Vietnam National University

HaNoi University of Science

Project Report

PARALLEL COMPUTING

Topic: IMAGE STITCHING

Group: 10

Lecturer: **Trần Hà Nguyên**

Students:

Nguyễn Trọng Đức	**21002133**
Lê Trường Giang	**21002135**
Trần Vũ Minh Đăng	**21002132**
Nguyễn Văn An	**21002115**
Nguyễn Quỳnh Anh	**21002119**

1. ABSTRACT

Image stitching is used to combine several individual images that have some overlap into a composite image. There are many methods that have been used, but each method has its own weaknesses. There are methods that give good image quality but take a long time to execute. There are methods that have fast execution time but the image quality is not guaranteed. Therefore, we propose a new calculation method using GPU to perform parallelization, reducing execution time but still ensuring the quality of the resulting image. Experimental results show that our method is 3 - 7.5 times faster than popular existing methods.

2. INTRODUCTION

Surely, we have all seen or used the panorama photo capture feature on smartphones. These panorama photos are relatively large in size, providing a wide view. With smartphones, we can create them by slowly panning the camera across the scene to be captured. The applications of this type of problem are numerous. In the past, when the quality and field of view of camera shots were limited, algorithms like these were used to generate high-resolution images with a panoramic view. For example, the image below gained attention in 2019, shared by neighbors, claiming to be a 24.9 billion-pixel photo taken from a "Quantum Technology Satellite." It could be zoomed in to reveal details of streets, windows, or people in Shanghai. It was later discovered that this image was actually composed of tens of thousands of photos created by the company Jingkun Technology. So, what is the technology behind these panorama photos?

This topic is known as Image Stitching or Photo Stitching, involving combining a set of images to create a large, seamless picture, with overlapping regions. Previously, the approach was to "directly minimize pixel-to-pixel dissimilarities," meaning finding the overlapping region by matching two areas on two images with the smallest pixel differences. However, this method proved to be less effective. Subsequently, the Feature-based approach emerged.

Some feature-based methods include SIFT (Scale-Invariant Feature Transform), known for its stability and high feature description. However, serious distortion can still occur, leading to panoramas seemingly approaching infinite size. Images may be overexposed and blurry, and transitions between images can be difficult and messy. At the same time, SHIFT has a very large amount of calculation, so the execution time is very long

ORB (Oriented FAST and Rotated BRIEF) is also a feature-based method with fast and stable processing speed, suitable for applications requiring rapid computation and real-time processing. However, ORB may be more susceptible to noise and lighting variations as its algorithm depends on pixel brightness, causing the quality of the composite image to be poor

We propose a new method using parallel computing to optimize to ensure stability in both time and quality. Our method includes:

- Use the Gauss and Pyramid of Gauss filters to identify key points in the image, next use the BRIEF (Binary robust independent elementary feature) descriptor for the keypoints, then match the key points and combine the image.

- Using CUDA to parallelize the Pyramid of Gauss filter calculation process and the keypoint matching process helps reduce calculation time many times.

Our proposed algorithm is 3-7.5 times faster. And the image quality stitched by our proposed method always maintains good quality with different numbers of images.

3. LITERATURE REVIEW

3.1 SIFT (Scale-Invariant Feature Transform)

Many methods have been implemented to combine images in recent years. In this paper, they propose a method that can create beautiful panoramas without requiring any additional information from the user and can even work with images arranged in any order. any.
The final algorithm implemented in Python is very efficient, with many advanced features that provide a good experience.

SIFT [2] (Scale-Invariant Feature Transform) is a powerful feature detection and description algorithm widely used in image processing. It was introduced by David Lowe [8] in 1999 and has since become fundamental in many different applications. Since SIFT features do not change with changes in rotation and scale, the model can process images with different orientations and zooms.
The next step is to use those matches to determine which images will overlap in the panorama and calculate the identity matrices.

Image matching using SIFT and RANSAC [7] allows the recognition of many different panoramas from a set of images and can also remove any noisy images that are not present in any panorama. The SIFT algorithm has a time complexity of O($N^{2log(N)}$), where N is the number of key points detected in the image.

The previous two steps have given us a panorama, where each image is accurately placed relative to each other. However, the resulting image is far from pleasing to the eye, and the transitions between each image are anything but seamless. To reduce the difference, they calculate the overall gain between images and apply a gain compensation that gives a more natural look to the final panorama, reducing the difference between images.

Finally, using linear and multiband blending improves panoramas by making the transitions between composite images smoother.

The implemented algorithm can be used on almost any type of image, as long as the total viewing angle is not too large. In most cases, the results of simple blending are very satisfying, and the boundaries between images are nearly seamless, as long as there are no major changes in the environment.

However, quite serious distortion still occurs and sometimes even causes the size of the panorama to tend to become infinite. Images tend to be overexposed and blurry, and transitions between images are sometimes a bit difficult and messy.

3.2 ORB (Oriented FAST and Rotated BRIEF)

In the paper "ORB: An efficient alternative to SIFT or SURF" by E. Rublee, V. Rabaud, K. Konolige, and G. Bradski [9], their research focuses on the efficiency of the ORB algorithm as an alternative to SIFT or SURF in the context of feature point extraction and description, particularly in the application of image stitching. This is also the major distinction of this approach compared to traditional image stitching. A notable aspect of this algorithm lies in its ability to ensure reasonable alignment between feature points in two different images, creating stitched images that maintain naturalness and accuracy.

The ORB's solution method begins with the use of the FAST algorithm (Features from Accelerated Segment Test) [10] to rapidly determine feature points in the image. FAST is an efficient algorithm with linear complexity, providing an advantageous condition for quickly extracting feature points from the image.

Subsequently, ORB utilizes the BRIEF method (Binary Robust Independent Elementary Features) [11] to describe the feature points. BRIEF generates a binary representation vector for each feature point by comparing pixel values of a selected set of points. One of the strengths of BRIEF is its speed and efficiency, especially when compared to other description methods like SIFT.

For image stitching, ORB leverages geometric information about the oriented nature of feature points to enhance flexibility in the stitching process. The image comparison and stitching process use the RANSAC technique (Random Sample Consensus) to estimate the transformation matrix between feature points in two images. This helps minimize the impact of noise and increase the accuracy of the stitching process.

In the best-case scenario, the ORB algorithm can achieve O(1) time complexity when no feature points are satisfied, and at worst, O(N) when all pixels in the image satisfy the feature points, where N is the number of pixels in the image. The execution time for image stitching depends heavily on the image size, making the algorithm's performance stable and relatively fast for datasets with few input images. This is suitable for applications that require fast computation and real-time processing. However, ORB may be more affected by noise and lighting variations as its algorithm relies on pixel brightness.

4. METHOD

Background

Based on the literature review, we observed that the two methods mentioned still have some drawbacks. Therefore, we propose incorporating the Difference of Gaussian (DoG) method along with CUDA to not only enhance processing speed but also improve the quality of the output images. In this report, we propose an optimization method for GPU-accelerated "Image Stitching" using CUDA. The proposed method consists of the following main steps:

1) Combining a set of Gaussian filters on the image.
2) Building the Gaussian pyramid's difference (DoG).

3) Identifying key points.
4) Computing BRIEF (Binary Robust Independent Elementary Features) for key points.
5) Matching key points across images.
6) Calculating the Homography matrix.
7) Warping images and stitching them together to create a panoramic image.

4.1 Keypoint detection

Gaussian filter

A Gaussian filter is a low-pass filter (a filter that only allows certain frequencies to pass, by reducing or blocking high frequencies) used to reduce noise (high frequency components) and blur regions of an image.

When working with images, we need to use a 2-dimensional Gaussian function:

$$G(x,y) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

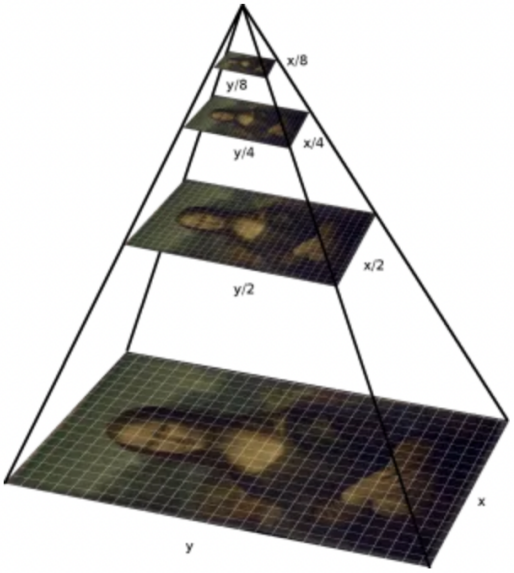
Where:

- G(x,y) is the Gaussian kernel at coordinates (x,y).
- σ is the standard deviation that determines the spread of the distribution.
- e is the base of the natural logarithm.

Next, we will create a set of Gaussian filters with different standard deviations and then apply each Gaussian filter to the input image. This will create an output image for each filter and the resulting images as an array.

Gaussian Pyramid

Sometimes, to detect an object in an image, we need to resize or sub-sample the image and run further analysis. In such cases, we maintain a collection of the same image with different resolutions. This collection is the Image Pyramid. The reason behind calling it a pyramid is that when we arrange the images in decreasing order of their resolution, we obtain a shape like a pyramid with a square base.



If the size of the base image (high resolution or level 0) is MXN, then the size of the upper level will be (M/2 XN/2).

In the Gaussian Pyramid, there are two types of operations: reduction and expansion. As the name suggests, in a shrink operation, we reduce (by half the width and height) the image resolution, and in an expand operation, it's the opposite.

The images, after being subjected to Gaussian filters with different scales, will be used to build the image pyramid. Lower levels of the pyramid correspond to larger sigma values, resulting in a smoother image, and vice versa for high levels, while higher levels correspond to smaller sigma values, providing more detailed images.

Difference of Gaussians (DoG)

“Difference of Gaussians (DoG)” is an image enhancement algorithm that works by subtracting two different Gaussian blurs on the image, with a different blur radius for each, and calculating their difference. Subtracting one image from the other preserves spatial information that lies between the frequency range preserved in the two blurred images. The most important parameters are the blur radius of the two Gaussian blur effects. Increasing the smaller radius tends to make the edges look thicker, and decreasing the larger radius tends to increase the “threshold” for identifying something as an edge.

In its operation, the Gaussian algorithm difference is said to mimic the way neural processing in the eye's retina extracts details from images transmitted to the brain.

To help identify important places, the generated DoG picture shows areas where there were notable differences between the two blurry versions of the image.

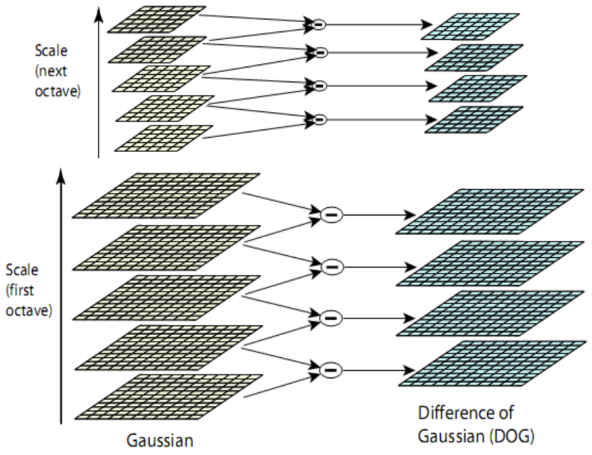
Specifically, we want to find:

$$D_1(x,\ y,\ \sigma_l)\ = G(x,\ y,\ \sigma_{l-1})\ -\ \ G(x,\ y,\ \sigma_l)\ *\ I(x,\ y)$$

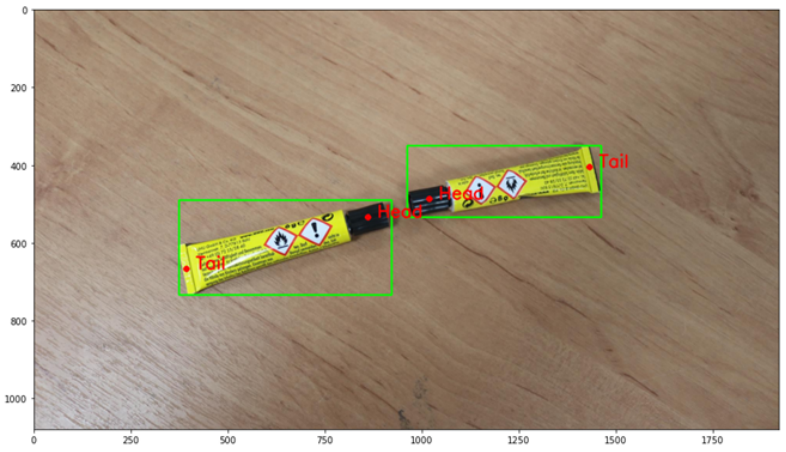
where $G(x, y, \sigma_l)$ is the Gaussian filter used at level l and $*$ is the convolution operator.
Due to the distributive property of convolution, this simplifies to

$$D_1(x, y, \sigma_l) = G(x, y, \sigma_{l-1}) * I(x, y) - G(x, y, \sigma_l) * I(x, y) = GP_l - GP_{l-1}$$

where GP_l denotes level l in the Gaussian pyramid.



Keypoint detection[1]**:** involves identifying specific, distinct points or locations within an image or frame in a video. These distinctive points, often referred to as "key points".



Keypoint definition[1]**:** Key points are typically defined by certain characteristics that set them apart from the surrounding pixels, specifically:

- Uniqueness: Key points should be unique and easily distinguishable from other points in the image. They stand out due to specific visual attributes, such as color, intensity, or texture.

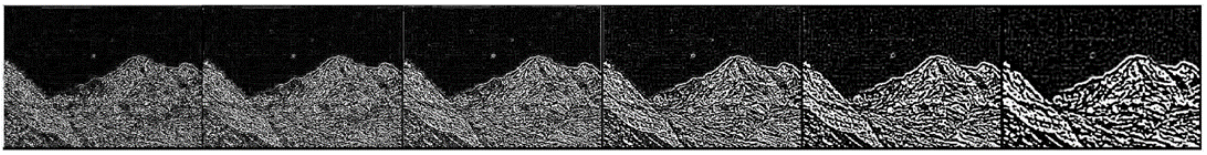
- Invariance: For example: rotate, scale or change lighting conditions. Thus, the same keypoint should be detectable in different versions of the same object or scene.

- Repeatability: Key points should be reliably detectable across different instances of the same object or scene.

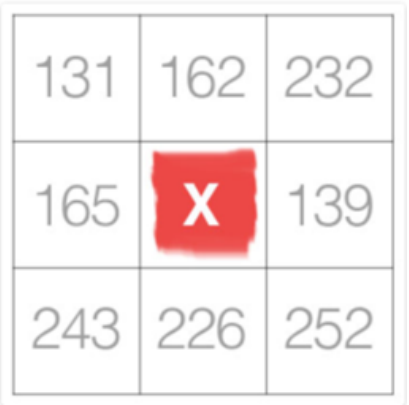
Build a keypoint detector:

We convolve a set of Gaussian filters on the image to produce a Gaussian Pyramid. The Gaussian pyramid of the example input image is shown in the following figure:

Then we obtain the Difference of Gaussian (DoG) by subtracting adjacent levels of the Gaussian Pyramid. The difference of Gaussian of the example input image is shown in the following figure:



Now that we have constructed the difference of Gaussians, we can move on to finding local maxima and minima in the DoG images.



So now for each pair of Difference of Gaussian images, we are going to detect local minima and maxima. Consider the pixel marked X in the following figure, along with its 8 surrounding neighbors:

We only check neighboring pixels with a patch size of 3x3. This pixel X can be considered a “keypoint” if the pixel intensity value is larger or smaller than all of its 8 surrounding neighbors[3]. The cost of this check is reasonably low due to the fact that most sample points will be eliminated following the first few checks[2].

Once a keypoint candidate has been found by comparing a pixel to its neighbors, the next step is to perform a detailed fit to the nearby data for location, scale, and ratio of principal curvatures. This information allows points to be rejected that have low contrast (and are therefore sensitive to noise) or are poorly localized along an edge. The initial implementation of this approach (Lowe, 1999)[5] simply located keypoints at the location and scale of the central sample point.

Contrast can appear in many forms. It can be color contrast, shape contrast, or proportional contrast[4]. For example:

If we follow the color wheel we will find color pairs with low contrast, such as yellow and orange. If we take two colors opposite each other, it will create the highest contrasting color pair.

For stability, it is not sufficient to reject keypoints with low contrast. The difference-of-Gaussian function will have a strong response along edges, even if the location along the edge is poorly determined and therefore unstable to small amounts of noise.[4]

A poorly defined peak in the difference-of-Gaussian function will have a large principal curvature across the edge but a small one in the perpendicular direction[4]. The principal curva-tures can be computed from a 2x2 Hessian matrix, H, computed at the location and scale of the keypoint:

$$\mathbf{H} = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix}$$

The derivatives are estimated by taking differences of neighboring sample points.

The eigenvalues of H are proportional to the principal curvatures of D. Borrowing from the approach used by Harris and Stephens (1988)[6], we can avoid explicitly computing the eigenvalues, as we are only concerned with their ratio. Let α be the eigenvalue with the largest magnitude and β be the smaller one. We have:

$$\begin{aligned} \mathrm{Tr}(\mathbf{H}) &= D_{xx} + D_{yy} = \alpha + \beta, \\ \mathrm{Det}(\mathbf{H}) &= D_{xx}D_{yy} - (D_{xy})^2 = \alpha\beta. \end{aligned}$$

In the unlikely event that the determinant is negative, the curvatures have different signs so the point is discarded as not being an extremum. Let r be the ratio between the largest magnitude eigenvalue and the smaller one, so that α = rβ. We have:

$$\mathrm{R} = \frac{\mathrm{Tr(H)}^2}{\mathrm{Det(H)}} = \frac{(\alpha+\beta)^2}{\alpha\beta} = \frac{(r\beta+\beta)^2}{r\beta^2} = \frac{(r+1)^2}{r}$$

=> R depends only on the ratio of the eigenvalues rather than their individual values. $\frac{(r+1)^2}{r}$ is at a minimum when the two eigenvalues are equal and it increases with r. Therefore, to check that the ratio of principal curvatures is below some threshold, r, we only need to check $\mathrm{R} < \frac{(r+1)^2}{r}$

Here is an example of detected keypoints:

(a) Key points detected for Fig 1(a)

(b) Key points detected for Fig 1(b)

4.2 Feature Descriptor

What is a feature descriptor? Feature descriptors are a very important part in image processing in particular and in computer vision in general. It is used to provide descriptions for a particular point we are considering in the image. The feature descriptor will then create a numerical representation or a vector for each particular point, with the purpose of describing specific information related to a region surrounding that point with the goal of having can create representations that can distinguish those particular points from each other in image space.

And in this problem, we consider the BRIEF feature descriptor with special points being the keypoints discovered above. BRIEF stands for “Binary Robust Independent Elementary Features”, this is a method of describing keypoint features in the form of binary vectors to be able to detect and distinguish keypoints in images. The way BRIEF works is: randomly select a pair of images, then use a function to compare whether the values of this pair of pixels are greater or less than each other. The result of the entire comparison process will be represented in binary form and form a unique feature descriptor for that keypoint.

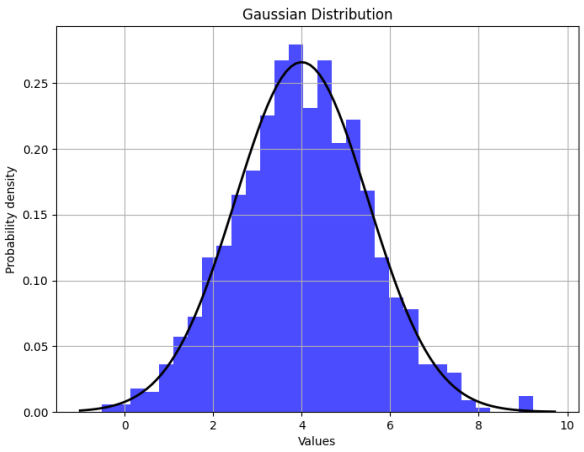
We will take a closer look at how BRIEF produces a descriptor in this image matching problem:

- Step 1: Extract important regions:

We will consider a 9 x 9 matrix with the key point being the center of the matrix extracted from the image matrix with the index positions along with the values of the elements remaining the same as the image origin. Then, we will transfer all the values of the elements of this matrix into a new 9 x 9 matrix in the correct position corresponding to the row and column index running from 1 to 9.

- Step 2: Create pairs of sample points:

Applying to the problem: We create 256 coordinate pairs [A, B]: $[A_1(x_1, y_1)\ B_1(x'_1, y'_1)]$, $[A_2(x_2, y_2)\ B_2(x'_2, y'_2)]$,..., $[A_{256}(x_{256}, y_{256})\ B_{256}(x'_{256}, y'_{256})]$. With $x_i, y_i, x_i, y'_i \in [1,9]$ ($i \in [1, 256]$) and coordinates are randomly generated according to Gaussian distribution (another name is normal distribution). We should know that the Gaussian distribution or normal distribution is called a bell curve because: The probability of occurrence of values near the midpoint will appear higher than points located at the edge and gradually decreases in a bell shape.

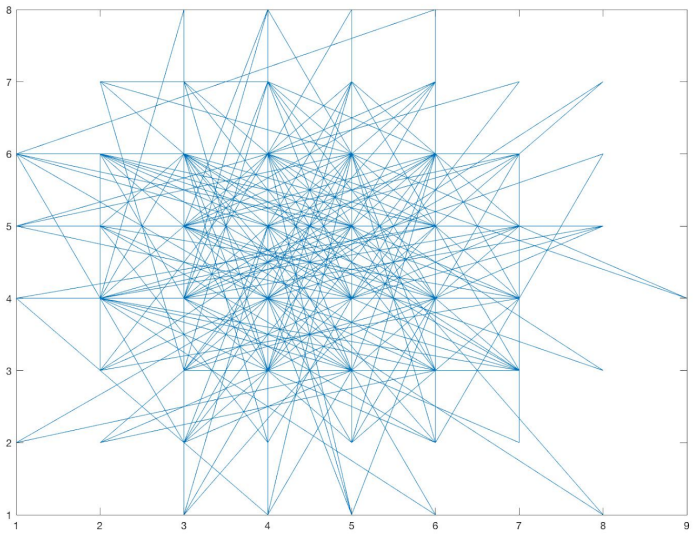


So we can see that the points created will be closer to the center of the matrix, and the center of the matrix is the keypoint we are considering. This will lead to a more accurate descriptor.

- Step 3: Compare intensities and initialize the binary descriptor:

Applying to the problem: Next, with a pair of coordinates $[A_i, B_i]$ in 256 pairs, we will extract the value of their corresponding pixel in the 9 x 9 matrix of the keypoint and compare with each other. . If the value of Ai < Bi then the result will return 1 and vice versa if $A_i > B_i$ then the result will return 0.

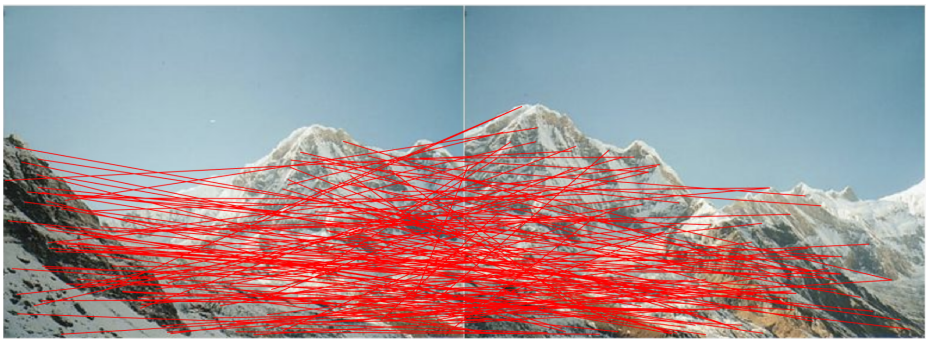
After 256 comparisons, a binary string will be returned and this binary string is the descriptor of the key point under consideration. The figure below visually shows the sampled locations, where each line connects a pair $[A_i, B_i]$.



4.3 Matching Key Points

We then match key points using the distance of their BRIEF descriptors. Hamming Distance, i.e. the number of positions where two descriptors differ, is used as the distance metric. Given the descriptors of two images, we find the best matching point in the second image for each key point in the first image.

We find both the minimum and the second minimum distance between the descriptor to be matched and some other descriptor in the second image. To reduce the likelihood of incorrect matching, we only consider the matching as valid if the difference between the minimum distance and second minimum distance is large enough.The matched key points are shown in the following figure.



4.4 Compute Homography between Image Pairs

Compute the homography (H), a transformation matrix that warps one image onto another, between each adjacent image. This is done by randomly sampling four pairs of matching key points of two images and computing an estimated homography until a good homography is found.

$$\mathbf{p' = Hp}$$

$$\begin{bmatrix} wx' \\ wy' \\ w \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

4.5 Stitching Image

Finally, we stitch the images by warping and blending the images onto a common plane to produce the panorama. The stitching result for the example input images is shown in the following figure.

4.6 Optimize with GPU

The five stages in the pipeline are dependent on each other and should be carried out sequentially, yet there is a lot of parallelism in each stage, both within a single image and across multiple images. Filter convolution considers the local neighborhood centered around each pixel, and thus exploiting strong spatial locality. Processing the image involves performing the same operations on each pixel, so it fits well with the data-parallel model. Therefore, the program can benefit greatly from parallel implementation. We will use C++ and CUDA for programming.

4.6.1 Gaussian Pyramid Computation

The complexity of computing Gaussian Pyramid can be up to O(n1*n2*fh*fw*l) where n1, n2 are the vertical and horizontal image sizes, fh and fw are the height and width of the Gauss filter, l is the

number of floors of the Pyramid. The convolution in Gaussian pyramid computation was performed by iterating over each pixel in the image and computing the weighted sum of the local neighborhood of the pixel. This computation fits naturally with the data-parallel model of CUDA, so we choose to parallelize within each image. We spawn a threaded-block with the size of the image, and each thread in the block performs the convolution for a single pixel. We synchronize the CUDA threads at the end of the computation of the Gaussian Pyramid.

4.6.2 Parallelization of Key Point Matching

The key point matching process uses the hamming distance between two BRIEF descriptions as the distance metric. For each key point in image 2, the algorithm finds its closest key point in image 1. In our baseline sequential implementation, a BRIEF descriptor is implemented as a vector of 256 integers with 0/1 values. Therefore, comparing two descriptors needs to iterate over the vector, which is very time-consuming.

Our first decision is to improve the computation time for hamming distance by replacing the data structure that represents the BRIEF descriptor with a bit array, i.e. std::bitset in C++. In this way, computing the hamming distance between two descriptors is essentially XORing the two bit arrays and counting the number of set bits of that bit array. Counting the number of set bits on a bit array is simply calling the std::bitset::count function.

We decided to parallelize the key point matching process using CUDA. However, it is not possible to simply utilize std::bitset in CUDA device code since bitset is not implemented in CUDA. Thus we implement our own version of bitset in CUDA. We use 4 64-bit unsigned integers to represent the 256-bit descriptor, and use bit-manipulation to set and count the bits in the descriptor efficiently.

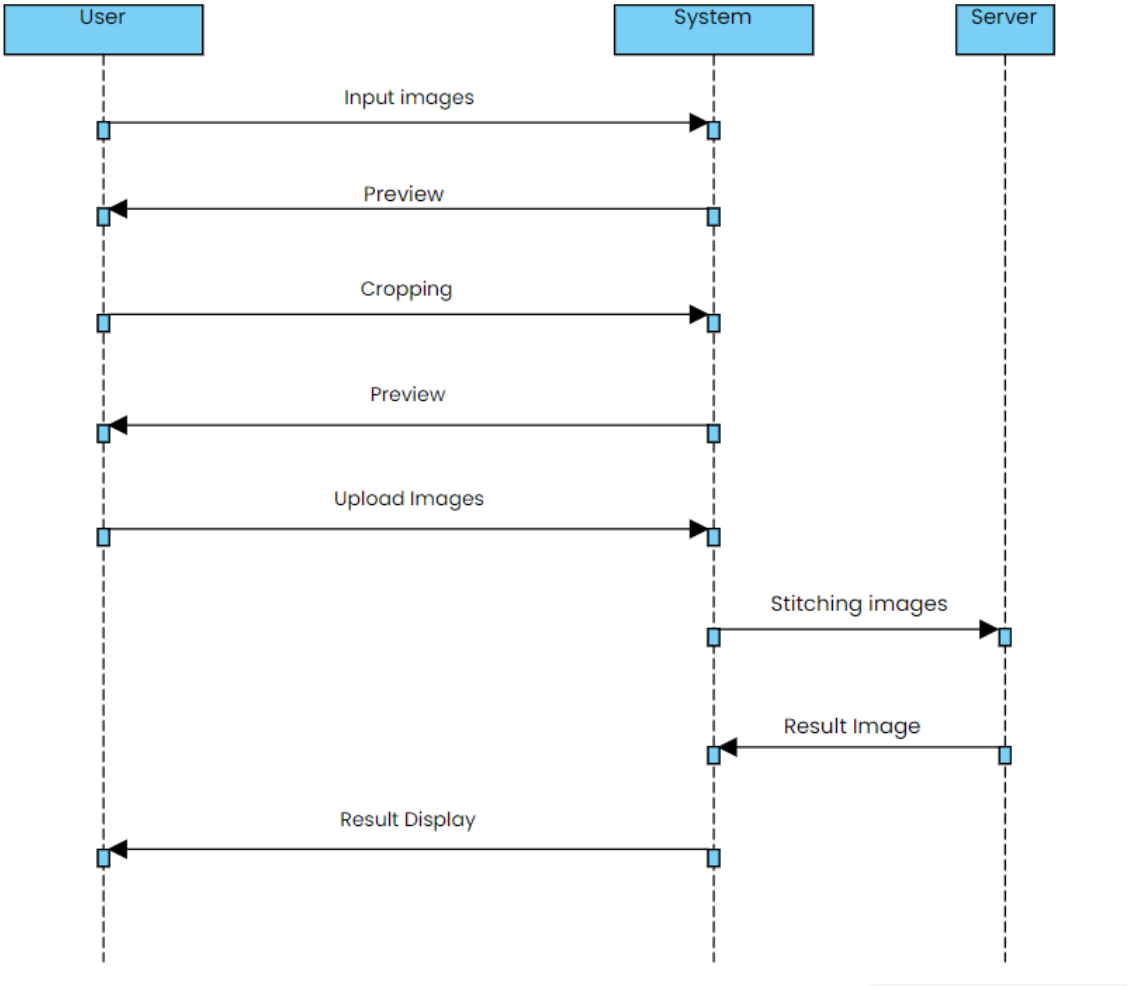
```
struct Descriptor {
  uint64_t num0;
  uint64_t num1;
  uint64_t num2;
  uint64_t num3;
};
```

To match key points between two images in parallel, our approach is to parallelize all the key points of the first image. For each key point in the first image, a CUDA thread is spawned. Each thread iterates over all the key points in the second image, and computes the hamming distances between these key points and the key point in the first image to select the closest matching key point.

4.7 Analyze design applications:

- **Component Diagram:**

- **Sequence Diagram:**



- **Workflow:**

1. Input Images: User select images from their computer and then upload it to the app. After the image is uploaded, the app will display all the images for the user and store those images.

2. Edit Images: User will have the option whether to cut the images or not. After the user has changed the images, it will replace the old one and being displayed again for the user.

3. Upload and stitching images: When user has confirmed all the images used for stitching, the system will then upload it to the server using API. The server then will call a stitching algorithm to stitch all the uploaded images and return a stitched image as the result.

4. Display result: The server will send back the result image to the app which then will be displayed and can be download by the user for other purpose.

- **User Interface:**

The user interface will be divided into 2 parts:

Section 1: Will display basic information of the website as well as the purpose and information of the working group. There will be a button:

- **LET'S GET STARTED**: Will move on to section 2

Section 2: This is where users will work with data images and perform image stitching

- **Input Images**: Allows users to select and upload photos they want to merge from the device
- **Stitch images**: Carry out the process of sending images to the server and when the results are returned, a Download button will be created
- **Uploaded Images**: Allows users to customize photo cropping as desired
- **Confirm**: Appears after pressing button Uploaded Images. Allows the old photo to be changed to the edited photo
- **Cancel**: Also appears after pressing button Uploaded Images. Has the function of canceling the cropping process
- **Reset page**: Will return to the first interface of the website
- **Reset pre images**: The most recent photo will be deleted
- **Confirm Stitched Images**: Performs latching of images to be merged as well as to activate button A
- **Download**: Appears after the composite image has been returned and allows the user to download the resulting image.

5. EXPERIMENT

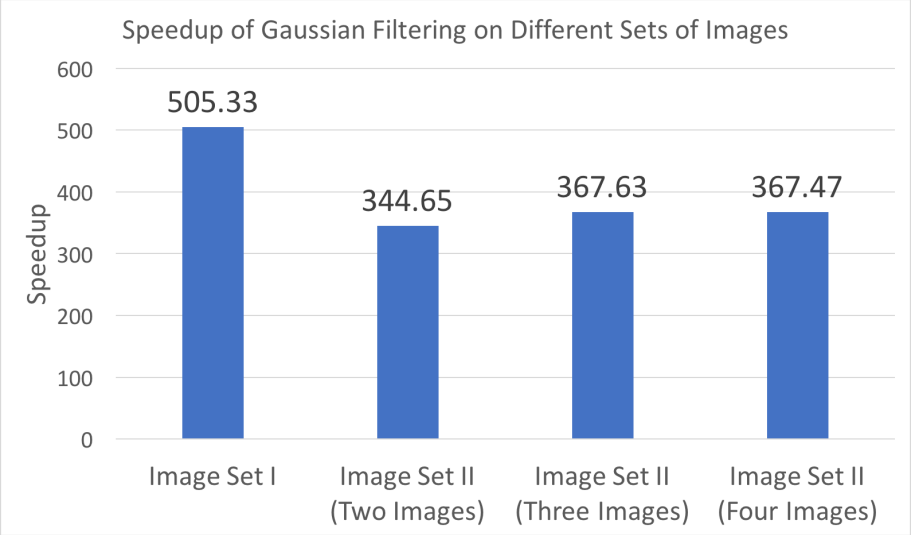
Our test device uses NVIDIA GeForce GTX 1650 and Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz, 2592 Mhz, 6 Core(s), 12 Logical Processor(s).

Below is information about the Dataset used:

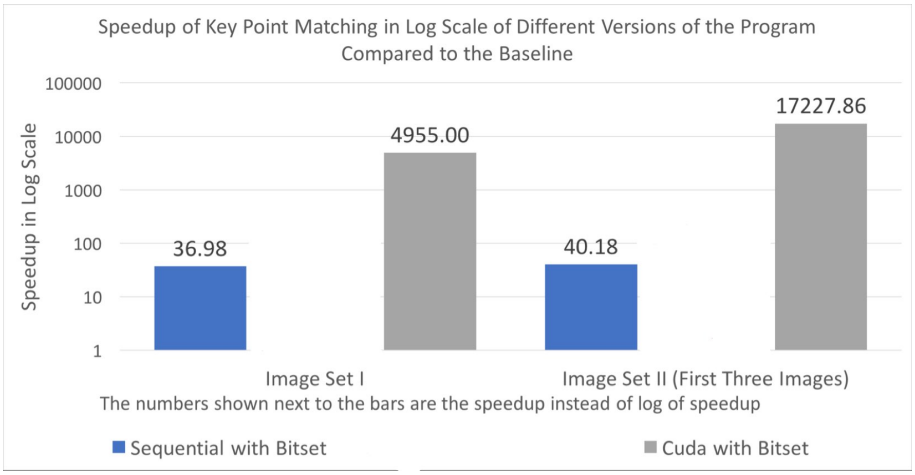
Data	Number of images	Image size (pixel)
Dataset 1	2	1064 x 576
Dataset 2	3	1440 x 1080
Dataset 3	4	320 x 480
Dataset 4	5	1440 x 1080

The results of the experiments are shown in the following figures:

Figure 0: The speedup of our method on GPU compared to the sequential version



Because we parallelize the computation within each image, the scale of parallelism increases as the number of pixels in the image increases, we expect the speedup to scale with the average size of the input images. However we notice that the speedup is greater for Image Set I which has smaller images. We reason that this is due to memory transfer overhead between the CUDA host and device, and synchronization overhead of CUDA threads. Because Image Set I has only two images of smaller size, much less data is transferred between the host and the device. There are also fewer threads spawned, so there is less synchronization overhead after each Gaussian filtering. To confirm our conjecture, we perform a fine-grain time measure on the parallelized computation of Gaussian filtering on Image Set II.



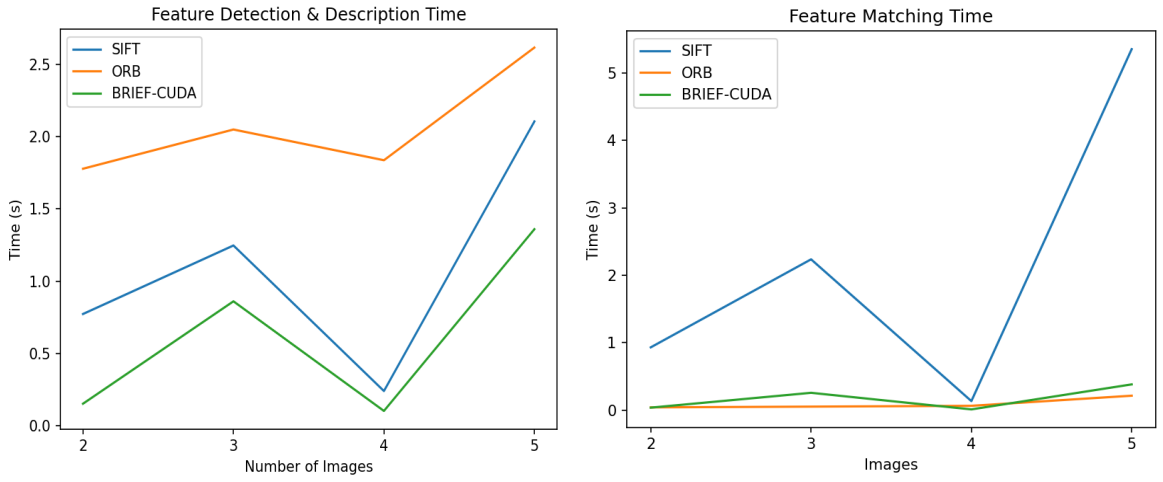
We obtain a significant speedup on CUDA with self-implemented bitset. The parallelized key point matching algorithm scales with the number of key points detected in the input image. Because we compress the descriptor structure from 256 32-bit integers to 4 64-bit integers, we significantly reduce the amount of memory transferred between CUDA host and device. Given that there are tens of thousands of key points in each image, this optimization greatly reduces the memory overhead.

Figure 1: Time comparison table detailing each step and total execution time with difference method

Method	Feature Detection & Description Time (s)	Feature Matching Time (s)	Homography Calculation Time (s)	Stitching Image Time (s)	Total Time (s)
Dataset 1					
SIFT	0,772	0,931	0,115	1,465	3,283
ORB	1,779	0,040	0,023	0,013	1,855
BRIEF - CUDA	0,150	0,036	0,080	0,150	0,416
Dataset 2					
SIFT	1,247	2,237	0,120	1,864	5,468
ORB	2,051	0,051	0,047	0,036	2,185
BRIEF - CUDA	0,860	0,256	0,330	0,390	1,836
Dataset 3					
SIFT	0,238	0,133	0,120	0,547	1,038
ORB	1,838	0,063	0,060	0,010	1,971
BRIEF - CUDA	0,100	0,010	0,070	0,210	0,390
Dataset 4					
SIFT	2,107	5,356	0,127	6,720	14,310
ORB	2,619	0,213	0,080	0,045	2,957
BRIEF - CUDA	1,360	0,381	0,580	1,630	3,951

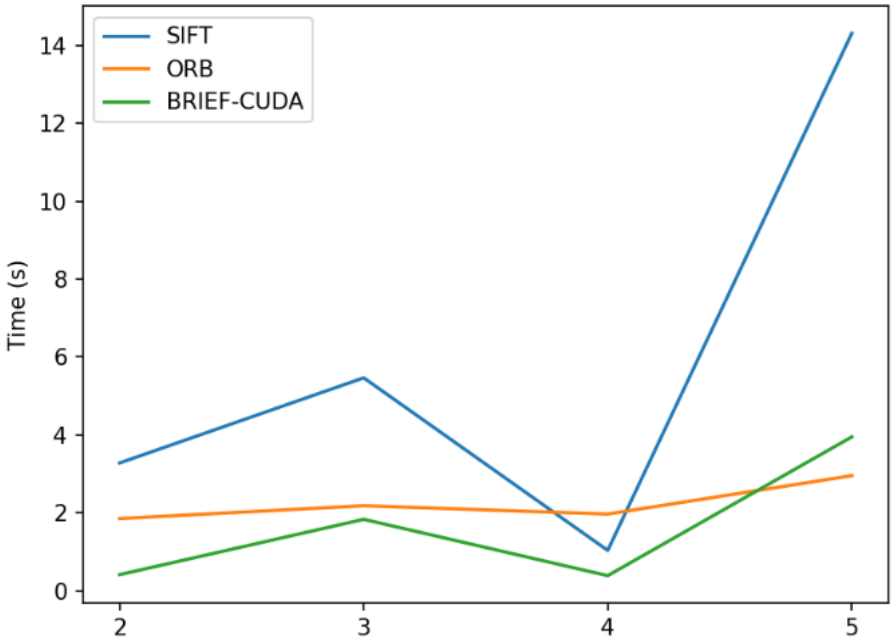
It can be seen that our proposed algorithm is 3-7.5 times faster in most test cases.

Figure 2: The chart compares the execution time of steps using parallelism



The chart above shows that BRIEF-CUDA's computational efficiency is better in feature detection & description and feature matching steps than SIFT and ORB. This results in significant efficiency gains, both in terms of the output image quality due to the DoG algorithm and the execution time thanks to parallelization, in the image stitching task.

Figure 3: Chart comparing total execution time



SIFT computes floating-point descriptors, which involves complex calculations that are computationally intensive and require computation at various levels. In addition, SIFT also requires more memory storage, so in memory-constrained environments or real-time applications, memory efficiency will be lower.
SIFT generates larger descriptors, making the matching process more computationally demanding than BRIEF's simpler binary descriptor matching. Therefore, the calculation time for SIFT will be longer.

Some image of the results after stitching:

Dataset 2

- SIFT

- BRIEF-CUDA

- ORB

We found that the correct matching ability of the methods for a small number of images (2 or 3 images) is equivalent.

Dataset 4

- SIFT

- BRIEF-CUDA

- ORB

In the case of a larger number of images, although the method using ORB shows a slightly faster speed than our method, the resulting images are skewed quite a lot. It can be seen that our proposed method has very high stability.

6. CONCLUSION

In this paper, we have proposed a method of using the BRIEF descriptor combined with parallelization on GPU to increase computational performance. Our proposed algorithm is 3-7.5 times faster in most test cases. At the same time, the image quality stitched by our proposed method always maintains good quality with different numbers of images.

Although the results we obtained are relatively good, the image stitching process is still taking too much time and the BRIEF descriptor, even though it uses GPU computing, will still have difficulty with large or large image sets. complicated. We will continue to develop other parallel computing methods with suitable algorithms as well as higher accuracy.

7. REFERENCES

1. Petru Potrimba (2023), What is Keypoint Detection?, available at: https://blog.roboflow.com/what-is-keypoint-detection/

2. David G. Lowe (2004), Distinctive Image Features from Scale-Invariant Keypoints, 5 January, available at: https://www.cs.ubc.ca/~lowe/papers/ijcv04.pdf, 7, 10, 12

3. WordPress (2020), DoG, 24 July, available at: https://cvexplained.wordpress.com/2020/07/24/dog/

4. Beau (2022), Tìm hiểu về tương phản (Contrast) và nhận thức thị giác (Visual Perception) trong thiết kế, available at: https://beau.vn/vi/tim-hieu-ve-tuong-phan-contrast-va-nhan-thuc-thi-giac-trong-thiet-ke

5. Lowe, D.G. (1999), Object recognition from local scale-invariant features, In International Conference on Computer Vision, Corfu, Greece, 1150-1157.

6. Harris, C. and Stephens, M. (1988), A combined corner and edge detector, In Fourth Alvey Vision Conference, Manchester, UK, 147-151.

7. Robert C Bolles and Martin A Fischler. A ransac-based approach to model fitting and its application to finding cylinders in range data. In IJCAI, volume 1981, pages 637–643. Citeseer

8. Matthew Brown and David G Lowe. Automatic panoramic image stitching using invariant features. International Journal of Computer Vision

9. E. Rublee, V. Rabaud, K. Konolige and G. Bradski, "ORB: An efficient alternative to SIFT or SURF," *2011 International Conference on Computer Vision*, Barcelona, Spain, 2011.

10. Rosten, Edward; Drummond, Tom (2006). "Machine Learning for High-speed Corner Detection". *Computer Vision – ECCV 2006*. Lecture Notes in Computer Science. Vol. 3951.

11. Calonder, M., Lepetit, V., Strecha, C., Fua, P. (2010). “BRIEF: Binary Robust Independent Elementary Features”. In: Daniilidis, K., Maragos, P., Paragios, N. (eds) Computer Vision – ECCV 2010. ECCV 2010. Lecture Notes in Computer Science, vol 6314.