GPU Project

PatchMatch - Finding Correspondences in 3D Regions

Team No: 12
Vibhu 2018116, Pragya 2018067

INDRAPRASTHA INSTITUTE *of*
INFORMATION TECHNOLOGY **DELHI**

PatchMatch Algorithm

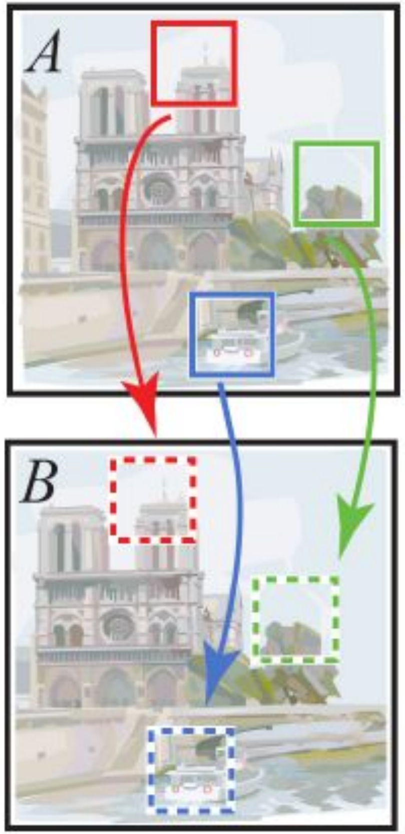
The algorithm tries to find the segments of one image matching the segments of the other image best.

In the most crude sense, given two input images A and B, patchmatch algorithm tries to generate image A from the sections of image B.

It synthesizes complex texture and image structures that resembles input imagery.

Key Idea

1. Large number of random sampling will yield some good guesses.
2. Neighboring pixels have coherent matches.

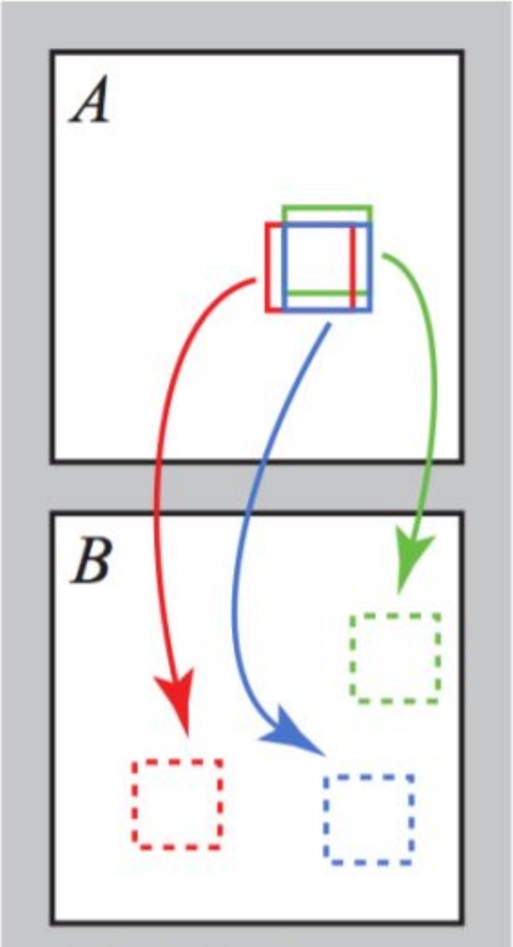


Algorithm

Random Initialization

1. The output image patches are initialised with some random patches of image B.
2. The distances between the guessed patch of B and patch of A are calculated.

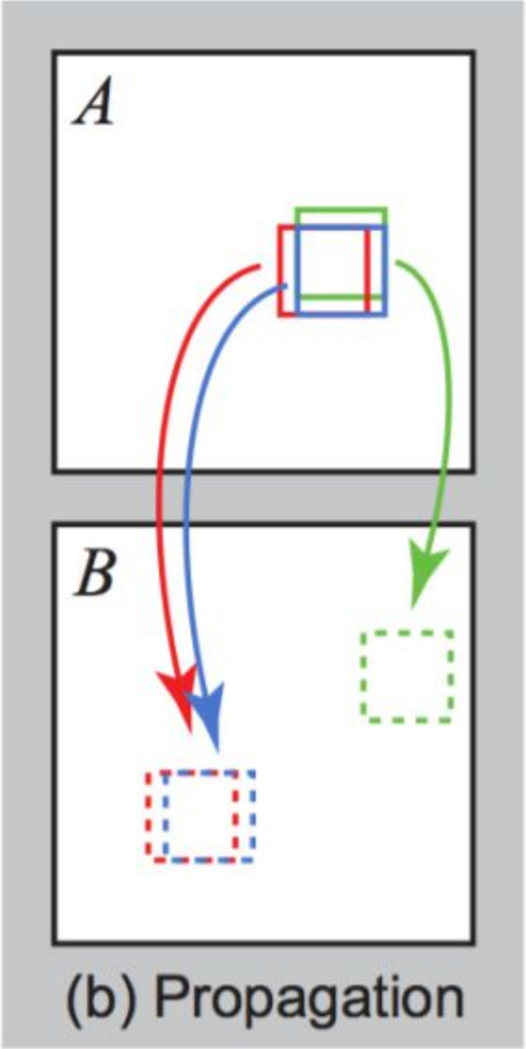
Distance = Σ(r*r) + (g*g) + (b*b) for each pixel in the patch.



(a) Initialization

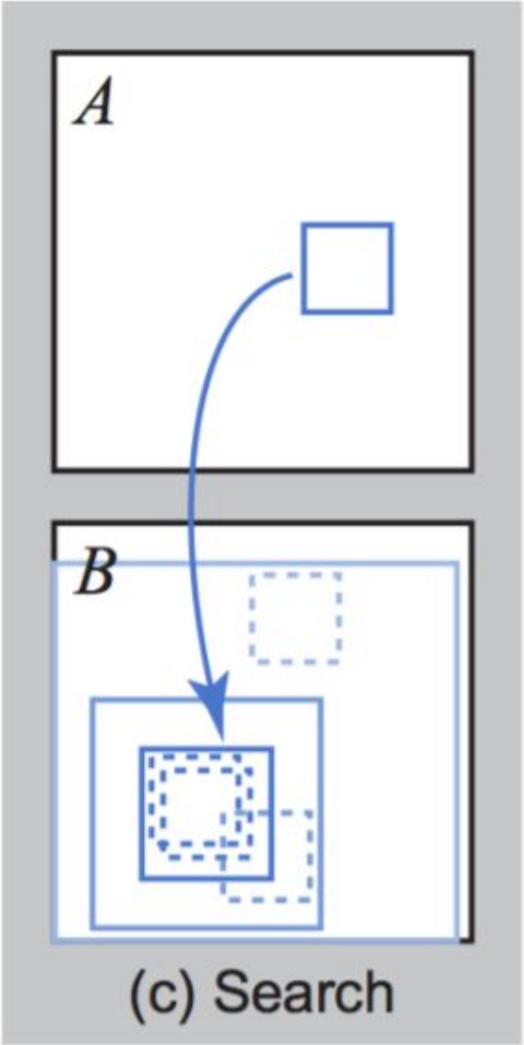
Propagation

1. It is based on the idea that if a particular patch is a good match, then the probability of the surrounding patches to be good is high.
2. Hence the patch mapped to the left and above patches are explored to improve the current patch.



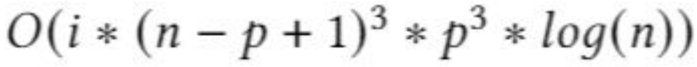
Random Search

1. In order to prevent getting stuck in local minima, a random search over the whole image is performed and the current patch is updated with a patch of smaller distance value if found.



Algorithm Complexity

1. i is the number of iterations
2. n is the size of image
3. p is the patch size



$$O(i * (n - p + 1)^3 * p^3 * log(n))$$

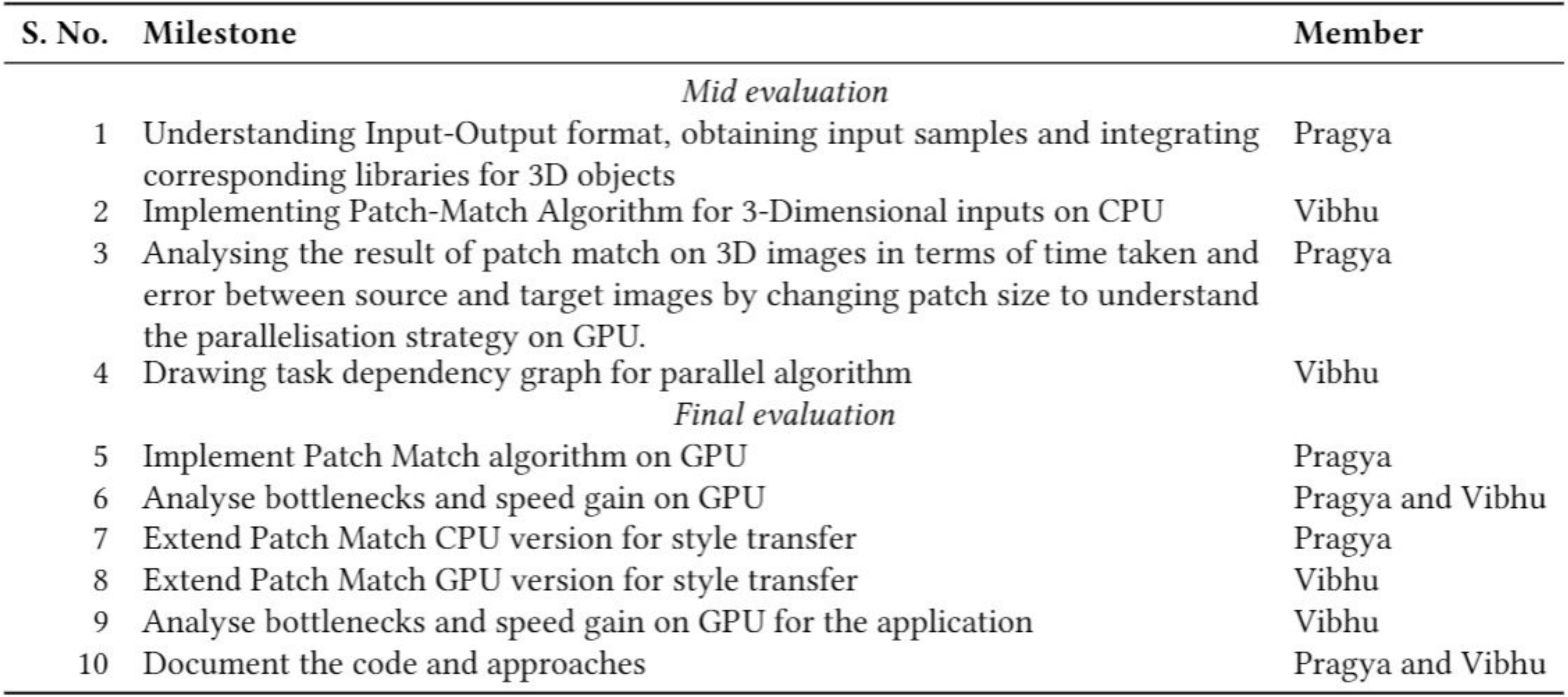
Double Buffer

Modification Proposed for GPU Implementation

Double Buffer

1. The algorithm in its naive implementation is sequential in nature due to checking the previous cells in the same iteration.
2. However, the algorithm can be modified to update distances from previous iteration by double buffering the results.
3. This modification makes the algorithm highly parallel as now each patch can perform propagation and random search independently and parallely.
4. Also, all surrounding patches can be explored now rather than just the ones in scanline.

Milestones



S. No.	Milestone	Member
	Mid evaluation	
1	Understanding Input-Output format, obtaining input samples and integrating corresponding libraries for 3D objects	Pragya
2	Implementing Patch-Match Algorithm for 3-Dimensional inputs on CPU	Vibhu
3	Analysing the result of patch match on 3D images in terms of time taken and error between source and target images by changing patch size to understand the parallelisation strategy on GPU.	Pragya
4	Drawing task dependency graph for parallel algorithm	Vibhu
	Final evaluation	
5	Implement Patch Match algorithm on GPU	Pragya
6	Analyse bottlenecks and speed gain on GPU	Pragya and Vibhu
7	Extend Patch Match CPU version for style transfer	Pragya
8	Extend Patch Match GPU version for style transfer	Vibhu
9	Analyse bottlenecks and speed gain on GPU for the application	Vibhu
10	Document the code and approaches	Pragya and Vibhu

Demo

1. Run code to produce output for slicer

Results

Colour Correspondences

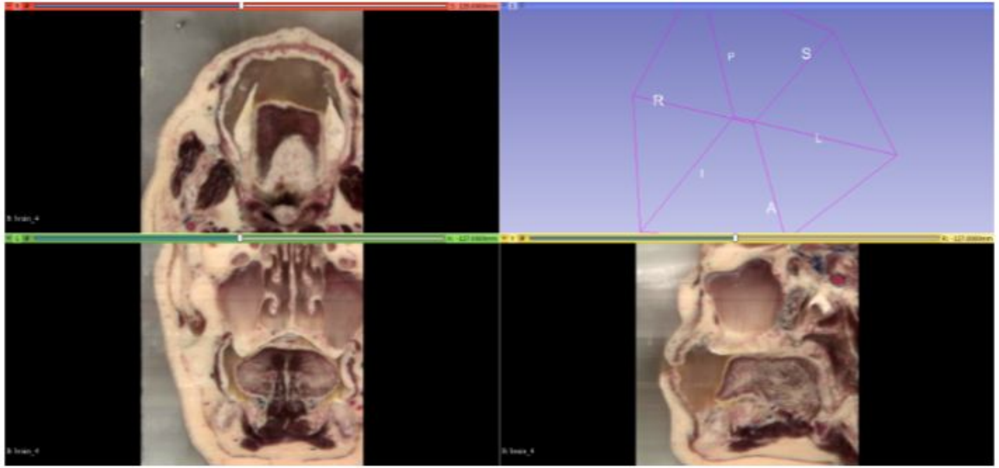


Fig. 8. Input Volume 1

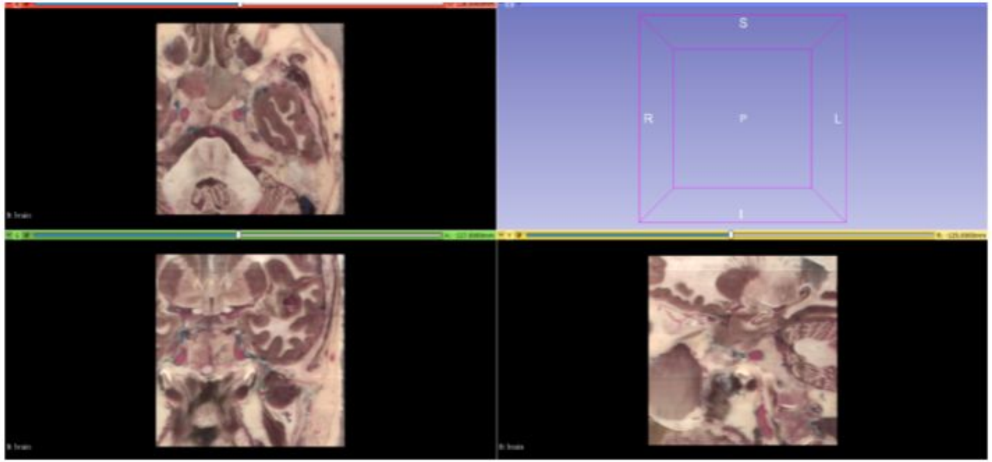


Fig. 9. Input Volume 2

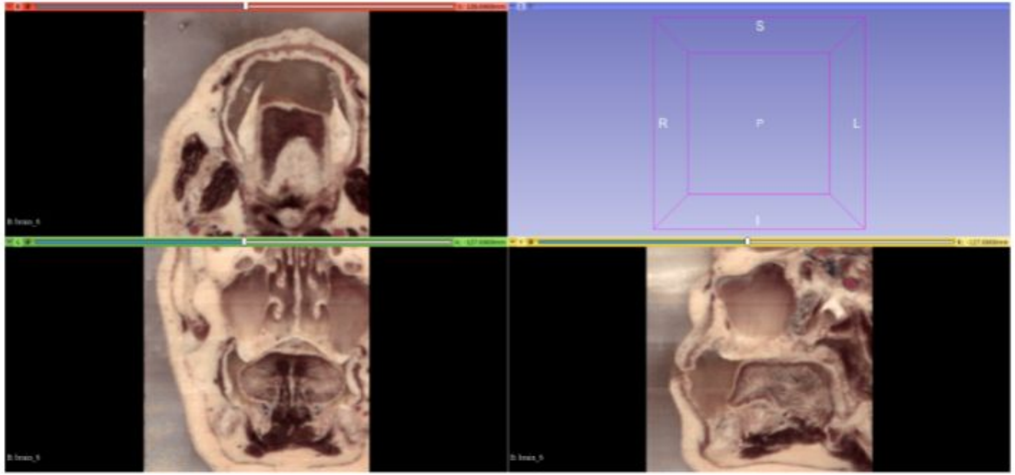


Fig. 10. Output

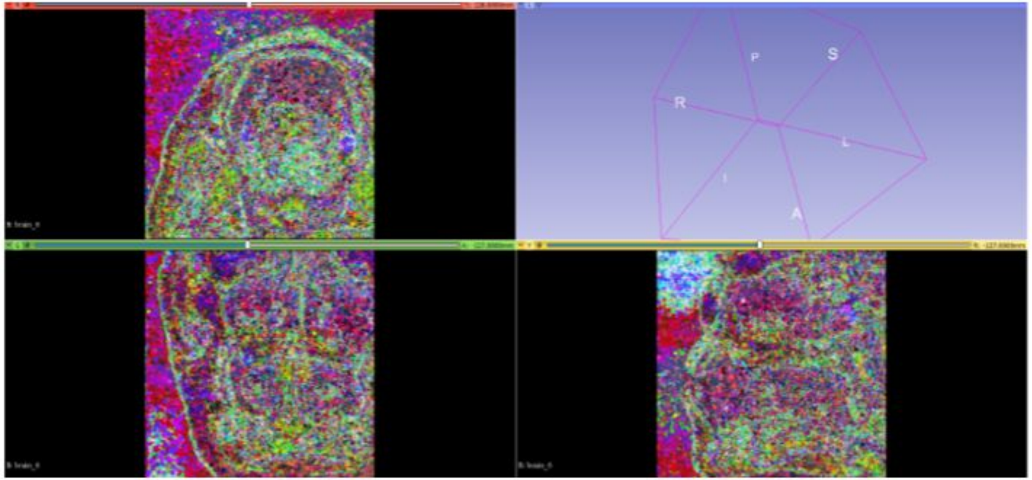
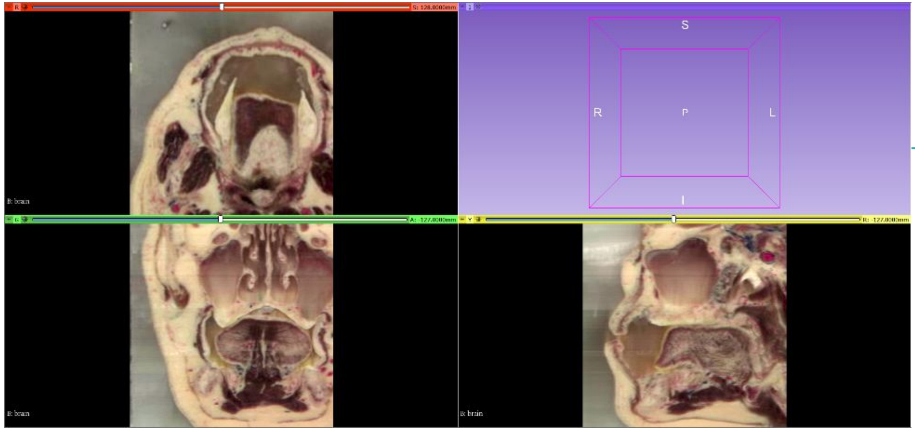
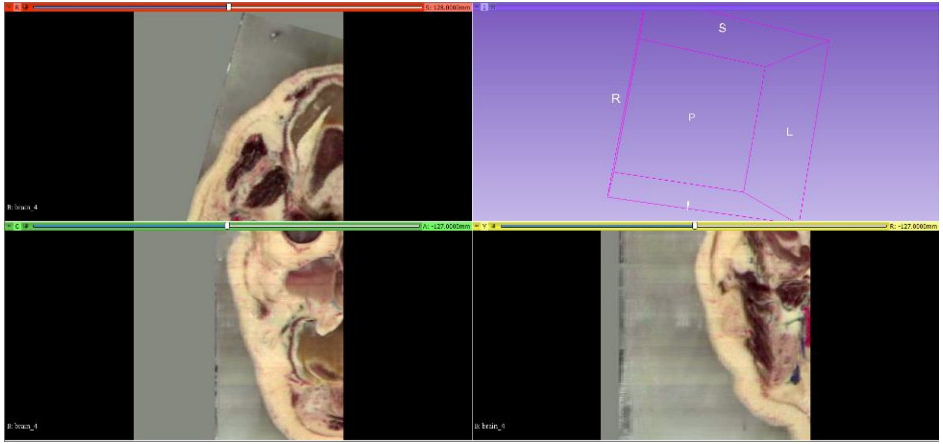


Fig. 11. ANN

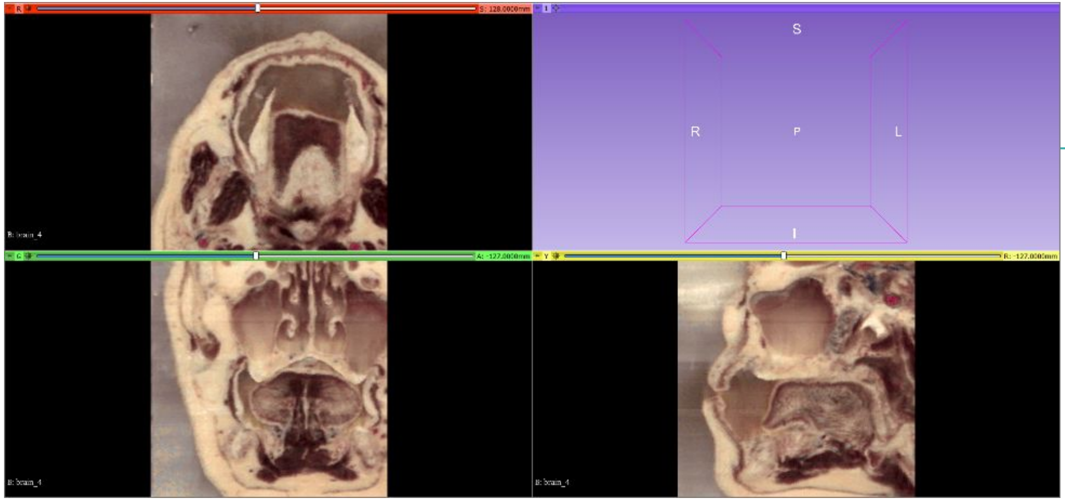
Structure Correspondences



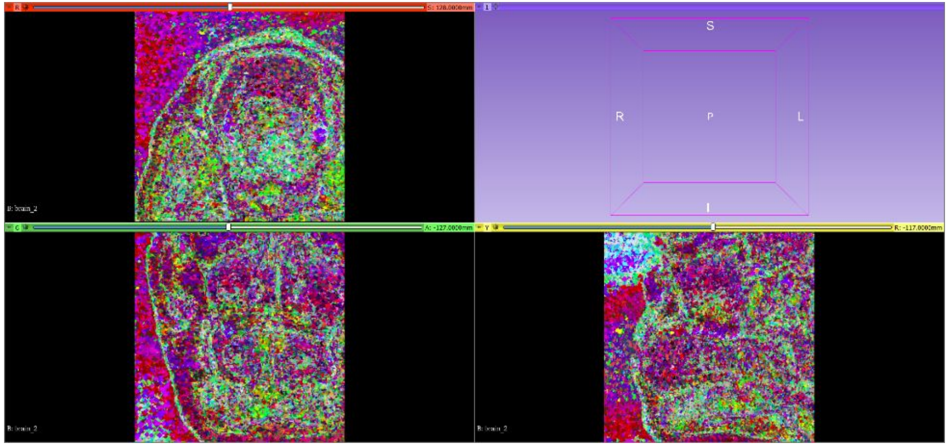
Input Image A



Input Image B (A rotated by 15 degree)



Output Image constructed from rotated image



ANN

Analysis Graphs - CPU

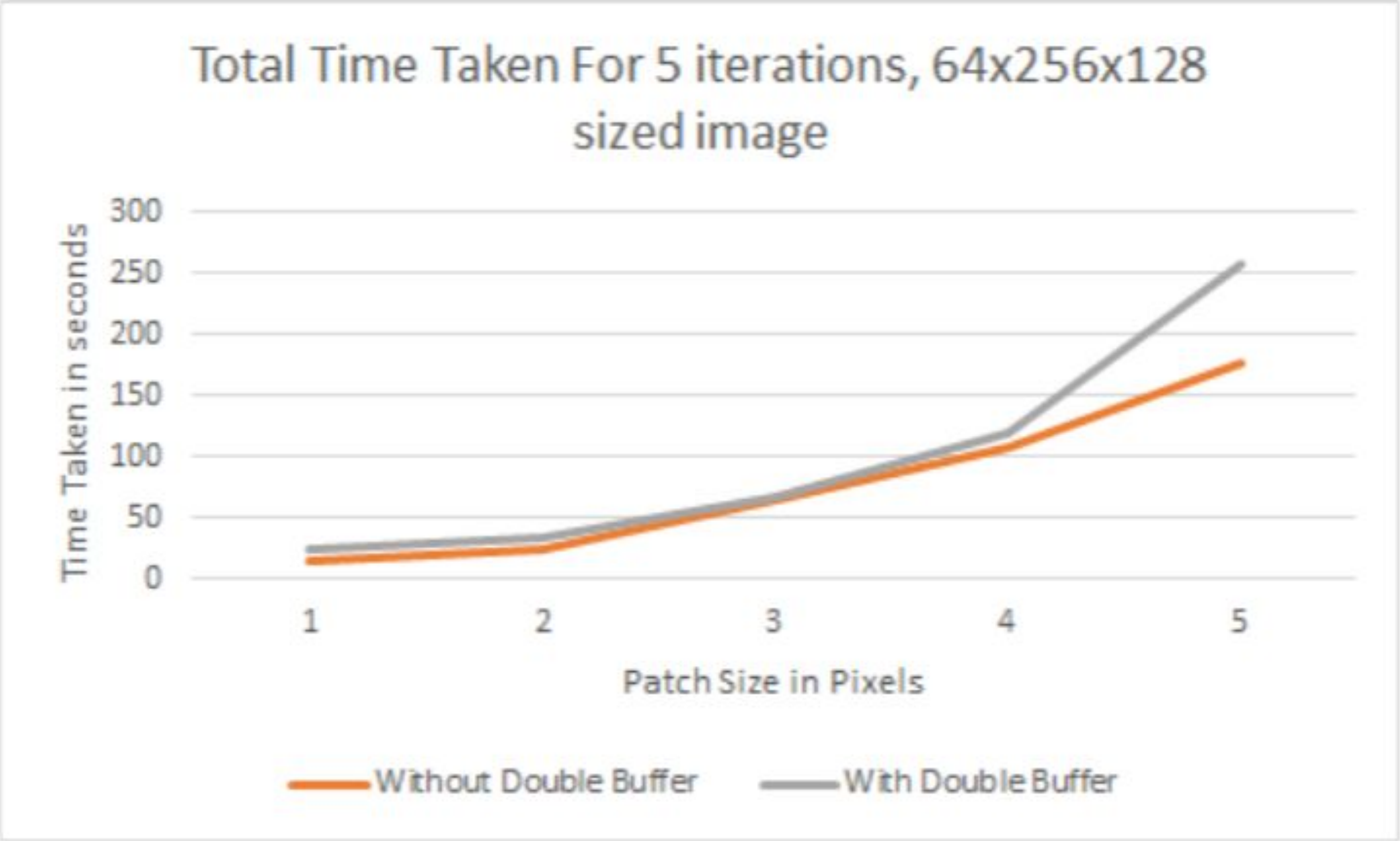


Fig. 4. Comparison of total time taken with and without double buffer on varying patch sizes.

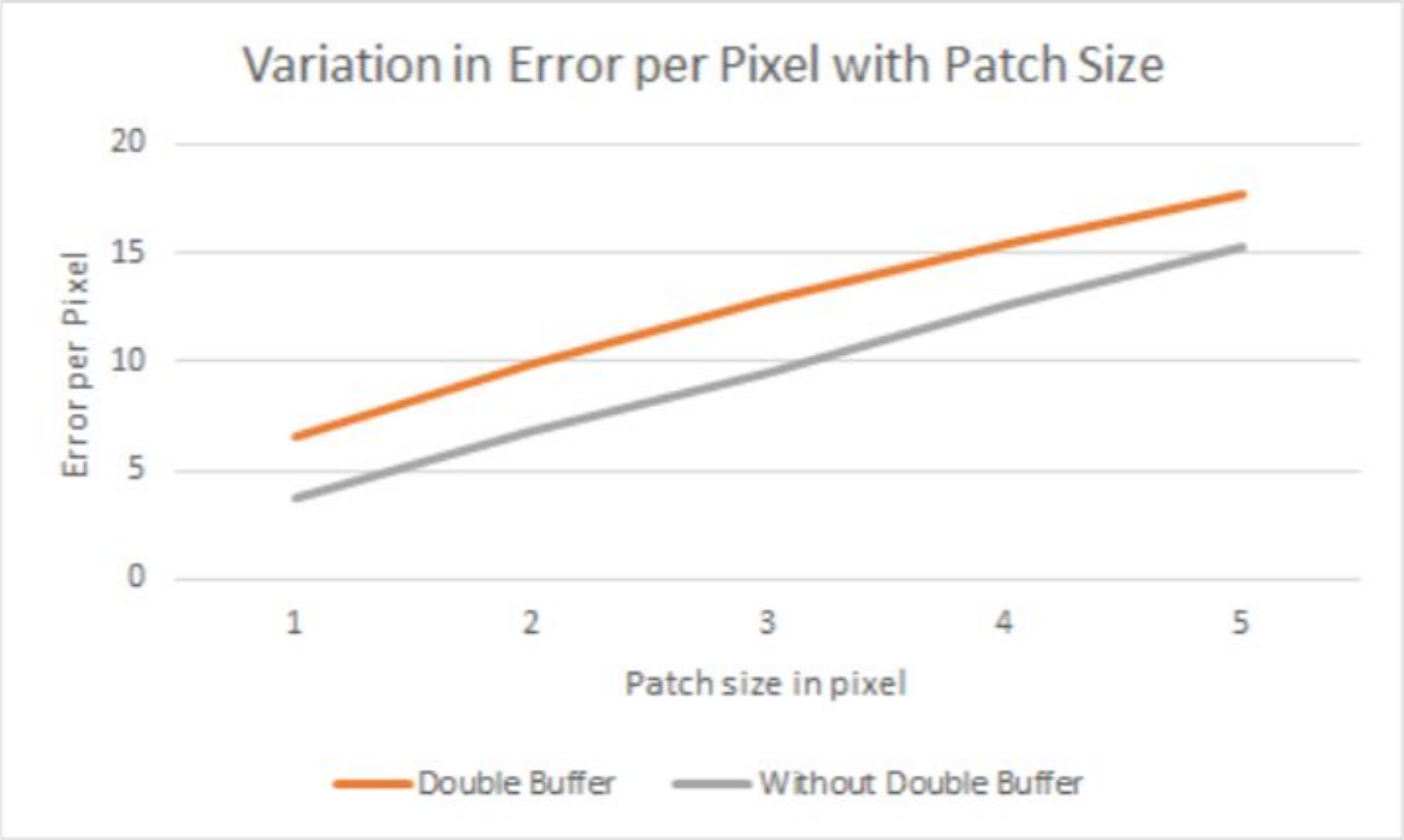


Fig. 5. Variation in error for different patch sizes with double buffer.

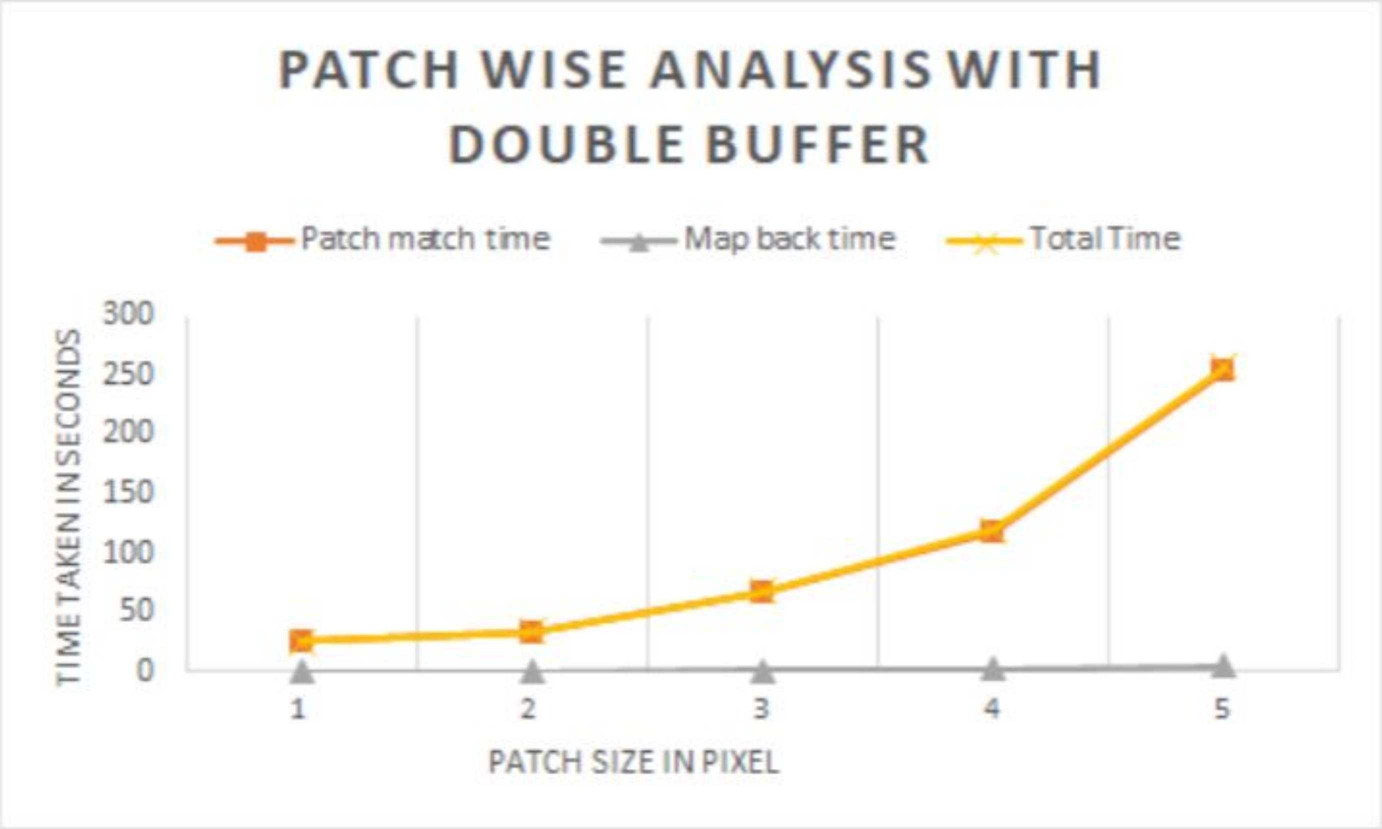
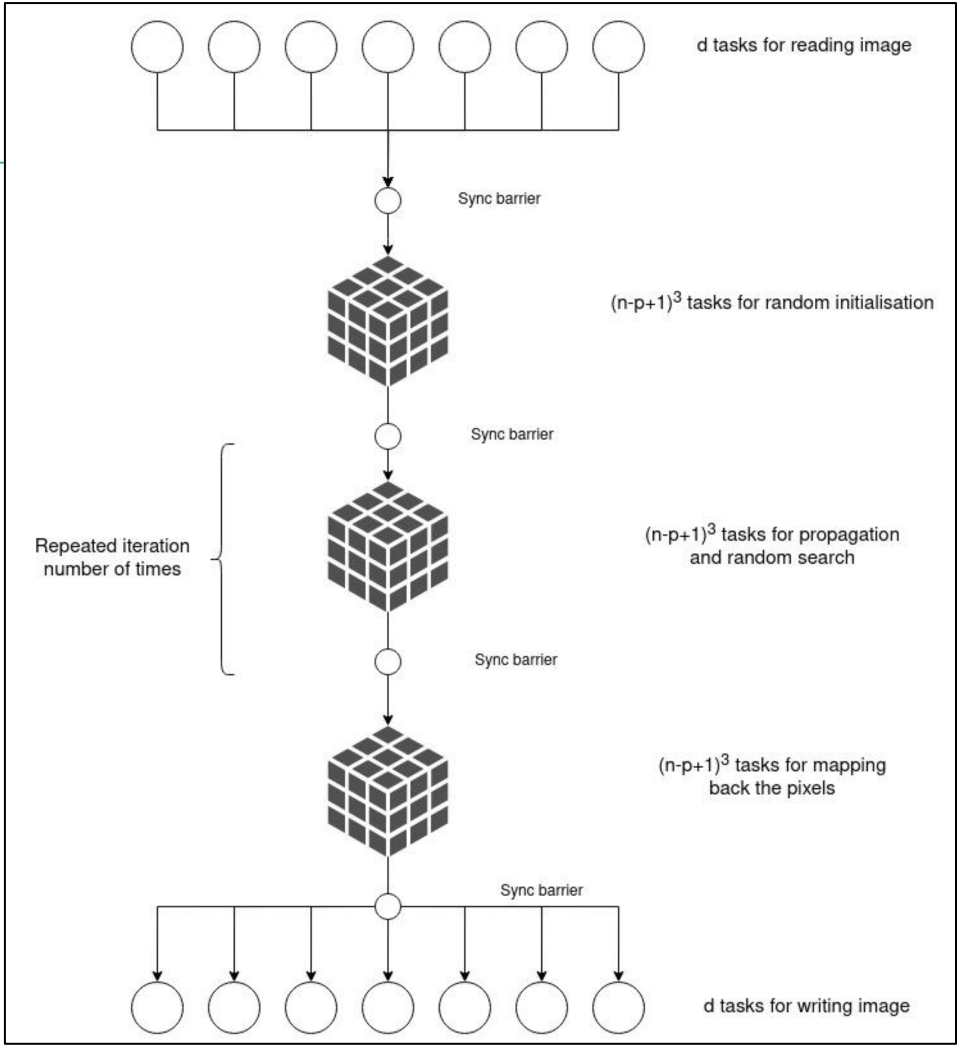


Fig. 3. Variation in time taken for different patch sizes with double buffer.

Task Dependency Graph


d tasks for reading image
Sync barrier
(n-p+1)^3 tasks for random initialisation
Sync barrier
Repeated iteration number of times
(n-p+1)^3 tasks for propagation and random search
Sync barrier
(n-p+1)^3 tasks for mapping back the pixels
Sync barrier
d tasks for writing image

Analysis Graphs - CPU vs GPU

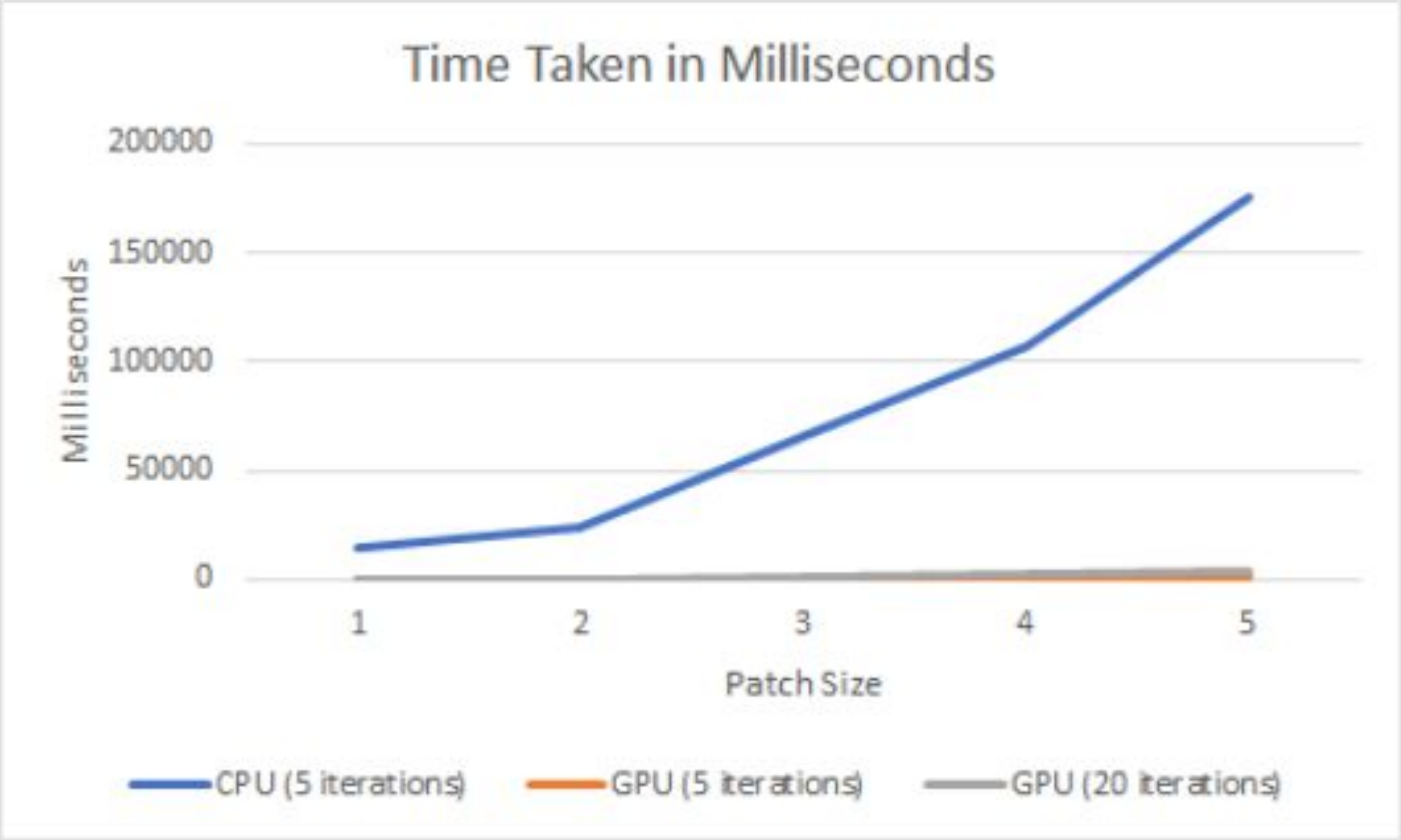


Fig. 16. Variation of Time CPU vs GPU.

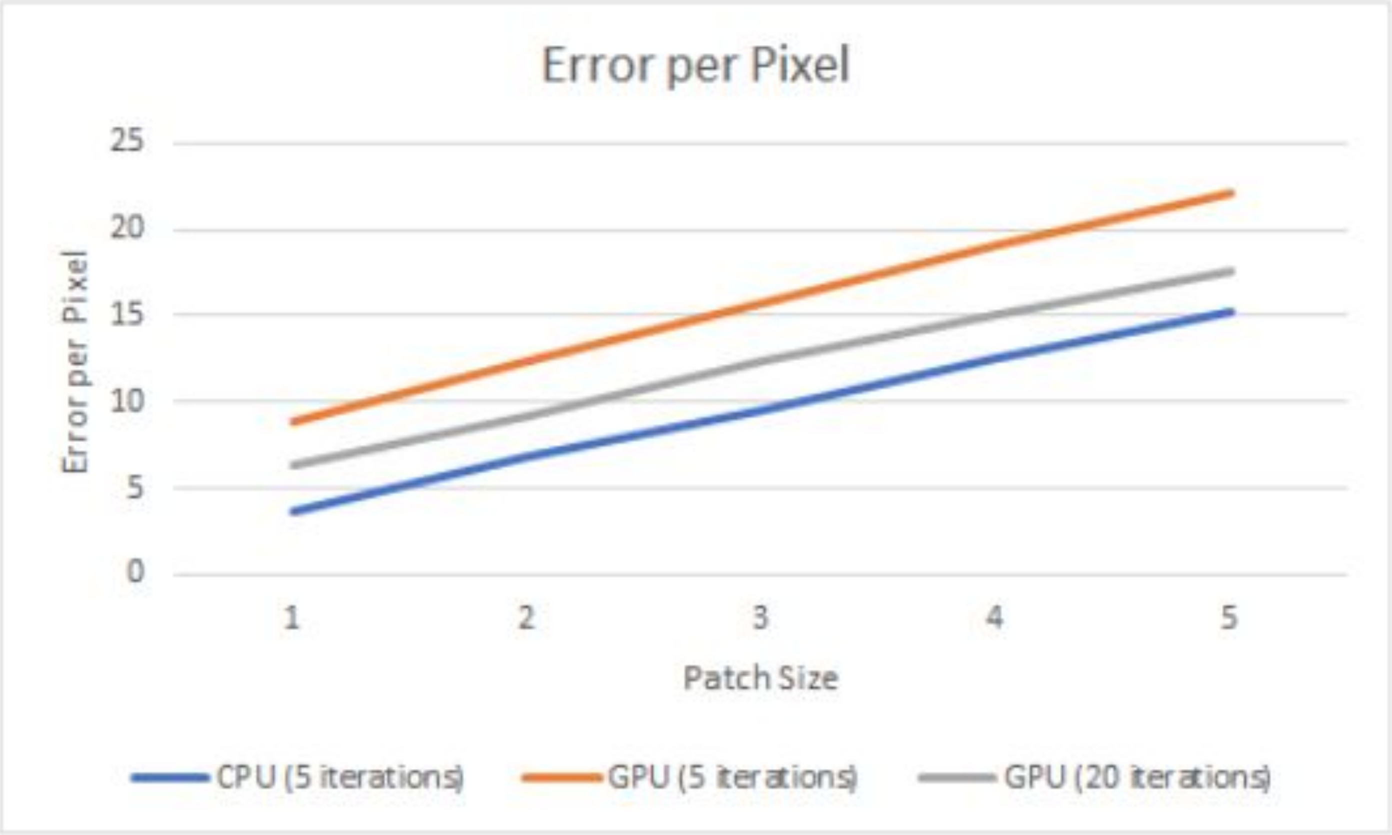


Fig. 17. Variation of Error per Pixel CPU vs GPU.

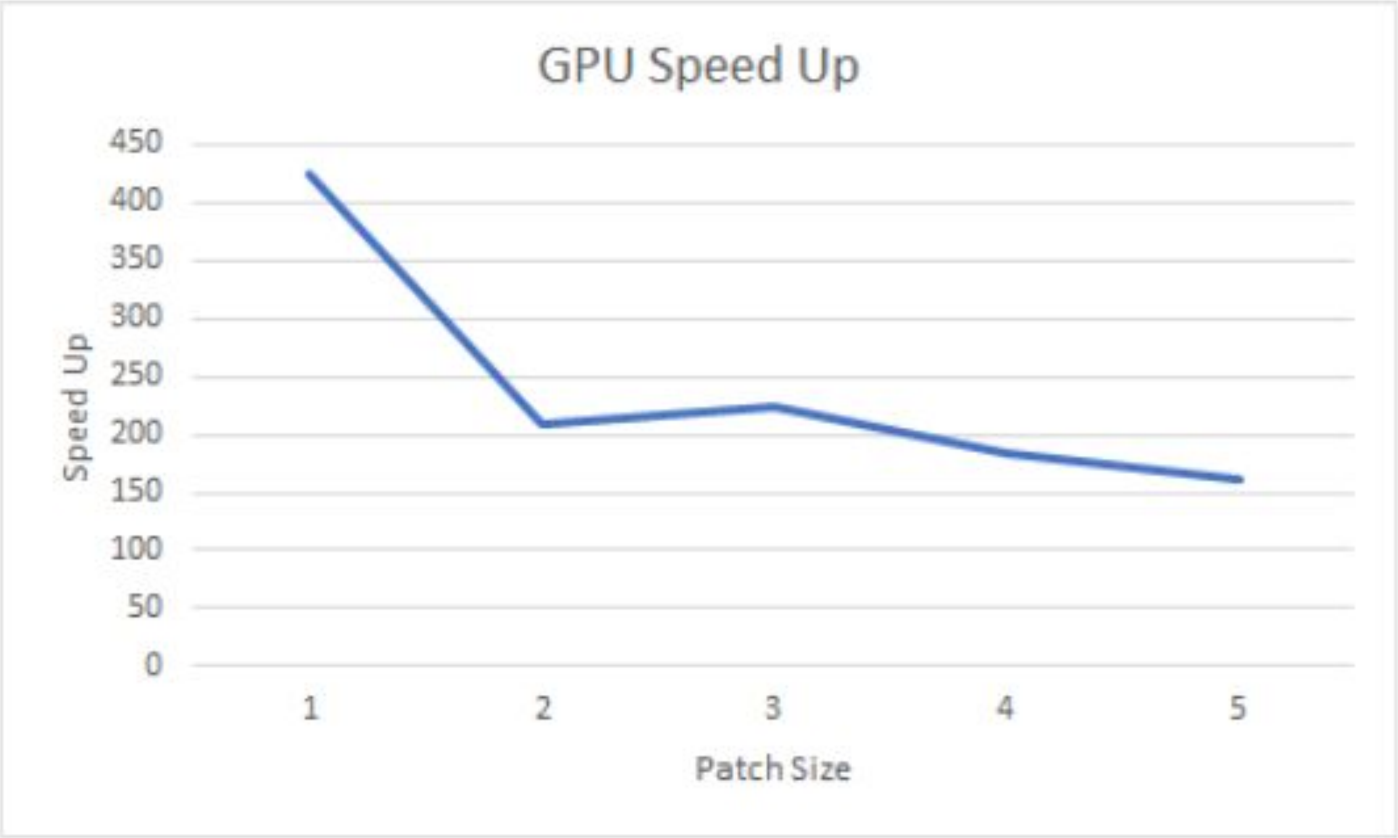


Fig. 18. Speed up on GPU with patch size