SCITECH
RESEARCH ORGANISATION|

Volume 6, Issue 2
November 21, 2017

Journal of Information Sciences and Computing Technologies
www.scitecresearch.com/journals

Reconstruction Improvements on Compressive Sensing

Yan Zhang[1], Suxia Cui[2], Yonghui Wang[3]
[1]Prairie View A&M University, Prairie View, TX, 77446, USA, powerzhangyan@gmail.com,
[2] Prairie View A&M University, Prairie View, TX, 77446, USA, sucui@pvamu.edu,
[3] Prairie View A&M University, Prairie View, TX, 77446, USA, yowang@pvamu.edu

Abstract

This paper presents the design of a system, which can improve the reconstruction of Compressive Sensed images. The proposed techniques can reduce the reconstruction time for a compressive sensing based image. Those improvements utilize matrix simplification, multi-thread and GPU computations. Implementing those techniques achieve gains on time consumption, compared to the baseline. This paper also presents a novel scheme of buffering streamed image (video) to achieve optimum performance.

Keywords: Compressive Sensing; GPU; multi-thread.

1. Introduction

Wireless spectrum is becoming increasingly scarce as more and more mobile devices are being used with new innovations to support multimedia applications. The evolution of new applications is further eroding the ability to make spectrum available to ever increasing mobile users [1]. The design of mobile devices for inclusion of new and evolving multimedia applications faces big challenges due to: a) limited power supply from battery; b) wireless transmission impairments for sustained image transmissions; and c) limited CPU capability in each device. Therefore, data compression is becoming more and more important. This research takes image as a sample multimedia data type to explore some improvement.

Compressive Sensing (CS) technique was initially proposed to enable signal sampled at sub-Nyquist rate to be perfectly recovered. For the past decade, CS has been widely adopted in various signal-processing areas[2-3].Compressive sensing of images allows sampling to occur at a sub-Nyquist rate onto a random basis and it could be reconstructed to the original image with the condition of sparsity [2-3].The N-dimensional signal x is assumed to be K-sparse with respect to some orthogonal matrix V. The "sampling" of x is a linear transformation by using a matrix ϕ to produce a vector $y = \phi x$. Let ϕ be an M-by-N matrix where $M<<N$, so y has M elements; we call each element of y as a measurement of x. The decoder recovers the signal x from y with known V and ϕ [4-5].

For the mobile application of CS scenario, there are two challenges: 1) the wireless transmission of compressed signals in noisy channels and 2) the reconstruction of the original content at the receiver side. The first challenge had been discussed in the works 2010 through 2014 [6-8]; the second challenge is critical in some scenarios, e.g, when the receiver side is mobile device with limited CPU capability.

Most of CS based reconstruction depends on iterations [9], which are very time consuming and inefficient. On average, processing 100 times more than encoding time, from previous implementation experiences. Thus, improvements of reconstruction of CS are very important in real world implementations of CS. This paper presents several approaches to speed up the reconstruction process of Compressive Sensing.

2. System and Methods

The reconstruction of Compressive Sensed data is very time consuming and inefficient. This paper presents several approaches to speed up the reconstruction process of Compressed Sensing.

2.1 Baseline Performance Time and Reconstruction Algorithm

"BCS-SPL—Block Compressed Sensing with Smooth Projected Landweber Reconstruction", which is developed by Sungkwang Mun and James E. Fowler following the method describe in [5], was chosen as the starting point for the CS reconstruction. It marks the baseline of the reconstruction time using single core microprocessor.
The reconstruction is recursive executed using following methods described in [5]. It is an iterative process.

$$\hat{x}^i = Wiener(x^i) \quad (1)$$

$$\hat{x}^i = \hat{x}^i + \Phi_B^T(y - \Phi_B\hat{x}^i) \quad (2)$$

$$\hat{x}^i = \psi\hat{x}^i \quad (3)$$

$$\hat{x}^i = Threshold(\hat{x}^i) \quad (4)$$

$$\bar{x}^i = \Psi^T\hat{x}^i \quad (5)$$

$$x^{i+1} = \bar{x}^i + \Phi_B^T(y - \Phi_B\bar{x}^i) \quad (6)$$

2.2 Hardware Upgrade

In order to speed up the processing time, multi-core microprocessor was adopted. The computer used for the reconstruction was a Lenovo IdeaPad 700 80RU00CYUSlaptop computer system. The system uses an Intel Core™ i7-6700HQ Processor. The base frequency is 2.6G Hz, and it has 4 cores support up to 8 threads.

The GPU of the system is a NVIDIA GeForce GTX-950M video card. It has 640 CUDA cores with a base clock of 914 MHz. Inside the GPU, there are 2 GB DDR5 memory.

The Random Access Memory (RAM) of the system contains 12 GB of DDR4 RAM, with frequency up to 2133 MHz.The hard drive of the system is a Samsung MZLVL256 SSD NVME PCI Solid State Drive. The read speed is up to 1258 MB/s.

3. Acceleration Approaching's and Experiment Results

3.1 Simplified Matrix Operation

3.1.1 Overview

From equations (2) and (6) in section 2.1, we could simplify the computation to

$$x^{i+1} = (I - \Phi_B^T\Phi_B)\bar{x}^i + \Phi_B^T y \quad (7)$$

By using equation (7), it may accelerate the reconstruction processing time.

3.1.2 Processing Complexity Analysis

Processing complexity analysis:

a) The CS block size is 32 by default, because $32 \times 32 = 1024$, therefore Φ_B is a $1024 \times (1024 \times subrate)$ matrix

b) Then $(I - \Phi_B^T\Phi_B)$ will be a 1024×1024 matrix.

c) In practice, x will be reshaped to a $1024 \times [\frac{m \times n}{1024}]$ matrix

d) Since $(I - \Phi_B^T\Phi_B)$ and $\Phi_B^T y$ are like "constant" in recursion and could be pre-calculated, thus the calculation cost and processing time can be saved by using equation (7).

Then for (7), the computational complexity will be 1024×1024 matrix multiply $1024 \times [\frac{m \times n}{1024}]$ matrix, the computational complexity is

$$1024 \times \left(m \times \frac{n}{1024}\right) \times 1024 = 1024 \times m \times n \quad (8)$$

In equation (2) or (6), the computational complexity will be

$$1024 \times 1024 \times subrate \times m \times n + 1024 \times 1024 \times subrate \times m \times \frac{n}{1024}$$

$$= 2 \times subrate \times 1024 \times m \times n \quad (9)$$

Comparing equations (8) and (9), it is concluded that if subrate is less than 0.5, using (2) and (6) is more efficient. When subrate is greater than 0.5, using (7) is more effective. This conclusion will be proved by experimental results in tables 1-2 and figures 1-2.

Table 1. Simplified Matrix Operation: Lenna as example for reconstruction time (in Second)

subrate	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
Baseline	9.9753	9.6059	8.1422	6.4530	5.9219	4.5965	4.2468	4.2238	3.3244	0.2869
Simplified Matrix operation	12.6014	11.2701	10.2928	6.9856	5.8617	4.9847	4.1751	3.5130	2.5165	0.2007

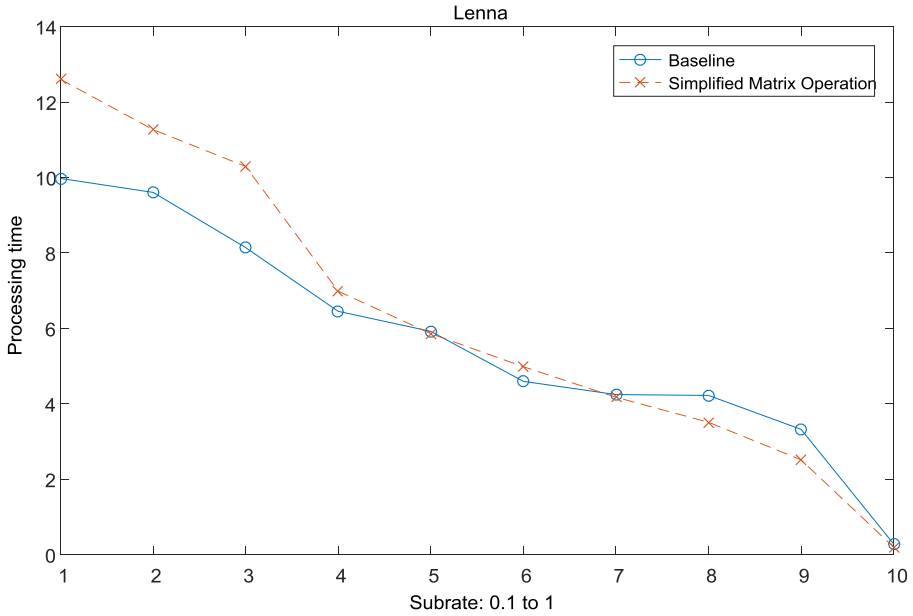

Fig 1: Simplified Matrix Operation: Lenna as example for reconstruction time

(Note: for all figures in this paper, the x-axis represents the percentage of measurements used to reconstruct the image, from 0.1 to 1, with 0.1 increments.)

Table 2. Simplified Matrix Operation:Barbara as example for reconstruction time (in Second)

subrate	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
Baseline	12.3059	9.4749	7.0272	5.6647	5.2568	4.9652	4.3774	3.9228	3.4502	0.2708
Simplified Matrix operation	18.1466	10.6697	8.1512	6.8408	5.6378	4.7552	4.0611	3.3409	2.7155	0.2055

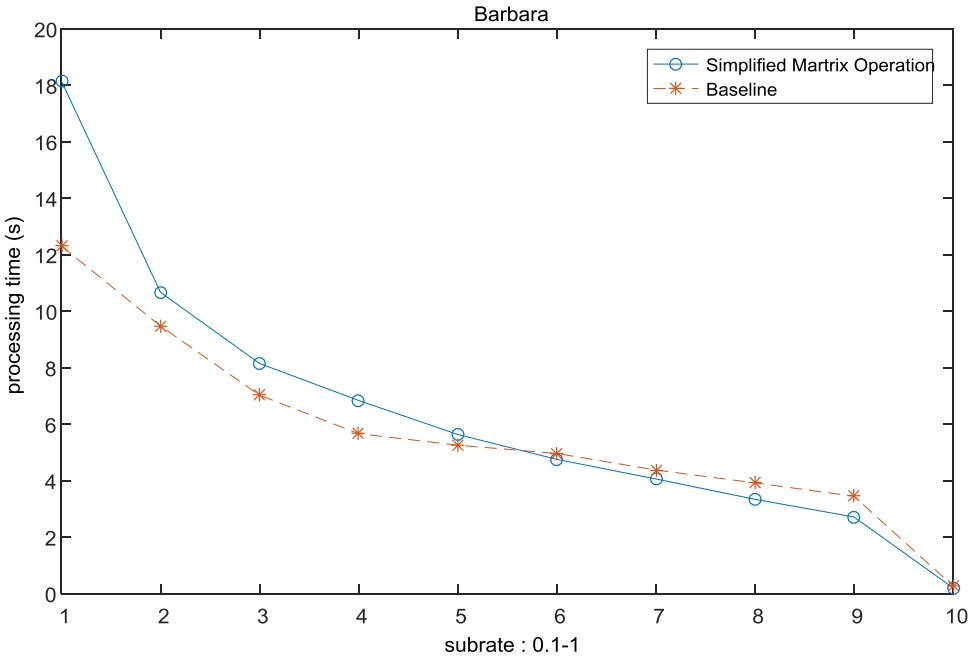

Fig 2: Simplified Matrix Operation: Barbara as example for reconstruction time

3.2 Multi Threads Computing

The system has 4 cores to support up to 8 threads of computing. By using multiple threads will speed up the processing. Tables 3-4 and figures 3-4 shows the experimental results using multi threads computing.

Table 3. Multi threads computing: Lenna as example for reconstruction time (in Second)

subrate	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
1 Thread (Baseline)	9.9753	9.6059	8.1422	6.4530	5.9219	4.5965	4.2468	4.2238	3.3244	0.28691
2 Threads	7.2586	6.8289	5.2330	4.4108	4.1842	3.5010	2.8858	2.5954	1.9682	0.17754
4 Threads	6.5446	6.0403	5.0301	4.1118	3.6344	3.1341	2.6066	2.4316	1.9384	0.15981
8 Threads	6.6134	6.1757	5.1376	4.1926	3.7457	3.2085	2.6468	2.4891	1.9439	0.16144

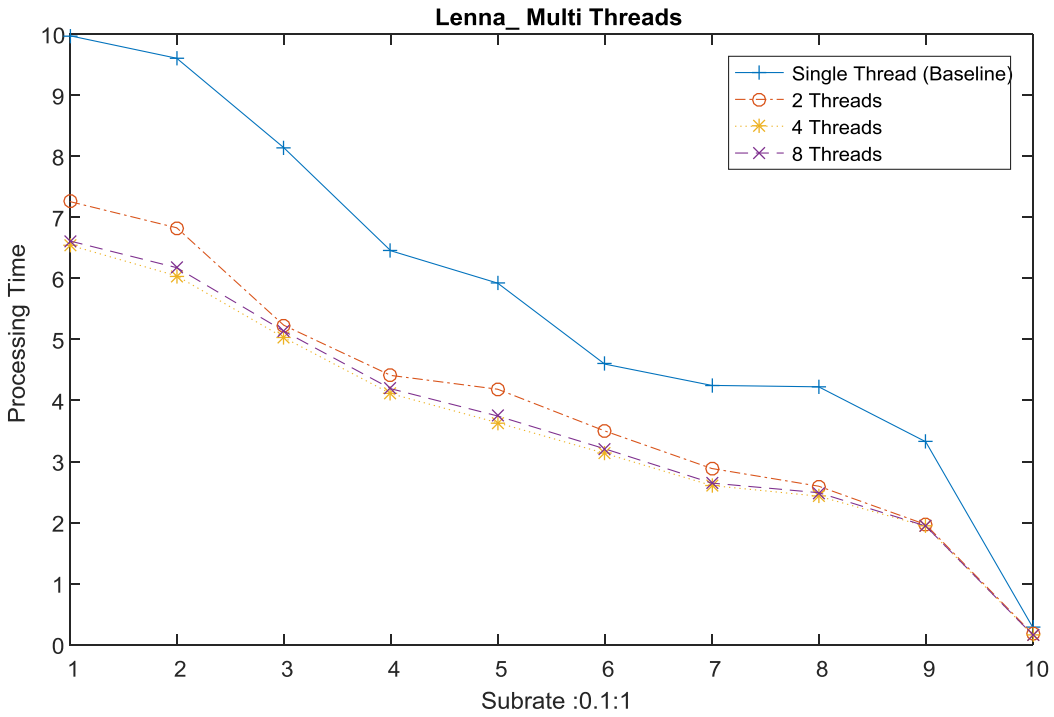

Fig 3: Multi threads computing: Lenna as example for reconstruction time

Table 4. Multi threads computing: Barbara as example for reconstruction time (in Second)

subrate	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
1 Thread (Baseline)	10.9453	7.9804	7.18028	6.8937	5.7913	5.6115	5.4693	4.8856	4.1857	0.81042
2 Threads	7.87528	5.8402	5.23701	4.9966	4.0884	3.8535	3.7689	3.3187	2.8091	0.58889
4 Threads	7.27363	5.4090	4.74991	4.4588	3.7206	3.5924	3.5323	3.0612	2.5866	0.56057
8 Threads	7.54904	5.6359	4.94224	4.6409	3.9460	3.6207	3.5204	3.0913	2.6460	0.57779

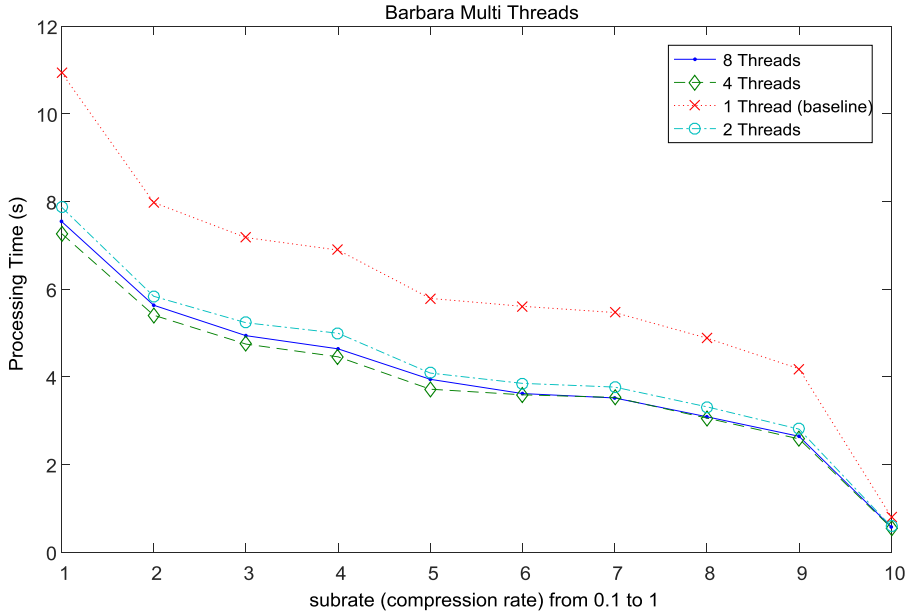

Fig 4:Multi threads computing: Barbara as example for reconstruction time

From above results, we can conclude that for a 4-core computer, using 4 threads is optimal. Therefore, the number of threads should be equal to number of CPU cores.

3.3 GPU Computing

Another approach to accelerate the processing is to use GPU since CS based algorithms include lots of matrix operation.

Originally, GPU (graphics processing unit) was used to accelerate graphics processing. Recently, GPUs are increasingly applied to scientific calculations. Unlike a CPU, which includes a few number of cores (1,2, 4, or 8, etc.), a GPU has a great number of parallel array of integer and floating-point processors as shown in Figure 5. A typical GPU comprises hundreds of these smaller processors [10]. Because of the nature of GPU architecture, using a GPU may speed up the CS reconstruction, by speeding up its matrix processing.

The results in Table 5 and Figure 6 show that, by using GPU, it saves around 50% of processing time. Table 5 shows the reconstruction time comparison for 512×512 grey scale image ‘Lenna’:

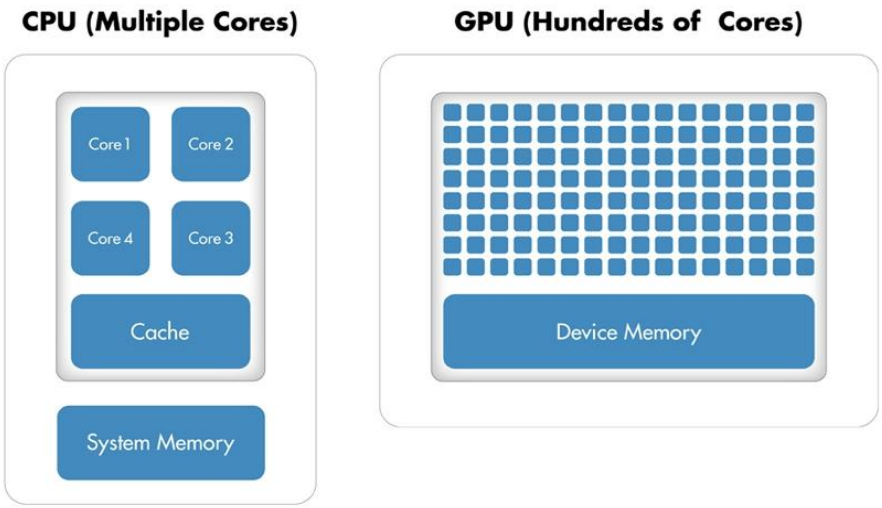

Figure5: CPU and GPU from [11]

Table 5: GPU vs CPU in reconstruction time (in seconds)

subrate	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
CPU_time (baseline)	9.9753	9.6059	8.1422	6.4530	5.9219	4.5965	4.2468	4.2238	3.3244	0.2869
GPU_time	4.2459	3.7375	3.1959	2.2181	1.9722	1.6035	1.2885	1.1070	0.7959	0.0879

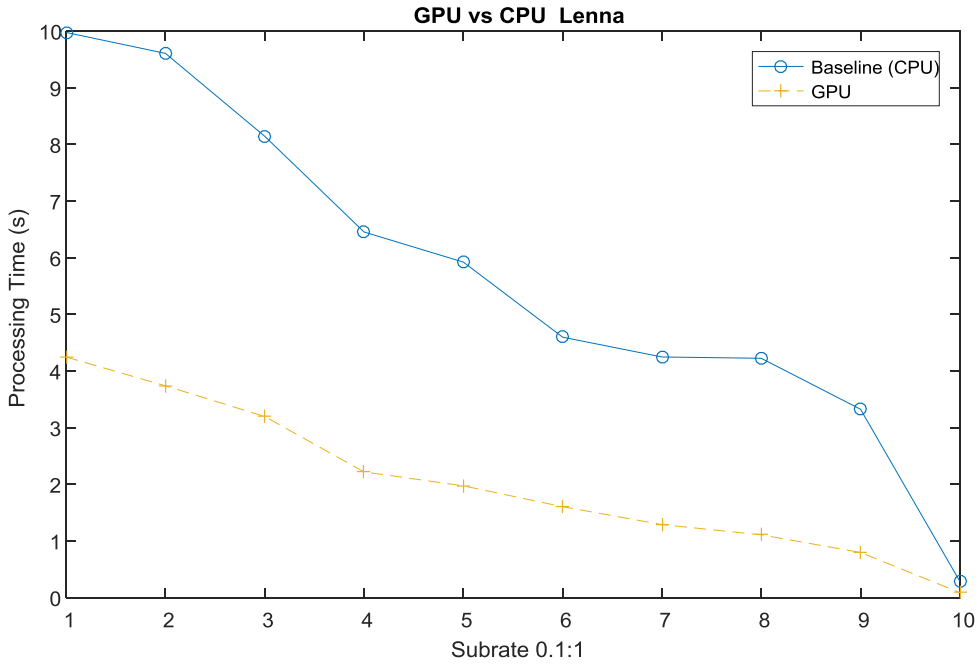

Fig 6:GPU vs CPU in reconstruction time (in seconds)

3.4 Buffering and Optimized Image Size for GPU Computing:

In stream applications, e.g., online video like YouTube, instead of decoding frame by frame, we could buffer several frames and decode in a whole. For example, for 512 by 512 video, we can buffer 4 frames together, to a 1024 by 1024 larger frame, then decode the larger frame. By doing this, it can save lots of decoding time.

If we compare average reconstruction time of a 256×256 slot, the experiment result shows the optimized image size will be 1024×1024. Thus, in stream applications, for example, if the video size is 512×512, we can combine 4 together to a 1024×1024 one, then do the reconstruction to achieve the minimum average reconstruction time.

Considering the quality of service (QOS), we should not buffer a very large size of frames. The experiment result shows 1024 by 1024 may be a near optimal size.

Table 6: Optimal Buffering for GPU (reconstruction time in seconds)

subrate		0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
airplane (256x256)	raw	1.27	1.26	1.23	1.21	1.12	0.938	0.829	0.652	0.548	0.044
	256*256 average	1.27	1.26	1.23	1.21	1.12	0.938	0.829	0.652	0.548	0.044
clock (256x256)	raw	1.29	1.29	1.24	1.25	1.26	1.25	1.25	0.896	1.03	0.042
clock (256×256)	256*256 average	1.29	1.29	1.24	1.25	1.26	1.25	1.25	0.896	1.03	0.042
Barbara (512x512)	raw	2.29	2.38	2.42	2.33	2.04	1.86	1.48	1.31	1.06	0.087
	256*256 average	0.57	0.59	0.606	0.582	0.510	0.467	0.370	0.329	0.265	0.021
Lenna (512x512)	raw	2.31	2.30	2.404	2.347	2.05	1.89	1.52	1.35	1.00	0.0909
	256*256 average	0.578	0.576	0.601	0.586	0.514	0.474	0.381	0.339	0.250	0.022
man (1024x1024)	raw	6.34	6.75	6.91	5.62	5.03	5.23	3.62	3.02	2.35	0.260
Lenna (512×512)	256*256 average	0.396	0.421	0.431	0.351	0.314	0.326	0.226	0.188	0.146	0.0162
man (2048x2048)	raw	22.7	24.5	29.0	21.1	18.0	16.4	14.0	10.5	8.48	0.938
	256*256 average	0.355	0.383	0.453	0.330	0.282	0.256	0.219	0.164	0.132	0.014
subrate		0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
airplane (256x256)	raw	1.27	1.26	1.23	1.21	1.12	0.938	0.829	0.652	0.548	0.044
	256*256 average	1.27	1.26	1.23	1.21	1.12	0.938	0.829	0.652	0.548	0.044
clock (256x256)	raw	1.29	1.29	1.24	1.25	1.26	1.25	1.25	0.896	1.03	0.042
	256*256 average	1.29	1.29	1.24	1.25	1.26	1.25	1.25	0.896	1.03	0.042

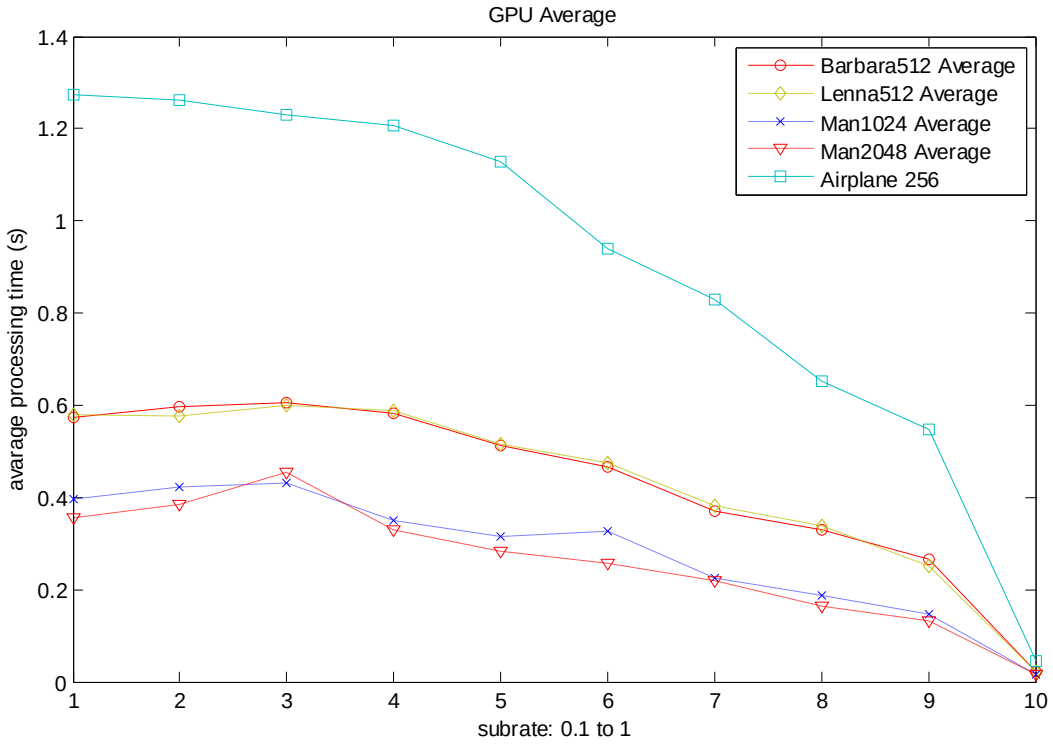

Fig 7. Average GPU Processing Time

4. Conclusion

In this paper, we discussed several methods to accelerate the reconstruction of Compressive Sensed images. Some of these methods could process the reconstruction up to 3-4 times faster.

Comparing those methods, GPU computing may be the most promising. Since the GPU used in the test system, NVidia GTX 950M, is not the most advanced one with very limited RAM. Thus GPU computing seems to be a well fit platform for rapid CS reconstruction.

Future development could implement Video processing using CS with GPU reconstruction and explore optimal algorithms.

References

[1] M. Stanley,"The mobile Internet report," Morgan Stanley Research, New York, NY, Dec. 2009.

[2] E. Candès and T. Tao, "Near optimal signal recovery from random projections: Universal encoding strategies," *IEEE Trans. on Inform. Theory*, vol. 52, no. 12, pp. 5406 - 5425, Dec. 2006.

[3] D. Donoho, "Compressed sensing," *IEEE Trans. on Inform. Theory*, vol. 52, no. 4, pp. 1289 - 1306, Apr. 2006

[4] V. K. Goyal, A. K. Fletcher and S. Rangan, "Compressive sampling and lossy compression," *IEEE Signal Process. Mag*., vol. 25, no. 2, pp. 48-56, Mar. 2008.

[5] S. Mun and J. E. Fowler, "Block compressed sensing of images using directional transforms," in Proc. Int. Conf. on Image Process., Cairo, Egypt, Nov. 2009, pp. 3021-3024.

[6] Y. Zhang, S. Cui and D.R. Vaman, "Compressive sensing based optimized image transmission over wireless gaussian channel," ITIP 2010 Conf., Changsha, China, 2010.

[7] Y. Zhang, S. Cui, and D. R. Vaman, "Optimized compressive image sensing system over mobile wireless noisy channel," Proc. 2012 Int. Conf. on Image Process., Comput. Vision, and Pattern Recognition, Las Vegas, NV, Jul. 2012.

[8] S. Olanigan and L. Cao, "Multi-scale image compressed sensing with optimized transmission", IEEE Workshop on Signal Process. Syst., Taipei City, China, Oct. 2013.

[9] S. Qaisar et al., "Compressive sensing: From theory to applications, a survey," *J. Commun. and Networks*, vol. 15, no. 5, pp. 443-456, Oct., 2013.

[10] NVIDIA Corporation, "GPU_Programming_Guide", Version 2.5, 2006

[11] J. Reese and S. Zaranek, "GPU Programming in Matlab," Mathworks Tech Report, 2011.