enhance parallel processing performance. Not only does our research allow enthusiasts to efficiently process images to ASCII, our application can be used in various industries such as healthcare, security, and digital humanities. We also intend to make our research available as an open-source application, allowing other developers to build upon and expand its current capabilities. By advancing these technologies, we can continue to push the boundaries of image processing, art, and pave the way for artistic innovative applications.

VI. CONTRIBUTIONS

This research was collaboratively conducted by two junior students from Rensselaer Polytechnic Institute, both pursuing their Bachelor's degrees in Computer Science.

Terry Lin: Terry's primary responsibility was the development of the core code of our image processing application. Terry also played a crucial role in integrating CUDA, MPI, OpenCV technologies, and optimization of the image-to-ASCII conversion algorithm. Additionally, he provided assistance in data gathering, research, and proofreading the final paper.

Oscar Li: Oscar focused on managing the debugging, testing, and data gathering. Oscar was also primarily responsible for conducting the research that supported our hypotheses and writing the draft of this paper. He reviewed related works and integration of our findings into the broader academic context.

REFERENCES

[1] Dougherty, G. (2009). Digital image processing for medical applications. Cambridge University Press.

[2] Kapur, J. (2013). Security using image processing. International Journal of Managing Information Technology (IJMIT) Vol, 5.

[3] Mudge, M., Malzbender, T., Chalmers, A., Scopigno, R., Davis, J., Wang, O., ... & Barbosa, J. (2008). Image-Based Empirical Information Acquisition, Scientific Reliability, and Long-Term Digital Preservation for the Natural Sciences and Cultural Heritage. Eurographics (Tutorials), 2(4).

[4] van Heel, M., Harauz, G., Orlova, E. V., Schmidt, R., & Schatz, M. (1996). A new generation of the IMAGIC image processing system. Journal of structural biology, 116(1), 17-24.

[5] Van der Walt, S., Schönberger, J. L., Nunez-Iglesias, J., Boulogne, F., Warner, J. D., Yager, N., ... & Yu, T. (2014). scikit-image: image processing in Python. PeerJ, 2, e453.

[6] Boyko, N., Basystiuk, O., & Shakhovska, N. (2018, August). Performance evaluation and comparison of software for face recognition, based on dlib and opencv library. In 2018 IEEE Second International Conference on Data Stream Mining & Processing (DSMP) (pp. 478-482). IEEE.

[7] Song J, Jeong H, Jeong J. Performance Optimization of Object Tracking Algorithms in OpenCV on GPUs. Applied Sciences. 2022; 12(15):7801.

[8] Markuš, N., Fratarcangeli, M., Pandžić, I. S., & Ahlberg, J. (2015, September). Fast rendering of

image mosaics and ascii art. In Computer graphics forum (Vol. 34, No. 6, pp. 251-261).

[9] Naz, F., Shoukat, I. A., Ashraf, R., Iqbal, U., & Rauf, A. (2020). An ASCII based effective and multi-operation image encryption method. Multimedia Tools and Applications, 79, 22107-22129.

[10] Gioiosa, R., Petrini, F., Davis, K., & Lebaillif-Delamare, F. (2004, December). Analysis of system overhead on parallel computers. In Proceedings of the Fourth IEEE International Symposium on Signal Processing and Information Technology, 2004. (pp. 387-390). IEEE.