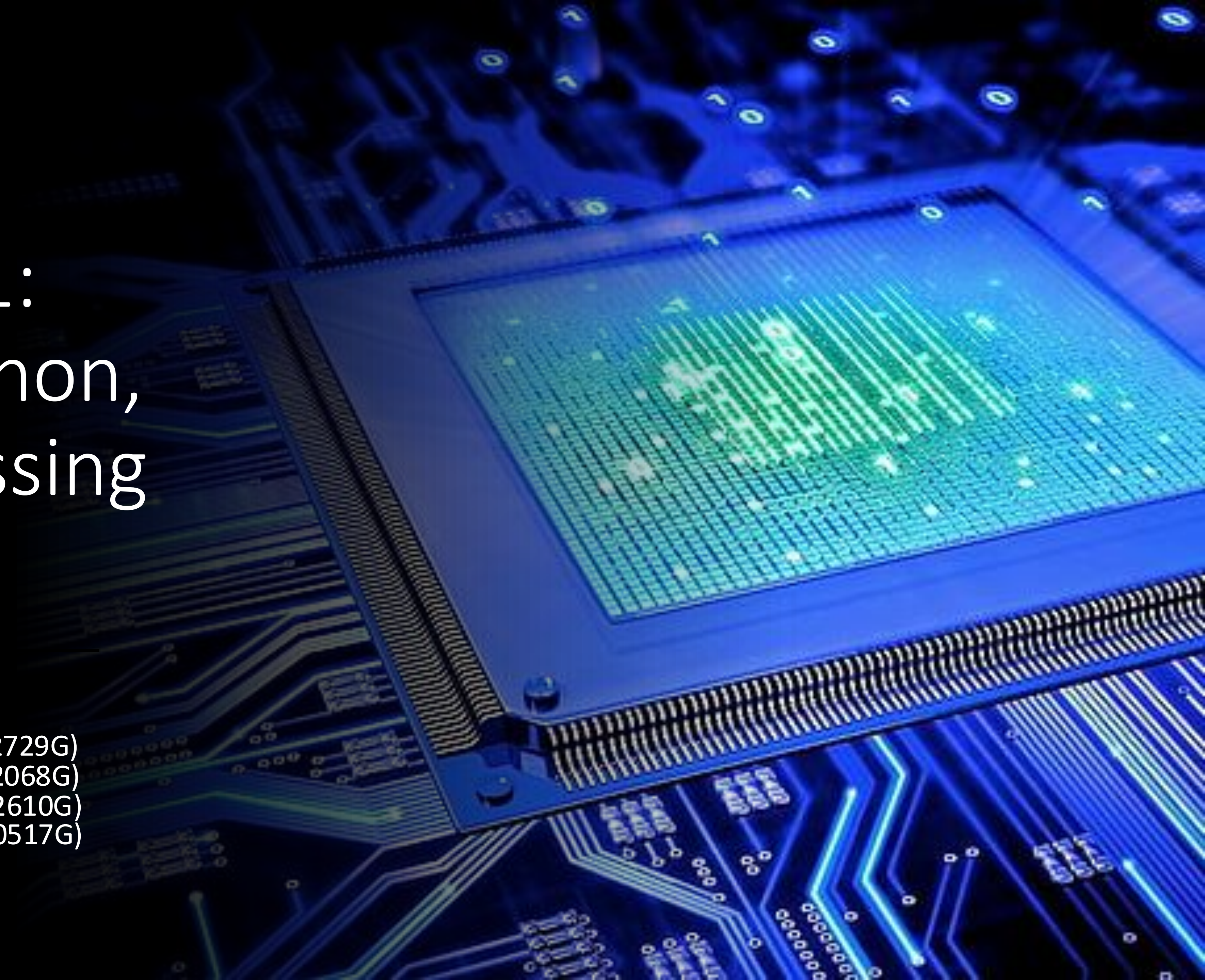


MILESTONE 1: Halide vs Python, Image Processing with tiling

Created by:

Ubaid Dalvi	(2021A7PS2729G)
Apurva Patil	(2021A7PS2068G)
Suryaansh Tripathi	(2021A7PS2610G)
Amaan Sajid Nalakath	(2021A7PS0517G)

Project Description

- Image processing is a critical task in computer vision, medical imaging, and multimedia applications.
- It is essential to optimize image processing algorithms for efficiency.
- This project aims to compare image processing in Halide and Python by using the concept of image tiling.

Project Description

- Halide is a Domain-Specific language built for Image processing
- Halide uses multiple pipelines for faster Array and Image processing Algorithms.
- Python, which is a multi-purpose language has several libraries for computer vision which enable faster Image processing techniques.
- Halide v/s Python in processing the Images
- First break the Images down into chunks and then perform the Image processing on these chunks.

Timeline

Project Proposal

Project Idea Overview and Details about Algorithmic Implementation and Benchmarking to compare the 2 programming languages.

Milestone 1

Image processing using Halide and Python for gray-scaling, brightness boosting, and box-blur etc.

Milestone 2

Tiling for image processing and performance comparison

Python v/s Halide

Aspect	Halide	Python (OpenCV /NumPy)
Language	Domain-Specific Language for Image processing	General-purpose Scripting language
Performance	Optimized for performance with efficient loop scheduling	Generally slower due to interpreted nature
Parallelization	Built-in support	Support varies depending on libraries and functions used.
Memory Management	Can explicitly manage memory for image data	Handled by the language and libraries
GPU Support	Supports GPU acceleration with proper hardware and setup.	Limited GPU support and may require additional libraries or extensions.
Image Tiling	Offers control through fine-grained optimization options.	May require manual implementation depending on the task and library used.
Typical Use case	High-performance image processing applications.	Suitable for a wide range of image processing tasks

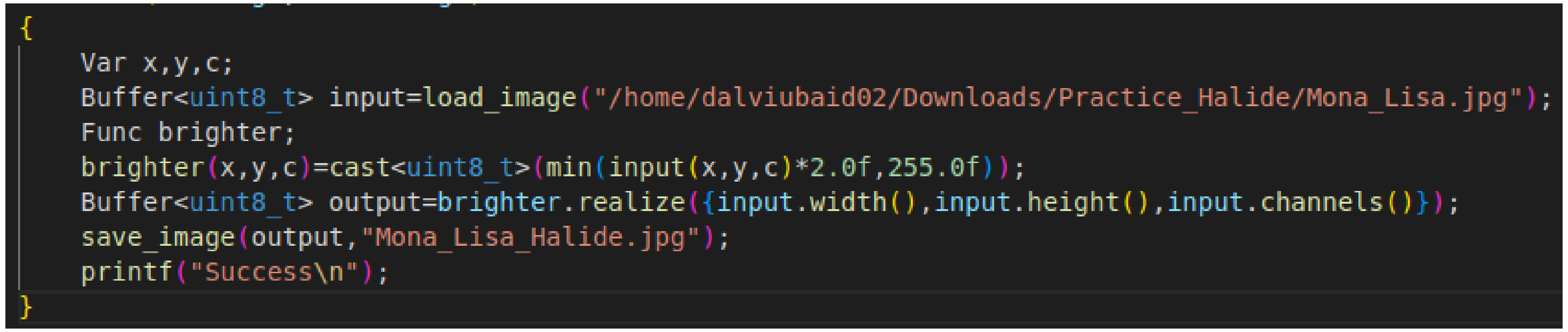
Progress so far

Implemented Halide and Python code for increasing brightness, blurring, gray scaling etc

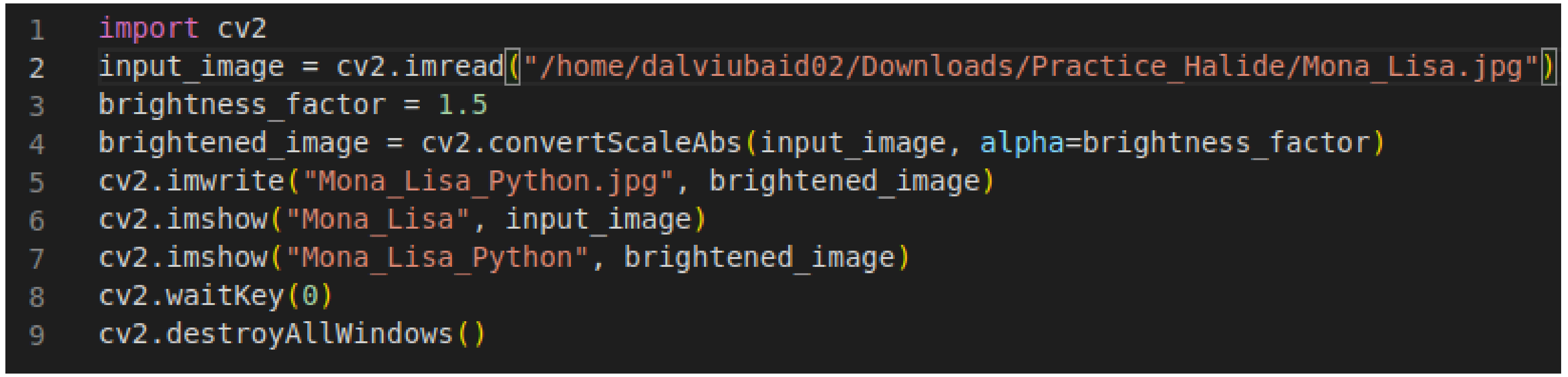
Halide provides more control over different types of scheduling like vectorization, tiling, and unrolling

Pythons openCV has in-built functions for performing the same but they restrict the scheduling parameters.

Increasing brightness in Halide



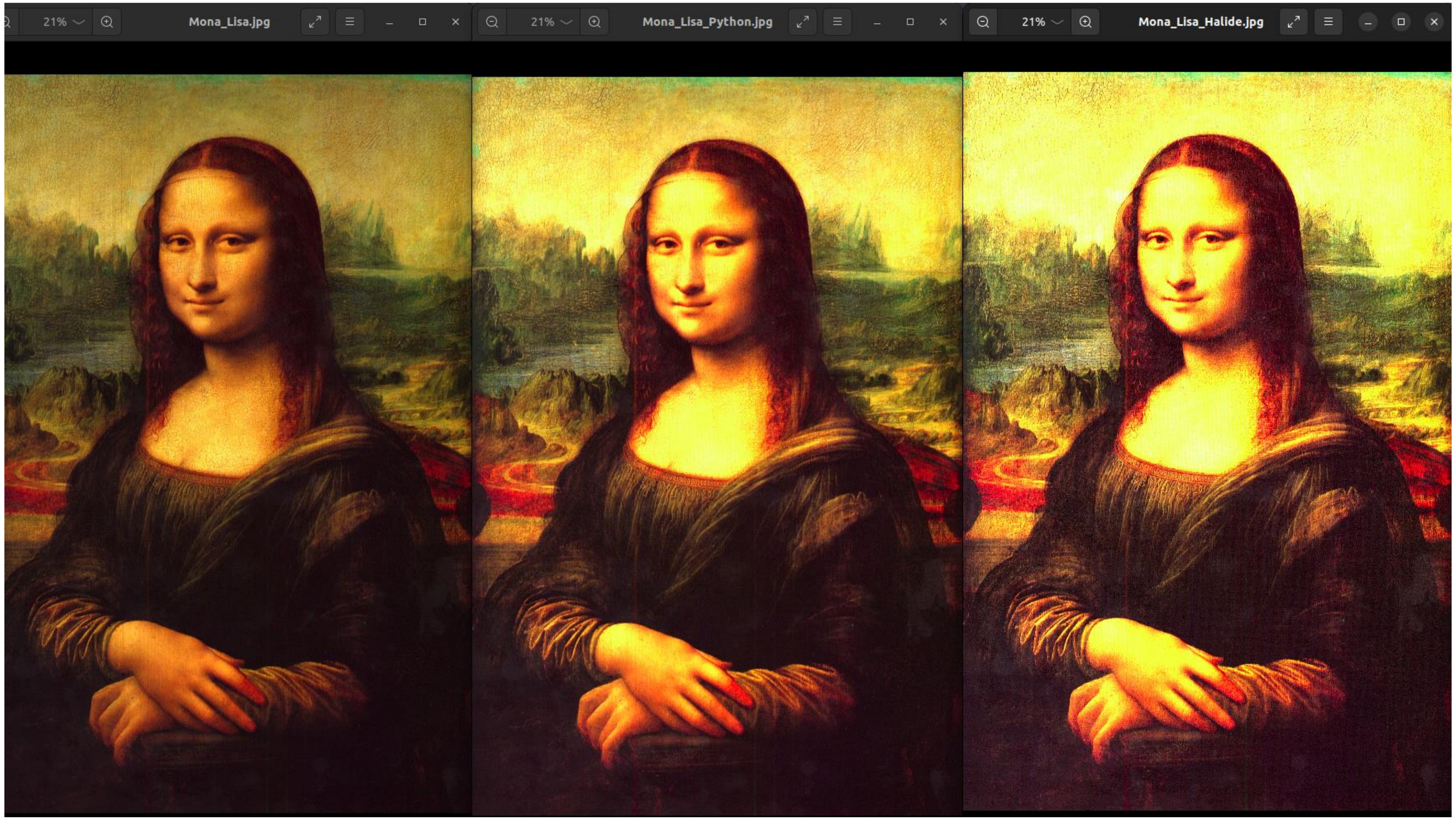
```
{
    Var x,y,c;
    Buffer<uint8_t> input=load_image("/home/dalviubaid02/Downloads/Practice_Halide/Mona_Lisa.jpg");
    Func brighter;
    brighter(x,y,c)=cast<uint8_t>(min(input(x,y,c)*2.0f,255.0f));
    Buffer<uint8_t> output=brighter.realize({input.width(),input.height(),input.channels()});
    save_image(output,"Mona_Lisa_Halide.jpg");
    printf("Success\n");
}
```



```
import cv2
input_image = cv2.imread("/home/dalviubaid02/Downloads/Practice_Halide/Mona_Lisa.jpg")
brightness_factor = 1.5
brightened_image = cv2.convertScaleAbs(input_image, alpha=brightness_factor)
cv2.imwrite("Mona_Lisa_Python.jpg", brightened_image)
cv2.imshow("Mona_Lisa", input_image)
cv2.imshow("Mona_Lisa_Python", brightened_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Increasing brightness in Python

Results


21%
Mona_Lisa.jpg
21%
Mona_Lisa_Python.jpg
21%
Mona_Lisa_Halide.jpg

Halide: Advantages and Limitations

Advantages of Halide:

- Performance: Designed for high-performance image processing.
- Automatic Optimization: Efficient code generation for CPU and GPU.
- Parallelism: Easily expresses parallelism for multi-core and SIMD.
- Custom Optimization: Fine-tune code for specific hardware.

Limitations of Halide:

- Complexity: Steeper learning curve and expertise required.
- Limited Ecosystem: Smaller community and fewer pre-built libraries.
- Integration: More challenging to integrate with other systems.

Python: Advantages and Limitations

Advantages of Python:

- Ease of Use: High-level language with straightforward syntax, accessible to many developers.
- Vast Ecosystem: Active community, extensive libraries, and resources for image processing.
- Integration: Easily integrates into software projects and complete applications.
- Cross-Platform: Runs on various operating systems.

Limitations of Python:

- Performance: Slower for high-performance applications due to interpretation.
- Limited Low-Level Control: Provides less fine-grained low-level control and optimization compared to Halide.
- Manual Optimization: Performance optimization often requires manual intervention and tuning, which is potentially more time-consuming than Halide's automatic optimizations.

Future Works:

Implementing Tiling of Images in Halide and Python , which would help in breaking down images in small sections and then processing the Images on these individual sections.

Calculating effective GPU and CPU usage for performance comparison.

Comparing overall scalability of Python and Halide for the purpose of Large-Scale Image processing such as Satellite Imaging.

THANK YOU