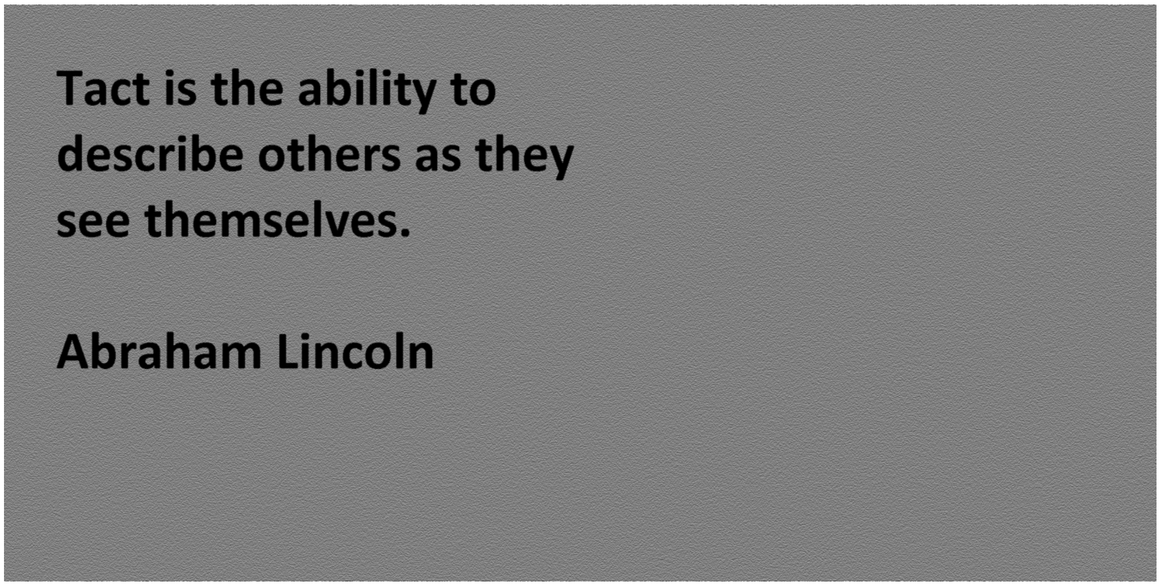
Term Assignment Report

Visual Cryptography using GPU



Tact is the ability to describe others as they see themselves.

Abraham Lincoln

SANDEEP LANKA

EE5496

Electrical and Computer Engineering

INTRODUCTION

Visual Cryptography is a secret sharing scheme where an image can be reconstructed “visually by superimposing two shares over one another. The two shares together should not reveal information about the original image. This is achieved by coding each individual pixel by two or four sub pixels. The width and height of the two shares are therefore double the size of the original image. In the two sub pixel encryption, the width is doubled and thus changes the aspect ratio. In four sub pixel encryption, the aspect ratio remains the same. This scheme can also be implemented with one than one share. That is the original Image can be encoded into *n* different shares and all the shares when superimposed reconstructs the original image. This project only deals with two shares.

The Encoding scheme as shown

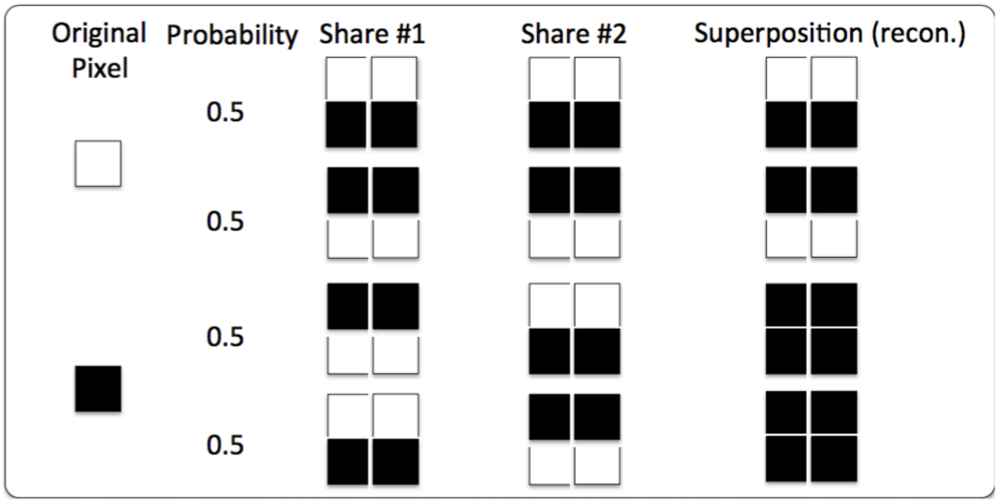


Figure 1: Four Pixel Encryption Patterns

The two Shares that are created with a probability 0.5 when superimposed creates the Original pixel. The Other Parts appear as noise. The encoding should be done in such a way that the individual shares do not reveal any part of the original image. In other words the image should be as random as possible.

ALGORITHM

The Algorithm followed in the project is as follows:

Encoding: 1. Traverse through all the pixels of the Original Image. If the pixel encountered is Black, Randomly pick one Share for that pixel that corresponds to black and encode that pixel with sub pixels corresponding to Black.

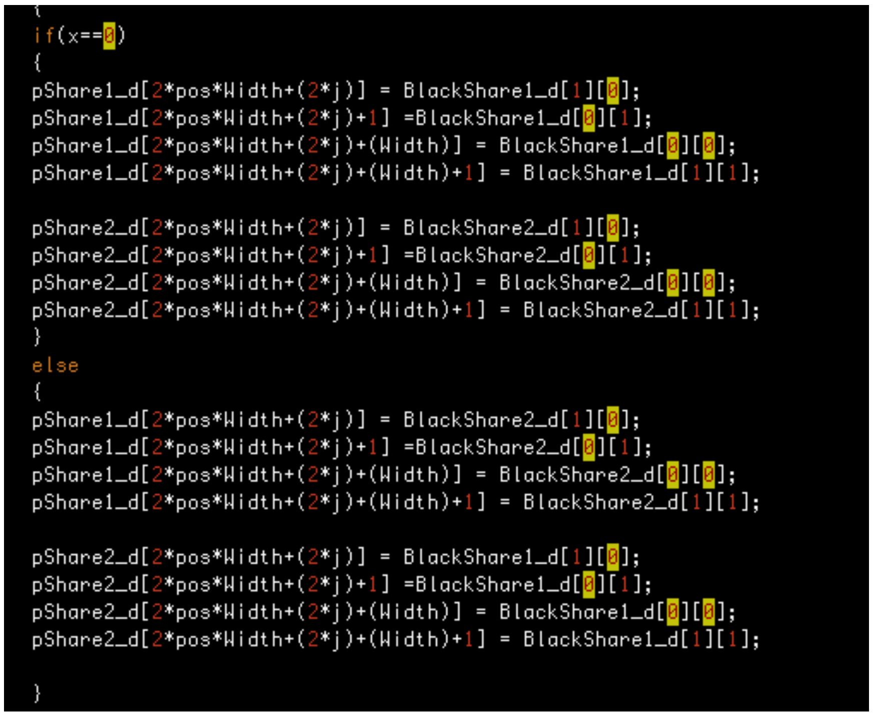
2. If the encountered pixel in the original image is White, encode the pixel with a randomly chosen Share corresponding to the White Pixel.

Decoding: 1. Traverse through every pixel of the first Share and add the pixel on the First Share with the corresponding pixel of the second Share by using bitwise and operator &.

IMPLEMENTATION

Encoding:

The program implemented on the GPU follows the same Algorithm as described above. The encoding is done. The program checks each pixel of the original Image. It encodes the corresponding sub pixels to the original pixel.



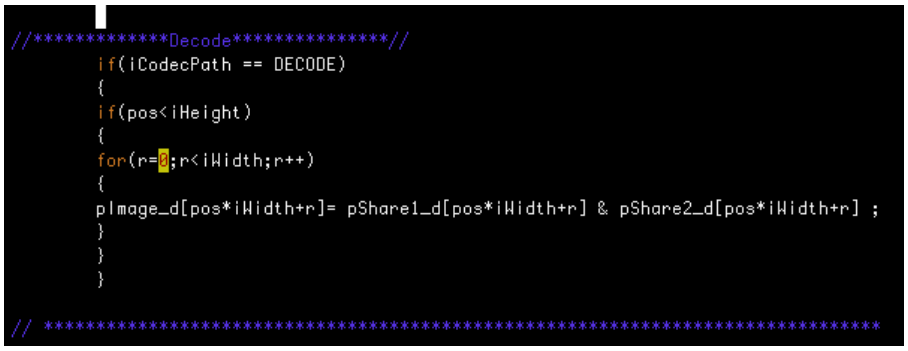
```
{
if(x==0)
{
pShare1_d[2*pos*Width+(2*j)] = BlackShare1_d[1][0];
pShare1_d[2*pos*Width+(2*j)+1] =BlackShare1_d[0][1];
pShare1_d[2*pos*Width+(2*j)+(Width)] = BlackShare1_d[0][0];
pShare1_d[2*pos*Width+(2*j)+(Width)+1] = BlackShare1_d[1][1];

pShare2_d[2*pos*Width+(2*j)] = BlackShare2_d[1][0];
pShare2_d[2*pos*Width+(2*j)+1] =BlackShare2_d[0][1];
pShare2_d[2*pos*Width+(2*j)+(Width)] = BlackShare2_d[0][0];
pShare2_d[2*pos*Width+(2*j)+(Width)+1] = BlackShare2_d[1][1];
}
else
{
pShare1_d[2*pos*Width+(2*j)] = BlackShare2_d[1][0];
pShare1_d[2*pos*Width+(2*j)+1] =BlackShare2_d[0][1];
pShare1_d[2*pos*Width+(2*j)+(Width)] = BlackShare2_d[0][0];
pShare1_d[2*pos*Width+(2*j)+(Width)+1] = BlackShare2_d[1][1];

pShare2_d[2*pos*Width+(2*j)] = BlackShare1_d[1][0];
pShare2_d[2*pos*Width+(2*j)+1] =BlackShare1_d[0][1];
pShare2_d[2*pos*Width+(2*j)+(Width)] = BlackShare1_d[0][0];
pShare2_d[2*pos*Width+(2*j)+(Width)+1] = BlackShare1_d[1][1];

}
```

Figure 2: Encoding Scheme

The program uses the Black and White Shares given in the program template to encode the pixels correspondingly.

Decoding:

Decoding is simply done by using the Bitwise & Operator on Share1 and Share2 for each individual pixels.



```
//*************Decode**************//
        if(iCodecPath == DECODE)
        {
        if(pos<iHeight)
        {
        for(r=0;r<iWidth;r++)
        {
        pImage_d[pos*iWidth+r]= pShare1_d[pos*iWidth+r] & pShare2_d[pos*iWidth+r] ;
        }
        }
        }

// ******************************************************************************
```

Figure 3: Decoding Scheme

If the Image Decodes correct, The Output displays a “*GPU Passed*” statement and if it fails, it displays a “*GPU Failed*” statement.

OUTPUTS

The Outputs Share1G, Share2G and ReconG are verified to check for the Output and the robustness of the scheme.

The Outputs for the image Img128x256.bmp are shown below:

Share1G_Img128x256.bmp

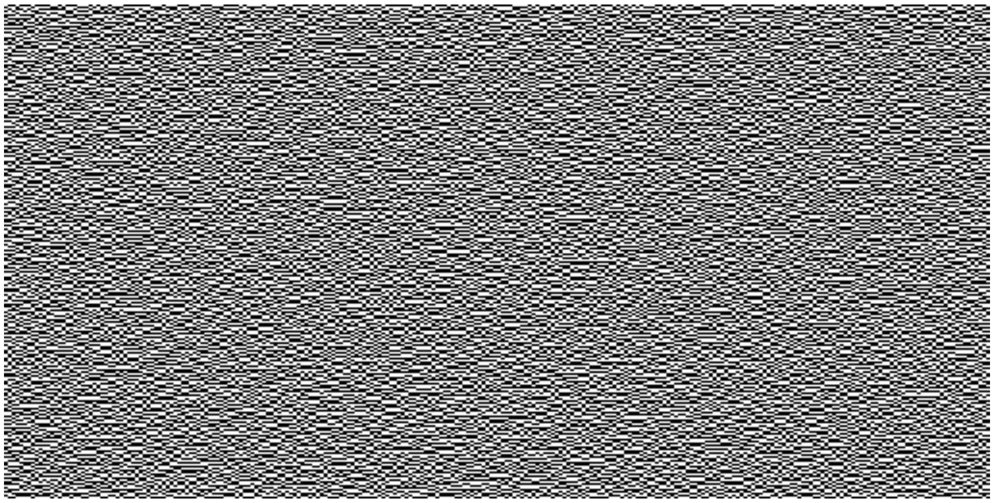


Figure 4: Share1G_Img128x256

Share2G_Img128x256.bmp

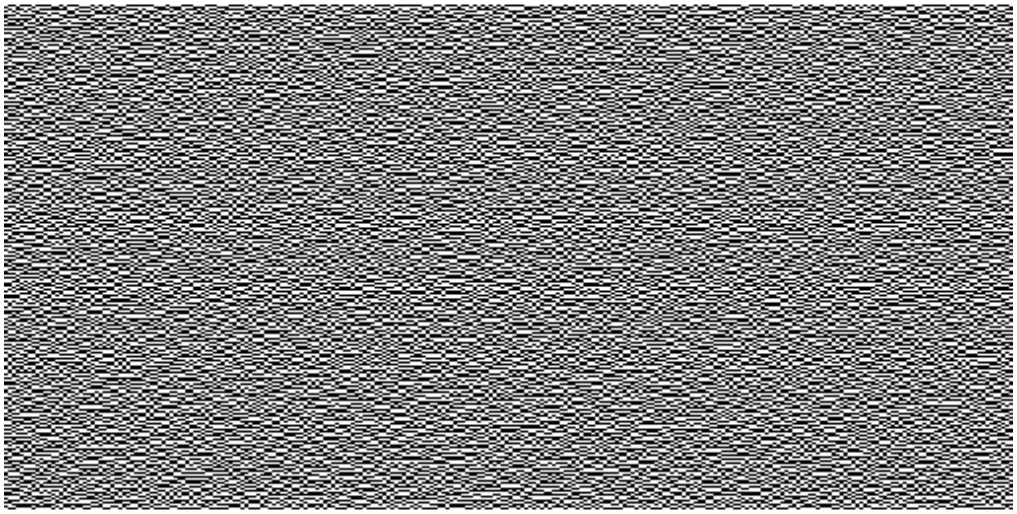


Figure 5: Img128x256.bmp

The Reconstructed Image

ReconG_Img128x256.bmp

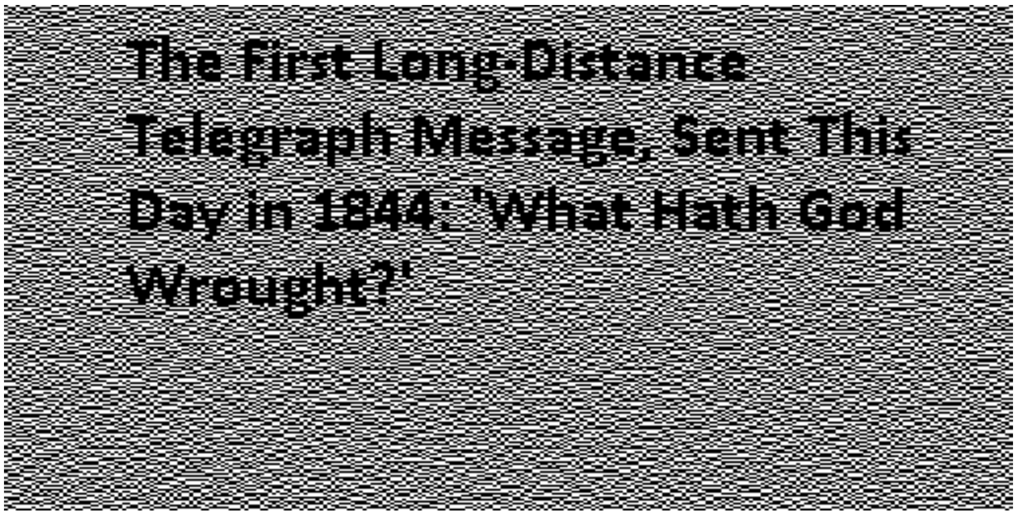


Figure 6: ReconG_:Img128x256.bmp

Similar Images are obtained for all the Images from Img128x256 to Img8192x1638. The random algorithm needs to be implemented in order for the images to be seemingly random. A good Encoding Technique does not reveal any underlying images traces.

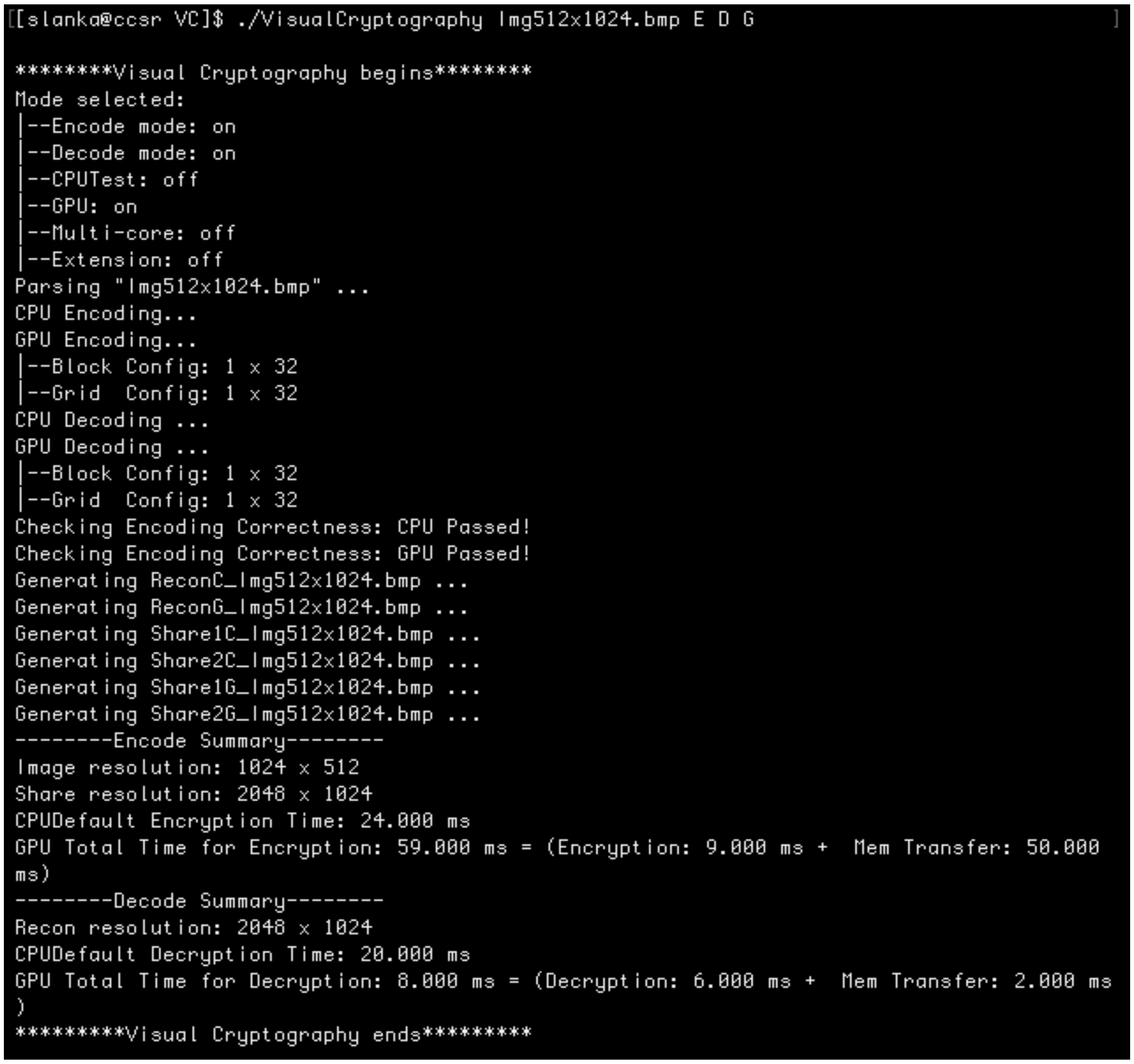
The Block Configuration:

The Block configuration affects the Speed of computation. If all the threads are in the same block they use the same memory and hence it computes faster. Although it may not always be possible to include all threads in a single block. Most Configurations are tested out and the results are formulated.

For The purpose of demonstration,

The Image512x1024 is chosen and the timings are recorded.

For the Configuration of 32x32, The Encryption time is 59ms and the Decryption Time is 8 ms.

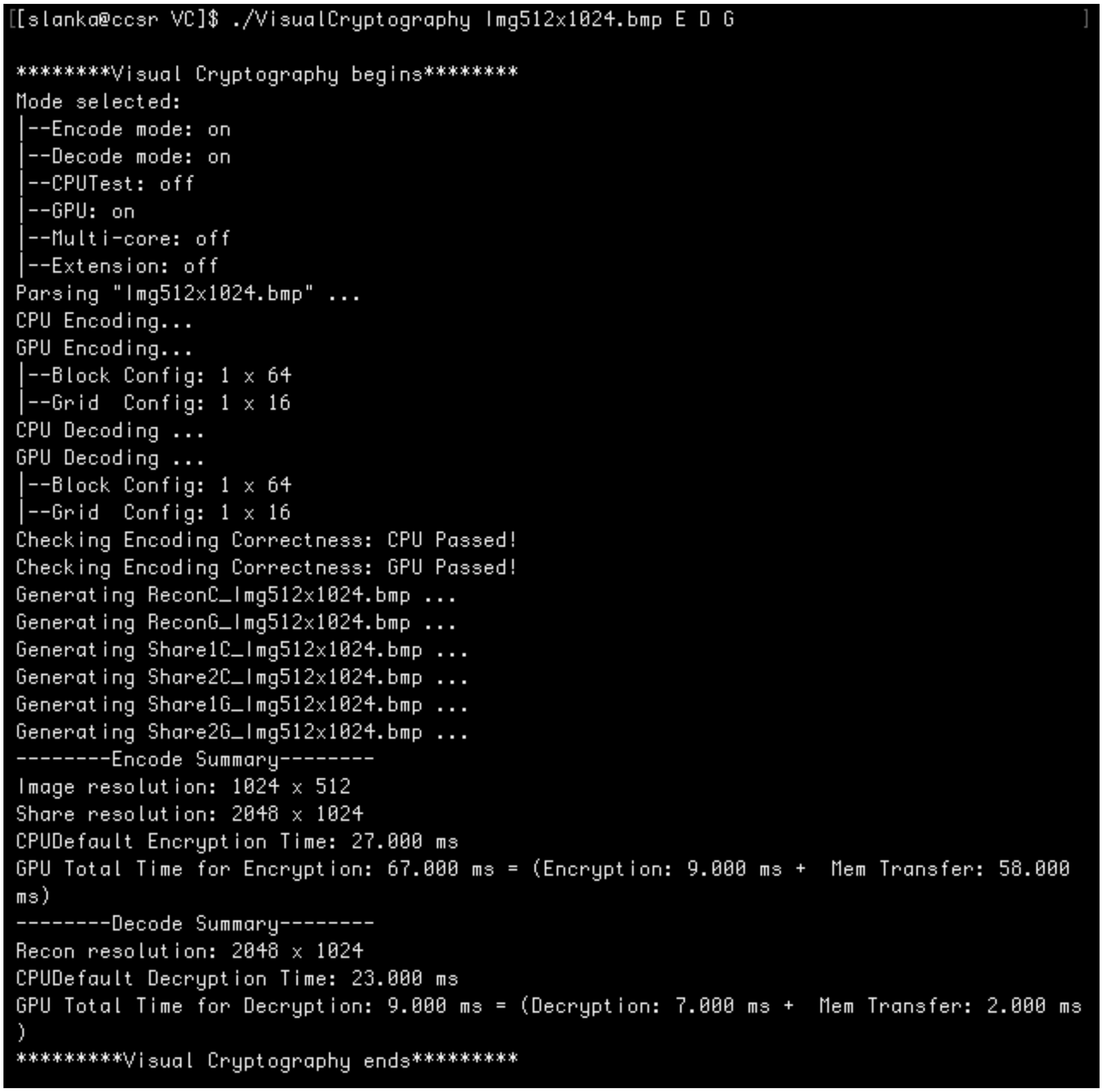


```
[[slanka@ccsr VC]$ ./VisualCryptography Img512x1024.bmp E D G                ]

********Visual Cryptography begins********
Mode selected:
|--Encode mode: on
|--Decode mode: on
|--CPUTest: off
|--GPU: on
|--Multi-core: off
|--Extension: off
Parsing "Img512x1024.bmp" ...
CPU Encoding...
GPU Encoding...
|--Block Config: 1 x 32
|--Grid  Config: 1 x 32
CPU Decoding ...
GPU Decoding ...
|--Block Config: 1 x 32
|--Grid  Config: 1 x 32
Checking Encoding Correctness: CPU Passed!
Checking Encoding Correctness: GPU Passed!
Generating ReconC_Img512x1024.bmp ...
Generating ReconG_Img512x1024.bmp ...
Generating Share1C_Img512x1024.bmp ...
Generating Share2C_Img512x1024.bmp ...
Generating Share1G_Img512x1024.bmp ...
Generating Share2G_Img512x1024.bmp ...
--------Encode Summary--------
Image resolution: 1024 x 512
Share resolution: 2048 x 1024
CPUDefault Encryption Time: 24.000 ms
GPU Total Time for Encryption: 59.000 ms = (Encryption: 9.000 ms +  Mem Transfer: 50.000
ms)
--------Decode Summary--------
Recon resolution: 2048 x 1024
CPUDefault Decryption Time: 20.000 ms
GPU Total Time for Decryption: 8.000 ms = (Decryption: 6.000 ms +  Mem Transfer: 2.000 ms
)
*********Visual Cryptography ends*********
```

The number of threads are kept constant(1024) and different configurations are tested.

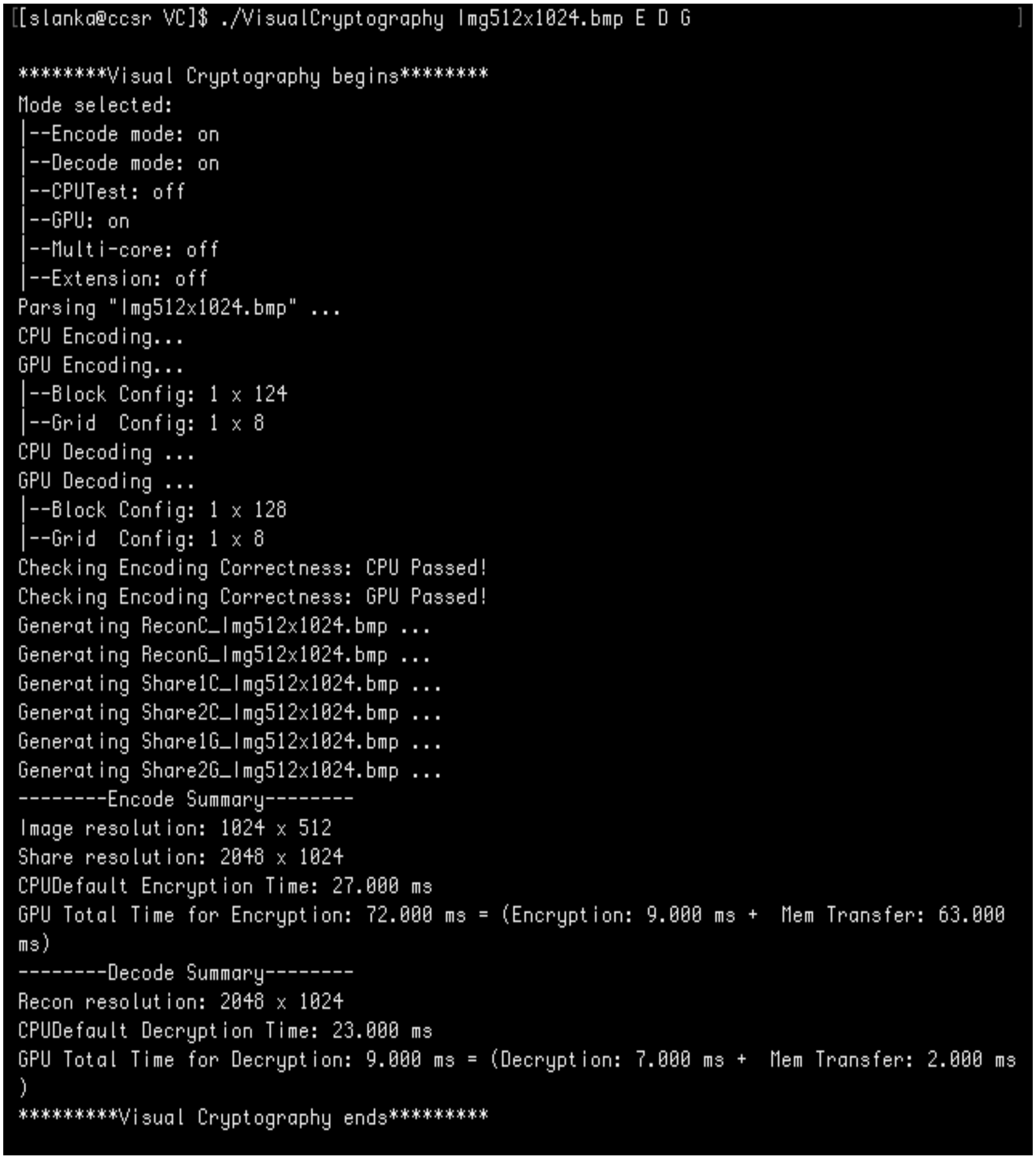
For the Configuration of 64x16, The Encryption time is 67ms. Decryption Time is 9 ms.

The Decryption Time took exactly 1 ms more than the previous configuration.



```
[[slanka@ccsr VC]$ ./VisualCryptography Img512x1024.bmp E D G                                    ]

********Visual Cryptography begins********
Mode selected:
|--Encode mode: on
|--Decode mode: on
|--CPUTest: off
|--GPU: on
|--Multi-core: off
|--Extension: off
Parsing "Img512x1024.bmp" ...
CPU Encoding...
GPU Encoding...
|--Block Config: 1 x 64
|--Grid  Config: 1 x 16
CPU Decoding ...
GPU Decoding ...
|--Block Config: 1 x 64
|--Grid  Config: 1 x 16
Checking Encoding Correctness: CPU Passed!
Checking Encoding Correctness: GPU Passed!
Generating ReconC_Img512x1024.bmp ...
Generating ReconG_Img512x1024.bmp ...
Generating Share1C_Img512x1024.bmp ...
Generating Share2C_Img512x1024.bmp ...
Generating Share1G_Img512x1024.bmp ...
Generating Share2G_Img512x1024.bmp ...
--------Encode Summary--------
Image resolution: 1024 x 512
Share resolution: 2048 x 1024
CPUDefault Encryption Time: 27.000 ms
GPU Total Time for Encryption: 67.000 ms = (Encryption: 9.000 ms +  Mem Transfer: 58.000
ms)
--------Decode Summary--------
Recon resolution: 2048 x 1024
CPUDefault Decryption Time: 23.000 ms
GPU Total Time for Decryption: 9.000 ms = (Decryption: 7.000 ms +  Mem Transfer: 2.000 ms
)
*********Visual Cryptography ends*********
```

For the Configuration of 128x8, The encryption time is 72ms and Decryption Time is 9 ms

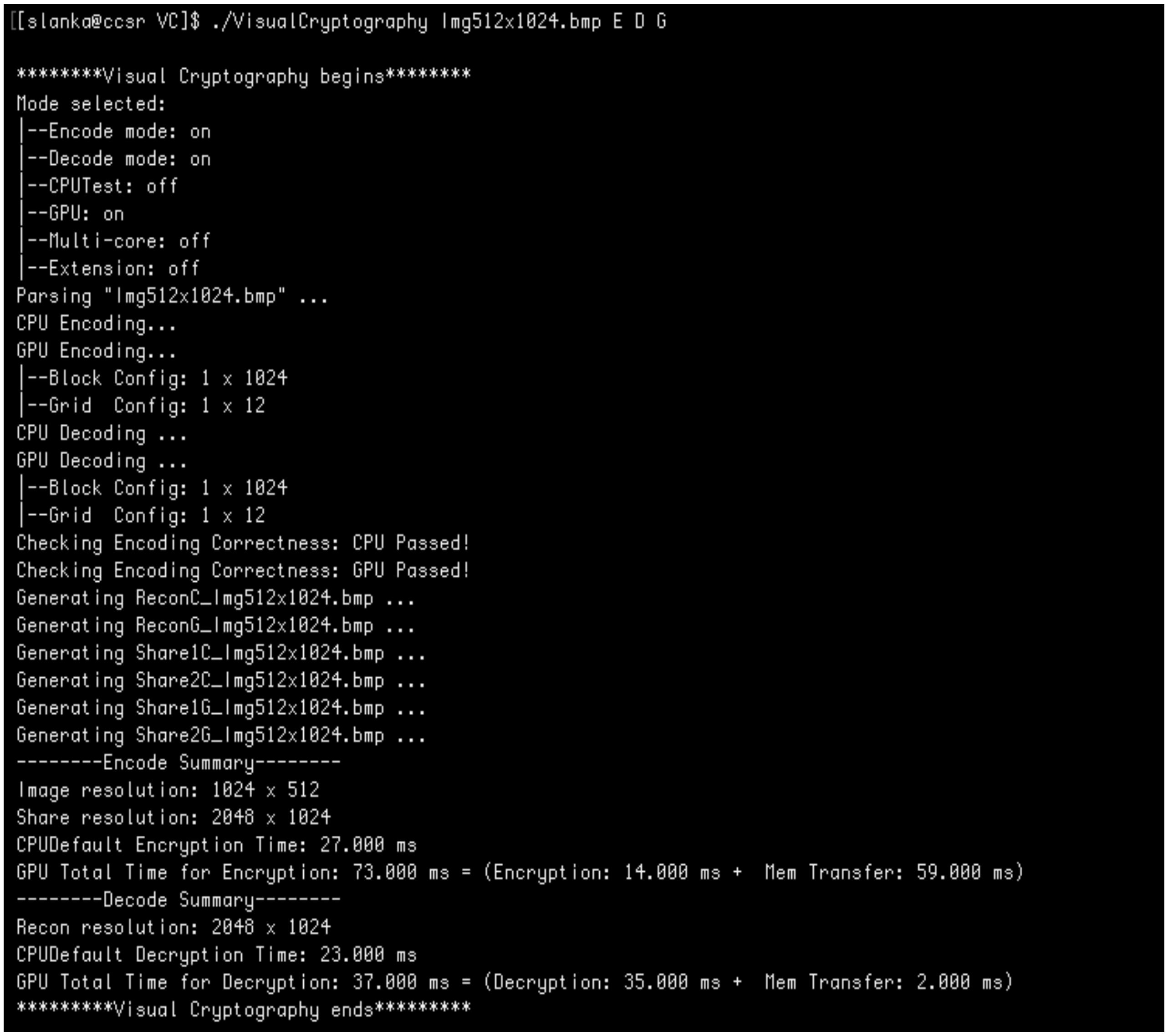


```
[[slanka@ccsr VC]$ ./VisualCryptography Img512x1024.bmp E D G                    ]

 ********Visual Cryptography begins********
 Mode selected:
 |--Encode mode: on
 |--Decode mode: on
 |--CPUTest: off
 |--GPU: on
 |--Multi-core: off
 |--Extension: off
 Parsing "Img512x1024.bmp" ...
 CPU Encoding...
 GPU Encoding...
 |--Block Config: 1 x 124
 |--Grid  Config: 1 x 8
 CPU Decoding ...
 GPU Decoding ...
 |--Block Config: 1 x 128
 |--Grid  Config: 1 x 8
 Checking Encoding Correctness: CPU Passed!
 Checking Encoding Correctness: GPU Passed!
 Generating ReconC_Img512x1024.bmp ...
 Generating ReconG_Img512x1024.bmp ...
 Generating Share1C_Img512x1024.bmp ...
 Generating Share2C_Img512x1024.bmp ...
 Generating Share1G_Img512x1024.bmp ...
 Generating Share2G_Img512x1024.bmp ...
 --------Encode Summary--------
 Image resolution: 1024 x 512
 Share resolution: 2048 x 1024
 CPUDefault Encryption Time: 27.000 ms
 GPU Total Time for Encryption: 72.000 ms = (Encryption: 9.000 ms +  Mem Transfer: 63.000
 ms)
 --------Decode Summary--------
 Recon resolution: 2048 x 1024
 CPUDefault Decryption Time: 23.000 ms
 GPU Total Time for Decryption: 9.000 ms = (Decryption: 7.000 ms +  Mem Transfer: 2.000 ms
 )
 *********Visual Cryptography ends*********
```

Thread Configuration of 1024x12 is considered:

The Encryption Time is: 73ms

The Decryption Time is: 23 ms



```
[[slanka@ccsr VC]$ ./VisualCryptography Img512x1024.bmp E D G

********Visual Cryptography begins********
Mode selected:
|--Encode mode: on
|--Decode mode: on
|--CPUTest: off
|--GPU: on
|--Multi-core: off
|--Extension: off
Parsing "Img512x1024.bmp" ...
CPU Encoding...
GPU Encoding...
|--Block Config: 1 x 1024
|--Grid  Config: 1 x 12
CPU Decoding ...
GPU Decoding ...
|--Block Config: 1 x 1024
|--Grid  Config: 1 x 12
Checking Encoding Correctness: CPU Passed!
Checking Encoding Correctness: GPU Passed!
Generating ReconC_Img512x1024.bmp ...
Generating ReconG_Img512x1024.bmp ...
Generating Share1C_Img512x1024.bmp ...
Generating Share2C_Img512x1024.bmp ...
Generating Share1G_Img512x1024.bmp ...
Generating Share2G_Img512x1024.bmp ...
--------Encode Summary--------
Image resolution: 1024 x 512
Share resolution: 2048 x 1024
CPUDefault Encryption Time: 27.000 ms
GPU Total Time for Encryption: 73.000 ms = (Encryption: 14.000 ms +  Mem Transfer: 59.000 ms)
--------Decode Summary--------
Recon resolution: 2048 x 1024
CPUDefault Decryption Time: 23.000 ms
GPU Total Time for Decryption: 37.000 ms = (Decryption: 35.000 ms +  Mem Transfer: 2.000 ms)
*********Visual Cryptography ends*********
```

The Timings are summarized as follows in the table:

Img128x256.bmp

Configuration	Encryption Time	Decryption Time
512x1	46	2
512x64	55	2
1024x4	42	1
1024x12	56	1

Img128x256.bmp

Configuration	Encryption Time	Decryption Time
256x2	48	5
256x512	56	8
512x1	57	8
1024x4	45	8
1024x12	54	8

The Largest Image Img8192x16348.bmp shows the following:

Configuration	Encryption Time	Decryption Time
512x64	2386	3425
1024x32	2448	3578
Configuration	Encryption Time	Decryption Time
512x1	139	206
1024x4	182	224
1024x12	191	228

The above table is for Img2048x4096.

Thus, We see from the tables that the Performance is good when the configuration has correct number of threads for the Image. If there are less threads, the Decryption does not pass.

For the Right number of threads, The Encryption and Decryption Times for all the images are tabulated in below:

Image.bmp	CPU Encryption Time(ms)	GPU Encryption Time(ms)	CPU Decryption Time(ms)	GPU Decryption Time(ms)
Img128x256	1	46	1	2
GPU	7	46	6	8
Img256x512	6	48	5	5
Img512x1024	24	59	20	8
Img1204x2048	99	112	96	86
Img2048x4096	428	182	365	224
Img4096x8192	1582	510	1462	878

Img4096x8192-A	1566	510	1283	891
Img4096x8192-C	1589	510	1451	880
Img8192x16348	6171	2386	5117	3425

We see that the as the Image size Increases the GPU performs better than the CPU.

Thus, the goal of the GPU that is to acquire throughput is satisfied. This means that the when there a lot of workload that is available GPU’s are a better option.

Multi Core Programming

pThreads are used to parallelize code for multi Core processors. The objective is to achieve thread level parallelism on a multicore processor. Thread Level parallelism is where threads distribute workload, join together at a future meeting point. This enables speed up and efficient use of all the cores in the CPU.

The Workloads are compared between all the threads for a serial code with certain parallelism.

The Workload distribution and Timings are compared for the Image 4096x8192-A

Number of Threads	CPU Encryption Time(ms)	MC Encryption Time(ms)	CPU Decryption Time(ms)	MC Decryption Time(ms)
1	1714	1355	1461	49455
2	1562	1184	1806	49289
4	1564	1196	1277	49485
16	1564	17891	1332	1925
32	1564	16452	1275	3845
64	1575	14237	1293	6246

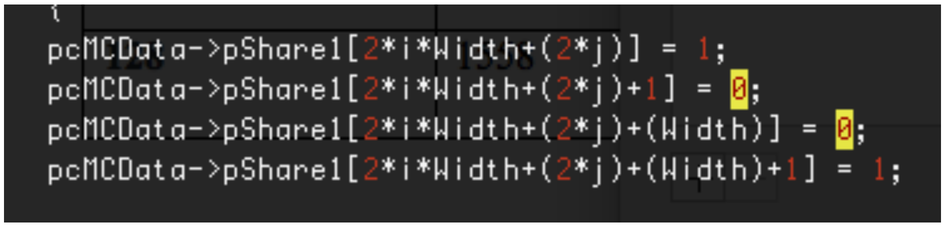
128	1558	17004	1373	12524

We Clearly see that when the threads are around 4 – 16 The encryption time is better. The Decryption Time is never better than the CPU for two reasons:

1. There is almost no available parallelism in the Decryption Code.
2. The more number of threads, the more computation involved and synchronization for join and fork.

There is thread unrolling involved in the Encryption Process. Each thread can handle the tasks separately parallel to each other.

For the Encryption:



```
{
pcMCData->pShare1[2*i*Width+(2*j)] = 1;
pcMCData->pShare1[2*i*Width+(2*j)+1] = 0;
pcMCData->pShare1[2*i*Width+(2*j)+(Width)] = 0;
pcMCData->pShare1[2*i*Width+(2*j)+(Width)+1] = 1;
```

Each thread can work on individual elements, There is no dependencies and the only dependency is the Jump from the loop variables.

For Decryption,

The statements are dependent on each other. The statements thus have to wait to execute till the statement before has finished executing. This causes severe lag and hence, we see that in order for multicore architectures to enhance performance, Available Thread-Level Parallelism should be reasonably good. Or It does not perform well. The goal is to write efficient algorithms that can reorder the existing serial Algorithm into efficient parallel Algorithm that has good Available Thread Level Parallelism.

Conclusion:

From the Data seen, GPU’s are more efficient when there is a lots of workload since GPU’s goal is to achieve throughput. We see that as the image sizes increase, GPU outruns CPU in terms of speed. Multi Core Processors on the other hand can implement more number of parallel threads if there is sufficient available parallelism in the program. If there is no parallelism in the program, Multi Cores perform equal to or sometimes worse than the CPU because, there is time and energy consumption for thread migration and thread switching and joining etc.

Acknowledgements

I thank Professor Saeid Nooshabadi for the tutorials and lectures and all the resources that are provided online and the guidance and support throughout this course.