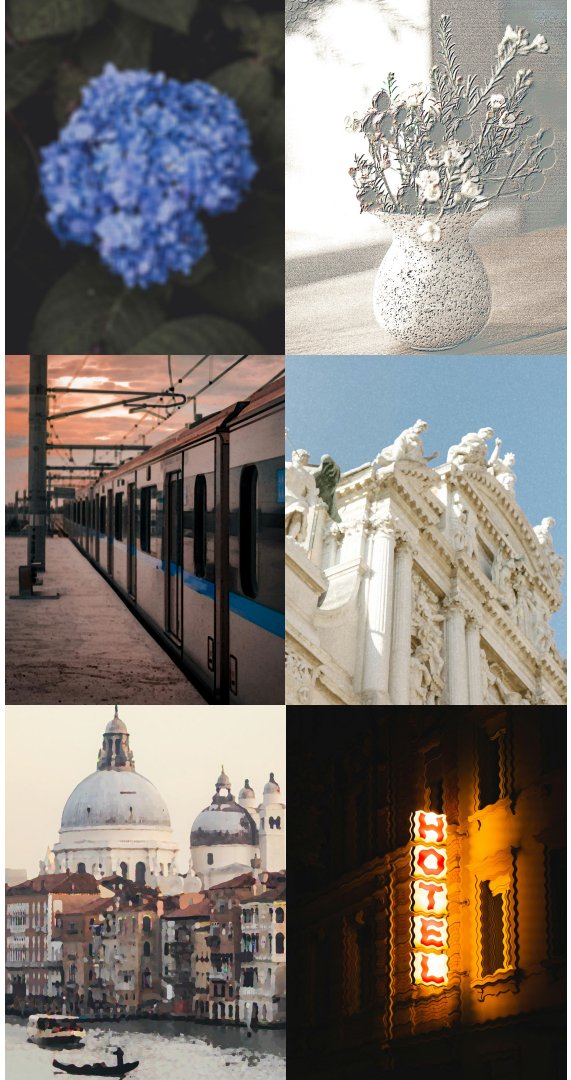


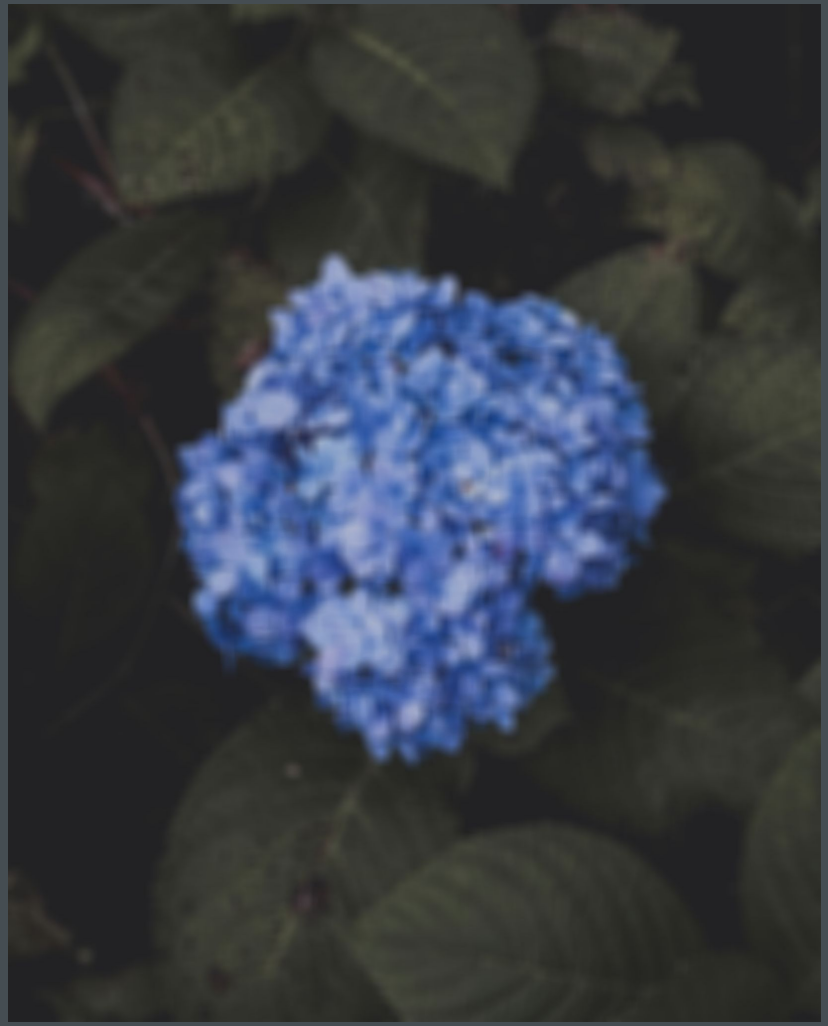
Parallel Programming Final Project

Team 8
112062520 戴維恩
111062698 戴樂為

Table of contents

#1
Image Filters

- Gaussian Blur
- Emboss
- Erosion
- Dilation
- Wave
- Oil Painting



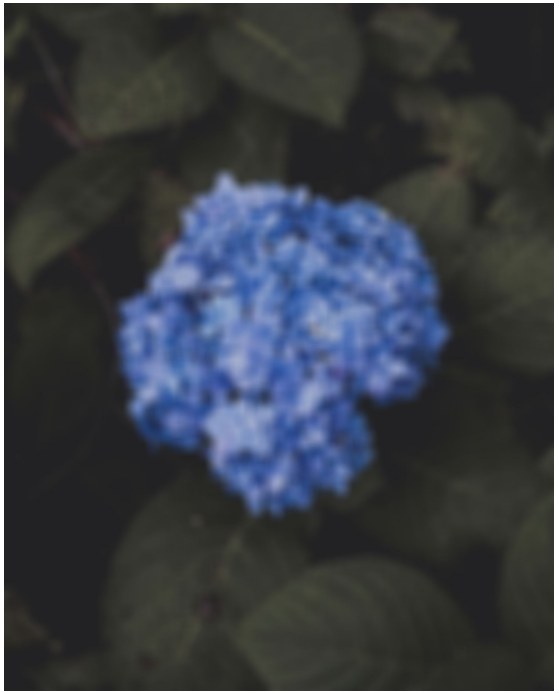
01

Gaussian Blur

Parameters: Radius, Sigma

input

Radius = 10
Sigma = 30



Radius = 30
Sigma = 30

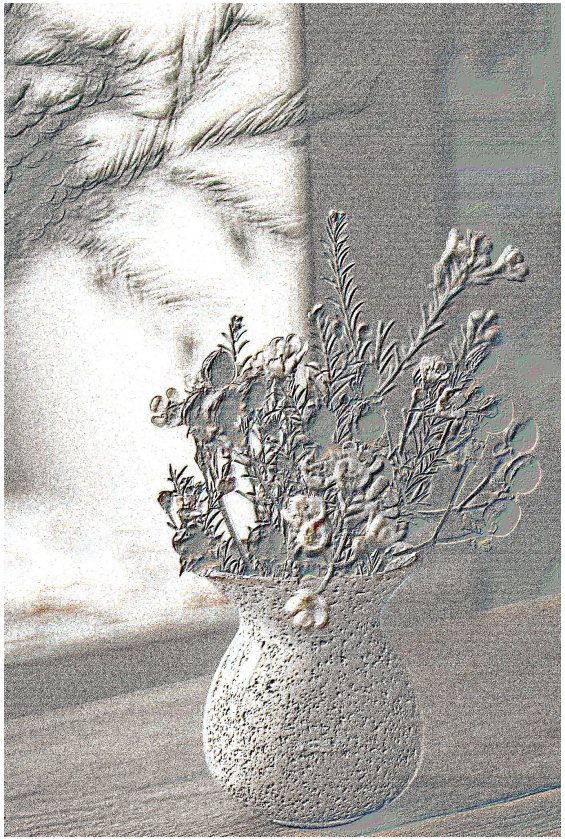
02

Emboss

Parameters: Intensity

input

Intensity = 5



Intensity = 20

03

Erosion

Parameters: Radius

input

Radius = 5

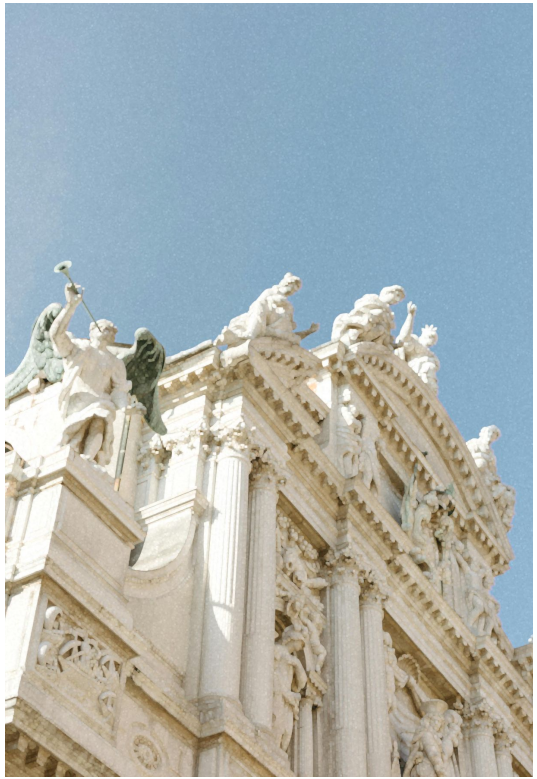
Radius = 10

04

Dilation

Parameters: Radius

Input



Radius = 5

Radius = 10

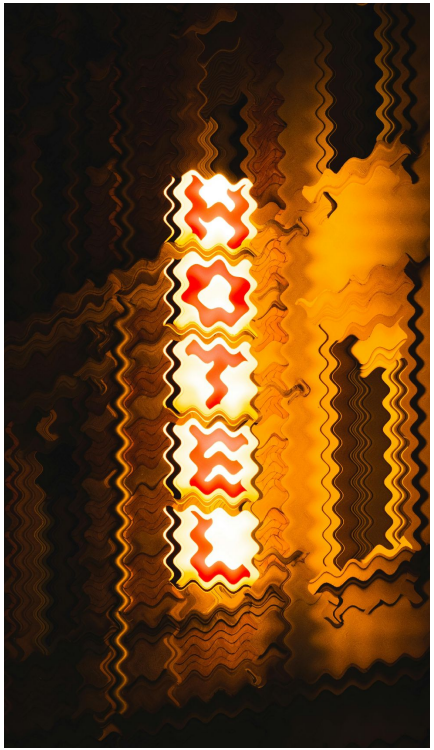
05

Wave

Parameters: Frequency, Amplitude

Input

Frequency = 0.05
Amplitude = 10



Frequency = 0.05
Amplitude = 20

Frequency = 0.1
Amplitude = 20

06

Oil Painting

Parameters: Radius

Input

Radius = 10

Radius = 15

#2
Implementation

- Min/Max Operations
  - Erosion
  - Dilation
- Weighted Combination
  - Gaussian Blur
  - Emboss
- Special Operations
  - Wave (Pixel Displacement)
  - Oil Painting (Intensity-based operation)

Setup

- Calculate Dimension

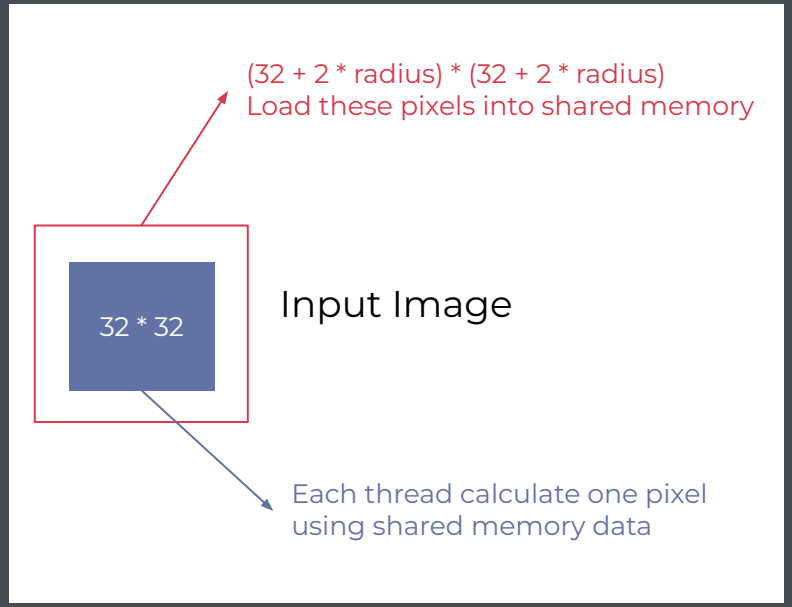
```
// code in kernel
const int tile_w = BLOCK_DIM_X + 2 * radius;
const int tile_h = BLOCK_DIM_Y + 2 * radius;
const int kernelSize = 2 * radius + 1;
```

- Coordinate Clamping

```
// clamp y to 0 ~ height-1
gy = max(0, min(gy, height - 1));
// clamp x to 0 ~ width-1
gx = max(0, min(gx, width - 1));
```

- Kernel Launch

```
// thread block & grid dimension
dim3 block(32, 32);
dim3 grid(
       (width + 32 - 1) / 32,
       (height + 32 - 1) / 32
);
```



Min/Max Operation - Erosion

- CUDA Code

```
unsigned char min_val = 255;
for(int ky = -radius; ky <= radius; ky++) {
    for(int kx = -radius; kx <= radius; kx++) {
        int py = min(max(y + ky, 0), height - 1);
        int px = min(max(x + kx, 0), width - 1);

        unsigned char val = src[(py * width + px) * channels + c];
        min_val = min(val, min_val);
    }
}
dst[idx] = min_val;
```

Min/Max Operation - Dilation

- CUDA Code

```
unsigned char max_val = 0;
for(int ky = -radius; ky <= radius; ky++) {
    for(int kx = -radius; kx <= radius; kx++) {
        int py = min(max(y + ky, 0), height - 1);
        int px = min(max(x + kx, 0), width - 1);

        unsigned char val = src[(py * width + px) * channels + c];
        max_val = max(val, max_val);
    }
}
dst[idx] = max_val;
```

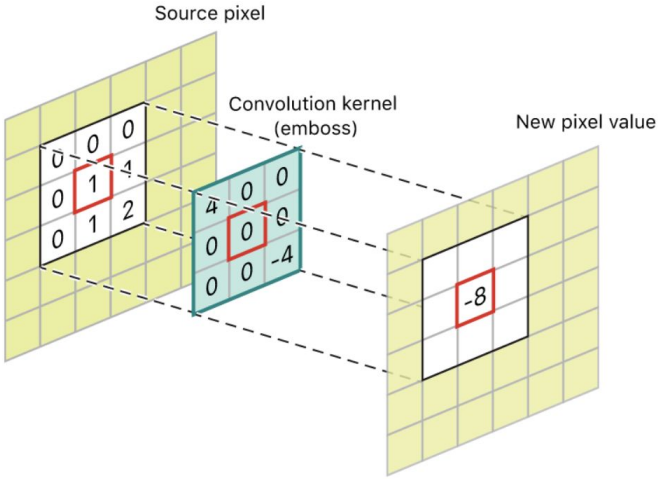
Weighted Combination - Gaussian Blur

- Filter

```
// compute filter on CPU
for(int y = -radius; y <= radius; y++) {
       for(int x = -radius; x <= radius; x++) {
              float value = exp(-(x*x + y*y)/(2*sigma*sigma));
              kernel[(y+radius) * kernelSize + (x+radius)] = value;
              sum += value;
       }
}
```

- CUDA Code

```
for(int ky = -radius; ky <= radius; ky++) {
    for(int kx = -radius; kx <= radius; kx++)
        // ...calculate index
        val += src[src_idx] * kernel[kernel_idx];
    }
}
```



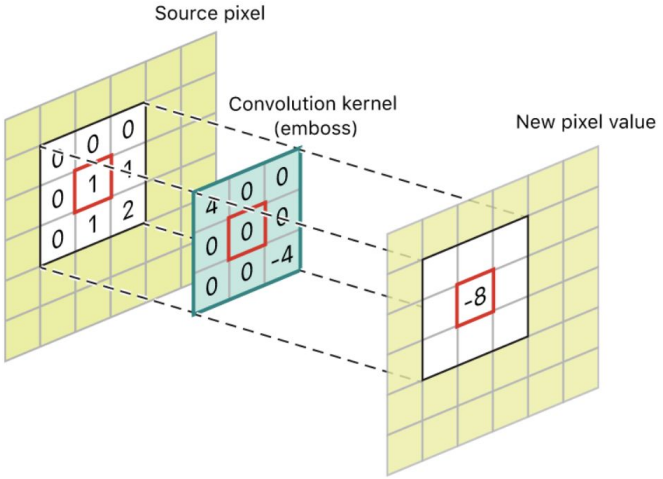
Weighted Combination - Emboss

- Filter

```
// compute filter on CPU
float side = -2.0f * intensity, corner = -1.0f * intensity;
float center = 1.0f, opposite = 2.0f * intensity;
float kernel[3][3] = {
       {side, corner, 0},
       {corner, center, intensity},
       {0, intensity, opposite}
};
```

- CUDA Code

```
for(int ky = -radius; ky <= radius; ky++) {
    for(int kx = -radius; kx <= radius; kx++)
        // ...calculate index
        val += src[src_idx] * kernel[kernel_idx];
    }
}
```



Special Operation - Wave

- CUDA Code

```
// kernel code
int x = blockIdx.x * blockDim.x + threadIdx.x;
int y = blockIdx.y * blockDim.y + threadIdx.y;
int sourceX = x + amplitudeX * sin(y * frequencyY);
int sourceY = y + amplitudeY * sin(x * frequencyX);
Output[y][x] = Input[sourceY][sourceX];
```

Special Operation - Oil Painting

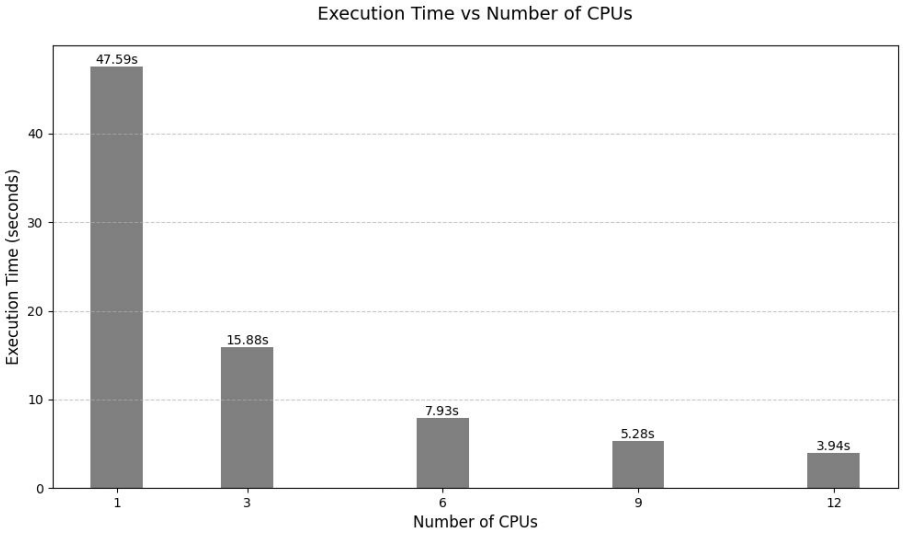
- CUDA Code

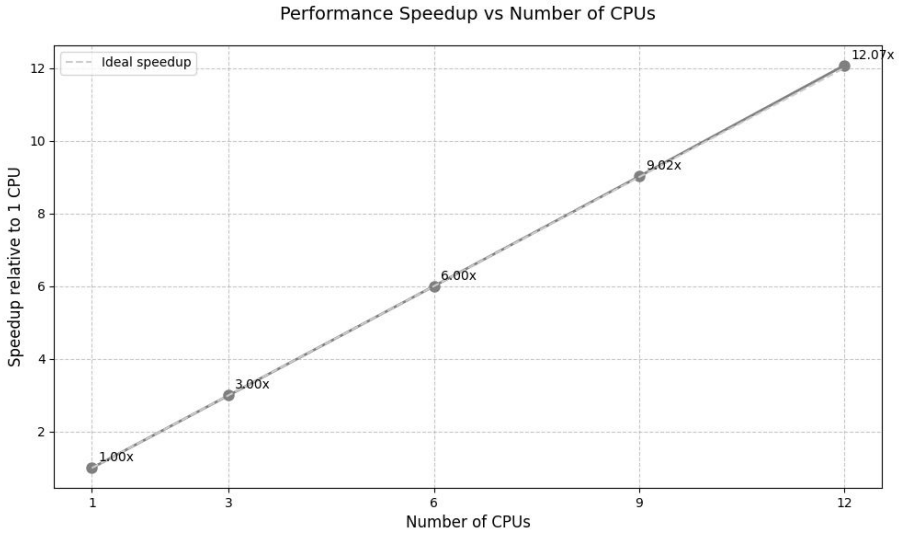
```
for (ky = -radius; ky <= radius; ky++) {
        for (kx = -radius; kx <= radius; kx++) {
                intensity = (img[py + ky][px + kx] * maxLevels) / 256;
                count[intensity]++;
                sum[intensity] += img[py + ky][px + kx];
        }
}
// ... find the intensity level I with maximum count.
Output[py][py] = sum[I] / count[I];
```

#3
Experiments

- OpenMP
  - Time profile & Speedup
- Cuda
  - Baseline vs Optimized
  - Blocking Factor
  - Time Distribution

OpenMP: Time Profile & Speedup

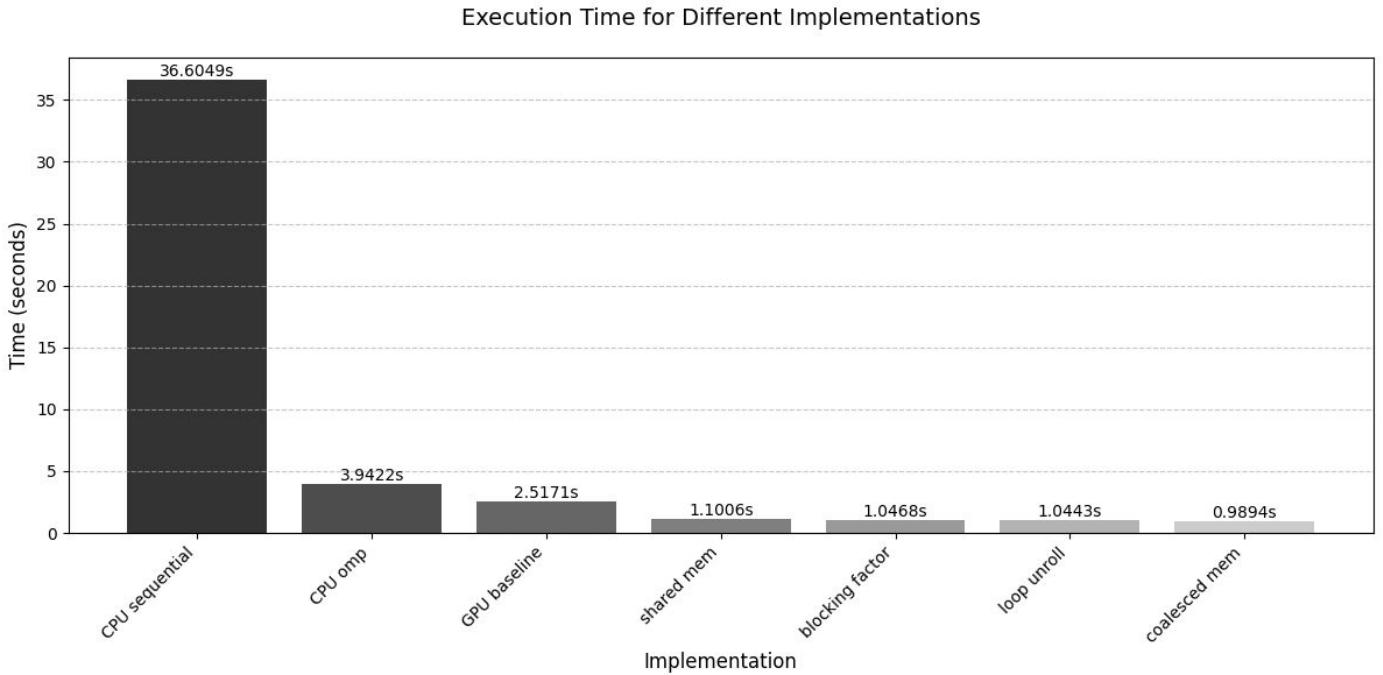




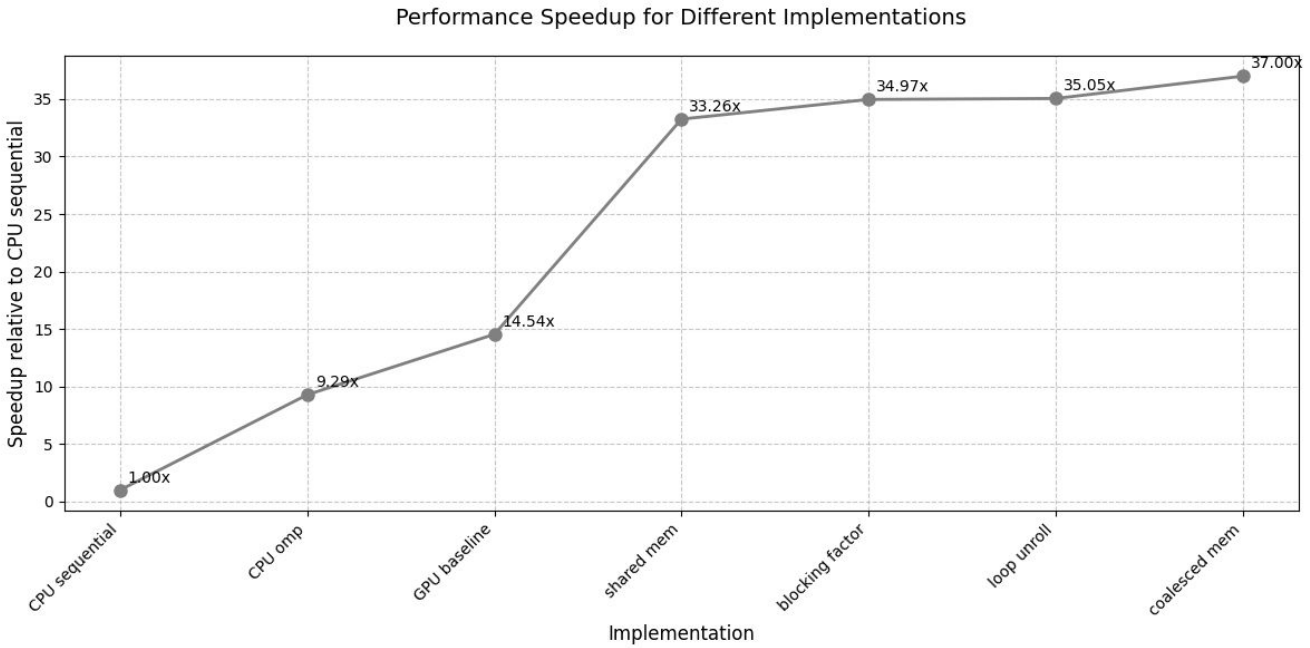
CUDA: nvprof

```
pp24s057@apollo-login:~/filter$ srun -p nvidia -N1 -n1 --gres=gpu:1 nvprof --metrics sm_efficiency,shared_load_throughput,shared_store_th
roughput,gld_throughput,gst_throughput ./filter flower.png out.png oil 5
==1519514== NVPROF is profiling process 1519514, command: ./filter flower.png out.png oil 5
==1519514== Some kernel(s) will be replayed on device 0 in order to collect all events/metrics.
GPU time: 9.5478 seconds
GPU time: 2.9458 seconds
==1519514== Profiling application: ./filter flower.png out.png oil 5
==1519514== Profiling result:
==1519514== Metric result:
Invocations                               Metric Name                        Metric Description         Min         Max         Avg
Device "NVIDIA GeForce GTX 1080 (0)"
    Kernel: oilPaintingKernelOptimized(unsigned char*, unsigned char*, int, int, int, int, int)
          1                            sm_efficiency                   Multiprocessor Activity      99.97%      99.97%      99.97%
          1                     shared_load_throughput             Shared Memory Load Throughput  25.299GB/s  25.299GB/s  25.299GB/s
          1                    shared_store_throughput            Shared Memory Store Throughput  562.00MB/s  562.00MB/s  562.00MB/s
          1                            gld_throughput                    Global Load Throughput  420.20MB/s  420.20MB/s  420.20MB/s
          1                            gst_throughput                   Global Store Throughput  160.57MB/s  160.57MB/s  160.57MB/s
    Kernel: oilPaintingKernel(unsigned char*, unsigned char*, int, int, int, int, int)
          1                     shared_load_throughput             Shared Memory Load Throughput  0.00000B/s  0.00000B/s  0.00000B/s
          1                    shared_store_throughput            Shared Memory Store Throughput  0.00000B/s  0.00000B/s  0.00000B/s
          1                            gld_throughput                    Global Load Throughput  4.7946GB/s  4.7946GB/s  4.7946GB/s
          1                            gst_throughput                   Global Store Throughput  21.285MB/s  21.285MB/s  21.285MB/s
```

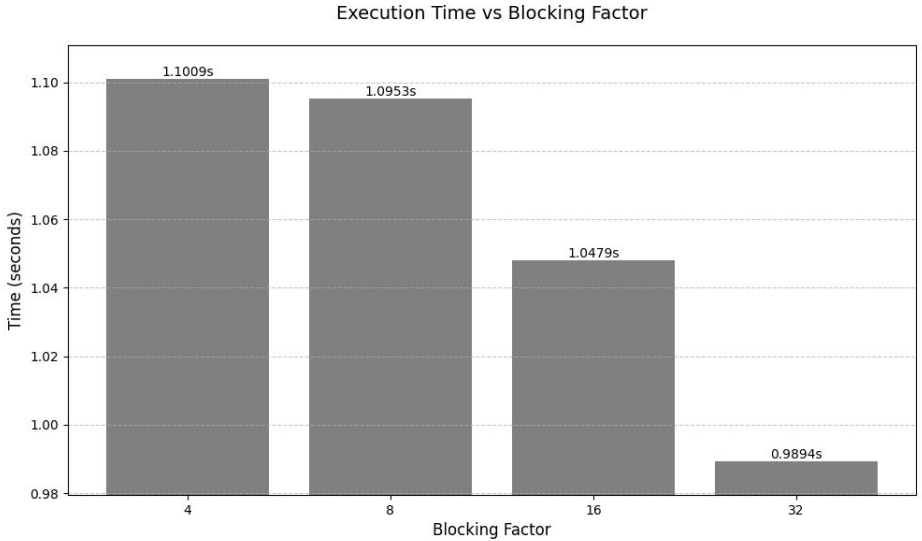
CUDA: Baseline vs Optimized

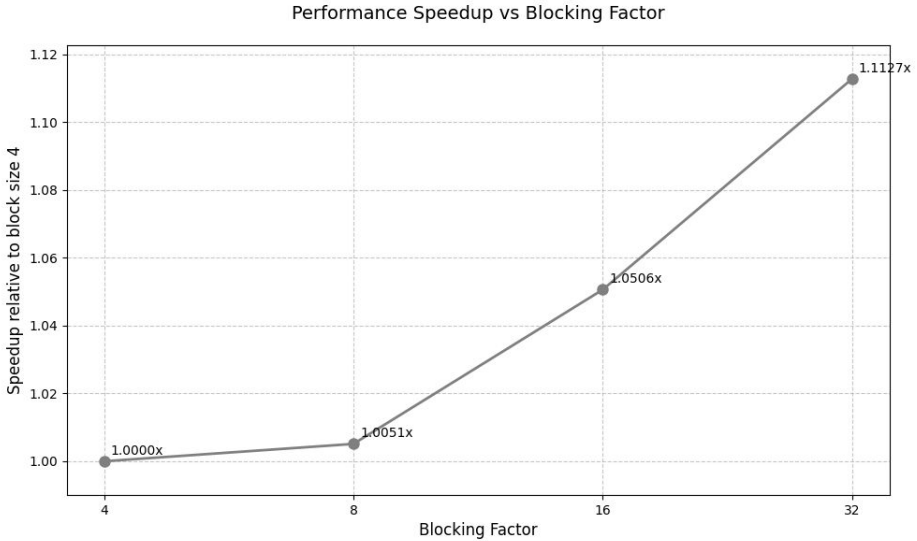


CUDA: Baseline vs Optimized

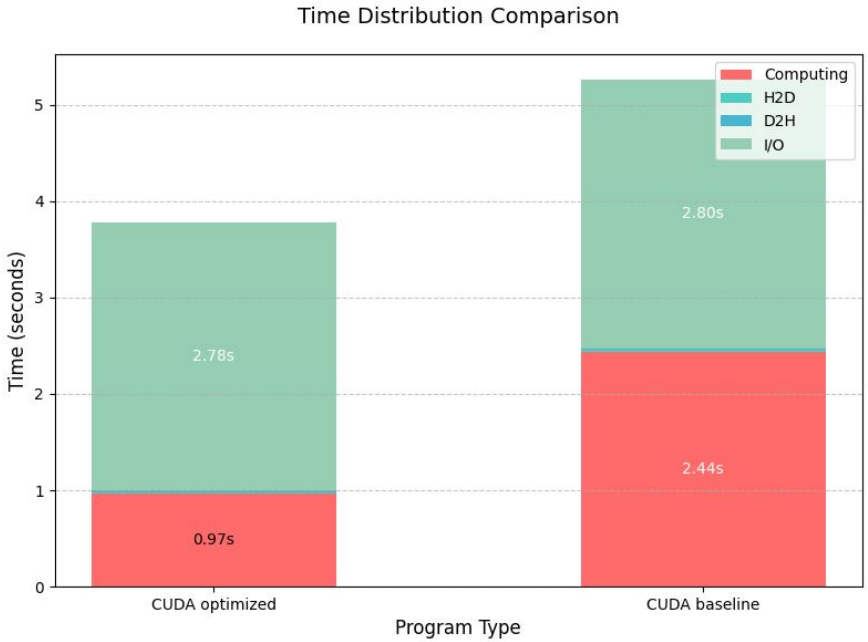


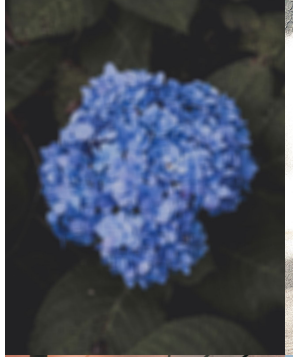
CUDA: Blocking Factor

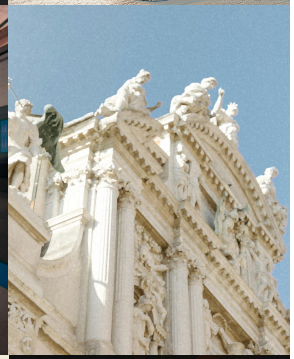
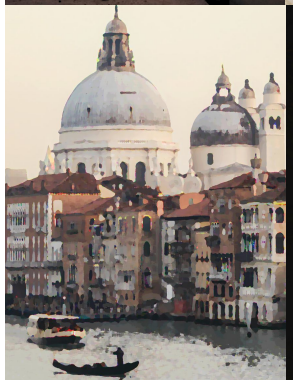
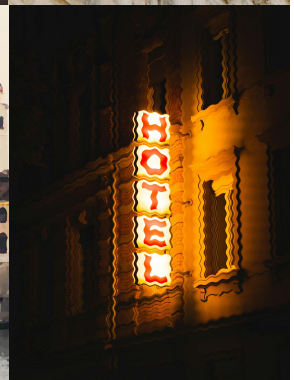




CUDA: Time Distribution





Thank You

Team 8
112062520 戴維恩
111062698 戴樂為