AIAA SciTech Forum
6-10 January 2025, Orlando, FL
AIAA SCITECH 2025 Forum

10.2514/6.2025-2714

Accelerated Image Stitching Via Parallel Computing for UAV Applications

Rick Ramirez[1], John Korah[2], and Subodh Bhandari[3],
California State Polytechnic University, Pomona, CA 91768, USA

Tu Nguyen[4] and Yuqi Chen[5]
University of California, Irvine, CA 92697, USA

Du D. Le[6]
Mt. San Antonio College, CA 91789, USA

In this work, we formulate an accelerated image fusion algorithm for Unmanned Aerial Vehicle (UAV) application which is based on image stitching using invariant features. By utilizing parallel computing techniques for distributed and shared memory architectures, we have improved both the run time of the feature-based image stitching procedure, and the quality of the results when compared to sequential processing. The images used to generate the panorama were obtained from a UAV at a frame rate of $30Hz$ and a resolution of 3840×2160. As the UAV traverses across the landscape, a video frame buffer is collected and sent to a multi-core Central Processing Unit (CPU) where individual CPU cores perform image fusion in parallel. Our algorithm can be applied to any multi-core CPU (or cluster of multi-core CPU architectures) with support for Graphics Processing Unit (GPU) accelerations. The procedure was tested on various hardware platforms including edge computing devices, ground stations, and high-performance computing environments. The results show that as we increase the number of available CPU cores, there is a decrease in the runtime by as much as 7x under the distributed memory design. There is also a decrease in runtime during the feature detection stage of the fusion process under the shared memory design by a factor of 10x. Finally, by incorporating GPU acceleration at various stages of the stitching procedure, we can decrease the runtime by another factor of 2x. Furthermore, as we increase the number of threads utilized in the shared memory design, we find that there is a significant improvement in the quality of the panorama.

I. Nomenclature

T_s	=	sequential processing runtime
T_p	=	parallel processing runtime
S	=	speedup
C	=	cost
E	=	efficiency
N	=	number of processors
q	=	number of threads

[1] Graduate Student, Computer Science Department.
[2] Assistant Professor, Department of Computer Science.
[3] Professor, Aerospace Engineering Department, and AIAA Associate Fellow.
[4] Undergraduate Student, Computer Science Department.
[5] Undergraduate Student, Informatics Department.
[6] Undergraduate Student, Mathematics and Computer Science Department.

II. Introduction

Unmanned Aerial vehicles (UAV) are widely used in many applications [1] that require remote sensing. In these applications, a UAV can be equipped with a camera to process images and generate a map, either onboard or remotely on a ground station. Real-time image processing is crucial in numerous applications, particularly during natural disasters [2] where rapid changes in the landscape, collapsing buildings, and outdated maps necessitate immediate and accurate updates. The process of feature-based map generation involves combining two or more images together and is referred to as 'image stitching' [3]. An example can be seen in Fig. 1 where 30 4k images are combined to form a cohesive panorama.

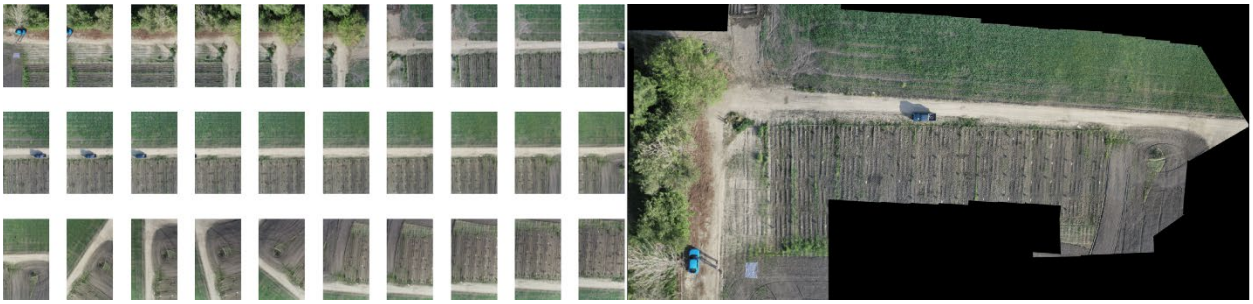

Fig. 1 Panorama generated from UAV video frames.

The image stitching workflow involves analyzing each image for characteristic features that are to be compared with each other. Since the number of features in an image is usually in the order of thousands, the feature detection and matching procedures between two images are computationally expensive tasks that can be very time-consuming. In time-critical applications such as search-and-rescue, it is important to process data within strict time constraints, but as more images are incorporated into the panorama, the runtime increases significantly due to redundant computations, and the quality of the results begins to degrade. The algorithms developed here can make use of all the available resources in a multi-core Central Processing Units (CPU) since most processors today are equipped with multiple cores. We investigate methods of distributing work across multiple processors via a distributed memory parallel programming tool called Message Passing Interface (MPI) [4,5] so as to decrease computation time and limit the number of times the data are transmitted between compute nodes. We also investigate accelerations involving shared memory architecture-based tools such as Open Multi-Processing (OpenMP) and Compute Unified Device Architecture (CUDA). MPI and OpenMP were chosen because of their portability and standardized functionality while CUDA was chosen due to its maturity as an application programming interface (API). We show that by employing various levels of parallelism, there is a corresponding decrease in the time it takes to fuse images as well as an enhancement in the quality of the panorama that is produced, regardless of the computing hardware that is used. Since each UAV and ground station is configured with different hardware components, the framework developed here is designed to be modular and adaptive.

While there have been many fusion algorithms that aim to accelerate the procedure for real-time applications, most of the literature fuse only a small number of images (between 2 and 10 on average) [6–11] or use images of low resolution (less than 3840 x 2160) [7,8,10,11] in their metrics. Many of the procedures developed in previous work also require specific hardware requirements such as a GPU [6,8–10] or even specialized hardware in the form of an FPGA [7]. In this work, we aim to fuse 64 4k images using an assortment of different hardware configurations. Each of the acceleration techniques outlined in Section IV can be utilized in tandem with each other, or as a standalone routine. This provides a great amount of flexibility in the types of hardware that the methodology can accommodate and offers means of choosing between processing onboard the UAV or transferring the video feed for remote processing (provided that the bandwidth of the network can support it).

Figure 2 shows the concept of operation where a UAV scans a landscape with a downward facing camera and the video data are to be transmitted to any single-core CPU, multi-core CPU, or cluster of multi-core CPUs for the fusion process to occur. From Fig. 2, “Black Box” refers to the fact that the image fusion process can take place onboard, on a ground control station, or on a high-performance computing cluster. In many applications involving UAV-generated maps, Global Positioning System (GPS) data are utilized [12] to orient the images relative to each other. Georeferencing an image provides a stable and reliable way to organize incoming images into a unified coordinate system because the procedure can by bypassing the feature detection and matching stages that are required to generate the transformation matrix that warps the images into a common space. However, if that system were to fail, either by

GPS spoofing [13], jamming [14] or a hardware failure, the feature-based approach will sustain map generation, and the algorithms developed in this work will help to ensure the speed and integrity of the results.

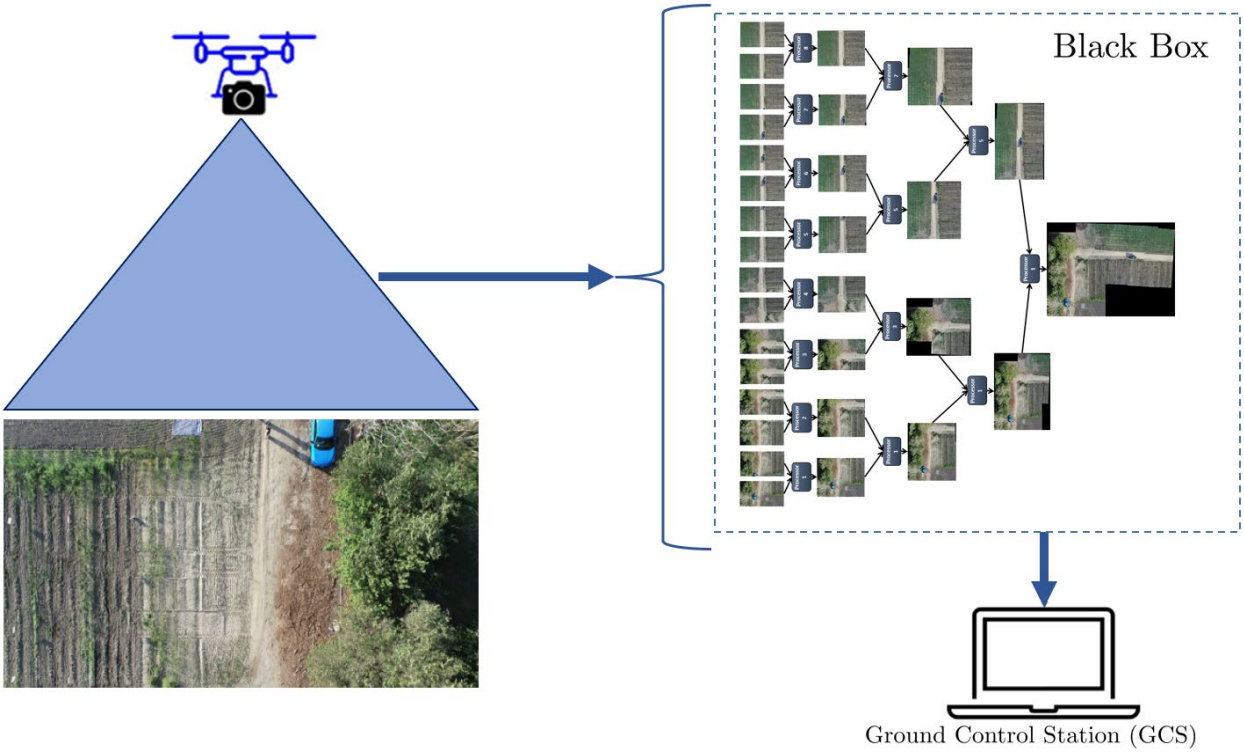

Fig. 2 Concept of operation.

Part of the challenge in feature-based image fusion algorithms [15] is ensuring that the results are not corrupted. This can happen when local features between two images are incorrectly matched, which will cause a miscalculation of the mapping between the coordinate frames of the two images. The method by which the images are captured also has a significant impact on the fusion procedure. Depending on the orientation of the camera, the coordinate system used to transform the incoming images into a unified space can be adjusted according to the way in which the camera is mounted to the aircraft. In this work, we utilize a downward-facing camera and a planar coordinate system. However, if the camera were mounted to the UAV via a gimbal, a different coordinate system would be more suitable. For example, if the camera can pan in the x and y-directions, a cylindrical system would be more suitable whereas a camera that can also move in the z-direction would benefit from spherical coordinates. By tracking features between sequential frames, it is possible to detect changes in the orientation of the camera and adjust the parameters of the algorithm accordingly to produce the best possible results.

Another challenge faced in this work is that the original implementation for image stitching by Ref. [3] assumed that the scene to be mapped is approximately an infinite distance from the camera lens such that there is no parallax and that the homography among all the images is calculated only once. The low altitude of the aircraft and the delayed arrival of frames directly violate these assumptions; however, we find that in most conditions, the results remain intact. Once the panorama has been generated, as shown in Fig. 2, it can then be transmitted to a ground station for remote viewing.

We address these challenges in the following sections which are organized as follows. Section III expands on the methodology of image stitching, Section IV explores our approach to performing image stitching in parallel for both shared and distributed memory architectures, Section V shows the results of our methodology, and Section VI closes with concluding thoughts and plans for future work.

III. Image Stitching Methodology

The image stitching workflow developed in Ref. [3] consists of four general stages. It begins by performing Scale Invariant Feature Transform (SIFT) detection to obtain unique features in each image [7]. The features are then compared and matched to the features in every other image. From these matches, a projective transformation [12,16] can be approximated, the solution of which is optimized to best agree with the features that were matched in the images. By holding one image as a frame of reference, every other image can be warped to the same coordinate system, thus producing a cohesive panorama. The final stage of the stitching procedure is to blend any harsh seams that may appear in the composite image. Figure 3 outlines the various stages of the image stitching workflow, each of which is expanded on below.

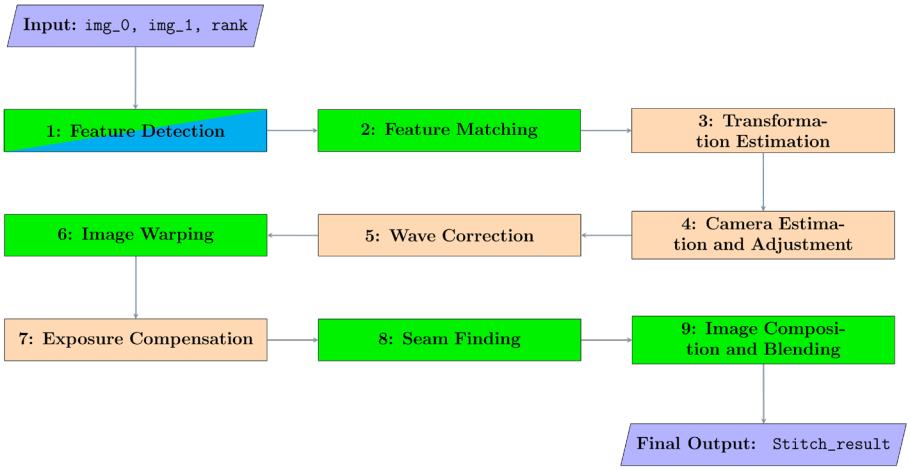

Fig. 3 Image stitching workflow.

1) Feature Detection

There are three common feature detectors available in our implementation.

a) Scale-invariant Feature Transform (SIFT): The original conception by Ref. [3] employed SIFT, which consists of detecting scale-space extrema and keypoint localization followed by orientation assignment and a 128-dimensional keypoint and descriptor construction. Once the keypoints and their descriptors are computed, they can be compared across images.

b) Speeded-Up Robust Features (SURF): The original implementation of the work in this paper utilized SURF for its enhanced speed compared to SIFT. It also has support for CUDA enhancement natively in OpenCV. The procedure begins by detecting a scale-space Hessian matrix followed by keypoint localization, orientation assignment and a 64-dimensional descriptor assignment. Given that the descriptors between SURF are half the size of the SIFT descriptors, there is an inherent speed boost when SURF is employed. This option also utilizes box filters and Haar wavelets, making it more suitable for real-time applications, however, the algorithm is currently patented and not suitable for commercial use.

c) Oriented FAST and Rotated BRIEF (ORB): ORB is a fast and efficient feature detection and description algorithm which combines two techniques: FAST for feature detection and BRIEF for feature description, with enhancements for rotation invariance and robustness. The procedure begins by utilizing FAST to perform keypoint detection followed by keypoint orientation assignment. Descriptors can then be extracted using BRIEF by sampling pairs of pixels in a circular region around the keypoint and generating a binary descriptor. Since ORB is opensource and designed for high efficiency while maintaining rotational invariance in real-time applications with limited resources, this work has opted for ORB in the results below.

2) Feature Matching

Pairwise matching of the descriptors between any two images can be performed in a number of ways. Our implementation offers three options:

a) Best of two nearest matching: Finds the two best matches for each feature and leaves the best one only if the ratio between descriptor distances is greater than a predefined confidence threshold.

b) Affine best of two nearest matching: Similar to best of two neatest matching with the expectation of an affine transformation between images.

c) Best of two nearest range matching: Similar to the two methods above with the restriction of sampling within a restricted location in the image. For more complex models, a homography matrix should be used; especially when the camera viewpoint changes, and the scene appears distorted.

3) Transformation Estimation

There are two options available to describe a projective transformation that relates the coordinates of points in one image to the coordinates of points in another image, assuming the scene is planar, or the transformation can be modeled as such. For simple transformations that do not exhibit perspective effects, a more restrictive affine transformation can be utilized.

4) Camera Estimation and Adjustment

As the UAV traverses across the landscape, the orientation of the camera is likely to change. This can be estimated by taking the features and pairwise matches between all images and estimating the rotations of all cameras with the goal of improving the accuracy of both the camera's internal parameters (such as focal length, lens distortion, and position) and the 3D scene geometry (such as the locations of feature points in the world). The procedure outlined by Ref. [3] assumes that the scene that is to be formed into a panorama is approximately an infinite distance from the camera. This is not the case for UAV acquired images with a downward facing camera. Nevertheless, the procedure is fairly forgiving in this regard.

5) Wave Correction

Since the homography and bundle adjustments are estimated and recalculated for each incoming video frame, there is often a geometrical misalignment that might appear along the stitching seams which can be addressed with a wave correction by globally adjusting the image grid to reduce these distortions. It essentially smooths the geometry of the image in a way that makes the seam artifacts between images less noticeable.

6) Image Warping

The implementation offers 16 warping schemes, 3 of which can be GPU accelerated. The 3 most common use cases utilized in this work are:

a) Plane Warping: A perspective warping is ideal for general-purpose image stitching with scenes that have perspective (e.g., architecture, detailed wide-angle shots). However, this method can introduce perspective distortion, especially on wide scenes or images with depth variation.
b) Cylindrical Warping: This option is ideal for stitching horizontal panoramas where the camera rotates around the vertical axis (e.g., 360° panoramas). Vertical distortion is minimized, but horizontal stretching may occur near the edges which tends to get worse as more images are added to the panorama.
c) Spherical Warping: This option is ideal for spherical panoramas or 360-degree photos, where the camera rotates around its center from a fixed viewpoint with no depth changes. This is rarely the case for UAV acquired images.

All of the above image warping options detailed above are offered with GPU support.

7) Exposure Compensation

When multiple images are taken from different angles or under different lighting conditions, there are likely varying brightness levels. This can cause harsh seams, especially in images with smooth gradients. The exposure of each image can be adjusted so that all images appear uniformly lit.

8) Seam Finding

A seam is the boundary line where two images are combined into a larger image. The goal of seam finding is to identify the best seam where the blending between images will be least noticeable and visually pleasing. This is done by representing each image as a graph where each pixel is a node. The edges between nodes represent the cost of transitioning from one pixel to another with the goal of minimizing the cost path between images. In most cases of UAV acquired images, there will still be a jagged seam that can be cleaned further in the blending stage of the stitching process.

9) Image Composition and Blending

During the composition stage, the images are warped via the transformation matrix that was calculated previously. The exposure compensator and seam finder can be used to adjust the images and minimize artifacts. Any harsh discrepancies between the images can further be cleaned with a number of blending techniques to obtain a smoother gradient across the panorama. The original concept of the stitching procedure assumed that this process would occur only once for a given set of images. However, since a real-time video feed provides images to be processed as a function of time, the incoming frames need to be incorporated sequentially. As a consequence, there is often a blurriness that propagates across the panorama as more images are incorporated into the result. To address this, our distributed memory approach to image stitching can be used to minimize overprocessing by a factor of $Log(N)$ where N is the number of available processing cores.

By exploiting the asynchronous nature of fusing a collection of images, the entire stitching workflow can be accelerated by distributing calculations across multiple compute nodes as outlined in the distributed memory design in Section IV. Under this approach, the results of each intermediate task is communicated between compute nodes and the computations are evenly distributed when the number of images to be processed is equal to half of the number of CPU cores available, as shown in Eq. 1.

$$N = \frac{I}{2} \tag{1}$$

where I is the number of images and N is the number of processes. The individual stages of the workflow can be further accelerated by utilizing finer grain levels of parallelism since each compute node utilizes a shared memory workspace. For example, the feature detection stage of the stitching workflow is considered to be "embarrassingly parallel" because the problem of detecting feature points can be split into individual tasks with no requirement of communicating between tasks. Similarly, the workload of the feature matching stage is perfectly parallel since each feature point can be simultaneously compared to every other feature point between two images.

The various stages of the workflow in Fig. 3 are color coded to reflect the type of accelerations that we have explored. From Fig. 3, green labels indicate that the particular stage can be performed with GPU support, while the blue label indicates the option for OpenMP threading. For example, the first stage can be accelerated by the shared memory design outlined Section IV, where each image is divided into sub-sections to be processed by individual threads or accelerated with a GPU if that hardware is available. By offering different levels of parallelism, workflow accelerations can be adapted to increase performance in any computing environment.

With the details of the image stitching workflow outlined and some insight into the interdependency between various tasks, we are ready to provide details in our approach to choreographing image stitching among a set or processors. In the next section we expand on the motivation for our approach to performing image stitching in parallel and present our algorithm designs for both shared and distributed memory considerations.

IV. Parallel Processing Designs for Image Stitching

The motivation for this work comes from the fact that as more images are incorporated into the panorama, there is an increase in the computation time that a single processor must execute. A purely feature-based approach to image fusion requires that each new image be compared to every other image in the set, as we are not guaranteed that the new incoming image belongs to a particular region of the partly generated map. For example, as images are fused together, the camera can abruptly change its orientation, which will cause a disjoint in the panorama - meaning the new image will have no overlap with the previous one. Obviously, the components of the composite image must be continuous for a feature-based approach to work, but since we are not guaranteed that incoming images belong to a particular location in the map, every feature detected in a new image should be compared to all previously calculated feature points. The accelerated algorithms described below do not consider this kind of situation, however we still compare all new incoming features with all previously generated ones. We also assume that each image collected from the video feed belongs in the order that it is received in relation to the partly generated map. This is critical for the distributed memory design, but not required for the shared memory design or GPU accelerations.

As the images are fused together, the composite image will increase in size with each iteration, which requires more and more pixels to be analyzed against each new incoming frame. As a result, the time it takes to fuse each new image increases as in Table 1, which shows the fusion time for 10 images on a single processor. As more images are added to the resulting panorama, the runtime increases without bound.

Table 1 Sequential processing runtime for image stitching algorithm.

Fusion of Images $I_0 \dots I_n$	$I_0 + I_1$	$I_{01} + I_2$	$I_{012} + I_3$	$I_{0\dots3} + I_4$	$I_{0\dots4} + I_5$	$I_{0\dots5} + I_6$	$I_{0\dots6} + I_7$	$I_{0\dots7} + I_8$	$I_{0\dots8} + I_9$	Overall
Feature Detection	0.526	0.130	0.112	0.099	0.094	0.089	0.082	0.080	0.077	1.288
Feature Matching	0.354	0.288	0.253	0.213	0.197	0.179	0.160	0.150	0.135	1.929
Warping	0.004	0.003	0.002	0.002	0.002	0.002	0.002	0.002	0.002	0.022
Compositing	1.036	1.187	1.423	1.680	1.872	2.056	2.401	2.672	2.876	17.203
Total Time:	1.919	1.607	1.791	1.994	2.165	2.326	2.645	2.904	3.090	20.442

Another issue that motivates this work is the quality of the results. As mentioned, when two images are fused together, there is likely going to be a harsh seam that can be removed with blending techniques. If the images are to be fused as a single batch, this does not present an issue. But for images obtained from UAV video data, the blending stage begins to degrade the resulting panorama as more images are incorporated into the result. The left image in Fig. 4 shows an example of this, where the images are fused, starting from the left and progressing to the right. As each new image is added to the right end of the composite, the portions to the left end begin to be overprocessed, resulting in a blur that begins to propagate across the scene. The right image in Fig. 4 shows the same results after applying our distribute memory acceleration techniques.

Fig. 4 Serial processing results (left) versus Parallel processing results (right).

To address both issues described above, we have developed a tree-based algorithm which utilizes all available CPU cores that a given hardware might have available. This effectively reduces the computational complexity of the fusion process as well as enhances the quality of the blending stage of the fusion algorithm. Furthermore, we have incorporated a shared memory design of the original stitching algorithm developed in Ref. [3] to further decrease the runtime.

It should be noted that fusing images in parallel does not guarantee quality results. In general, the more processor cores involved in the calculation, the less reliable the results can be [4]. This is indeed the case for this work. Processing subsections of the map separately can cause miscalculations in the global homography. But under our shared memory architecture approach to feature detection, we actually see an improvement in the quality of the resulting scene.

A. Distributed Memory Design

Under the distributed memory design, each process utilizes its own memory space and executes independently from each other. This allows each node (processor core, or cluster of processor cores) to work on a subsection of the larger composite image asynchronously. To communicate with each other, each node is assigned a label or 'rank' which is used to send and receive information to/from other nodes. We employ a distributed memory framework called Message Passing Interface (MPI) which is a portable message-passing standard designed to function on parallel computing architectures. For this work, we use the terms node, process, and processor interchangeably, but it should be noted that the algorithm does support nodes that consist of multiple processor cores.

The procedure begins by collecting images from video data taken from a UAV. The images are then divided evenly among the group of nodes. Each node will then use the stitching algorithm to fuse its initial images. Once that is completed, depending on the rank of the node, it will either transmit its result to an adjacent node, or receive a result from an adjacent node. The senders and receivers alternate for each round of the computation and at each round, half of the processors become idle. In the last stage of the algorithm, the final two ranks send and receive their result for the final stitching to occur on a single node. Figure 5 shows an example of the distributed memory design for a set of

16 images and 8 processor cores. By the distributed memory architecture of the CPU, we can increase the throughput of the traditional feature-based image fusion algorithm. Depending on the number of CPU cores available, our algorithm will adjust itself to stitch multiple images among all available cores which decreases the computational complexity from $\mathcal{O}(\mathrm{n})$ for the sequential algorithm to $\mathcal{O}(log(n))$ under the distributed memory design. This approach also decreases the amount of overprocessing that each video frame must go through by a factor of $\mathcal{O}(log(n))$, which produces clearer results as shown in Fig. 4.

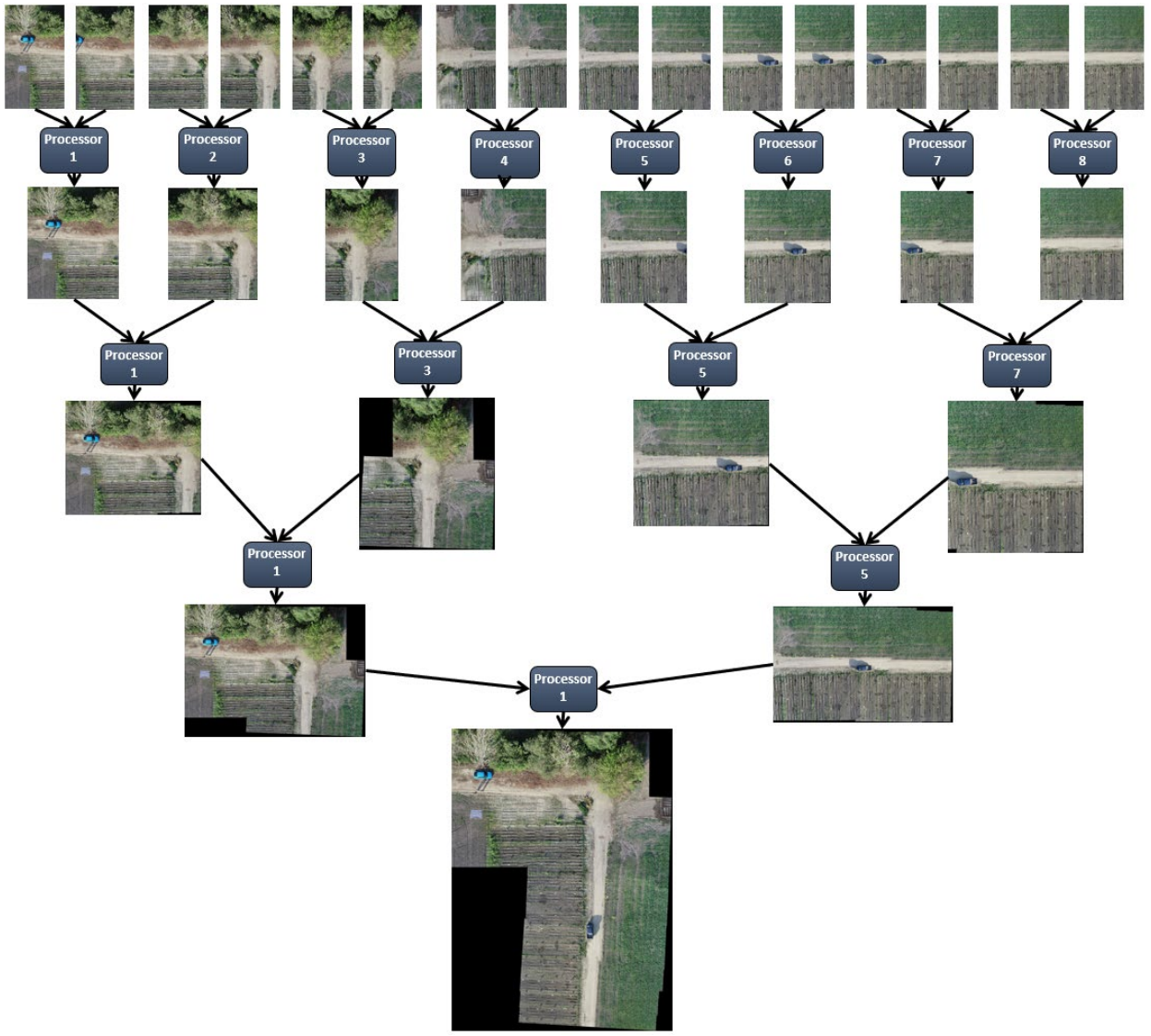

Fig. 5 Tree-based image fusion algorithm.

Tree Based Algorithm

Input: $\boldsymbol{Images\ \ A(m \times n), B(m \times n), Set\ of\ processors\ Q = \{Q_0, Q_1, \ldots, Q_{N-1}\}}$

Output: $\boldsymbol{D}(u \times \boldsymbol{v})$

1 Function: $\boldsymbol{Tree\_Stitch\_Alg(A, B, Q)}$

2 Stride $= 1$

3 for $\boldsymbol{i = 0\ \ to\ \ log(N)}$ do

4 $\quad \boldsymbol{list\_of\_active\_processors} = \{\}$

5 $\quad \boldsymbol{list\_of\_senders} = \{\}$

```
    list_of_receivers = {}
    for k = 0 to N − 1 do
        list_of_active_processors += {k}
    for k = 0 to list_of_active_processors.length − 1 do
        if k % 2 == 0 then
            list_of_receivers += k
        else
            list_of_senders += k
    if Q(rank) belongs to list_of_senders then
        MPI_SEND to Q(rank − 2^(stride−1))
    else if Q(rank) belongs to list_of_receivers then
        MPI_RECV from Q(rank + 2^(stride−1))
        IMAGE_STITCH
    stride ++
return D
```

B. Shared Memory Designs

Every stage of the feature-based image fusion algorithm offers an opportunity to accelerate the process with a finer grain level of parallelism. Because Message Passing Interface does not explicitly leverage the strengths of shared memory, the internal components of the stitching procedure cannot easily be parallelized with the distributed memory design. Therefore, we have utilized a shared memory design for the feature detection stage of the image stitching procedure that can be run in tandem with the distributed memory algorithm outlined above. Figure 6 shows that under this approach, when a node receives two images, they can be partitioned into regions that individual threads can work on simultaneously. In this case, 8 threads are used to process two 4K images, effectively reducing the number of pixels that a single thread would need to consider by one-quarter.

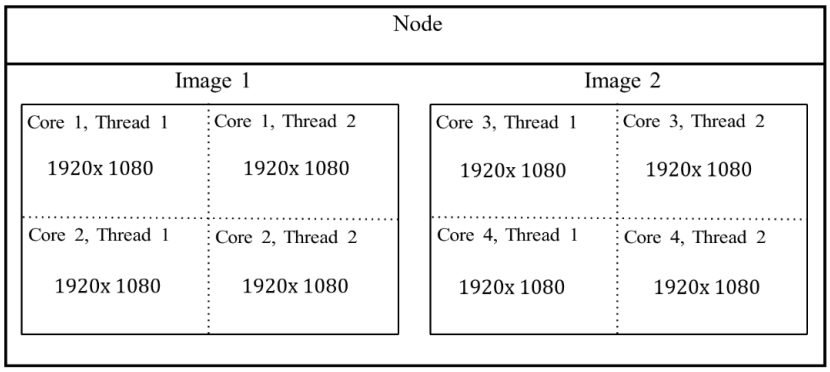

Fig. 6 A single node processing two 4k images with multi-threading.

The number of threads that a system can run in parallel is not guaranteed. Most systems today offer two threads per CPU core. Figure 7 shows the general case of the shared memory design. In this case, the images are partitioned vertically. This decision was made to simplify the algorithm design, but the partitioning method in Fig. 6 is also suitable. By leveraging the "embarrassingly parallelizable" nature of the feature detection stage of the image stitching workflow [3], we can break the procedure into smaller sub-problems to decrease the computational complexity from $\mathcal{O}(mn)$ [17] for the sequential algorithm to $\mathcal{O}\left(m\frac{n}{q} + \frac{\text{total\_features}}{q}\right)$ for the shared memory design,

where m and n are the dimensions of the image, q is the number of threads and $total\_features$ is defined beforehand by the user.

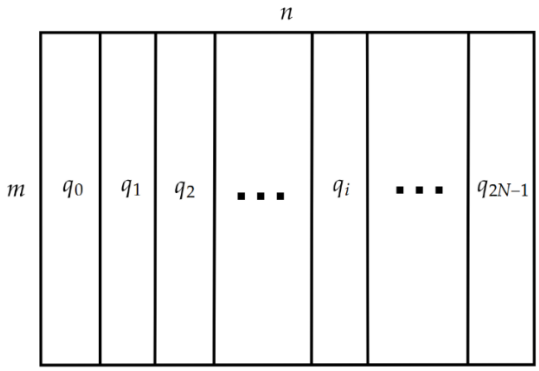

Fig. 7 Parallel feature detection over an m x n (in pixels) image performed by q threads, where q_i corresponds to the i^{th} thread and N is the number of processor cores.

To ensure portability, we have opted to utilize Open Multi-Processing (OpenMP) for the implementation of the shared memory design since it supports shared-memory multiprocessing programming in C, C++, and Fortran, on multiple platforms, instruction-set architectures and operating systems. This approach offers a scalable platform that can be built as a hybrid model alongside MPI.

The details of the shared memory design can be seen in the algorithm below. The procedure begins by partitioning each incoming image into a number of smaller regions equal to half of the number of threads available on a particular compute node. Once the features are detected by each thread, the x-position of the keypoints are to be adjusted to reflect the location of detection with respect to the thread that detected it. Note that if the partitioning system shown in Fig. 6 were used, the y-positions would also need to be corrected. Once all keypoints have been processed, they are concatenated into a final vector that can then be used in the feature matching stage of the workflow. Figure 8 shows an example where each thread is color-coded and assigned a specific region of the image to perform feature detection based on the individual thread-ID.

Fig. 8 Feature points detected via individual threads.

Parallel Feature Detection

Input: $Images\ A(m \times n), B(m \times n),\ Set\ of\ threads\ q = \{q_0, q_1, \dots, q_i\}$
Output: $D(u \times v)$
1 Function $IMG\_STITCH\_PARALLEL\_FEATURE\_DETECT(A, B, q)$
2 for 0 to num_images do

```
        for each thread q_i do
            tid = omp_get_thread_num()
            start_col = tid * (full_img.cols / num_threads)
            end_col = (tid + 1) * (full_img.cols / num_threads )
            roi(start_col, 0, end_col − start_col, ful_img.rows)
        for 0 to roi do
            FEATURE_DETECTION(temp_features[tid])
        for keypoint : temp_features[tid].keypoints do
            keypoint.pt.x += tid * (full_img.cols / num_threads)
            features[i].keypoints.push_back(keypoint)
        for 0 to num_threads do
            descriptor_vector.push_back(tem_features[tid].descriptors)
    concat(descriptor_vector, features[i].descriptors)
    FEATURE_MATCH
    IMAGE_WARP
    IMAGE_COMPOSITION
    return D
```

C. GPU Accelerations

As outlined in Fig. 3, various stages of the stitching algorithm can be accelerated with a Graphics Processing Unit if that resource is available. In this work we utilize the OpenCV implementations of GPU accelerated functionality. For example, the feature detection stage of the image stitching workflow can use CUDA supported versions of ORB and SURF. If there is no GPU access, the implementation will default to CPU usage. The results in Section IV C show a comparison of performance between computing the image stitching workflow on a CPU against a CPU with GPU support.

In this section we expanded on our design to distributing the workload of fusing a set of images across multiple processes and explored how we can use the shared memory of each node to perform subtasks in parallel. With each of the algorithm designs defined, we can proceed with details of our implementation and present results for both the shared and distributed memory approach to image stitching, as well as the results of a hybrid model that combines both techniques.

V. Experimental Results

The data set used to benchmark the algorithms outlined in Section IV consists of Open Computer Vision (OpenCV) library's Mat objects, which represent an n-dimensional numerical multi-channel array. Each test run will fuse 64 4k images. The metrics used to measure performance will consist of runtime, speedup, cost, and efficiency defined below. Sections V subsections A, B and C record the impact of the distributed memory design (utilizing MPI), shared memory design (utilizing OpenMP) and hybrid design (utilizing MPI and OpenMP) on the algorithm's speed respectively. To ensure optimal performance, C++ was chosen as the implementation language and the tested hardware platforms are shown in Table 2 below.

Table 2 Testing hardware configurations.

	First Setup	Seconds Setup	Third Setup	Fourth Setup
CPU	Intel i7-8750H	Intel i9-13900KF	Intel 17-7700K	Xeon Gold
GPU	GTX 1070 Max-Q	RTX 4060Ti	GTX 1070	Tesla-v100
OpenCV Ver.	OpenCV 4.10.0	OpenCV 4.10.0	OpenCV 4.5.5	OpenCV 4.5.5
RAM	16GB	32GB	64GB	192GB
GPU Memory	8GB	16GB	8GB	16GB

It should be noted that the number of available processors is not directly proportional to the decrease in runtime but depends on the percentage of the program that can be parallelized as described by Amdahl's law,

$$S = \frac{1}{(1-P)+\frac{P}{N}} \tag{2}$$

where S is the speedup ratio, N is the number of processor cores, P is the proportion of the task that can be parallelized and $(1 - P)$ is the proportion of tasks that must be performed sequentially. The four metrics used to in our analysis are:

1) *Serial and Parallel Runtime:* The serial and parallel runtime of a program are defined as follows:
    1) The Serial runtime of a program is the time between the beginning and the end of its execution on a sequential processor. We denote this by T_s.
    2) Parallel runtime is the time from the moment the first processor begins its execution to the moment the last processor ends its execution. We denote this by T_p.
2) *Speedup:* The overall speedup is defined as the ratio between the serial runtime to the time taken by the parallel runtime,

$$S = \frac{T_s}{T_p} \tag{3}$$

The speedup of a parallel algorithm measures how much faster an algorithm is than its sequential counterpart.

3) *Cost:* The cost of processing a program on a parallel system is defined as the product of runtime and the number of processes,

$$C = T_p * N \tag{4}$$

4) Efficiency: The efficiency of a parallel program that uses N processes is defined by

$$E = \frac{T_s}{N*T_p} = \frac{S}{N} \tag{5}$$

The benchmarks below are the result of fusing 64 sequential video frames. However, the resulting images shown in Subsections A and C were produced by fusing 64 preselected video frames while the resulting images in Subsection B were produced by fusing 64 sequential frames. The distance traveled between frames depends on the speed of the aircraft, and in general, the scene does not change significantly between frames. Therefore, in order to showcase the fusion process for a prolonged flight, the preselected frames are presented.

A. Results for Distributed Memory Design

By employing Message Passing Interface with the image stitching workflow, we can reduce the runtime of fusing 64 4k images from approximately 205 seconds to about 27 seconds as shown in Fig. 9a. This equates to a speedup of 7x as shown in Fig. 9b and was achieved using a setup consisting of an Intel Cascade-Lake CPU paired with a Nvidia Tesla-v100 GPU (fourth setup in Table 2). It should be noted that SURF was used in producing these results while the results of the remaining sub-sections utilized ORB. Because with each iteration, half of the compute nodes go idle, the cost in Fig. 9c increases severely followed by a reduction in efficiency as shown in Fig. 9d. The result of fusing 64 preselected nonsequential frames is shown in Fig. 10.

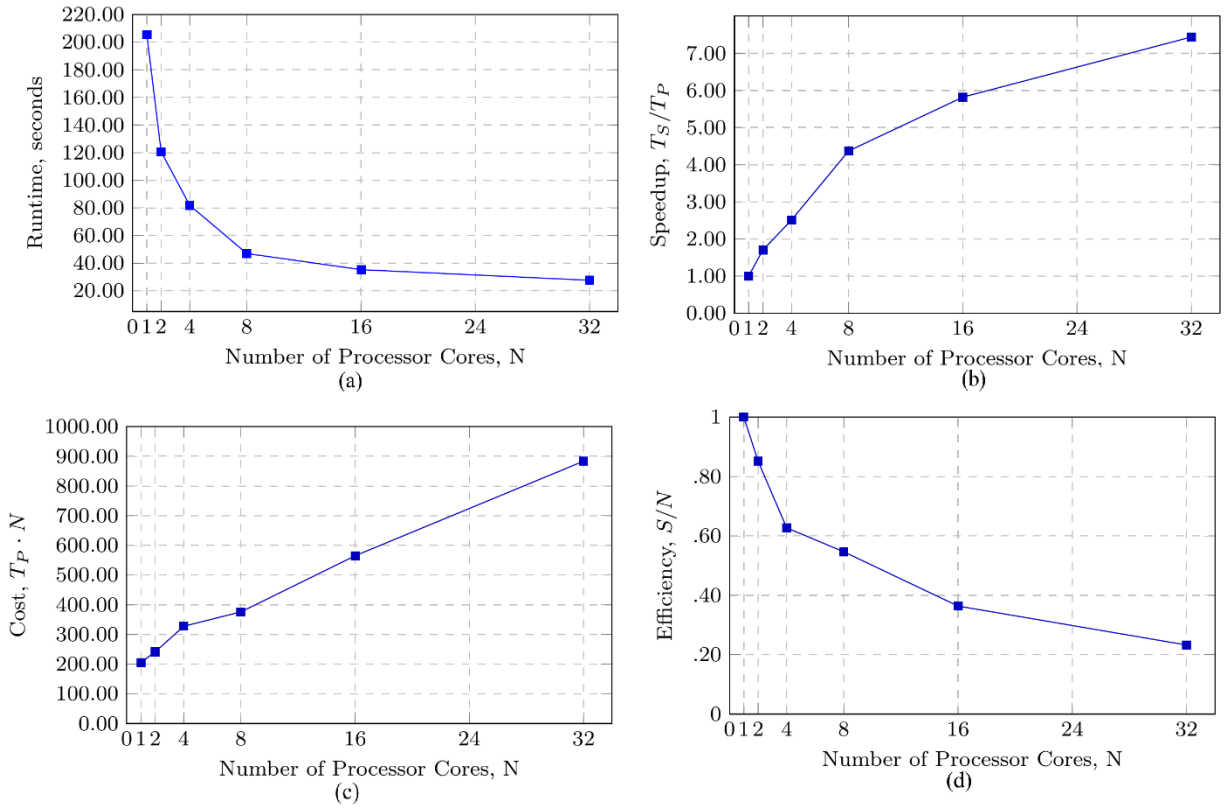

Fig. 9 Runtime, speedup, cost and efficiency of fusing 64 sequential 4K video frames for increasing MPI node count.

Fig. 10 Result of fusing 64 preselected video frames.

B. Results for Shared Memory Design

As shown in Fig. 11a, by increasing the number of threads that detect features on a single video frame, we can decrease the computation time of feature detection by more than 10x. These data were measured on a single Intel i9-12900H CPU and Nvidia RTX 3070Ti GPU (second setup in Table 2). Similar to the distributed memory design, there is an increase in cost and decrease in efficiency with increasing thread count.

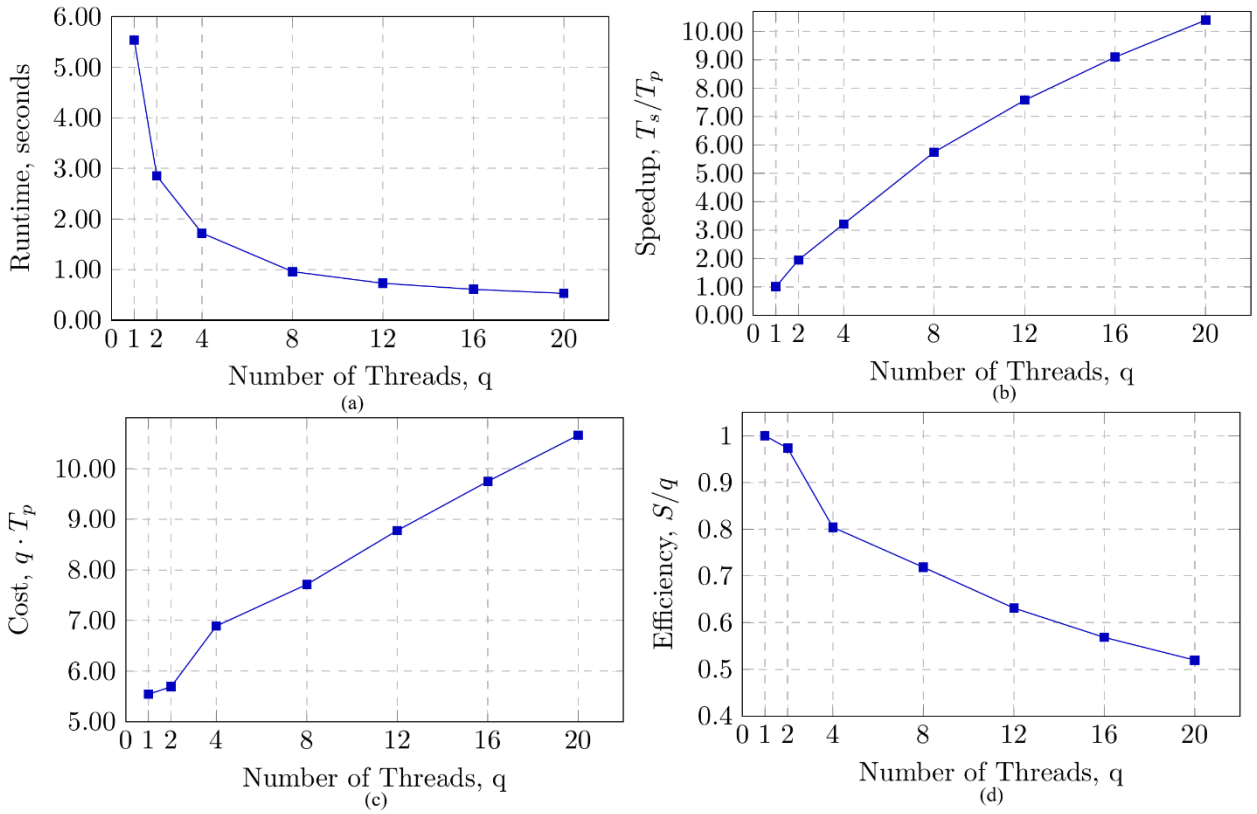

Fig. 11 Parallel feature detection runtime, speedup, cost, and efficiency for increasing thread count.

It is interesting to note that, as shown in Fig. 14 in Subsection IV C, while the total computation time of the image stitching workflow is not greatly decreased with the shared memory design, there is a significant improvement in the quality of the results. The leftmost image of Fig. 12 shows the result of fusing 64 sequential video frames with a single thread. The artifacts consisting of black streaks in the first three results are common when the images that are fused together are too similar. From Fig. 12, moving from left to right, we see that by increasing the thread count during the feature detection stage of the image stitching workflow, there is a corresponding increase in the overall quality of the results.

Fig. 12 Result of using 1, 4, 8, and 16 OpenMP threads respectively with a single CPU core to fuse 64 sequential 4k video frames.

C. Results for the Hybrid Design

By integrating both the distributed memory and shared memory designs with the image stitching workflow, we can decrease the computational complexity of the fusion algorithm while also preserving the quality of the results. Fig. 13a shows the runtime of fusing 64 sequential video frames under the hybrid design for increasing processor core/thread count. These results were produced using the second setup in table 2. Note the dip in speedup after 16

cores are used. This is because the Intel i9-13900KF has 24 cores. A processor with more cores will not show this behavior. Again, we see that there is a significant increase in speedup as shown in Fig. 13b. There is also a corresponding increase in cost and decrease in efficiency as before. Also shown in Fig. 13 are metrics for the GPU accelerations outlined in Fig. 3 in tandem with the distributed memory design. We see a 2x reduction in the computation time when compared to performance on a CPU and a significant reduction in cost when the GPU is enabled. However, if that particular hardware were unavailable, we can still decrease the computation with the methods outlined above.

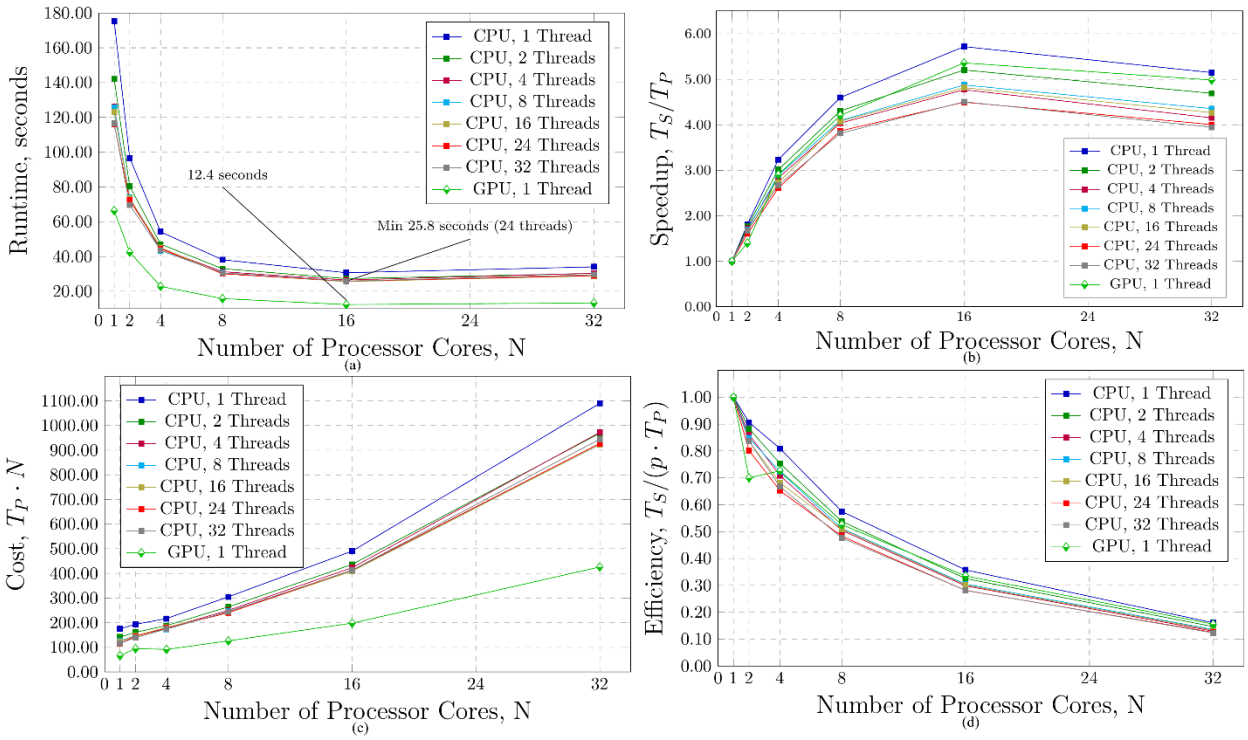

Fig. 13 Runtime, speedup, cost and efficiency of the hybrid design as a function of thread/CPU core count.

Figure 14 shows the results of fusing 64 preselected video frames under the hybrid design. These results agree with those in Fig. 10, which utilized only the distributed memory design.

Fig. 14 Result of fusing 16 video frames under the hybrid design.

In this section we presented and analyzed the results of the distributed memory, shared memory, and hybrid design to image stitching and compared those results to performance with and without GPU support. We found that as we incorporate more levels of parallelism and break the workflow into smaller subtasks, there is an decrease in the time it takes to produce results. We also found that as we increase the thread count of the shared memory approach to feature detection, there is an improvement in the quality of the results that are produced. In the next section, we provide some final thoughts and plans for future work.

VI. Conclusion

By utilizing parallel computing techniques for distributed and shared memory architecture, we have been able to reduce the runtime of feature-based image fusion while preserving the integrity of the panoramic images that are generated. We find that the distributed memory design helps reduce redundant calculations while the shared memory design helps to preserve the quality of the results. Overall, the best performance observed comes from utilizing the distributed memory approach along with GPU accelerations. The algorithms developed in this work will not only accelerate the image fusion process on any single-core CPU, multi-core CPU, or cluster of multi-core CPUs, but will also provide stability in map generation should GPS data become unavailable during UAV flight.

Future work will involve expanding the shared memory design to include more components of the image stitching workflow as well as integrating the GPU with the shared memory design. As of now, the shared memory and GPU accelerations are not designed to run simultaneously. Within the image stitching workflow, we plan to explore more optimizations involving GPU accelerated tools since as of now, only certain operations within the workflow utilize a GPU. We also plan to design a framework for storing, retrieving, and updating larger maps that exceed the memory of the hardware. Lastly, we would like to build a Graphical User Interface (GUI) where a remote user can see a live video feed from the UAV while observing the panorama being generated in real-time.

References

[1] Debnath, D., Vanegas, F., Sandino, J., Hawary, A. F., and Gonzalez, F., "A Review of UAV Path-Planning Algorithms and Obstacle Avoidance Methods for Remote Sensing Applications," *Remote Sensing*, Vol. 16, No. 21, 2024, p. 4019. https://doi.org/10.3390/rs16214019

[2] Bashir, M. H., Ahmad, M., Rizvi, D. R., and El-Latif, A. A. A., "Efficient CNN-Based Disaster Events Classification Using UAV-Aided Images for Emergency Response Application," *Neural Computing and Applications*, Vol. 36, No. 18, 2024, pp. 10599–10612. https://doi.org/10.1007/s00521-024-09610-4

[3] Brown, M., and Lowe, D. G., "Automatic Panoramic Image Stitching Using Invariant Features," *International Journal of Computer Vision*, Vol. 74, No. 1, 2007, pp. 59–73. https://doi.org/10.1007/s11263-006-0002-3

[4] Ramadhan, A. W., Aulia, F., Dewi, N. M. L. A., Winarno, I., and Sukaridhoto, S., "Distributed Aerial Image Stitching on Multiple Processors Using Message Passing Interface," *JOIV : International Journal on Informatics Visualization*, Vol. 8, No. 1, 2024, pp. 409–416. https://doi.org/10.62527/joiv.8.1.1890

[5] Rafael, C. G., and Richard, E. W., "Digital Image Processing (Fourth Edition)," Pearson Education, 2018.

[6] Abughalieh, K., Bataineh, O., and Alawneh, S., "Acceleration of Image Stitching Using Embedded Graphics Processing Unit," presented at the 2018 IEEE International Conference on Electro/Information Technology (EIT), 2018. https://doi.org/10.1109/EIT.2018.8500187

[7] Wang, G., Zhai, Z., Xu, B., and Cheng, Y., "A Parallel Method for Aerial Image Stitching Using ORB Feature Points," presented at the 2017 IEEE/ACIS 16th International Conference on Computer and Information Science (ICIS), 2017. https://doi.org/10.1109/ICIS.2017.7960096

[8] Du, C., Yuan, J., Dong, J., Li, L., Chen, M., and Li, T., "GPU Based Parallel Optimization for Real Time Panoramic Video Stitching," *Pattern Recognition Letters*, Vol. 133, 2020, pp. 62–69. https://doi.org/10.1016/j.patrec.2019.06.018

[9] Nakov, O., Mihaylova, E., Lazarova, M., and Mladenov, V., "Parallel Image Stitching Based on Multithreaded Processing on GPU," presented at the 2018 International Conference on Intelligent and Innovative Computing Applications (ICONIC), 2018. https://doi.org/10.1109/ICONIC.2018.8601253

[10] Yeh, S.-H., and Lai, S.-H., "Real-Time Video Stitching," presented at the 2017 IEEE International Conference on Image Processing (ICIP), 2017. https://doi.org/10.1109/ICIP.2017.8296528

[11] Huang, X., Tang, R., Zhou, Y., Yin, H., and Yan, C., "DSP-Based Parallel Optimization for Real-Time Video Stitching," *Journal of Real-Time Image Processing*, Vol. 20, No. 2, 2023, p. 28. https://doi.org/10.1007/s11554-023-01275-x

[12] Bang, S., Kim, H., and Kim, H., "UAV-Based Automatic Generation of High-Resolution Panorama at a Construction Site with a Focus on Preprocessing for Image Stitching," *Automation in Construction*, Vol. 84, 2017, pp. 70–80. https://doi.org/10.1016/j.autcon.2017.08.031

[13] AlAbidy, A., Zaben, A., Abu-Sharkh, O. M. F., and Noman, H. A., "A Survey on AI-Based Detection Methods of GPS Spoofing Attacks on UAVs," presented at the 2024 IEEE 12th International Conference on Intelligent Systems (IS), 2024. https://doi.org/10.1109/IS61756.2024.10705273

[14] "(PDF) Protecting Autonomous UAVs from GPS Spoofing and Jamming: A Comparative Analysis of Detection and Mitigation Techniques," *ResearchGate*, 2024. https://doi.org/10.9734/jerr/2024/v26i101291

[15] Lyu, W., Zhou, Z., Chen, L., and Zhou, Y., "A Survey on Image and Video Stitching," *Virtual Reality & Intelligent Hardware*, Vol. 1, No. 1, 2019, pp. 55–83. https://doi.org/10.3724/SP.J.2096-5796.2018.0008

[16] Guo, L., Zhu, H., Liu, Y., Sun, X., and Teng, X., "Aerial Image Stitching Based on Fusion of Geographic Coordinates and Image Features," presented at the 2022 IEEE International Conference on Unmanned Systems (ICUS), 2022. https://doi.org/10.1109/ICUS55513.2022.9986913

[17] Laishram, D., and Manglem Singh, K., "A Watermarking Scheme for Source Authentication, Ownership Identification, Tamper Detection and Restoration for Color Medical Images," *Multimedia Tools and Applications*, Vol. 80, No. 16, 2021, pp. 23815–23875. https://doi.org/10.1007/s11042-020-10389-4