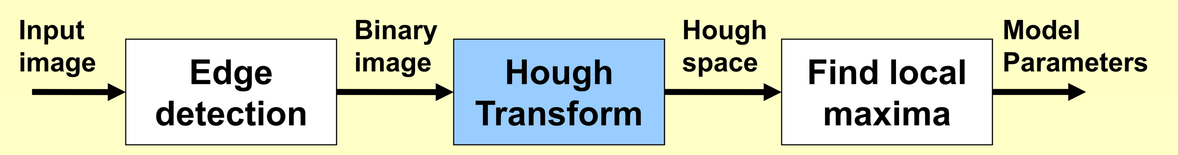
Parallelization of Hough Transform for Circles using CUDA

Faris Serdar TASEL[1,2] Alptekin TEMIZEL[1]
[1]Graduate School of Informatics, Middle East Technical University, Ankara, Turkey
[2]The department of Computer Engineering, Çankaya University, Ankara, Turkey

Hough Transform (HT)

- Hough Transform (HT) is a well-known technique used for detection of parametric shapes in image processing.
- However, various optimizations are necessary in its implementation due to large memory and computational requirements.

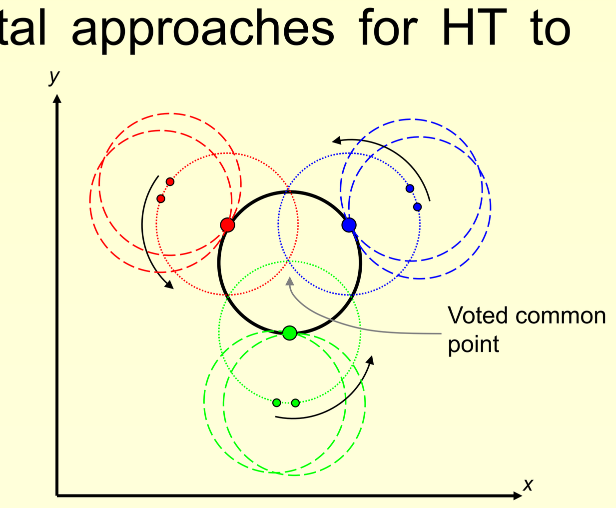


- Hough transform may simply be used with an edge detector to obtain local maxima of Hough space.
- The peak values which are greater than a threshold in Hough space give the model parameters of detected shapes.

Circle Detection using HT

- A 2D circle may be denoted by three parameters: center location (a, b) and radius (r).
- A circle is voted for if model parameters of the circle satisfy the points in input image (x, y).
- There are two fundamental approaches for HT to detect circles.

$$x = a + r\cos\theta$$
$$y = b + r\sin\theta$$



Approach 1

- Use parametric equation of circle.
- Determine a range for r.
- Solve circle centers (a, b) for different r and (x, y).
- Vote for (a, b, r).

$$(x-a)^2 + (y-b)^2 = r^2$$

Approach 2 (*Version 1*)

- Use conventional equation of circle.
- Solve circle radius r for different (a, b) and (x, y).
- Vote for (a, b, r).

$$a = x \pm \sqrt{r^2 - (y-b)^2}$$

Approach 2 (*Version 2*)

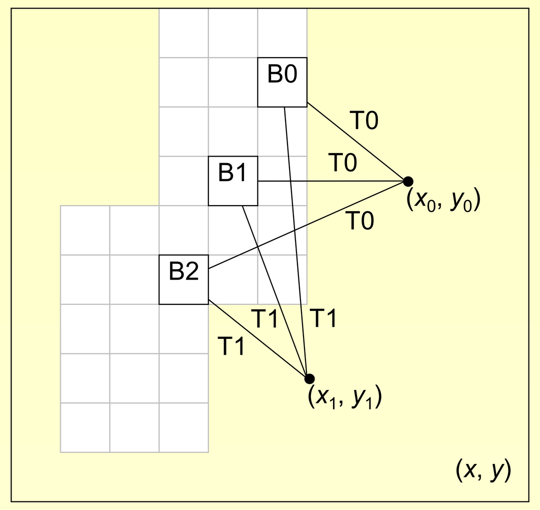
- Solve circle center abscissa a for different (r, b) and (x, y).
- Determine a range for r.
- Circle center ordinate b must satisfy: $y - r \le b \le y + r$
- Vote for (a, b, r).

Parallelization on CUDA

- The parallelization of HT may be accomplished by dividing and sharing the search space among thread-blocks and threads.
- In **approach 1**, accumulation is done on 2D array. Not suitable for shared memory!
- In **approach 2**, accumulation is done on 1D array. Suitable for shared memory. Three versions are proposed.

Version 1 (*Based on Approach 2 Version 1*)

- (a, b) pairs are shared by thread-blocks. (x, y) pairs are shared by threads.

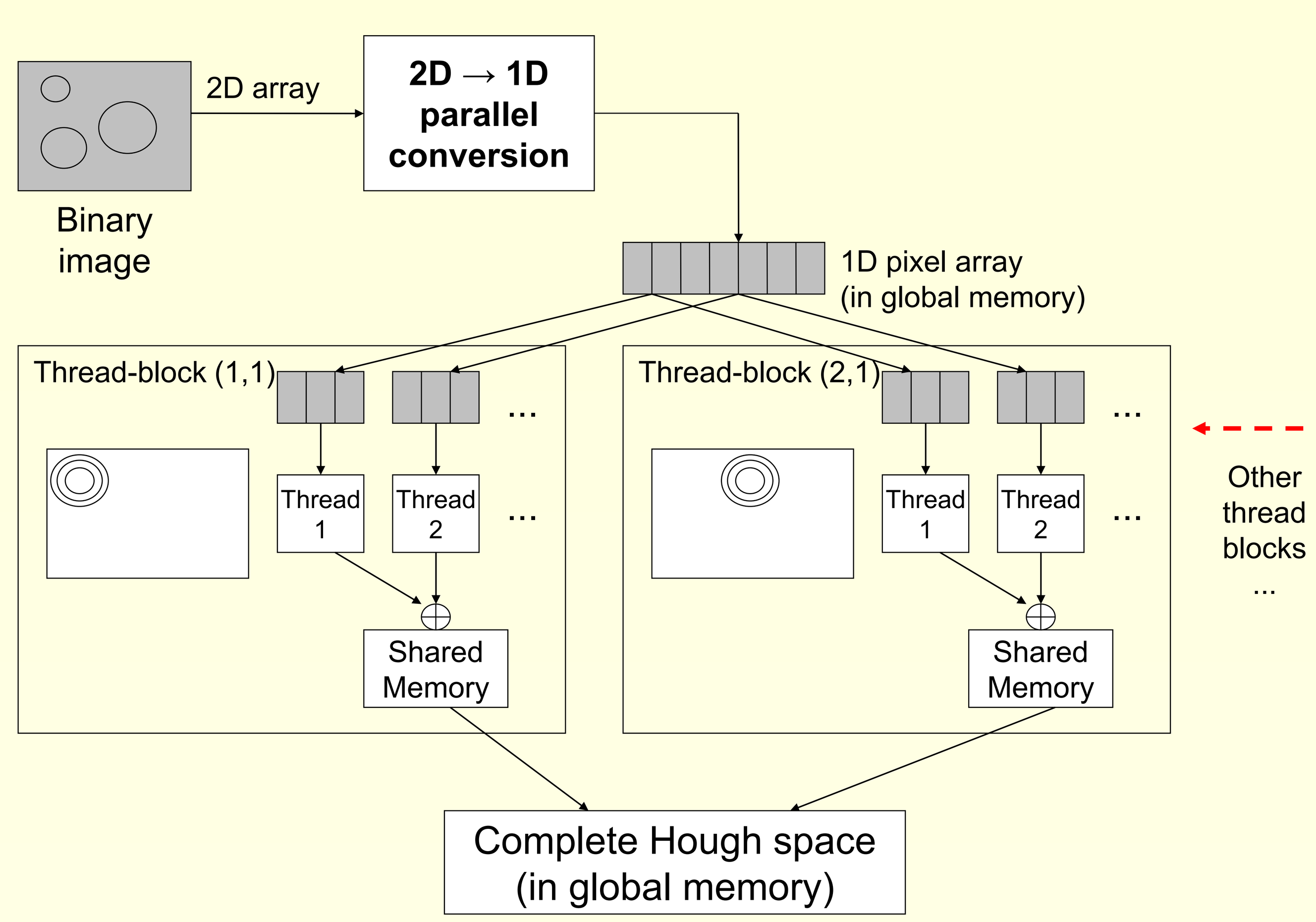


Thread-blocks {B0, B1, B2 ...} sharing circle centers (a, b) and threads {T0, T1 ...} sharing the points {(x_0, y_0), (x_1, y_1) ...} in the image.

- What if the image point (x, y) scanned by a thread is empty pixel?
- Most of the threads will be idle if we consider binary image as a sparse matrix.

Version 2 (*Based on Approach 2 Version 1*)

- HT is the same as Version 1.
- 2D binary image to 1D pixel array conversion is carried out before HT.



2D Binary image to 1D pixel array conversion

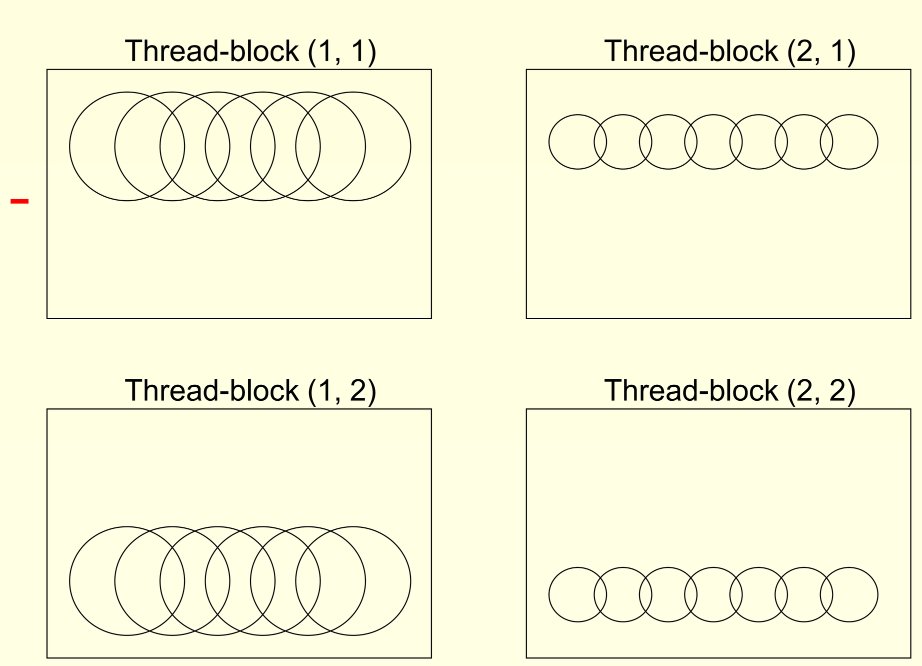
- Conversion process is based on the paper by Braak *et al.* ACIVS'2011.
- The image is divided into tiles for the thread-blocks to process. Threads in each thread-block search the pixels in collaboration and construct a 1D pixel array in the shared memory which comprises 4 byte integers where the coordinate data are packed.
- Then, the first thread in each thread-block calculates an offset position in the global list. Finally, the threads copy the 1D pixel array from the shared memory to the global memory.

```
register_index = -1
register_pixel_value = global_image[x,y]
if (register_pixel_value == 1) {
    do {
        register_index++
        shared_index = register_index
        shared_array[register_index] = (x,y)
    } while (shared_array[register_index] != (x,y))
}
index = shared_index
```

- Conversion is carried out by threads in race condition. Each thread writes an item into the array and check whether the item is correct. If not, then it tries to write into next location.

Version 3 (*Based on Approach 2 Version 2*)

- Conversion phase is applied similar to *Version 2*.
- (r, b) pairs are shared by thread-blocks. (x, y) pairs are shared by threads.
- Hough space initialization and copying operations are also done via unrolled loops to the contrary of the second version.



- The parameter b cannot be adjusted with respect to the inequality. All rows are searched instead.

This research was supported by CUDA Research Center of Middle East Technical University

Results

Used hardware: **Intel i3 3.33 Ghz CPU, NVIDIA Tesla C2070 GPU**

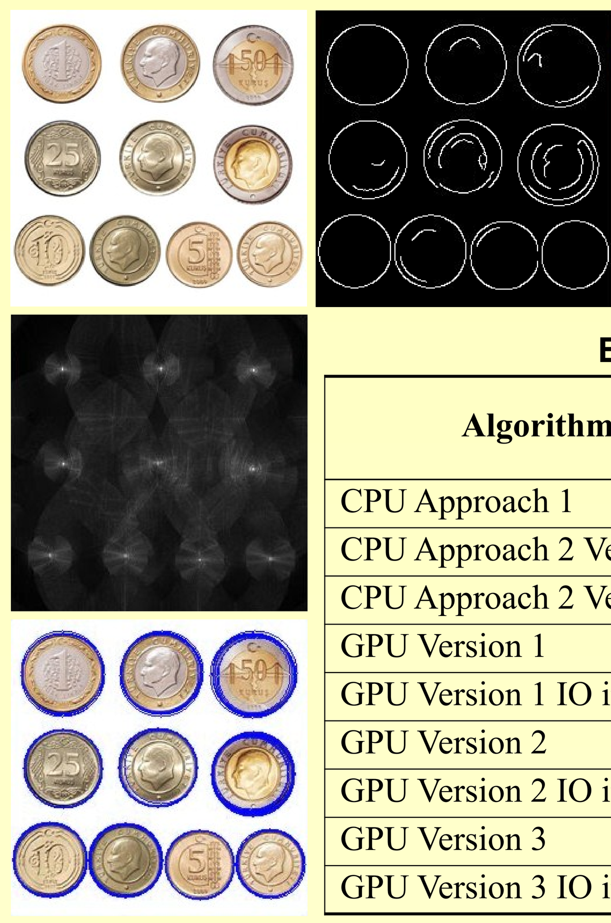


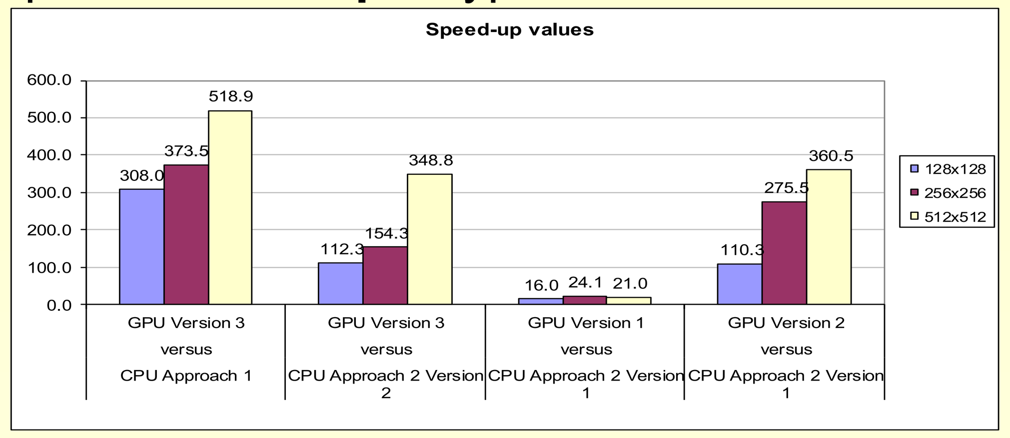
Image Details

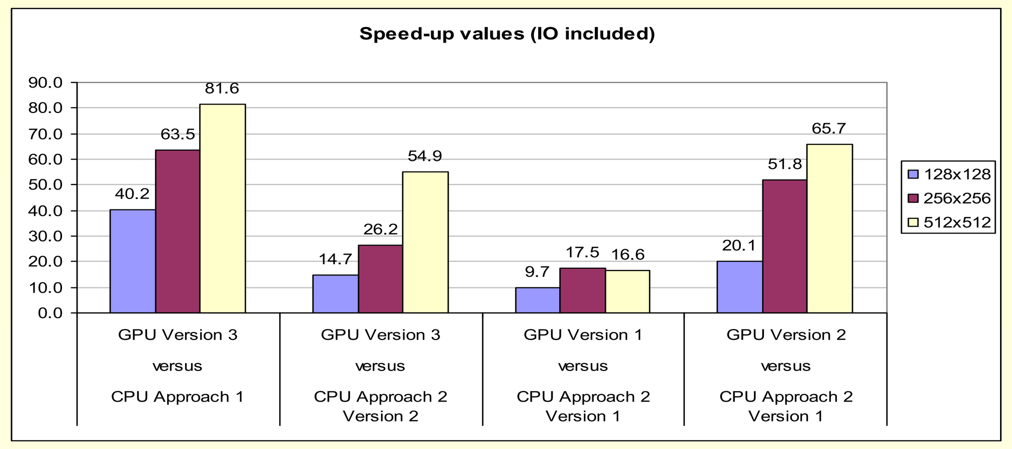
Image Size	# of edge pixels	Radius range
128x128	1466	[10, 25] px
256x256	3217	[20, 50] px
512x512	6059	[40, 100] px

Execution Times (ms)

Algorithm	Image Size		
	128x128	256x256	512x512
CPU Approach 1	92.4	933.8	9133.2
CPU Approach 2 Version 1	430.0	8815.3	87921.0
CPU Approach 2 Version 2	33.7	385.7	6139.2
GPU Version 1	26.9	365.7	4189.3
GPU Version 1 IO included	44.3	504.3	5282.1
GPU Version 2	3.9	32.0	243.9
GPU Version 2 IO included	21.4	170.3	1337.9
GPU Version 3	0.3	2.5	17.6
GPU Version 3 IO included	2.3	14.7	111.9

- Each algorithm was executed 30 times.
- The number of threads is 128 (found to be the best performing value). The grid dimension is equal to the image size for GPU Version 1&2 and (Image height x Radius range) for GPU Version 3.
- The algorithms were grouped and compared with respect to the **output** types.





- Speedup increases with image size.
- Up to **~360** times speedup for GPU Version 2 and up to **~350** times speedup for GPU Version 3.
- Speedups degrade significantly when IO transfer time is taken into consideration: up to **~66** times speedup for GPU Version 2 and up to **~55** times speedup for GPU Version 3.

References

[1] Gonzalez R.C., Woods R.E., 2007. Digital Image Processing, 3rd Edition, Prentice Hall.
[2] Hough P.V.C. Method and Means for Recognizing Complex Patterns. 1962.
[3] Ujaldon M., Ruiz A., Guil N. On the Computation of the Circle Hough Transform by a GPU Rasterizer. Pattern Recognition Letters, 29(3), 309-318, 2008.
[4] Braak G.-J., Nugteren C., Mesman B., Corporaal H. Fast Hough Transform on GPUs: Exploration of Algorithm Trade-Offs. Advances Concepts for Intelligent Vision Systems, Lecture Notes in Computer Science, 6915(2011), pp. 611-622, 2011.