# ECE 4772 / 5772:
# High-Performance Embedded Programming

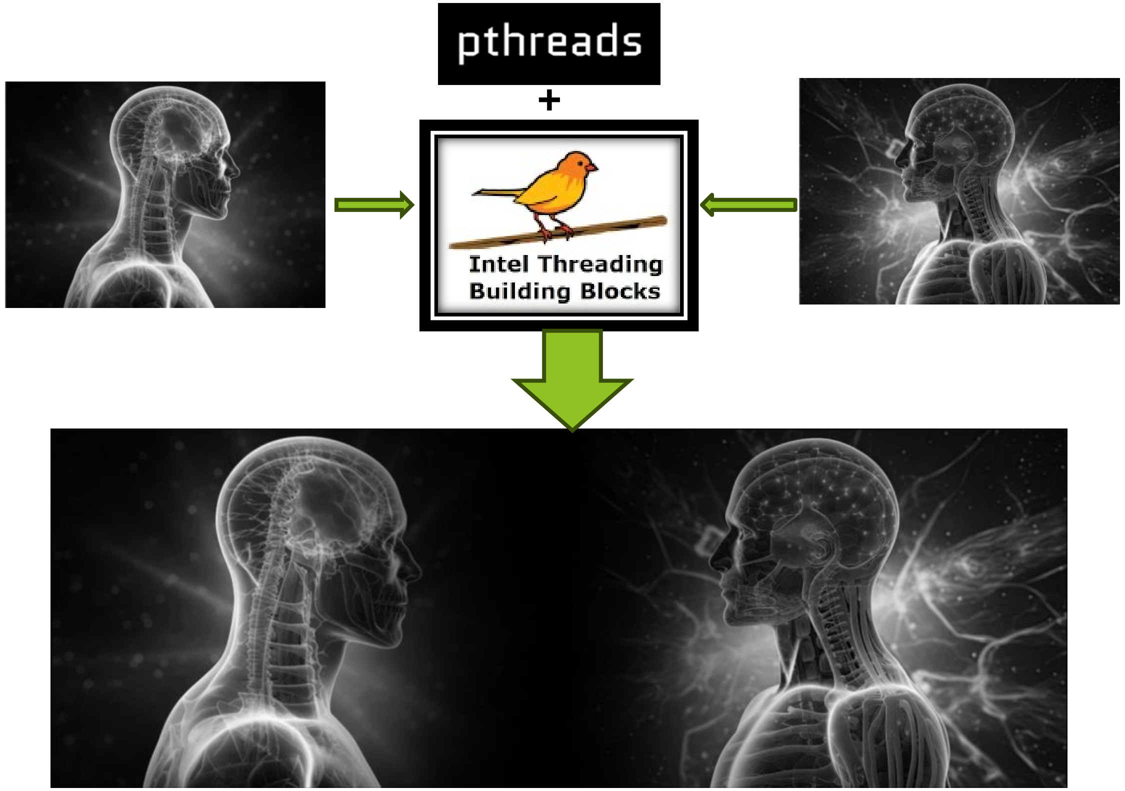
**Semester – Fall 2024**

**Professor: Dr. Daniel Llamocca**

## Project Topic:
## Image Stitching using Parallel programming

**Team:**

- Manoj Verma




OAKLAND UNIVERSITY™

School of Engineering and Computer Science

# Presentation Outline:

- Project Goal
- Introduction to Image-stitching
- Steps
- Image Stitching – Sequential Method
- Image Stitching – TBB Pipeline Method (2D Image)
- Image Stitching - TBB Pipeline (1-D Image)
- Image Stitching – Parallel_For (Row)
- Image Stitching – Parallel_For (Column)
- Image Stitching – Parallel_For (Row + Column)
- Image Stitching – PThreads
- Processing Time and Speedup
- Processing Time measurements for different size of input images
- Live Demo.
- References
- Appendix – Source Code Details
- Questions

# Project Goals:

- The purpose of this project is to develop an algorithm for stitching and blending images using parallel programming. During this semester, I gained a significant amount of knowledge about parallel programming, I have tried to used most of them in this project.

  - Using **Parallel Pipeline** parallel programming techniques.
    - Three stage pipeline stages
  - Using Parallel-For parallel programming techniques.
    - Parallel_For to parallelize Row
    - Parallel_For to parallelize the Column
    - Nested Parallel_For to parallelize both Row and Column.
  - Pthread – Multi threading programming Techniques.
    - Number of threads based on the user input.
    - Default – 50 threads
  - Data Acquisitions and Performance analysis between different programming techniques and speedup.
  - Python Script to Validate the result
    - Developed a python script to validate the results.
    - Demonstrate the stitched image result.

# Introduction Image-stitching:

- Image Stitching is the process of merging multiple images into a single high-resolution or panoramic image. An image blending technique is used in compute vision and image processing fields to create seamless images.

- While my research I found that there are many advance algorithms for both image stitching and blending, however I am focusing in developing an image stitching approach where I could apply the parallel programming techniques to optimize the performance and maximize the throughput.

- The human visual system is highly sensitive to changes in brightness, which helps to discern different visual artifacts. It is quite easy for our visual system to distinguish two different images if we just stitch the image without applying any blending, as shown in the Figure-1, it has no homogenous blending.


Figure-1

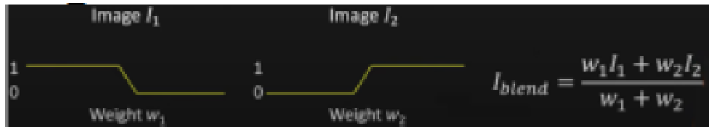


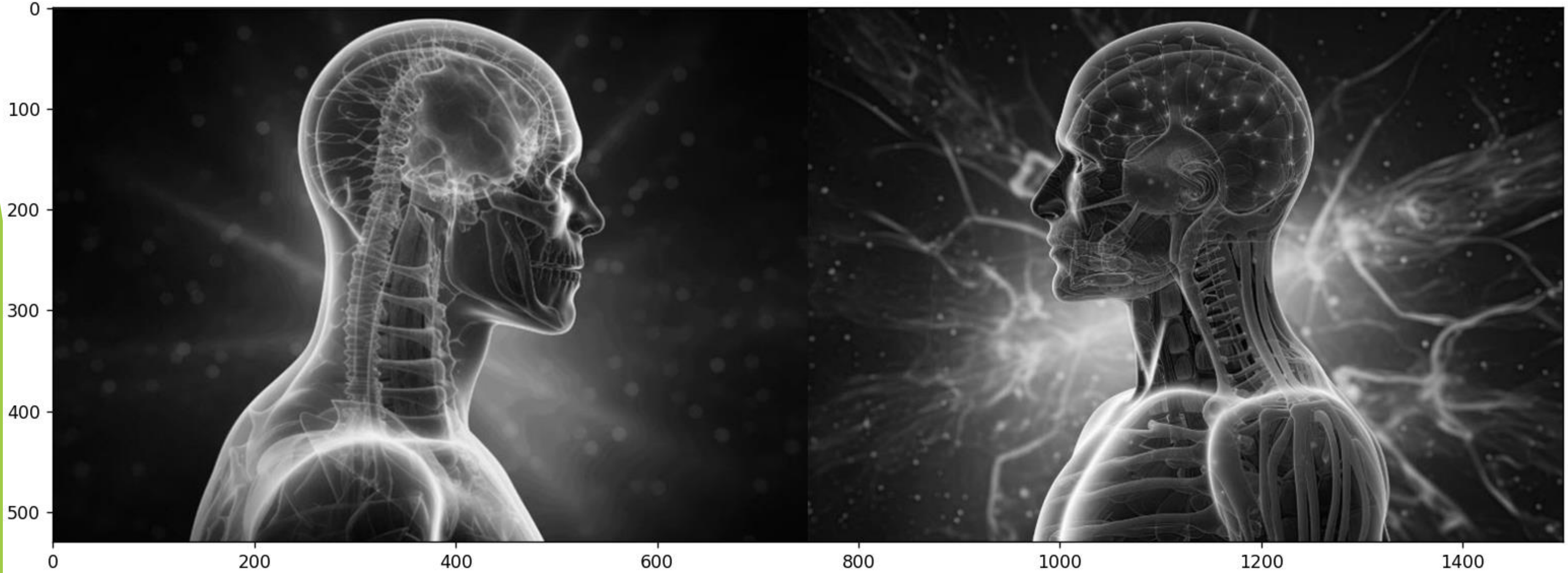

Figure-2

# Images that’s need to be stitched:

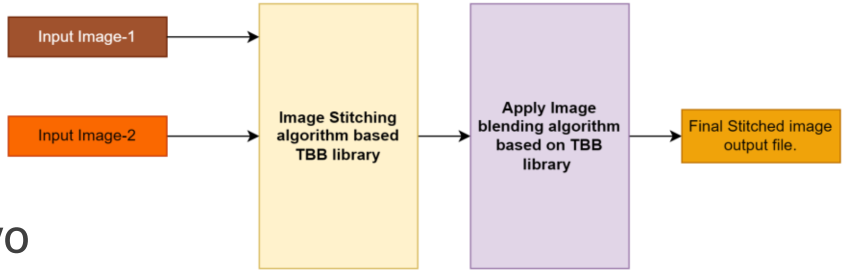
Image-1:
- Width: 750
- Height: 530

Image-2:
- Width: 750
- Height: 530

# Image stitching without weight function:



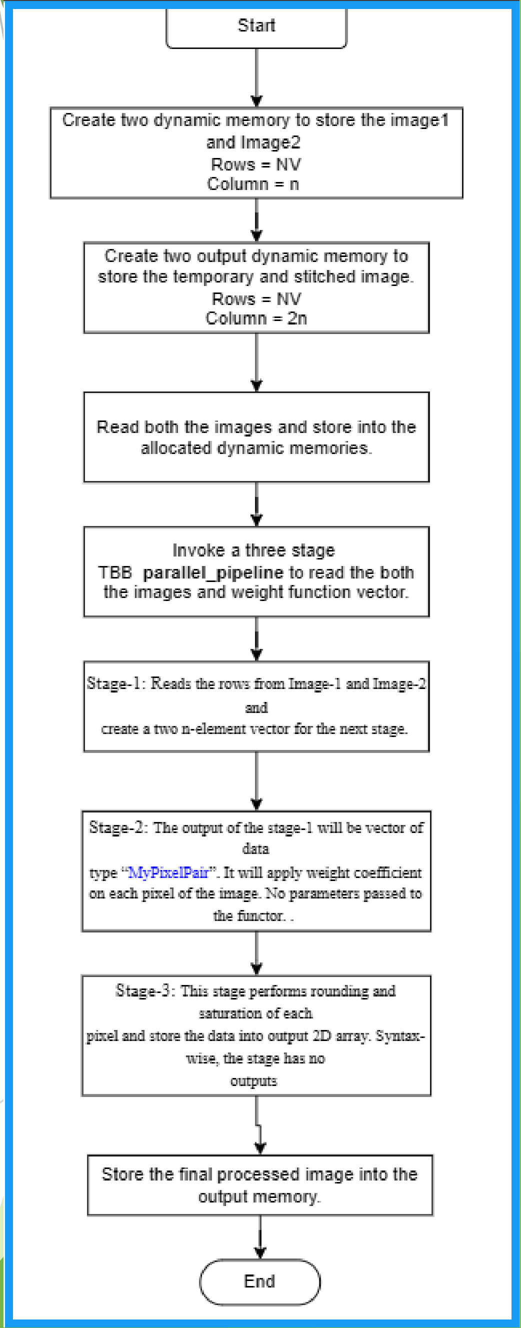
# Steps:



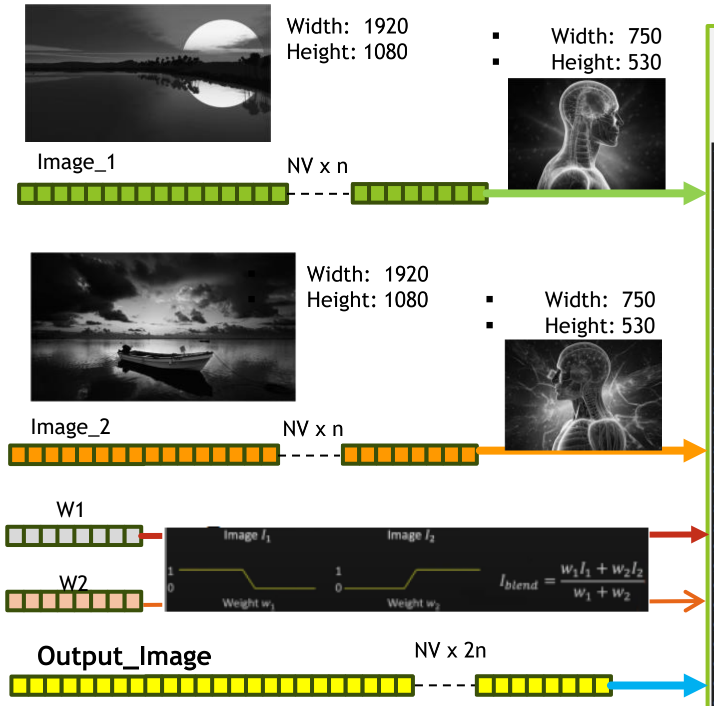

- Read two grayscale images and store into two dynamically allocated memory sections.
- Update weight function vectors w1 and w2 for Image1 and Image2.
- Create dynamic memory sections
- Apply different parallel programming techniques to process both the images.
- Store the intermediate processed image and final stitched output image.
- Write the processed output image into a file.
- De-allocate all the dynamic memories.
- Analyze the stitched image.

```
read_binfile(input_image1, (n*NV), im_filename1, 0);
read_binfile(input_image2, (n*NV), im_filename2, 0);

for (int i = 0; i < NV; i++){

    for (int j = 0; j<n; j++){

        *(image1[i] + j) = input_image1[(i*n) + j];
        *(image2[i] + j) = input_image2[(i*n) + j];
    }
}
```

```
int InitWeightFunc(double *w1, double *w2, int n){

    int ret = 0;
    int halfImageWidth = (n/2);
    double Factor1;//int Factor1;
    double Factor2;

    for (int x = 0; x < n; x++){

        Factor1 = x / halfImageWidth;
        //Factor1 = x /halfImageWidth;
        Factor2 = (((double)(n - 1)/(double)x) - 1);
        w1[x] = (double)(Factor1)*(Factor2);

    }

    for (int y = 0; y < n; y++){
    if(y < halfImageWidth){
        Factor2 = (double)y / (double)halfImageWidth;
        w2[y] = Factor2;
    }
    else{

        w2[y] = 1;

    }
 }

 return ret;

}
```



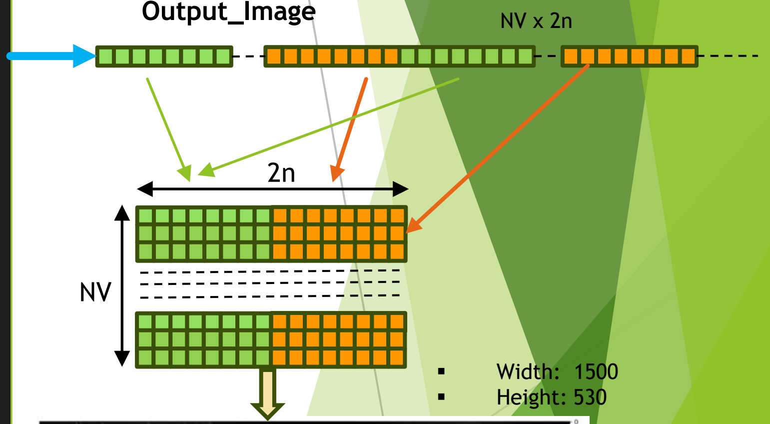
# Image Stitching – Sequential Method




## Sequential Image Stitching

```c
int sequential_ImageMerge(unsigned char *a, unsigned char *img1, unsigned char *img2, double *w1, double *w2, int n, int NV){

    int ret = 0;
    double temp1, pixel_value;

    for(int i = 0; i < NV; i++){

        for(int j = 0; j< n; j++){

            if (j > (n/2)){

                temp1 = (double)img1[(i*n) + j];
                temp1 = temp1*w1[j];
                pixel_value = (temp1 >= 0)?(temp1 + 0.5):(temp1 - 0.5);

                a[(i*n*2) + j] = (unsigned char)pixel_value;
                a[(i*n*2) + j + n] = img2[(i*n) + j];

            }
            else{

                temp1 = (double)img2[(i*n) + j];
                temp1 = temp1*w2[j];
                pixel_value = (temp1 >= 0)?(temp1 + 0.5):(temp1 - 0.5);

                a[(i*n*2) + j] = img1[(i*n) + j];
                a[(i*n*2) + j + n] = (unsigned char)pixel_value;
            }

        }
    }
    return ret;
}
```

- Width: 3840
- Height: 1080

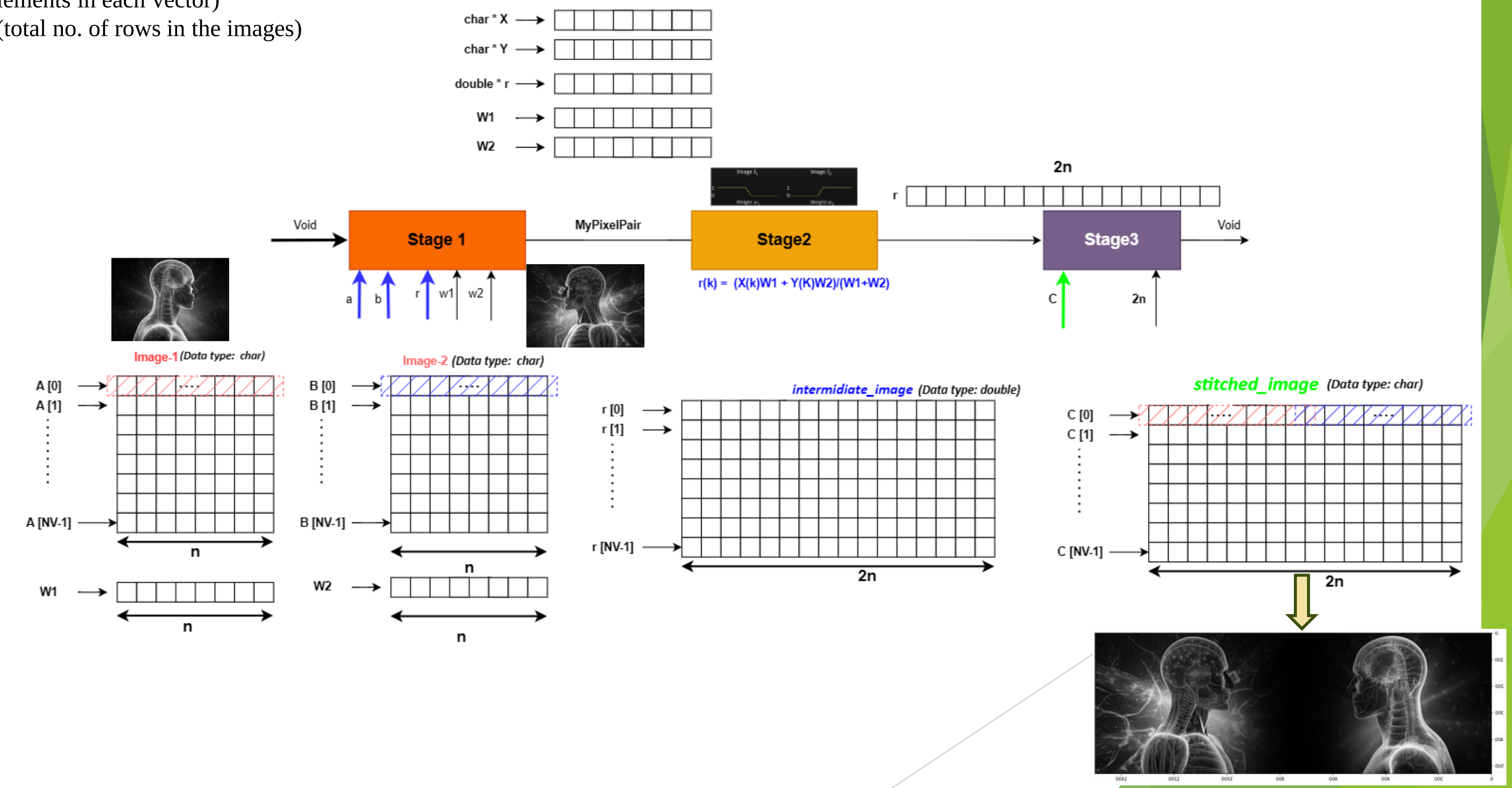



- Width: 1500
- Height: 530

| Mode | Image1 & Image2 Size | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Average Process time (in uSec) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Sequential | 750 x 530 | 4615 | 2999 | 5057 | 4623 | 4403 | 3773 | 2871 | 2989 | 4405 | 2974 | **3870,9** |
| | 1920 x 1080 | 11723 | 11717 | 11781 | 13940 | 11453 | 11627 | 11663 | 14135 | 14298 | 11721 | **12405,8** |

# Image Stitching – TBB Pipeline Method (2D Image):

- To reads the rows from Image-1 and Image-2 and create a two n-element vector for the next stage.
- The parameter passed this first stage are:
  - Two Images (two 2D data arrays – 2 NV x n matrices).
  - Two Vectors (Both vectors will have n elements)
  - Intermediate 2D array ( NV x 2n)
  - Value of n (Elements in each vector)
  - Value of NV (total no. of rows in the images)

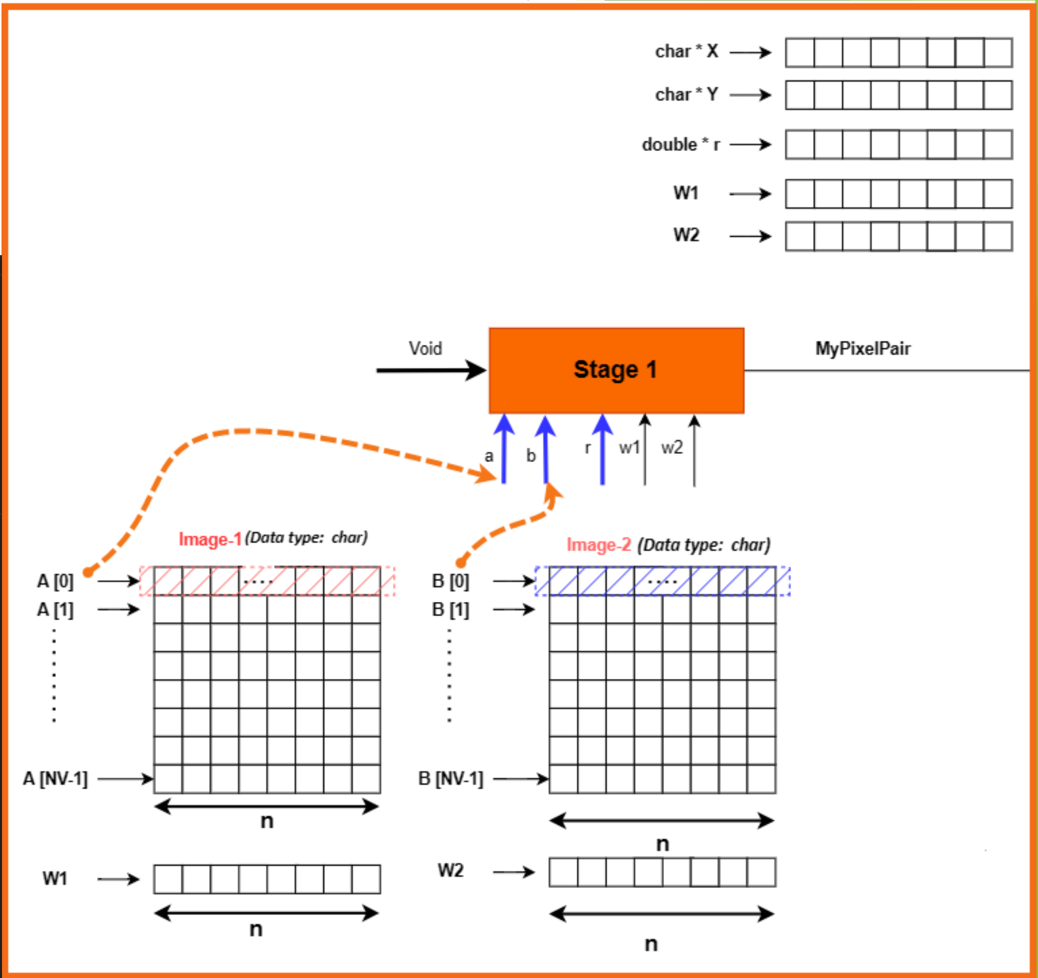


# Image Stitching using TBB Pipeline(2D Image): Stage-1

```
void RunPipeline (int ntoken, int n, int NV, unsigned char **a, unsigned char **b, unsigned char **r, double *w1, double *w2, unsigned char *c) {

    parallel_pipeline(ntoken,
        make_filter<void, MyPixelPair>(filter::serial_in_order, my_in(a, b, w1, w2, r, n, NV))
    &   make_filter<MyPixelPair, unsigned char*>(filter::parallel, PixelWeight_Calculator())
    &   make_filter<unsigned char*, void>(filter::serial_in_order, My_StitchedImage(c, n)));

}
```

```
/*Class which is used a return type from the stage-1*/
class MyPixelPair{
public:
    unsigned char *x;
    unsigned char *y;
    double *wx;
    double *wy;
    unsigned char *r;
    int n;
};
```

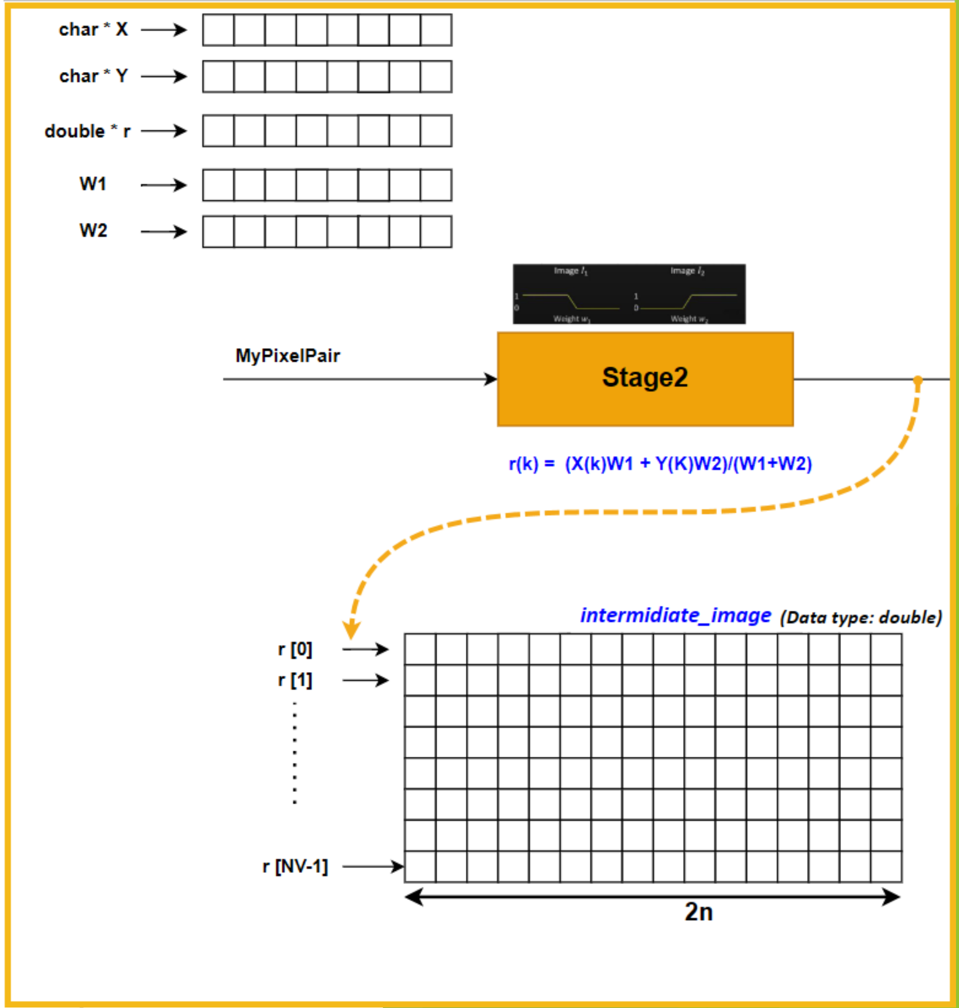
```
class my_in{

    unsigned char **a;
    unsigned char **b;
    double *w1;
    double *w2;
    unsigned char **r;
    int n;
    int NV;
    mutable int i;

public:
    my_in(unsigned char **ap, unsigned char **bp, double *w1p, double *w2p, unsigned char **rp, int np, int NVp) : a(ap), b(bp), w1(w1p), w2(w2p), r(rp), n(np), NV(NVp), i(0) {}

    MyPixelPair operator()(flow_control &fc) const {

        MyPixelPair t;
        const MyPixelPair ret_val = {.x = NULL, .y = NULL, .wx = NULL, .wy = NULL, .r = NULL, .n = 0};

        if (i < NV) {
            t.x = *(a + i);
            t.y = *(b + i);
            t.wx = w1;
            t.wy = w2;
            t.r = *(r + i);
            t.n = n;
            i++;
            return t;
        }
        else {
            fc.stop();
            return ret_val;
        }
    }
};
```



# Image Stitching using TBB Pipeline(2D Image): Stage-2

```
void RunPipeline (int ntoken, int n, int NV, unsigned char **a, unsigned char **b, unsigned char **r, double *w1, double *w2, unsigned char *c) {

    parallel_pipeline(ntoken,
        make_filter<void, MyPixelPair>(filter::serial_in_order, my_in(a, b, w1, w2, r, n, NV))
    &   make_filter<MyPixelPair, unsigned char*>(filter::parallel, PixelWeight_Calculator())
    &   make_filter<unsigned char*, void>(filter::serial_in_order, My_StitchedImage(c, n)));

}
```

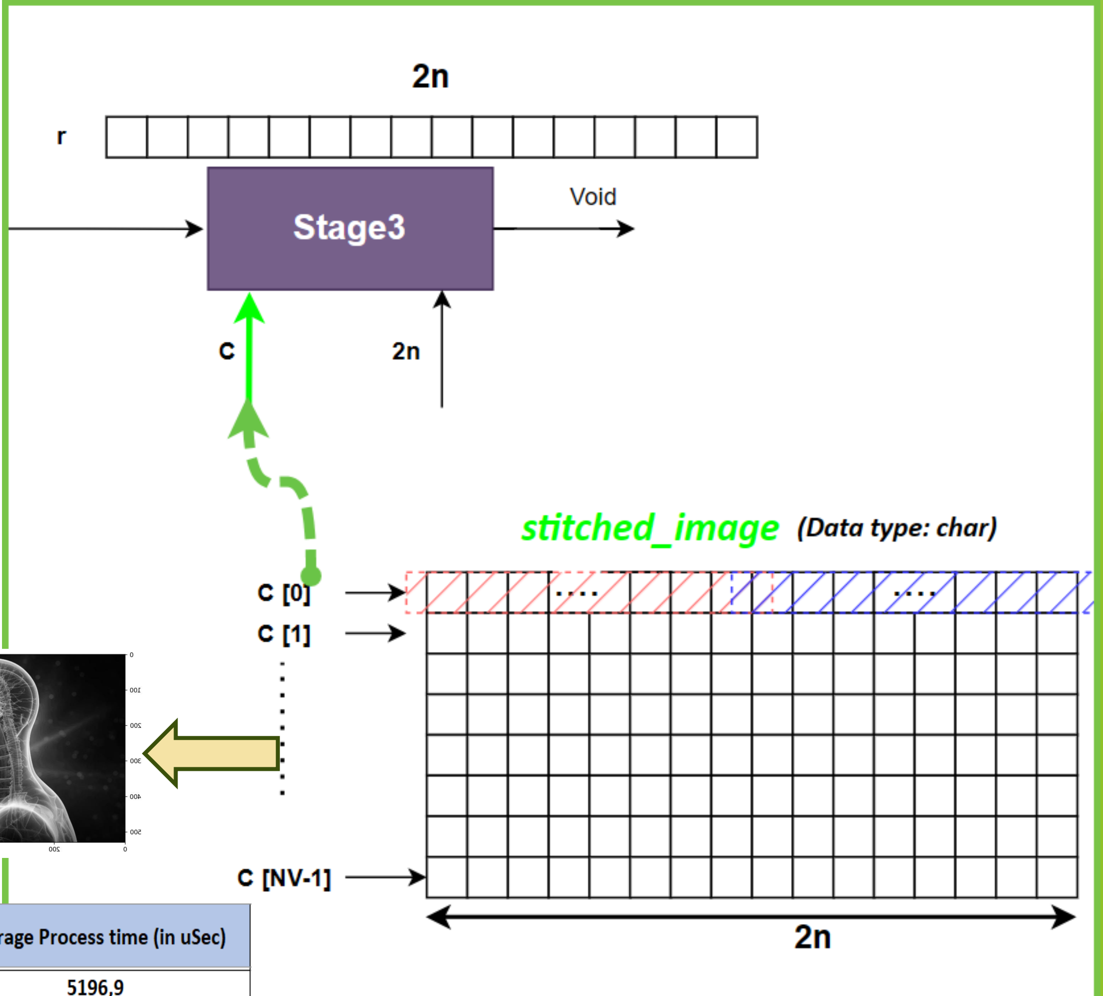
```
class PixelWeight_Calculator{

    public:
        unsigned char* operator()(MyPixelPair input) const{

            int i;
            int n1 = (input.n);
            double temp1, temp2;
            unsigned char *result = input.r;

            for(i = 0; i < input.n; i++){

                if( i < (input.n/2)){

                    result[i] = input.x[i];

                    temp1 = input.wy[i]*input.y[i];
                    temp2 = (temp1 >= 0)?(temp1 + 0.5):(temp1 - 0.5);
                    result[i + input.n] = (unsigned char)temp2;

                }
                else{
                    temp1 = input.wx[i]*input.x[i];
                    temp2 = (temp1 >= 0)?(temp1 + 0.5):(temp1 - 0.5);
                    result[i] = (unsigned char)temp2;

                    result[i + input.n] = input.y[i];

                }

            }
            return result;
        }

};
```

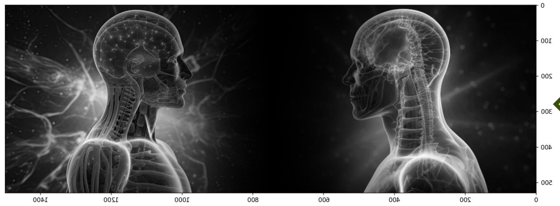


# Image Stitching using TBB Pipeline(2D Image): Stage-3

```
void RunPipeline (int ntoken, int n, int NV, unsigned char **a, unsigned char **b, unsigned char **r, double *w1, double *w2, unsigned char *c) {

    parallel_pipeline(ntoken,
        make_filter<void, MyPixelPair>(filter::serial_in_order, my_in(a, b, w1, w2, r, n, NV))
    &   make_filter<MyPixelPair, unsigned char*>(filter::parallel, PixelWeight_Calculator())
    &   make_filter<unsigned char*, void>(filter::serial_in_order, My_StitchedImage(c, n)));

}
```

```
class My_StitchedImage{
    public:
        mutable int j;
        unsigned char *ci;
        int n;

        My_StitchedImage(unsigned char *cp, int np): ci(cp), n(np), j(0){}

        void operator ()(unsigned char *rt) const{
            for(int k = 0; k < (2*n); k++){
                ci[(j*n*2) + k] = rt[k];

            }
            j++;
        }

};
```



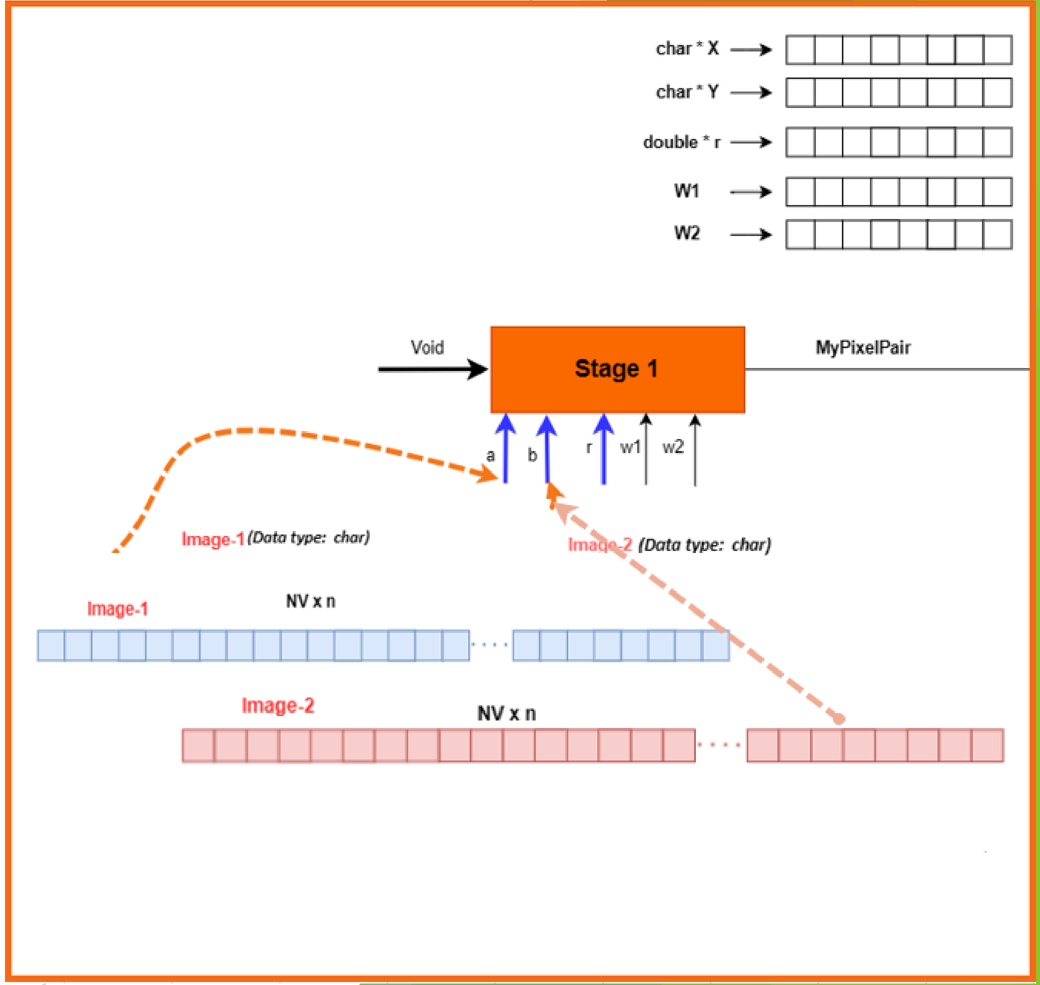
| Mode | Image1 & Image2 Size | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Average Process time (in uSec) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| TBB Pipeline (2D Array) | 750 x 530 | 5404 | 5040 | 5420 | 4819 | 5004 | 5290 | 5423 | 5334 | 4776 | 5459 | 5196,9 |
| | 1920 x 1080 | 19997 | 19878 | 19645 | 18806 | 19951 | 19996 | 20468 | 20107 | 20550 | 19408 | 19880,6 |

# Image Stitching using TBB Pipeline (1-D Image): Stage-1

```
/*Function where parallel_pipeline is invoked*/
void RunPipeline1 (int ntoken, int n, int NV, unsigned char *a, unsigned char *b, unsigned char *r, double *w1, double *w2, unsigned char *c) {

    parallel_pipeline(ntoken,
        make_filter<void, MyPixelPair1>(filter::serial_in_order, my_in1(a, b, w1, w2, r, n, NV))
    &   make_filter<MyPixelPair1, unsigned char*>(filter::parallel, PixelWeight_Calculator1())
    &   make_filter<unsigned char*, void>(filter::serial_in_order, My_StitchedImage1(c, n)));

}
```

```
/*Class which is used a return type from the stage-1*/
class MyPixelPair{
public:
    unsigned char *x;
    unsigned char *y;
    double *wx;
    double *wy;
    unsigned char *r;
    int n;
};
```

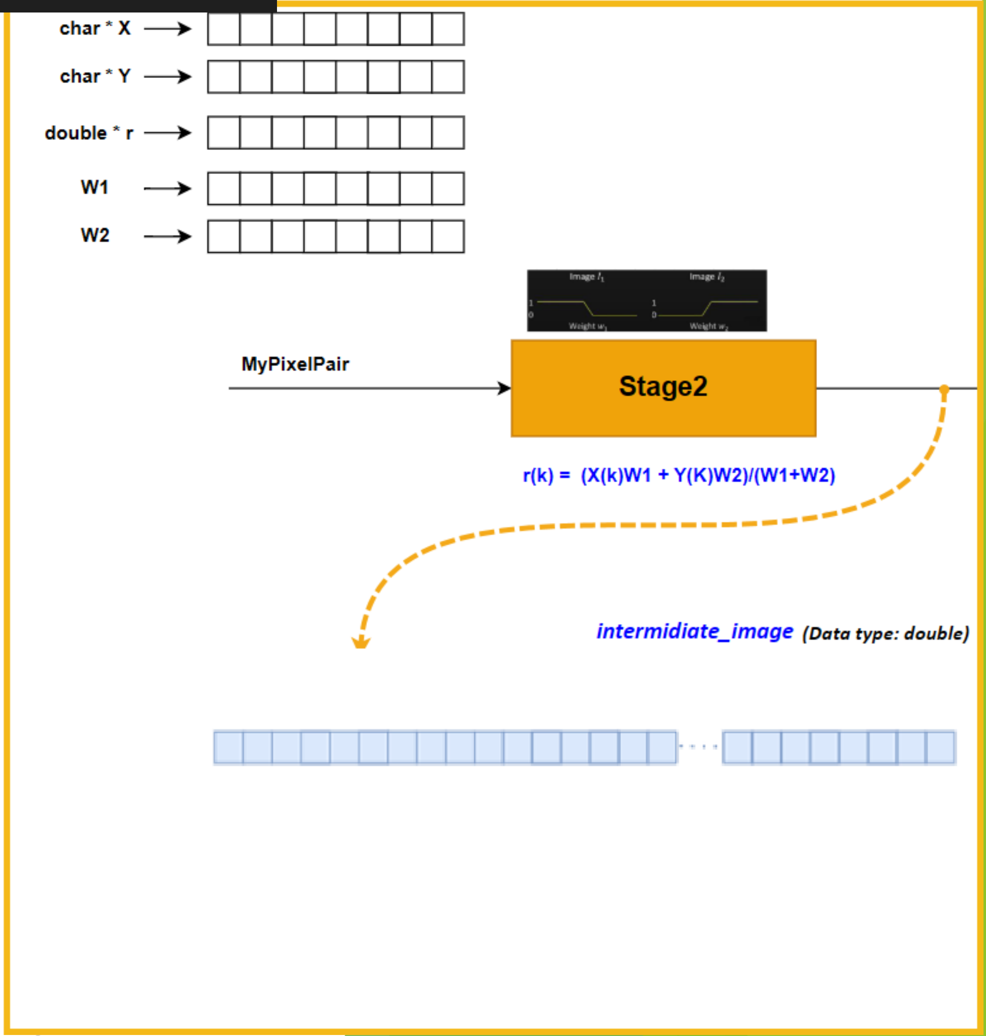
```
class my_in1{

    unsigned char *a;
    unsigned char *b;
    double *w1;
    double *w2;
    unsigned char *r;
    int n;
    int NV;
    mutable int i;

public:
    my_in1(unsigned char *ap, unsigned char *bp, double *w1p, double *w2p, unsigned char *rp, int np, int NVp) : a(ap), b(bp), w1(w1p), w2(w2p), r(rp), n(np), NV(NVp), i(0) {}

    MyPixelPair1 operator()(flow_control &fc) const {

        MyPixelPair1 t;
        const MyPixelPair1 ret_val = {.x = NULL, .y = NULL, .wx = NULL, .wy = NULL, .r = NULL, .n = 0};

        if (i < NV) {
            t.x = (a + (i*n));
            t.y = (b + (i*n));
            t.wx = w1;
            t.wy = w2;
            t.r = (r + (i*n*2));
            t.n = n;
            i++;
            return t;
        }
        else {
            fc.stop();
            return ret_val;
        }
    }
};
```



# Image Stitching using TBB Pipeline (1D- Image): Stage-2

```
/*Function where parallel_pipeline is invoked*/
void RunPipeline1 (int ntoken, int n, int NV, unsigned char *a, unsigned char *b, unsigned char *r, double *w1, double *w2, unsigned char *c) {

    parallel_pipeline(ntoken,
        make_filter<void, MyPixelPair1>(filter::serial_in_order, my_in1(a, b, w1, w2, r, n, NV))
    &   make_filter<MyPixelPair1, unsigned char*>(filter::parallel, PixelWeight_Calculator1())
    &   make_filter<unsigned char*, void>(filter::serial_in_order, My_StitchedImage1(c, n)));

}
```

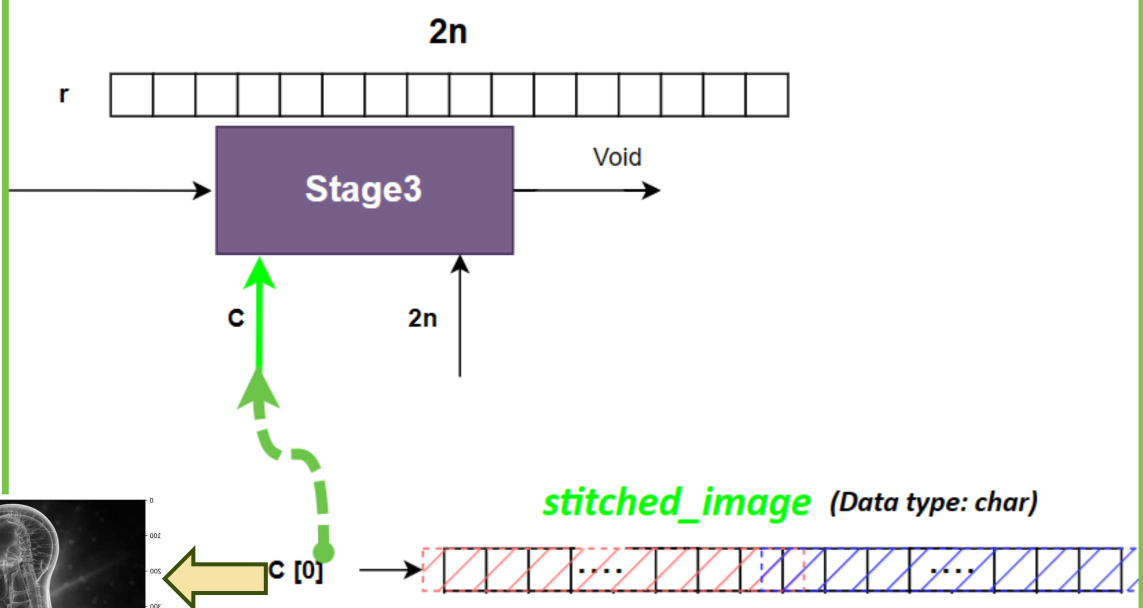
```
class PixelWeight_Calculator1{

    public:
        unsigned char* operator()(MyPixelPair1 input) const{

            int i;
            double temp1, temp2;
            unsigned char *result = input.r;

            for(i = 0; i < input.n; i++){

                if( i < (input.n/2)){

                    result[i] = input.x[i];

                    temp1 = input.wy[i]*input.y[i];
                    temp2 = (temp1 >= 0)?(temp1 + 0.5):(temp1 - 0.5);
                    result[i + input.n] = (unsigned char)temp2;

                }
                else{
                    temp1 = input.wx[i]*input.x[i];
                    temp2 = (temp1 >= 0)?(temp1 + 0.5):(temp1 - 0.5);
                    result[i] = (unsigned char)temp2;

                    result[i + input.n] = input.y[i];

                }

            }
            return result;
        }

};
```

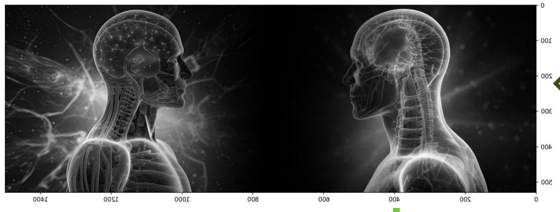


# Image Stitching using TBB Pipeline (1D-Image): Stage-3

```cpp
/*Function where parallel_pipeline is invoked*/
void RunPipeline1 (int ntoken, int n, int NV, unsigned char *a, unsigned char *b, unsigned char *r, double *w1, double *w2, unsigned char *c) {

    parallel_pipeline(ntoken,
        make_filter<void, MyPixelPair1>(filter::serial_in_order, my_in1(a, b, w1, w2, r, n, NV))
    &   make_filter<MyPixelPair1, unsigned char*>(filter::parallel, PixelWeight_Calculator1())
    &   make_filter<unsigned char*, void>(filter::serial_in_order, My_StitchedImage1(c, n)));

}
```

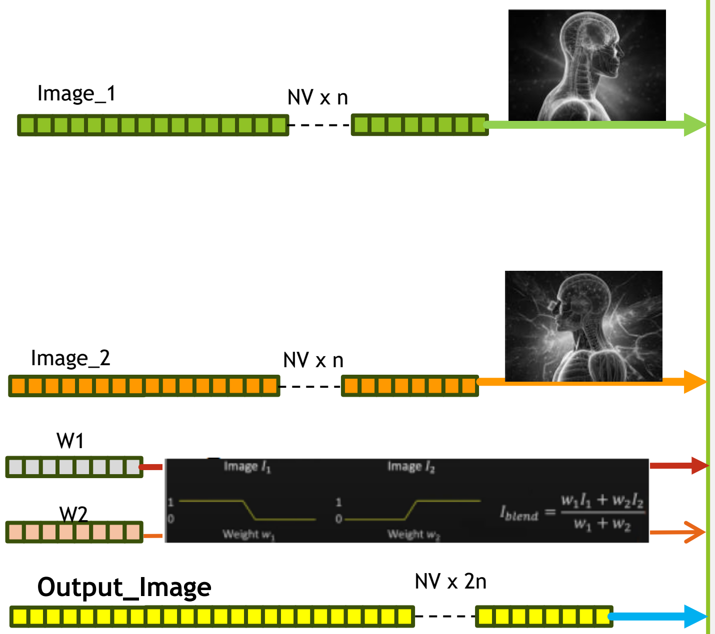
```cpp
class My_StitchedImage1{
    public:
        mutable int j;
        unsigned char *ci;
        int n;

        My_StitchedImage1(unsigned char *cp, int np): ci(cp), n(np), j(0){}

        void operator ()(unsigned char *rt) const{
            for(int k = 0; k < (2*n); k++){
                ci[(j*n*2) + k] = rt[k];

            }
            j++;
        }

};
```



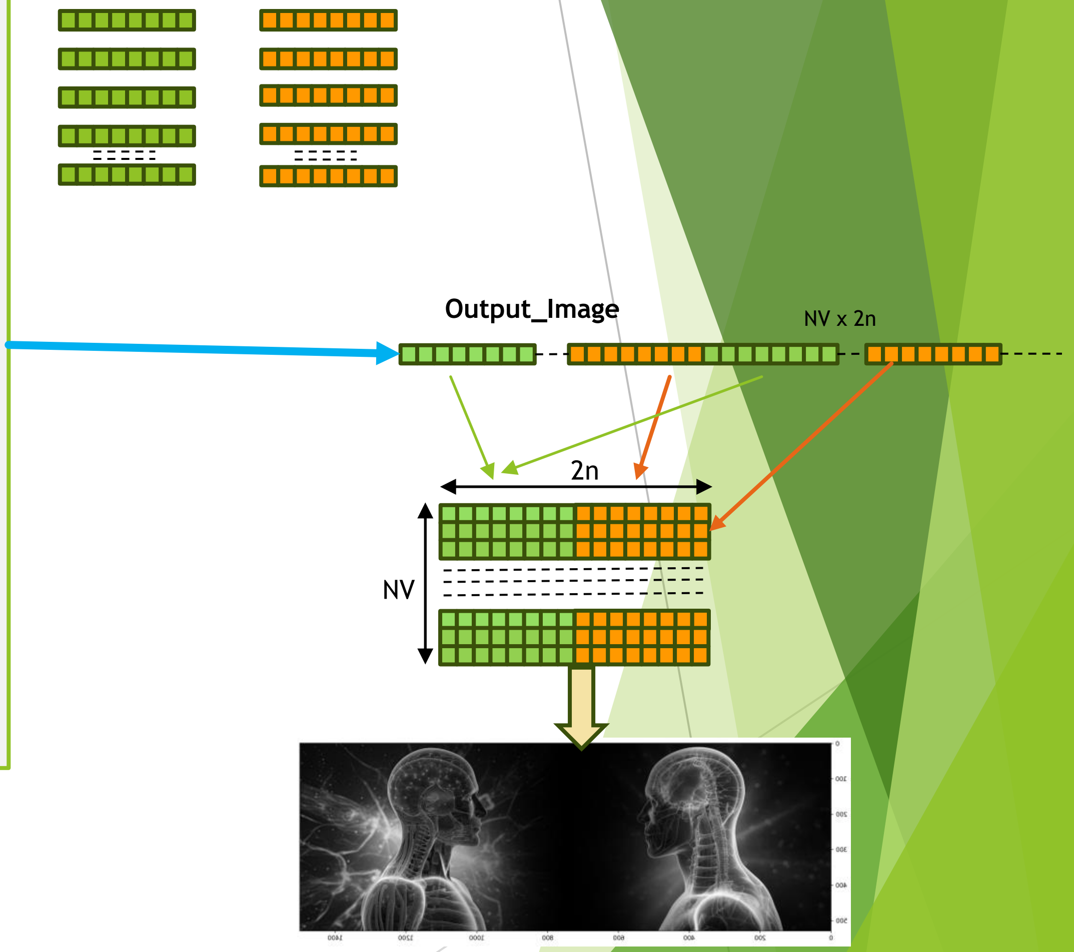
| Mode | Image1 & Image2 Size | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Average Process time (in uSec) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| TBB Pipeline (1D Array) | 750 x 530 | 5958 | 6422 | 5428 | 6583 | 6219 | 6206 | 6294 | 6148 | 6503 | 6273 | **6203,4** |
| | 1920 x 1080 | 21739 | 21950 | 21690 | 21947 | 21362 | 21693 | 21310 | 21548 | 21292 | 21294 | **21582,5** |

# Image Stitching - Parallel_For (Row)



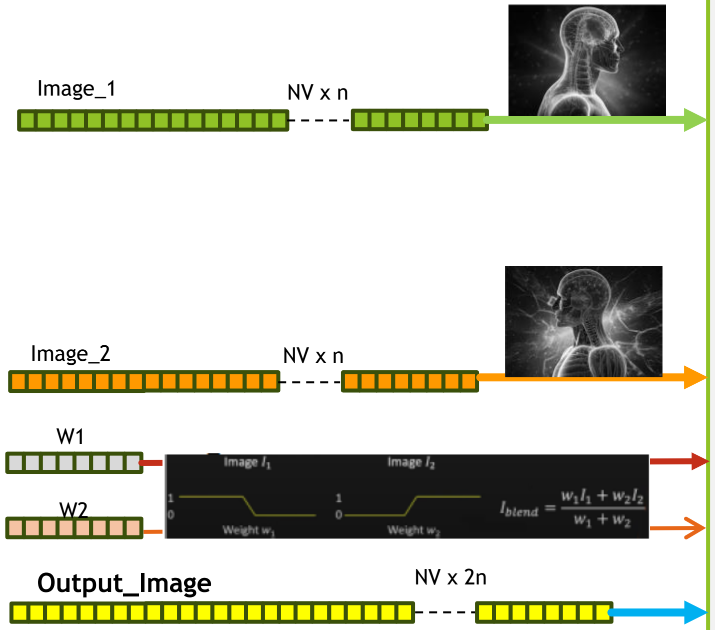

## Parallel_For Image Stitching

```
parallel_for(blocked_range<size_t>(0, NV), [&](blocked_range<size_t> r){

  for (size_t i = r.begin(); i <r.end(); ++i){

    for (int j = 0; j < n; j++){

      double temp1, pixel_value;

      if (j > (n/2)){

        temp1 = (double)input_image1[(i*n) + j];
        temp1 = temp1*w1[j];
        pixel_value = (temp1 >= 0)?(temp1 + 0.5):(temp1 - 0.5);

        out_img_tbb[(i*n*2) + j] = (unsigned char)pixel_value;
        out_img_tbb[(i*n*2) + j + n] = input_image2[(i*n) + j];

      }

      else{

        temp1 = (double)input_image2[(i*n) + j];
        temp1 = temp1*w2[j];
        pixel_value = (temp1 >= 0)?(temp1 + 0.5):(temp1 - 0.5);

        out_img_tbb[(i*n*2) + j] = input_image1[(i*n) + j];
        out_img_tbb[(i*n*2) + j + n] = (unsigned char)pixel_value;
      }

    }
  }

});
```



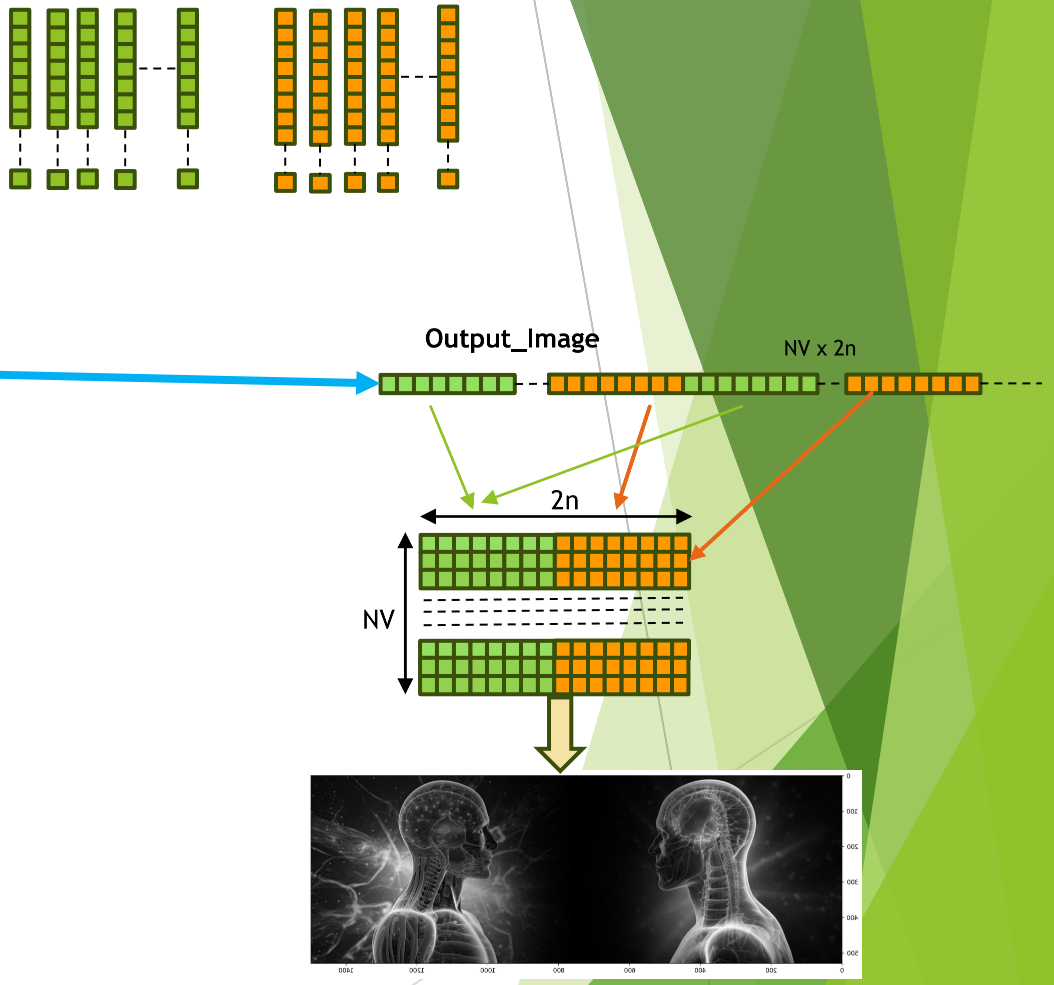

| Mode | Image1 & Image2 Size | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Average Process time (in uSec) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| TBB Parallel_For (Row Parallel) | 750 x 530 | 2252 | 1737 | 1954 | 2096 | 1938 | 2359 | 2003 | 2251 | 2443 | 2034 | **2106,7** |
| | 1920 x 1080 | 3641 | 5552 | 4116 | 3615 | 3919 | 3650 | 7011 | 3750 | 5325 | 4594 | **4517,3** |

# Image Stitching - Parallel_For (Column)



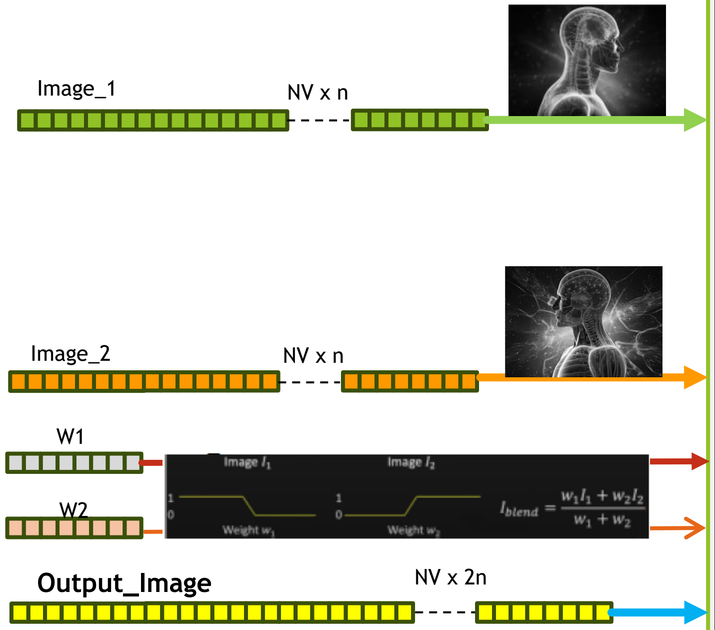

## Parallel_For Image Stitching

```
for(int i = 0; i < NV; i++){

  parallel_for(blocked_range<size_t>(0, n), [&](blocked_range<size_t> r){

    for (size_t j = r.begin(); j <r.end(); ++j){

      double temp1, pixel_value;

      if (j > (n/2)){

        temp1 = (double)input_image1[(i*n) + j];
        temp1 = temp1*w1[j];
        pixel_value = (temp1 >= 0)?(temp1 + 0.5):(temp1 - 0.5);

        out_img_tbb[(i*n*2) + j] = (unsigned char)pixel_value;
        out_img_tbb[(i*n*2) + j + n] = input_image2[(i*n) + j];

      }

      else{

        temp1 = (double)input_image2[(i*n) + j];
        temp1 = temp1*w2[j];
        pixel_value = (temp1 >= 0)?(temp1 + 0.5):(temp1 - 0.5);

        out_img_tbb[(i*n*2) + j] = input_image1[(i*n) + j];
        out_img_tbb[(i*n*2) + j + n] = (unsigned char)pixel_value;
      }

    }
  });

}
```



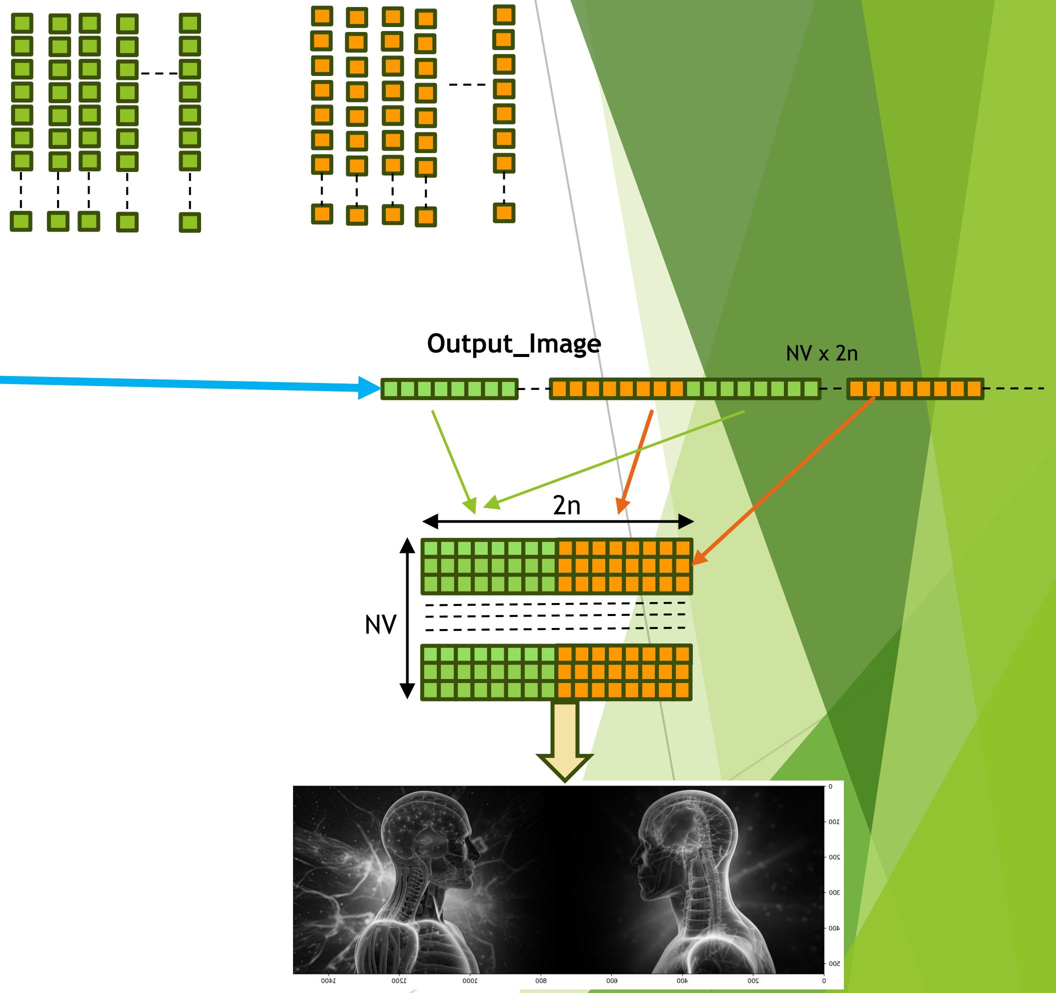

| Mode | Image1 & Image2 Size | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Average Process time (in uSec) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| TBB Parallel_For (Column Parallel) | 750 x 530 | 17155 | 17072 | 17377 | 17396 | 17882 | 20310 | 17298 | 17952 | 17377 | 17071 | **17689** |
| | 1920 x 1080 | 36797 | 38772 | 37052 | 38832 | 39951 | 39516 | 37033 | 36404 | 39229 | 39162 | **38274,8** |

# Image Stitching - Parallel_For (Row & Column both)




## Parallel_For Image Stitching

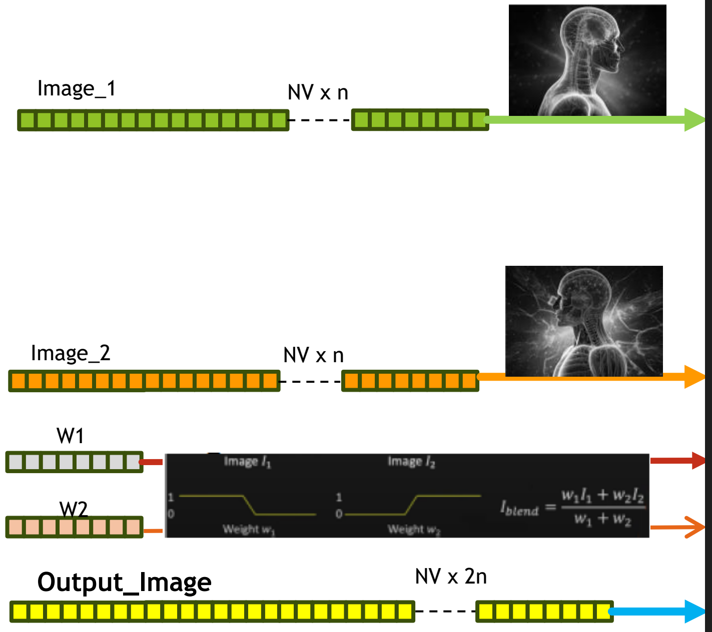
```
parallel_for(blocked_range<size_t>(0, NV), [&](blocked_range<size_t> r){

  for (size_t i = r.begin(); i <r.end(); ++i){

    parallel_for(blocked_range<size_t>(0, n), [&](blocked_range<size_t> r){

      for (size_t j = r.begin(); j <r.end(); ++j){

        double temp1, pixel_value;

        if (j > (n/2)){

          temp1 = (double)input_image1[(i*n) + j];
          temp1 = temp1*w1[j];
          pixel_value = (temp1 >= 0)?(temp1 + 0.5):(temp1 - 0.5);

          out_img_tbb[(i*n*2) + j] = (unsigned char)pixel_value;
          out_img_tbb[(i*n*2) + j + n] = input_image2[(i*n) + j];

        }

        else{

          temp1 = (double)input_image2[(i*n) + j];
          temp1 = temp1*w2[j];
          pixel_value = (temp1 >= 0)?(temp1 + 0.5):(temp1 - 0.5);

          out_img_tbb[(i*n*2) + j] = input_image1[(i*n) + j];
          out_img_tbb[(i*n*2) + j + n] = (unsigned char)pixel_value;
        }

      }
    });
  }

});
```



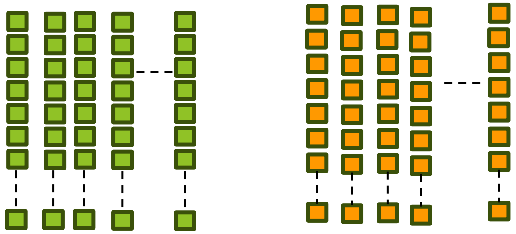

| Mode | Image1 & Image2 Size | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Average Process time (in uSec) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| TBB Parallel_For (Row + Column Parallel) | 750 x 530 | 5810 | 5525 | 7320 | 5706 | 5987 | 6656 | 6849 | 5332 | 5955 | 5779 | **6091,9** |
| | 1920 x 1080 | 13280 | 14179 | 14041 | 13366 | 13800 | 13428 | 13806 | 13369 | 14067 | 12987 | **13632,3** |

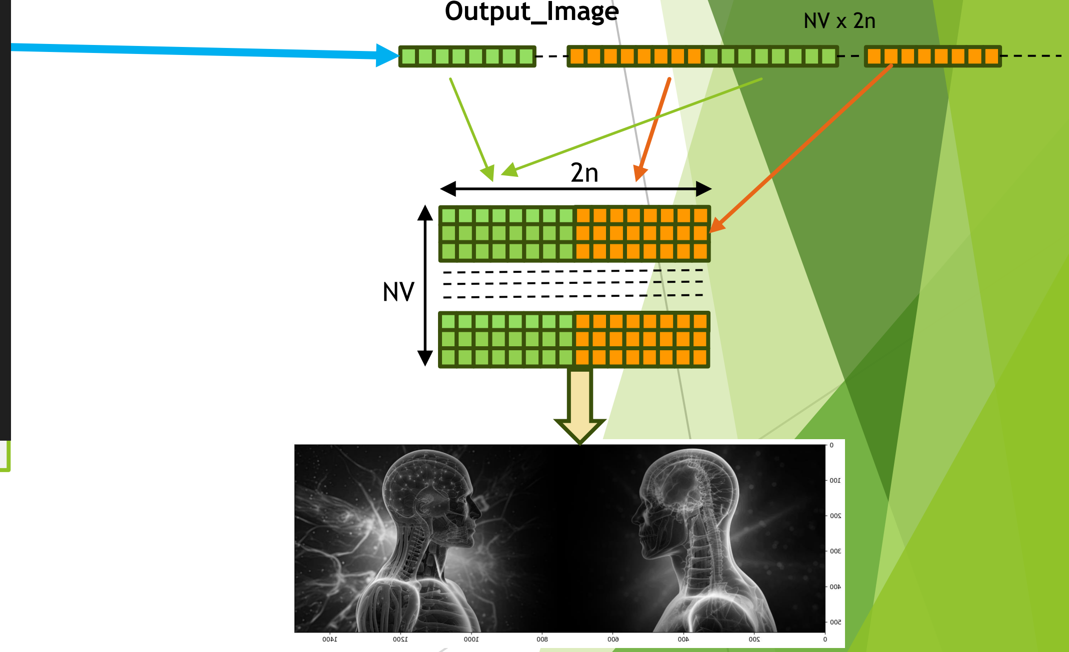
# Image Stitching - PThreads

## Image Stitching - Pthreads




```
void *thread_result;
tSt_Image *my_image = (tSt_Image*)calloc(num_threads, sizeof(tSt_Image));

pthread_t *thread_id = (pthread_t *)calloc(num_threads, sizeof(pthread_t));

 for (int i = 0; i < num_threads; i++){

     my_image[i].width = (int) IMAGE_WIDTH;

     my_image[i].height = (int) IMAGE_HEIGHT;

     my_image[i].slice = ceil((double)IMAGE_HEIGHT / (double)num_threads);

     my_image[i].a = input_image1;

     my_image[i].b = input_image2;

     my_image[i].id = i;

     my_image[i].wa = w1;

     my_image[i].wb = w2;

     my_image[i].out = out_img_pthread;

    pthread_create(&thread_id[i], NULL, Stitch_Images, (void *)&my_image[i]);
 }

 for (int i = 0; i < num_threads; i++){

     /*waits for thread to complete.*/
     pthread_join (thread_id[i], &thread_result);
 }
```





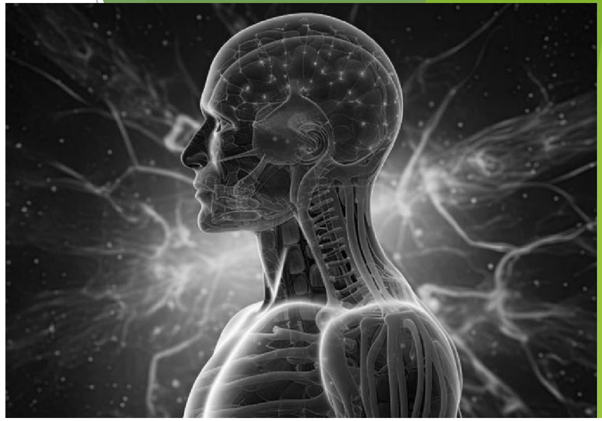

| Mode | Image1 & Image2 Size | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Average Process time (in uSec) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Pthreads (50 Threads)** | 750 x 530 | 1697 | 1990 | 1836 | 1935 | 1811 | 1900 | 1790 | 1914 | 1803 | 1767 | **1844,3** |
| | 1920 x 1080 | 2174 | 2285 | 2088 | 2083 | 2403 | 2467 | 2224 | 2314 | 2622 | 2156 | **2281,6** |

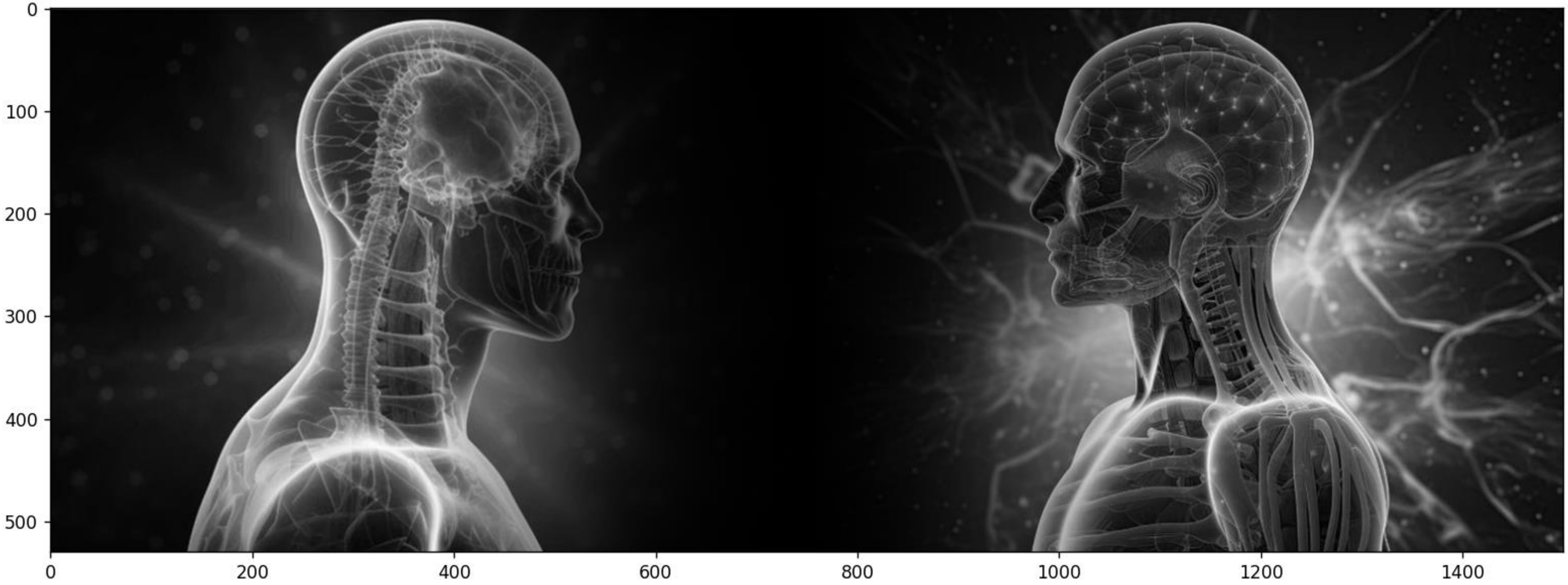
# Images that’s need to be stitched:

Image-1:
- Width: 750
- Height: 530

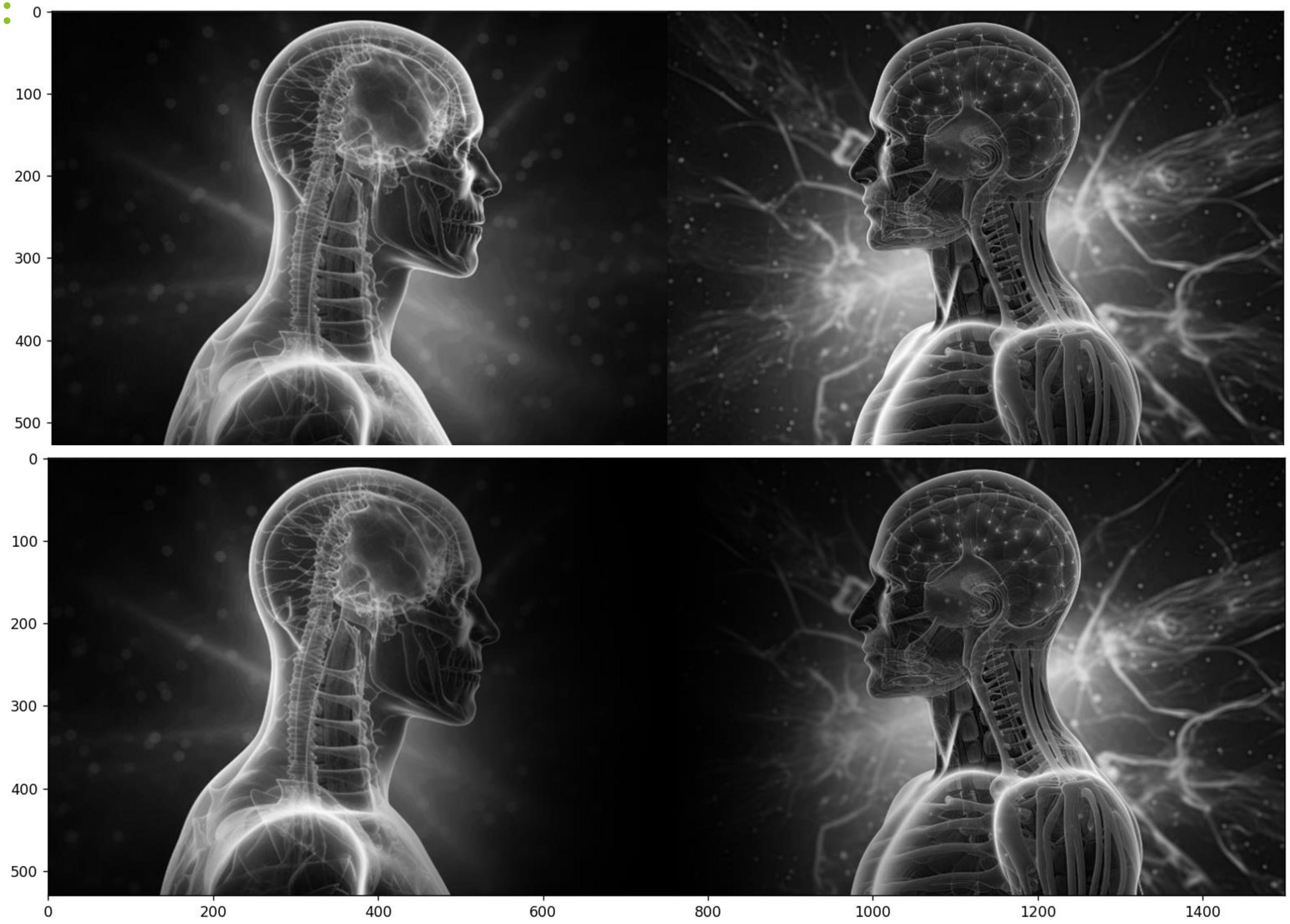
Image-2:
- Width: 750
- Height: 530

# Image stitching without weight function :





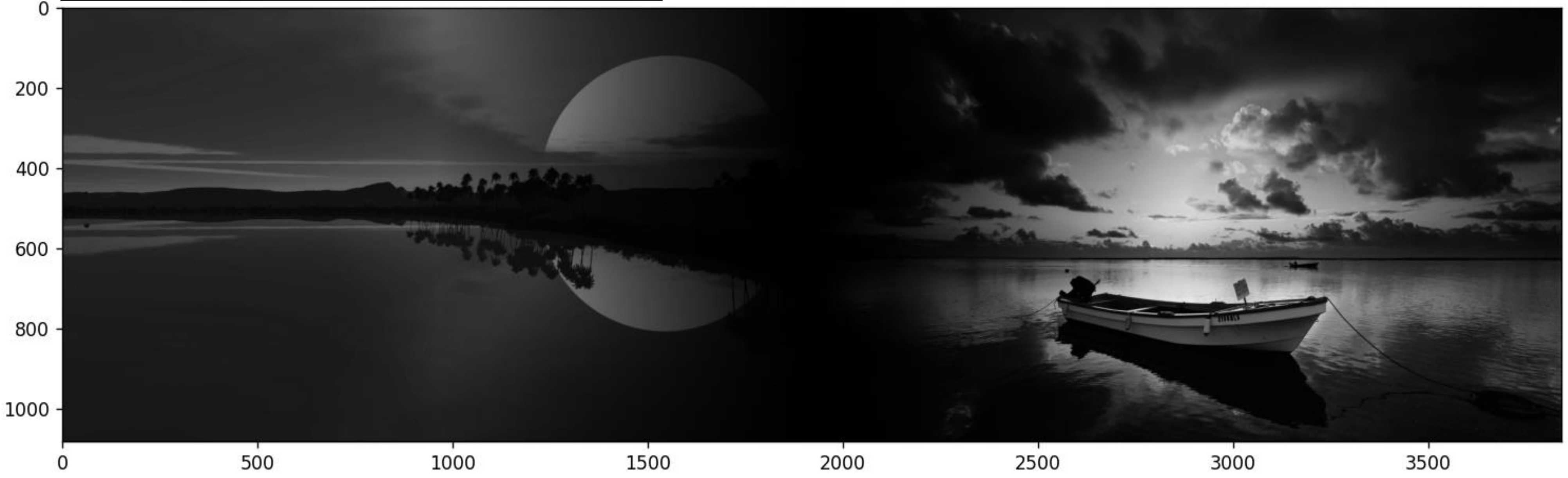
# Image comparison:



## Image Artifacts and processing information:

- Row: 1080
- Col: 1920

Figure 1




(x, y) = (1029., 512.)
[7.0]

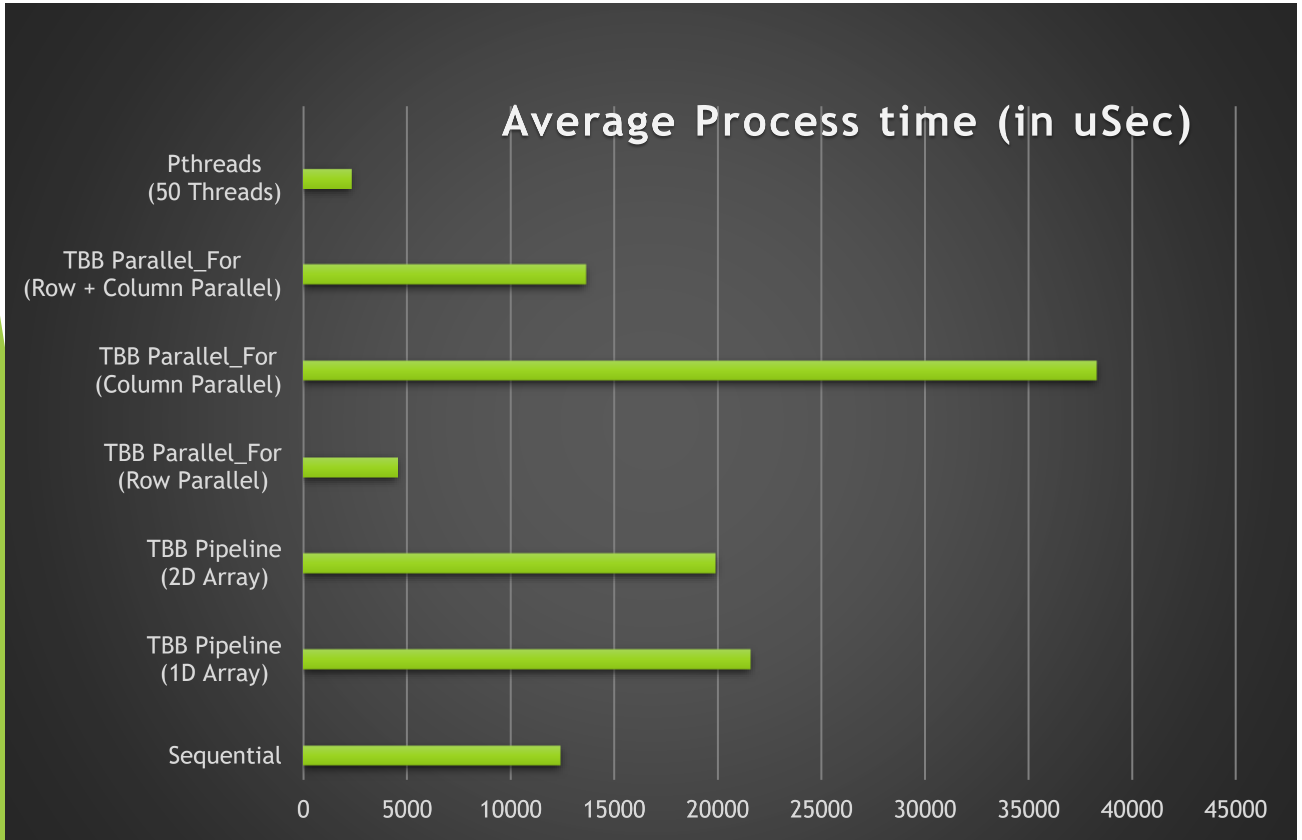
```
D:\masters\Fall_semester2024\Final_Project\matlab>python test_image.py
Stitch Image Width =  3840
11
232
```

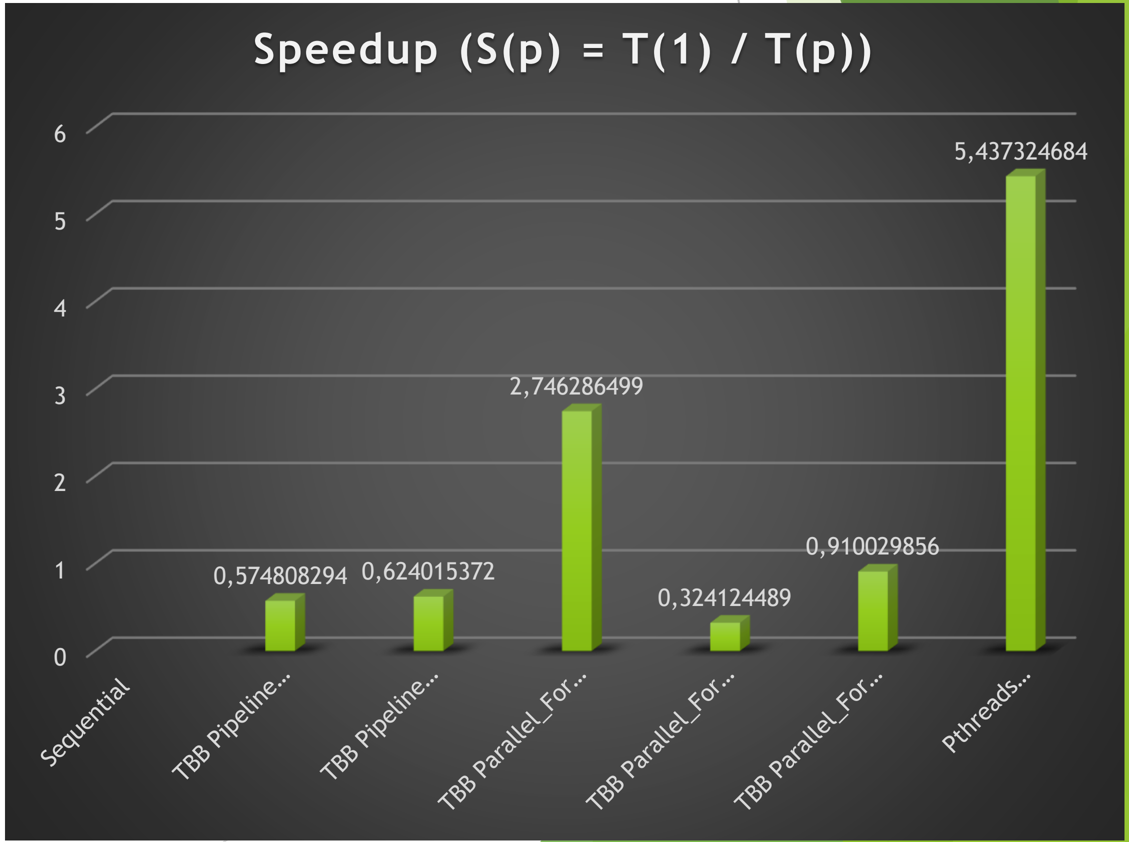
## Image Artifacts and processing information:

```
uidr4928@qhws04x:~/manoj/fall2024/lab_hw/final_project_Dec4$ ./image_stitch_tbb
(read_binfile) Input binary file 'photo8a.bif': # of elements read = 2073600
(read_binfile) Input binary file 'photo8b.bif': # of elements read = 2073600

start (Pthread): 332180 us
end (Pthread):334405 us
Elapsed time using pthread(only Computation Time): 2225 us

start (Parallel_For): 334405 us
end (Parallel_For):340001 us
Elapsed time using pthread(only Computation Time): 5596 us

start (Pipeline-1D): 355751 us
end (Pipeline-1D):377369 us
Elapsed time using 1-D parallel_pipeline(only Computation Time): 21618 us

start (Pipeline-2D): 379772 us
end (Pipeline-2D):397449 us
Elapsed time using parallel_pipeline(only Computation Time): 17677 us

start (Sequential): 341928 us
end (Sequential): 353588 us
Elapsed time using sequential(only Computation Time): 11660 us
uidr4928@qhws04x:~/manoj/fall2024/lab_hw/final_project_Dec4$
```

# Processing Time and Speedup:

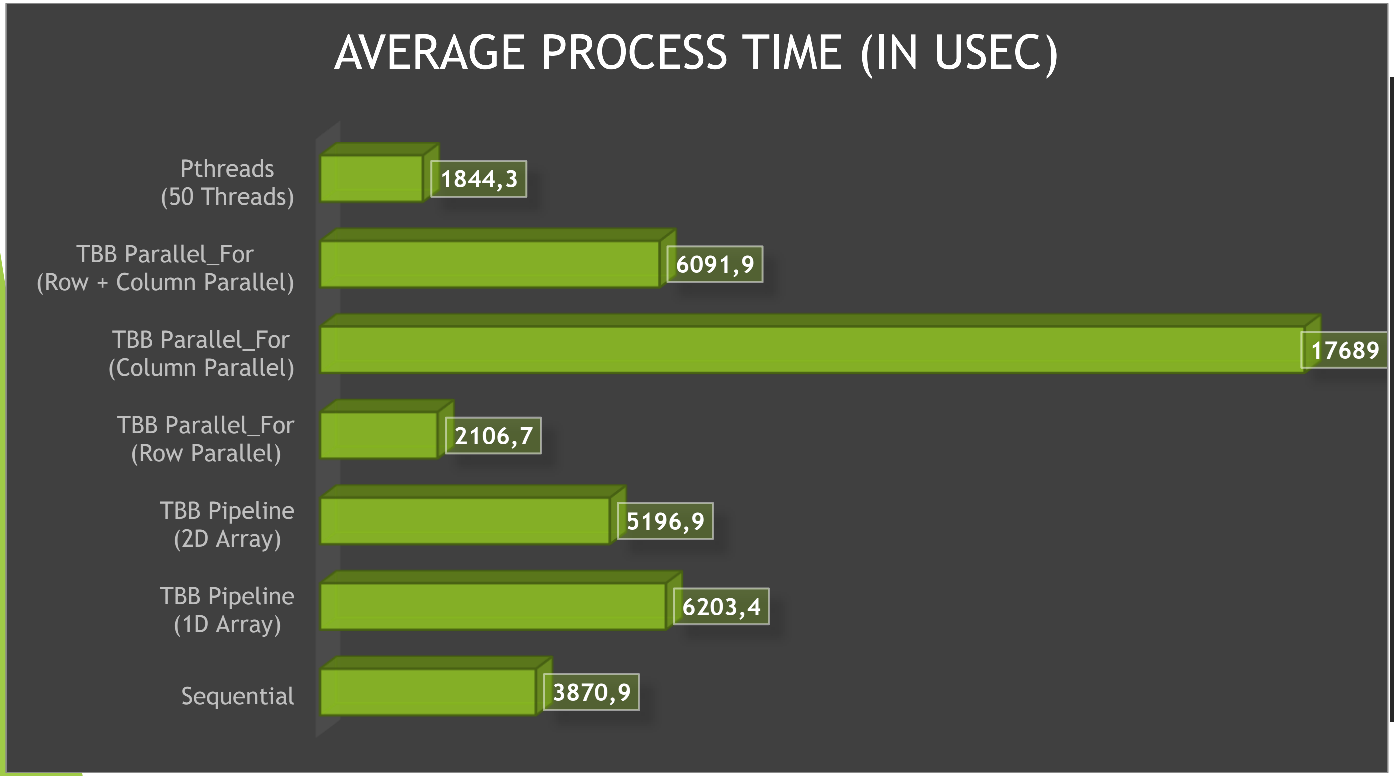
| Mode | Image1 & Image2 Size | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Average Process time (in uSec) | Speedup (S(p) = T(1) / T(p)) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Sequential** | 1920 x 1080 | 11723 | 11717 | 11781 | 13940 | 11453 | 11627 | 11663 | 14135 | 14298 | 11721 | **12405,8** | |
| **TBB Pipeline (1D Array)** | 1920 x 1080 | 21739 | 21950 | 21690 | 21947 | 21362 | 21693 | 21310 | 21548 | 21292 | 21294 | **21582,5** | **0,574808294** |
| **TBB Pipeline (2D Array)** | 1920 x 1080 | 19997 | 19878 | 19645 | 18806 | 19951 | 19996 | 20468 | 20107 | 20550 | 19408 | **19880,6** | **1,085606068** |
| **TBB Parallel_For (Row Parallel)** | 1920 x 1080 | 3641 | 5552 | 4116 | 3615 | 3919 | 3650 | 7011 | 3750 | 5325 | 4594 | **4517,3** | **4,400991743** |
| **TBB Parallel_For (Column Parallel)** | 1920 x 1080 | 36797 | 38772 | 37052 | 38832 | 39951 | 39516 | 37033 | 36404 | 39229 | 39162 | **38274,8** | **0,118022824** |
| **TBB Parallel_For (Row + Column Parallel)** | 1920 x 1080 | 13280 | 14179 | 14041 | 13366 | 13800 | 13428 | 13806 | 13369 | 14067 | 12987 | **13632,3** | **2,807655348** |
| **Pthreads (50 Threads)** | 1920 x 1080 | 2174 | 2285 | 2088 | 2083 | 2403 | 2467 | 2224 | 2314 | 2622 | 2156 | **2281,6** | **5,974886045** |

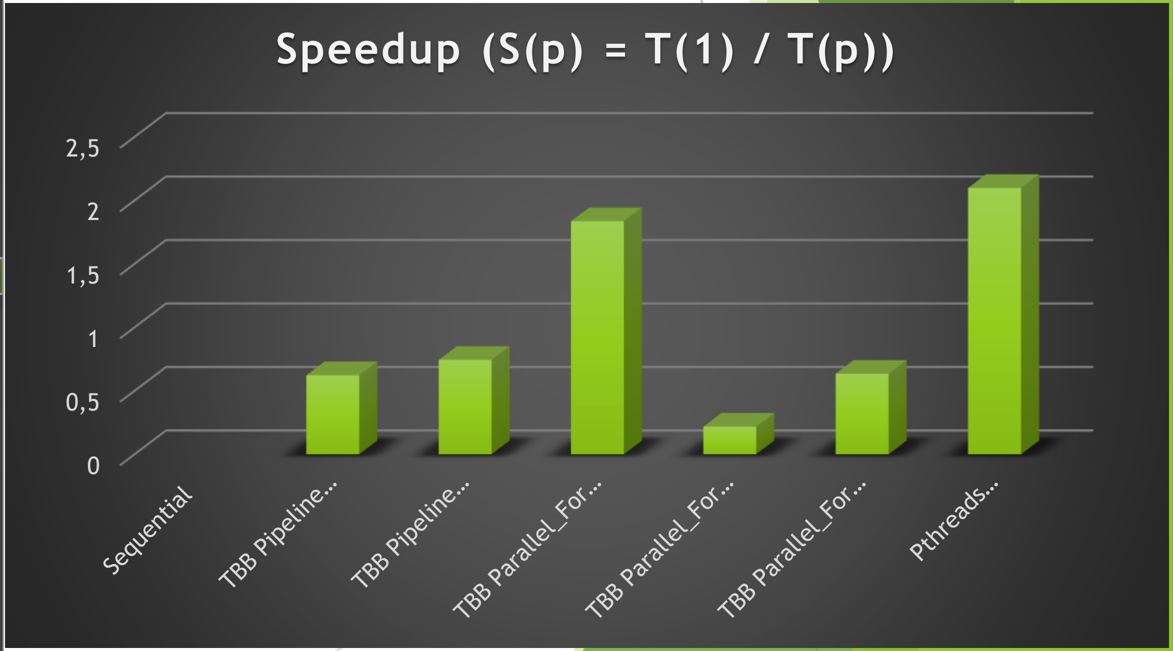


# Processing Time and Speed-up:

| Mode | Image1 & Image2 Size | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Average Process time (in uSec) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Sequential** | 750 x 530 | 4615 | 2999 | 5057 | 4623 | 4403 | 3773 | 2871 | 2989 | 4405 | 2974 | **3870,9** |
| **TBB Pipeline (1D Array)** | 750 x 530 | 5958 | 6422 | 5428 | 6583 | 6219 | 6206 | 6294 | 6148 | 6503 | 6273 | **6203,4** |
| **TBB Pipeline (2D Array)** | 750 x 530 | 5404 | 5040 | 5420 | 4819 | 5004 | 5290 | 5423 | 5334 | 4776 | 5459 | **5196,9** |
| **TBB Parallel_For (Row Parallel)** | 750 x 530 | 2252 | 1737 | 1954 | 2096 | 1938 | 2359 | 2003 | 2251 | 2443 | 2034 | **2106,7** |
| **TBB Parallel_For (Column Parallel)** | 750 x 530 | 17155 | 17072 | 17377 | 17396 | 17882 | 20310 | 17298 | 17952 | 17377 | 17071 | **17689** |
| **TBB Parallel_For (Row + Column Parallel)** | 750 x 530 | 5810 | 5525 | 7320 | 5706 | 5987 | 6656 | 6849 | 5332 | 5955 | 5779 | **6091,9** |
| **Pthreads (50 Threads)** | 750 x 530 | 1697 | 1990 | 1836 | 1935 | 1811 | 1900 | 1790 | 1914 | 1803 | 1767 | **1844,3** |



# Processing Time measurements for different size of input images:

```
Image-1 = photo8a.bif, Resolution = 1920 x 1080
Image-2 = photo8b.bif, Resolution = 1920 x 1080
(read_binfile) Input binary file 'photo8a.bif': # of elements read = 2073600
(read_binfile) Input binary file 'photo8b.bif': # of elements read = 2073600

Number of pThreads = 25
start (Pthread): 875286 us
end (Pthread):877832 us
Elapsed time using pthread(only Computation Time): 2546 us

start (Parallel_For): 877832 us
end (Parallel_For):884625 us
Elapsed time using pthread(only Computation Time): 6793 us

start (Pipeline-1D): 914243 us
end (Pipeline-1D):935494 us
Elapsed time using 1-D parallel_pipeline(only Computation Time): 21251 us

start (Pipeline-2D): 938727 us
end (Pipeline-2D):959543 us
Elapsed time using parallel_pipeline(only Computation Time): 20816 us

start (Sequential): 888175 us
end (Sequential): 899874 us
Elapsed time using sequential(only Computation Time): 11699 us
```

```
Image-1 = photo8aa.bif, Resolution = 3840 x 1080
Image-2 = photo8bb.bif, Resolution = 3840 x 1080
(read_binfile) Input binary file 'photo8aa.bif': # of elements read = 4147200
(read_binfile) Input binary file 'photo8bb.bif': # of elements read = 4147200

Number of pThreads = 50
start (Pthread): 659449 us
end (Pthread):664190 us
Elapsed time using pthread(only Computation Time): 4741 us

start (Parallel_For): 664190 us
end (Parallel_For):670905 us
Elapsed time using pthread(only Computation Time): 6715 us

start (Pipeline-1D): 727533 us
end (Pipeline-1D):761730 us
Elapsed time using 1-D parallel_pipeline(only Computation Time): 34197 us

start (Pipeline-2D): 766574 us
end (Pipeline-2D):800505 us
Elapsed time using parallel_pipeline(only Computation Time): 33931 us

start (Sequential): 675944 us
end (Sequential): 701587 us
Elapsed time using sequential(only Computation Time): 25643 us
uidr4928@qhws04x:~/manoj/fall2024/lab_hw/final_project_Dec4$
```

```
Image-1 = photo8aaa.bif, Resolution = 7680 x 1080
Image-2 = photo8bbb.bif, Resolution = 7680 x 1080
(read_binfile) Input binary file 'photo8aaa.bif': # of elements read = 8294400
(read_binfile) Input binary file 'photo8bbb.bif': # of elements read = 8294400

Number of pThreads = 1000
start (Pthread): 603378 us
end (Pthread):627887 us
Elapsed time using pthread(only Computation Time): 24509 us

start (Parallel_For): 627887 us
end (Parallel_For):635412 us
Elapsed time using pthread(only Computation Time): 7525 us

start (Pipeline-1D): 759198 us
end (Pipeline-1D):817666 us
Elapsed time using 1-D parallel_pipeline(only Computation Time): 58468 us

start (Pipeline-2D): 824953 us
end (Pipeline-2D):883728 us
Elapsed time using parallel_pipeline(only Computation Time): 58775 us

start (Sequential): 648990 us
end (Sequential): 702314 us
Elapsed time using sequential(only Computation Time): 53324 us
uidr4928@qhws04x:~/manoj/fall2024/lab_hw/final_project_Dec4$
```

```
Image-1 = photo8aaaa.bif, Resolution = 15360 x 1080
Image-2 = photo8bbbb.bif, Resolution = 15360 x 1080
(read_binfile) Input binary file 'photo8aaaa.bif': # of elements read = 16588800
(read_binfile) Input binary file 'photo8bbbb.bif': # of elements read = 16588800

Number of pThreads = 500
start (Pthread): 760480 us
end (Pthread):776711 us
Elapsed time using pthread(only Computation Time): 16231 us

start (Parallel_For): 776711 us
end (Parallel_For):790472 us
Elapsed time using pthread(only Computation Time): 13761 us

start (Pipeline-1D): 37241 us
end (Pipeline-1D):143942 us
Elapsed time using 1-D parallel_pipeline(only Computation Time): 106701 us

start (Pipeline-2D): 162394 us
end (Pipeline-2D):271024 us
Elapsed time using parallel_pipeline(only Computation Time): 108630 us

start (Sequential): 819083 us
end (Sequential): 921146 us
Elapsed time using sequential(only Computation Time): 102063 us
uidr4928@qhws04x:~/manoj/fall2024/lab_hw/final_project_Dec4$
```

```
Image-1 = photo8aaaaa.bif, Resolution = 30720 x 1080
Image-2 = photo8bbbbb.bif, Resolution = 30720 x 1080
(read_binfile) Input binary file 'photo8aaaaa.bif': # of elements read = 33177600
(read_binfile) Input binary file 'photo8bbbbb.bif': # of elements read = 33177600

Number of pThreads = 500
start (Pthread): 223304 us
end (Pthread):245581 us
Elapsed time using pthread(only Computation Time): 22277 us

start (Parallel_For): 245581 us
end (Parallel_For):262557 us
Elapsed time using pthread(only Computation Time): 16976 us

start (Pipeline-1D): 727115 us
end (Pipeline-1D):938058 us
Elapsed time using 1-D parallel_pipeline(only Computation Time): 210943 us

start (Pipeline-2D): 974785 us
end (Pipeline-2D):184116 us
Elapsed time using parallel_pipeline(only Computation Time): 209331 us

start (Sequential): 311579 us
end (Sequential): 507328 us
Elapsed time using sequential(only Computation Time): 195749 us
```

```
Image-1 = photo8aaaaa.bif, Resolution = 30720 x 1080
Image-2 = photo8bbbbb.bif, Resolution = 30720 x 1080
(read_binfile) Input binary file 'photo8aaaaa.bif': # of elements read = 33177600
(read_binfile) Input binary file 'photo8bbbbb.bif': # of elements read = 33177600

Number of pThreads = 50
start (Pthread): 850845 us
end (Pthread):867661 us
Elapsed time using pthread(only Computation Time): 16816 us

start (Parallel_For): 867661 us
end (Parallel_For):885494 us
Elapsed time using pthread(only Computation Time): 17833 us

start (Pipeline-1D): 370454 us
end (Pipeline-1D):582212 us
Elapsed time using 1-D parallel_pipeline(only Computation Time): 211758 us

start (Pipeline-2D): 630262 us
end (Pipeline-2D):840373 us
Elapsed time using parallel_pipeline(only Computation Time): 210111 us

start (Sequential): 941459 us
end (Sequential): 131377 us
Elapsed time using sequential(only Computation Time): 189918 us
uidr4928@qhws04x:~/manoj/fall2024/lab_hw/final_project_Dec4$
```

# Python Script:

**File Name**: test_image.py

**Location**:
In the source code folder.

D: > masters > Fall_semester2024 > Final_Project > report > image_stitching_project > test_image.py > ...

Configuration

```python
import numpy as np
import matplotlib.pyplot as plt

# Read the vector file

Image1 = "photo8a.bif"
Image2 = "photo8b.bif"
TBB_Generated_FileName = "Stitched_TBB_pipeline_1d.bin"

Img_width = 1920 #750
Img_height = 1080 #530


halfImageWidth = Img_width/2

Stitch_img_height = Img_height
Stitch_img_width = (Img_width*2)

n = Img_width

print("Stitch Image Width = ", Stitch_img_width)

Input_vector1 = np.fromfile(Image1, dtype=np.uint8)
Input_matrix1 = Input_vector1.reshape(Img_height, Img_width)

Input_vector2 = np.fromfile(Image2, dtype=np.uint8)
Input_matrix2 = Input_vector2.reshape(Img_height, Img_width)

output_matrix = np.zeros((Stitch_img_height*Stitch_img_width))

#c = Input_matrix1 + Input_matrix2
c = np.concatenate((Input_matrix1, Input_matrix2), axis=1)

w1 = np.zeros(Img_width)
w2 = np.zeros(Img_width)

for i in range (1, Img_width):
    Factor1 = np.uint8(i / halfImageWidth)
    Factor2 = (((Img_width - 1)/i) - 1)
    w1[i] = (Factor1)*(Factor2)
    #print("w1[",i,"] = ",w1[i])

for i in range (0, Img_width):
    if(i < halfImageWidth):
        Factor2 = i / halfImageWidth
        w2[i] = Factor2
        #print("w2[",i,"] = ",w2[i])
```

Restricted Mode ⊗ 0 ⚠ 2 0

D: > masters > Fall_semester2024 > Final_Project > report > image_stitching_project > test_image.py > ...


```python
    Factor2 = (((Img_width - 1)/i) - 1)
    w1[i] = (Factor1)*(Factor2)
    #print("w1[",i,"] = ",w1[i])

for i in range (0, Img_width):
    if(i < halfImageWidth):
        Factor2 = i / halfImageWidth
        w2[i] = Factor2
        #print("w2[",i,"] = ",w2[i])
```


```python
    else:
        w2[i] = 1
        #print("w2[",i,"] = ",w2[i])

for i in range (0, Img_height):
    for j in range (0, Img_width):
        if(j > halfImageWidth):
            temp1 = Input_vector1[(i*n) + j]*w1[j]
            output_matrix[(i*n*2) + j] = np.uint8(temp1 + 0.5)
            output_matrix[(i*n*2) + j + n] = Input_vector2[(i*n) + j]
        else:
            temp1 = Input_vector2[(i*n) + j]*w2[j]
            output_matrix[(i*n*2) + j] = Input_vector1[(i*n) + j]
            output_matrix[(i*n*2) + j + n] = np.uint8(temp1 + 0.5)
output_Image = output_matrix.reshape(Img_height, Stitch_img_width)

vector_data = np.fromfile(TBB_Generated_FileName, dtype=np.uint8)

image_data = vector_data.reshape(Img_height, Stitch_img_width)


# Display

plt.title("Project - Stitcheded Image")
plt.imshow(image_data, cmap='gray')
plt.show()

plt.title("Python - Stitcheded Image")
plt.imshow(output_Image, cmap='gray')
plt.show()

delta_image = image_data - output_Image

plt.title("Difference")
plt.imshow(delta_image, cmap='gray')
plt.show()
```

# Future Scope:

- Combine more than two images of any width and height.
- Compute the transformation matrix (homograph) that aligns the matched key point from different images.
- More sophisticated transformation to make it panoramic view.
- 360-degree stitching view.

# Conclusion:

- The stitching of images is a very practical and extremely useful area which requires a great deal of computing power to achieve seamless output.
- The Parallel programming techniques are most widely used in multi-core CPU based hardware accelerators designed for such SIMD-based computations, and it is capable of fulfilling the computing requirements of image processing.
- The knowledge we gained from High-performance Embedded Programming contributed greatly to the development of this project.

# References


- https://www.intel.com/content/www/us/en/docs/onetbb/developer-guide-api-reference/2021-6/onetbb-developer-guide.html
- https://en.wikipedia.org/wiki/POSIX
- https://en.wikipedia.org/wiki/Image_stitching
- https://pubs.opengroup.org/onlinepubs/7908799/xsh/pthread.h.html
- https://neptune.ai/blog/image-processing-python
- https://courses.cs.washington.edu/courses/cse576/05sp/papers/MSR-TR-2004-92.pdf
- https://github.com/natandrade/Tutorial-Medical-Image-Registration
- https://en.wikipedia.org/wiki/Image_stitching
- https://www.cmor-faculty.rice.edu/~zhang/caam699/p-files/Im-Align2005.pdf

# Appendix - Source Code Details

**Environment and tools:**

- Linux
- TBB Lib
- Pthread
- g++ Compiler
- Python
- make

**Executable File:**

- image_stitch_tbb

**Source Code Files:**

- image_stitch_tbb.cpp
- image_stitch_tbb.hpp
- read_file.c
- read_filrm e.h
- Makefile

- **Command to clean:** make clean
- **Command to build the program:** make all

- **Command to Run:** ./image_stitch_tbb "No. of Pthreads"
  - **Example:** See the screen-shot here

```
uidr4928@qhws04x:~/manoj/fall2024/lab_hw/image_stitching_project$ ^C
130 uidr4928@qhws04x:~/manoj/fall2024/lab_hw/image_stitching_project$ ./image_stitch_tbb 10
Image-1 = photo8a.bif, Resolution = 1920 x 1080
Image-2 = photo8b.bif, Resolution = 1920 x 1080
(read_binfile) Input binary file 'photo8a.bif': # of elements read = 2073600
(read_binfile) Input binary file 'photo8b.bif': # of elements read = 2073600

Number of pThreads = 10
start (Pthread): 325171 us
end (Pthread):328775 us
Elapsed time using pthread(only Computation Time): 3604 us

start (Parallel_For): 328775 us
end (Parallel_For):331979 us
Elapsed time using pthread(only Computation Time): 3204 us

start (Pipeline-1D): 359609 us
end (Pipeline-1D):381155 us
Elapsed time using 1-D parallel_pipeline(only Computation Time): 21546 us

start (Pipeline-2D): 383554 us
end (Pipeline-2D):401767 us
Elapsed time using parallel_pipeline(only Computation Time): 18213 us

start (Sequential): 335503 us
end (Sequential): 347151 us
Elapsed time using sequential(only Computation Time): 11648 us
uidr4928@qhws04x:~/manoj/fall2024/lab_hw/image_stitching_project$
```

No. of Pthreads

# Appendix - Source Code Details

- **Generated Files:**

| Sr. No. | File Name | Generation Methods |
|---|---|---|
| 1 | Stitched_Image_Sequential.bin | Generated using the sequential method. |
| 2 | Stitched_TBB_pipeline_1d.bin | Generated using the TBB parallel Pipeline and Images are passed as 1-D arrays |
| 3 | Stitched_TBB_pipeline_2d.bin | Generated using the TBB parallel Pipeline and Images are passed as 2-D arrays |
| 4 | Stitch_image_pthread.bin | Generated using the PThreads parallelization method. |
| 5 | Stitched_TBB_parallel_for.bin | Generated using the TBB parallel_For. |
| 6 | Noblending_Stitched_Image_Sequential.bin | Generated using the sequential method without applying the Weight Function. |

- The files 1 through 5 are generated using different parallelism methods, but the contents should be the same.
- I have generated File no. 6 in order to illustrate the difference between stitched images generated using weight functions and non-weight functions.

# Appendix - Source Code Details

**Make File:**

Makefile

D: > masters > Fall_semester2024 > Final_Project > report > image_stitching_project > Makefile

```makefile
# Compiler/Linker Setup -------------------------------------
# Linux-specific flags -------------------------------------

PLATFORM = Linux
CC = g++
CFLAGS = -O0 -Wall -g
OSLIBS =
LDLIGS =

OBJS = image_stitch_tbb
all: $(OBJS)

image_stitch_tbb: image_stitch_tbb.cpp image_stitch_tbb.hpp read_file.o
	$(CC) $(CFLAGS) -o image_stitch_tbb image_stitch_tbb.cpp image_stitch_tbb.hpp read_file.o -lm -ltbb -lpthread

#Library
read_file.o: read_file.c read_file.h
	$(CC) $(CFLAGS) -c read_file.c -lm -ltbb -lpthread

# Maintenance and cleaning stauff --------------------------------------

clean:
	rm -f $(OBJS) *.o core

```

# Thank You

# Questions??