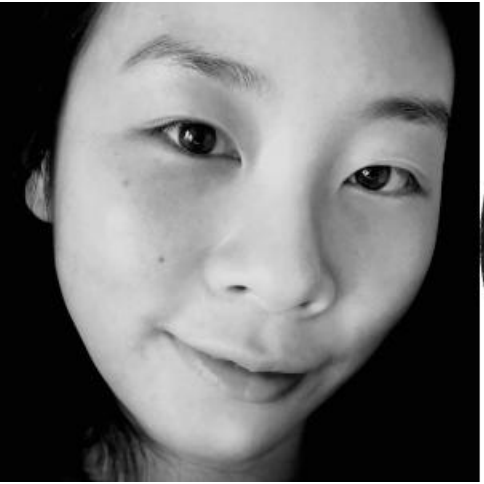
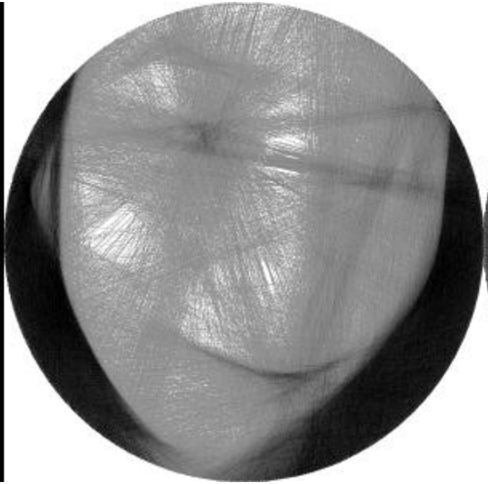
Parallel String Art

Catherine Yu & Nanxi Li

Background

String art is a technique for the creation of visual artwork where images emerge from a set of strings that are spanned between pins[1].

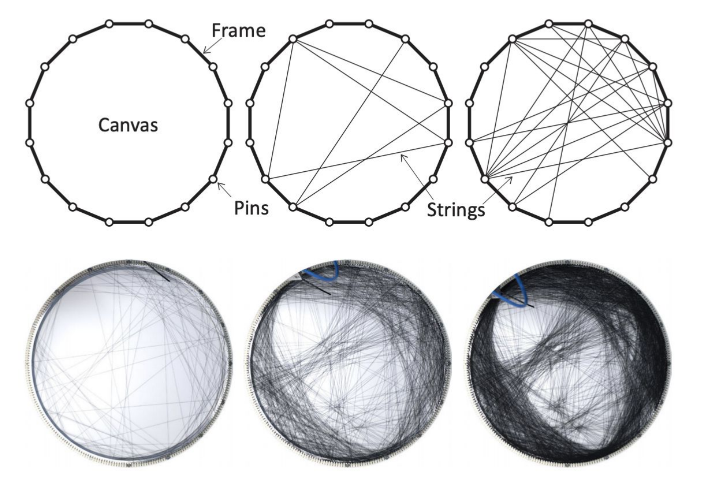
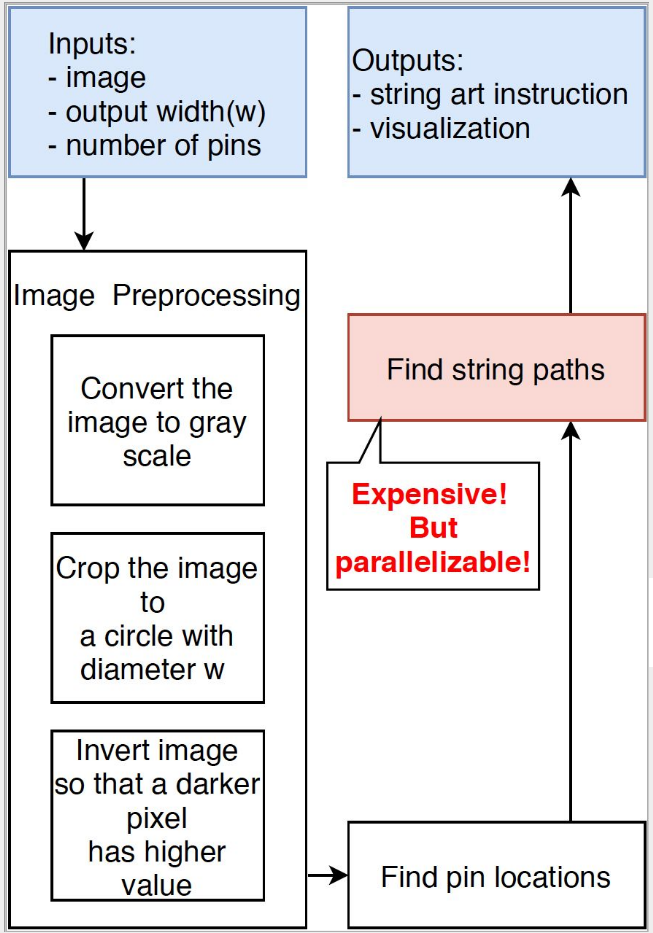


image taken from Brisak et al. [0]

Workflow



Optimization Problem:

Given input image, we denote it as column vector:

$$y \in [0,255]^m \subset \mathbb{Z}^m$$

where m is the number of pixels. The output is a binary vector: $x \in \mathbb{B}^n$

where n is the number of all possible pin pairs (P)(P-1), where P is the number of the pins. The goal is to find a best mapping F from the space of edges to the space of pixels:

$$F : \mathbb{B}^n \rightarrow [0,255]^m \text{ with } x \mapsto F(x)$$

and to determine the values of the elements of the vector x such that it delivers the best approximation of the input image.

$$\min_x \|F(x) - y\|^2 \text{ s.t. } x \in \mathbb{B}$$

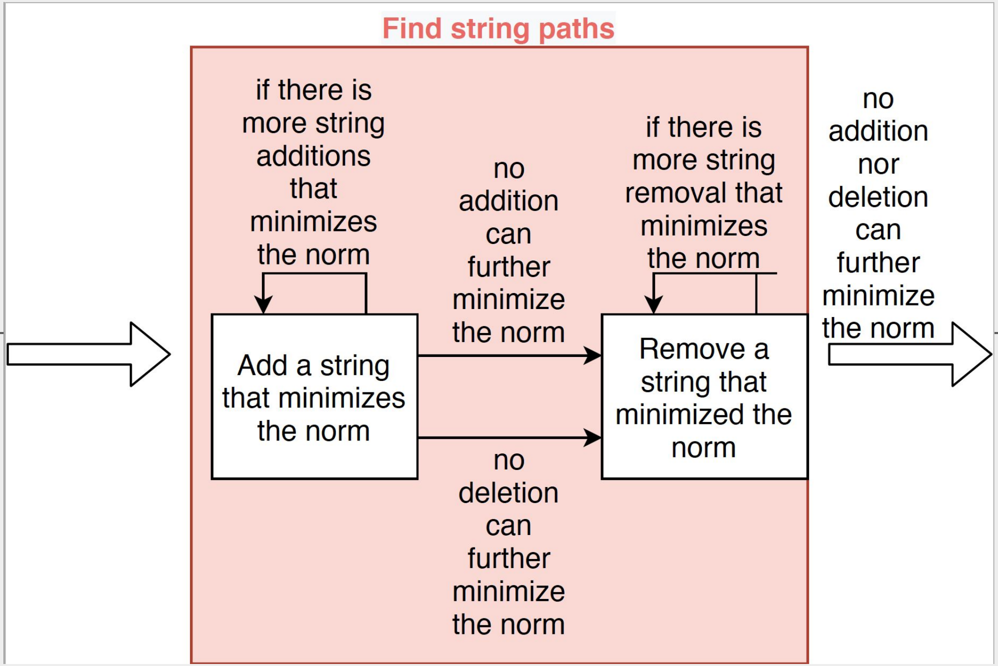
Approach

try adding by increasing the pixel value

try removing by decreasing the pixel value

sequential

parallel




```
BlockDim(16, 16)
gridDim((P+15)/16,(P+15)/16
```

thread ⇔ one of all pin pairs

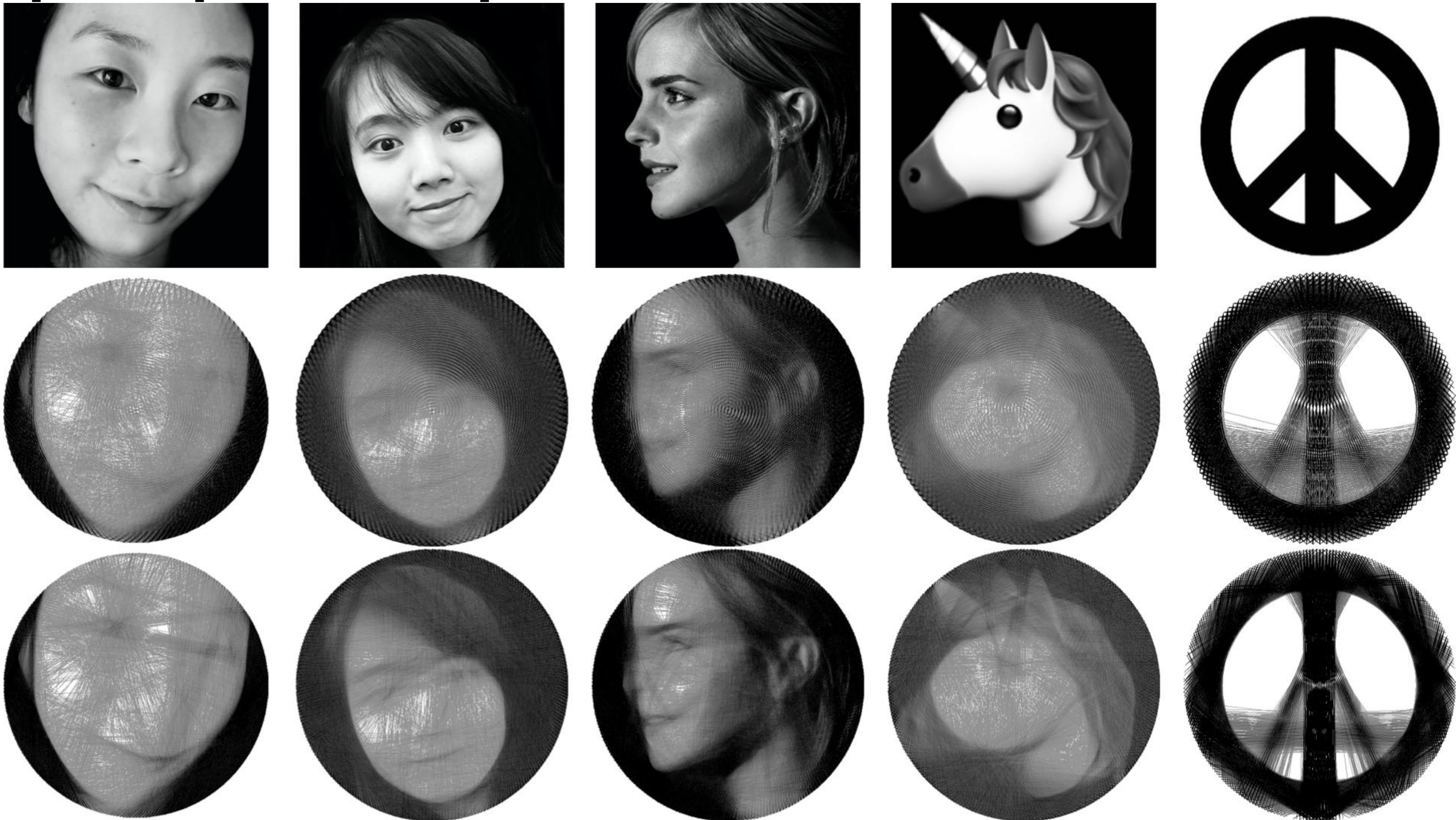
```
blockDim(256)
gridDim((L+15)/16
```

thread ⇔ one selected pin pair

What worked

- Implemented sequential version from scratch based on paper
- Changes made to proposed algorithm to exploit parallelism, achieve better speedup, reduce memory footprint, and produce 'prettier' output:
    - Data structure changes: queue => array
    - Algorithm change: instead of setting a pixel to be either white or black in the constructed image, use value in range [0, 255]; increase pixel value when adding; decreasing pixel value when removing
- **Parallel version: Line level parallelism**
    - finding a line to add: parallelize over all pin pairs (O(P^2))
    - finding a line to remove: parallelize over pin pairs added (O(P^2))

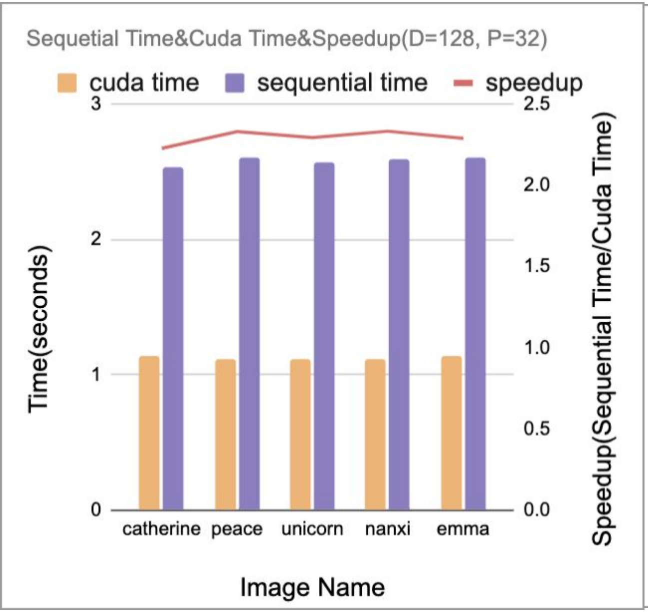
Example Inputs&Outputs

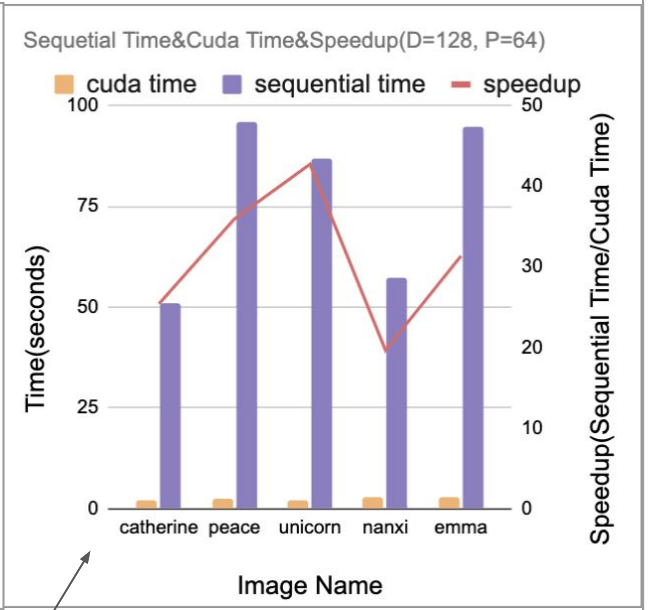


D=512
P=128

D=512
P=256

Speedup Comparisons





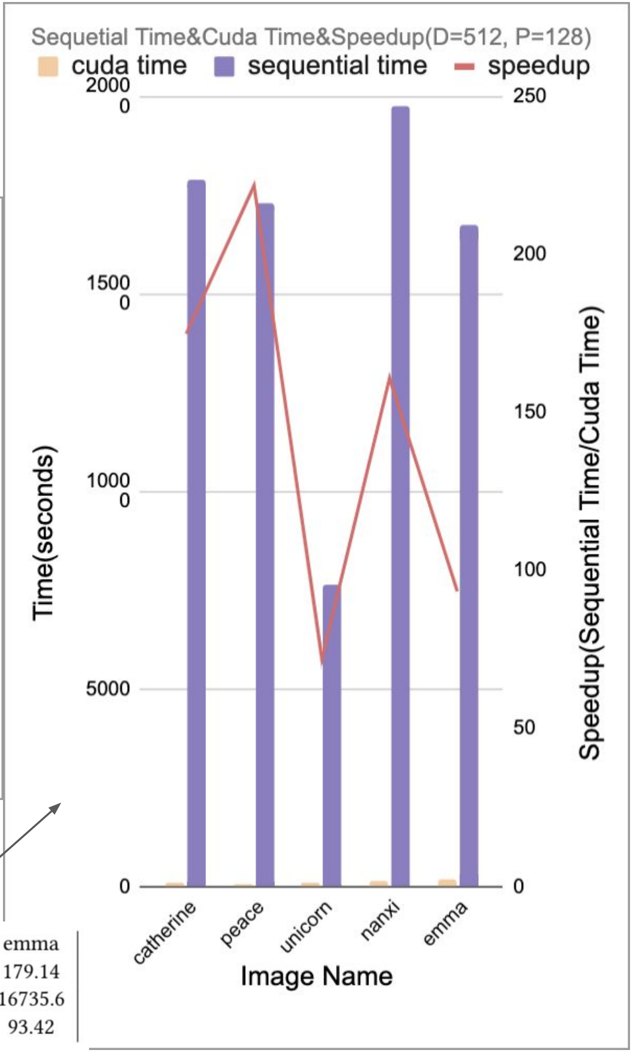


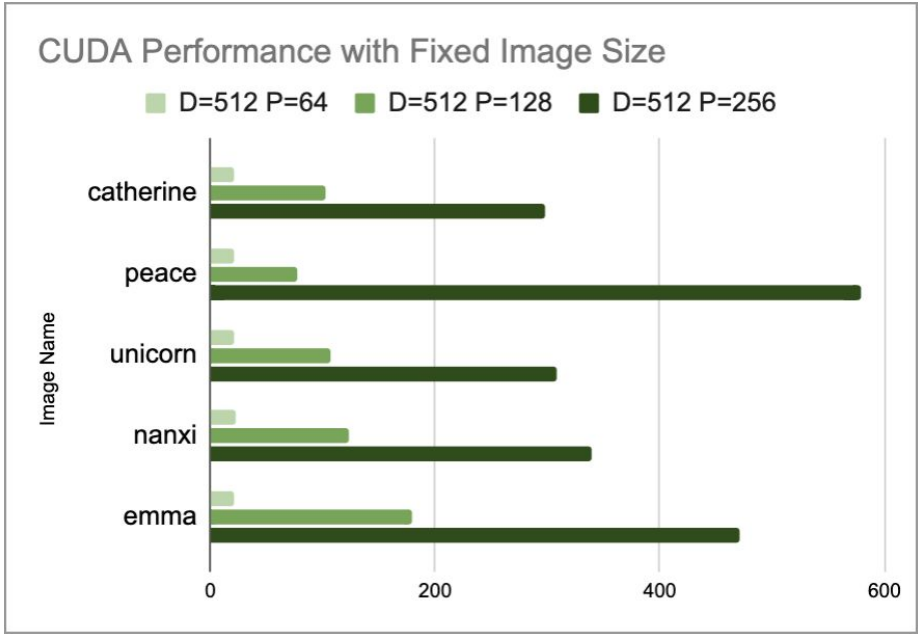
Image Name	catherine	peace	unicorn	nanxi	emma
CUDA time(s)	2.01	2.67	2.03	2.91	3.02
sequential time(s)	51.06	95.9	86.75	57.25	94.67
speedup	25.40	35.92	42.73	19.67	31.35

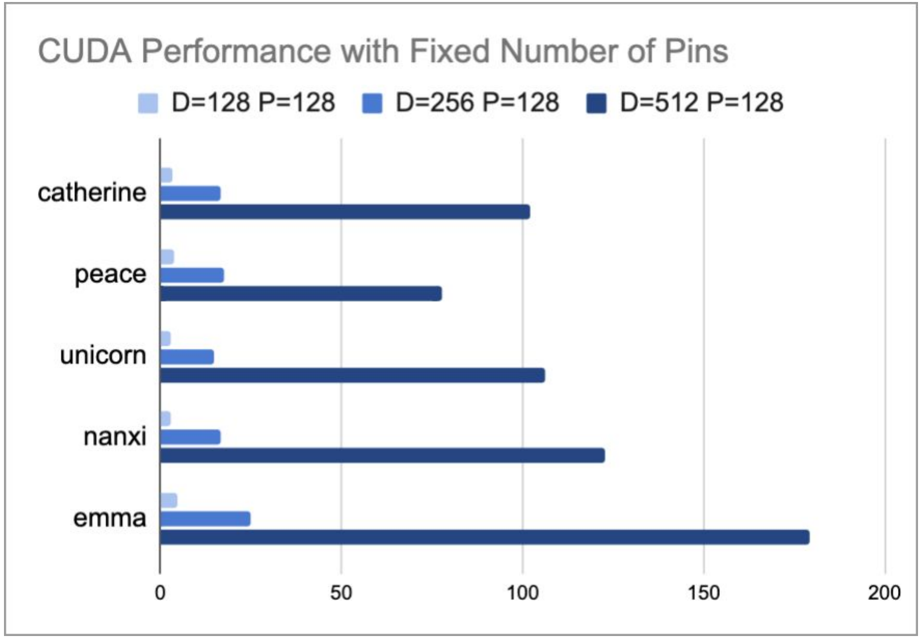
Image Name	catherine	peace	unicorn	nanxi	emma
CUDA time(s)	102.24	77.98	106.26	122.7	179.14
sequential time(s)	17883.5	17290.8	7644.04	19732.6	16735.6
speedup	174.92	221.73	71.94	160.82	93.42

CUDA execution time with one parameter fixed

Image Name	catherine	peace	unicorn	nanxi	emma
D=512 P=64 time(s)	21.46	21.81	21.82	21.93	21.84
D=512 P=128 time(s)	102.24	77.98	106.26	122.7	179.14
D=512 P=256 time(s)	298.08	578.63	308.9	340.03	471.63

Image Name	catherine	peace	unicorn	nanxi	emma
D=128 P=128 time(s)	3.23	4.03	2.78	3.05	4.62
D=256 P=128 time(s)	16.48	17.58	15.05	16.75	24.77
D=512 P=128 time(s)	102.24	77.98	106.26	122.7	179.14





References

[1] Michael Birsak et al. “String art: towards computational fabrication of string images”. In: *Computer Graphics Forum*. Vol. 37. 2. Wiley Online Library. 2018, pp. 263–274.

[2] Exception1984. *Exception1984/StringArt*. URL: https://github.com/Exception1984/StringArt.

[3] Jblezoray. *jblezoray/stringart*. URL: https://github.com/jblezoray/stringart.